Intel

Byte
Feedback
Write to Byte

Newsletter
Sign Up Now

BYTE Categories
Previous Weeks
Columns
Features
Audio

Search:
Byte
Research Center

Print Archives
By Issue By Topic

Resources
Downloads

Java:
Columns & Resources
Books
More Books
Java One Audio Report

History Of Byte:
Part I Part II Part III

BYTE Humor
Ian Shoales' Page

About Us
Byte Editorial Staff
Feedback
Sales Staff
Privacy Policy

TechWeb Sites
Byte.com
CMPmetrics
Data Communications
File Mine
InformationWeek
InternetWeek

ARTICLES

Beyond Pentium II

December 1997 / Cover Story / Beyond Pentium II

Here's the first detailed look at the new breakthrough microprocessor architecture from Intel and Hewlett-Packard -- and what it will mean for developers and users.

Tom R. Halfhill

For Intel and Hewlett-Packard, the year 2000 isn't a problem -- it's an opportunity. In late 1999, Intel plans to ship Merced, the first microprocessor based on a next-generation architecture jointly conceived by the two companies. Although this 64-bit architecture builds on years of research at Intel, HP, other companies, and universities, it is radically different from anything ever attempted on a mass-market scale. Whether it succeeds or fails, one thing is certain: It will change the computer industry forever.

Known as Intel Architecture-64 (IA-64), the new definition breaks clean with the past in a startling fashion. IA-64 is emphatically not a 64-bit extension of Intel's 32-bit x86 architecture. Nor is it an adaptation of HP's 64-bit PA-RISC architecture. IA-64 is something completely different -- a forward-looking architecture that uses long instruction words (LIW), instruction predication, branch elimination, speculative loading, and other advanced techniques to extract more parallelism from program code.

Although Intel and HP promise backward compatibility with today's x86 and PA-RISC software, they're still withholding those details. Compatibility will not be trivial because IA-64 goes far beyond the 32-bit extensions that Intel added to the x86 in 1985, as well

Network Computing
Planet IT
TechShopper
TechWeb News
Tele.com
WebTools
Winmag.com

extensions that Intel added to the x86 in 1985, as well as the 64-bit extensions that HP added to PA-RISC in 1996. It's worth remembering that the x86's much less radical transition from 16 to 32 bits has so far taken 12 years and still is not complete.

The migration to IA-64 won't concern most users in the short term, however, because Intel says it's designing Merced for high-end servers and workstations. Merced is not for mainstream PCs. In fact, Intel says IA-64 won't replace the x86 "for the foreseeable future." It's likely that Intel (and other x86 vendors) will continue to introduce new generations of x86 chips for years to come.

Superwide CPUs

Before plunging neck-deep into the technical innards of IA-64, it is critical to understand why Intel and HP are gambling their futures on such sweeping changes. It comes down to this: Intel and HP think CISC and RISC are running out of gas.

Intel' s x86 is an ancient CISC architecture that dates back to 1978. In those days, CPUs were scalar devices (able to execute only one instruction at a time) with little or no pipelining. Chips had tens of thousands of transistors. HP's PA-RISC architecture dates back to 1986, when superscalar (multiple-instruction) pipelines were just starting to sprout and chips had hundreds of thousands of transistors. In the late 1990s, leading-edge processors have millions of transistors.

By the time Merced makes its debut, Intel will be rolling out the next generation of process technology beyond today's latest 0.25-micron process -- 0.18-micron. Even the first IA-64 chips will have tens of millions of transistors. Future generations will have hundreds of millions.

CPU architects are eager to put those transistors to work. They want to design chips with many more functional units -- that execute more instructions in parallel -- but they're hitting a wall of complexity. As they add more units to make the CPU "wider ," they must also add more control circuitry to dispatch those instructions to the units. Today's best CPUs can't retire more than four instructions per clock and already waste too much silicon on this purely bookkeeping logic.

At the same time, the sequential nature of program code and the frequency of branches make it increasingly difficult to dispatch instructions in parallel. Today's CPUs devote enormous amounts of logic to minimizing branch penalties and to extracting as much hidden parallelism as possible from the code. CPUs reorder instructions on the fly, predict where branches will jump, and speculatively execute instructions beyond the branches. If the CPU guesses wrong, it must discard the speculative results, flush the pipelines, and reload the correct instructions -- paying a heavy penalty in lost cycles. CPUs that theoretically can retire four instructions per clock actually average less than two per clock.

To compound these problems, memory chips haven't come close to matching the s oaring clock speeds of CPUs. When Intel designed the first x86 chip, CPUs could fetch data from memory as fast as they could process the data. Today, CPUs spend hundreds of clock cycles waiting for data to arrive from memory, despite having large, fast caches.

Intel and HP are addressing all these problems. Here's what they divulged in two lengthy interviews with BYTE:

- The new IA-64 format packs three instructions into a single 128-bit-long bundle for faster processing. Usually this is called LIW encoding, but Intel shuns that label, saying LIW has "negative connotations." For the same reason, Intel does not like to describe the individual instructions as "RISC-like," even though they are fixed-length and presumably optimized for single-cycle execution in a core that doesn't need microcode. Intel calls its new LIW technology Explicitly Parallel Instruction Computing, or EPIC.

At any rate, IA-64 is nothing like the x86. An x86 instruction is a single unit that can vary from 8 to 108 bits long, and the CPU must tediously decode each instruction while scanning for the instruction boundaries. (See the figure "IA-64 Instruction Format" .)

- Each 128-bit IA-64 bundle contains a template of several bits -- placed there by the compiler -- that explicitly tells the CPU which instructions it can execute in parallel. No longer must the CPU hurriedly analyze the instruction stream at run time to uncover hidden parallelism. Instead, the compiler identifies the parallelism and binds this information into the machine code.

Each instruction contains three 7-bit general-purpose register (GPR) fields, and the fields are specific to integer and floating-point (FP) instructions. That means IA-64 processors will have 128 integer-type GPRs and 128 FP registers. All are programmer-visible, random-access registers. Compare that to the constipated x86, which has eight integer GPRs and an eight-entry FP stack. IA-64 processors can be much wider and will stall less often due to false dependencies (e.g., shortages of registers).

- IA-64 compilers will use a technique called *predication* to remove the penalties caused by mispredicted branches and the need to jump over blocks of code beyond branches. When the CPU encounters a predicated branch at run time, it will begin executing the code along all destinations of the branch, exploiting as much parallelism as possible. When the CPU discovers the actual branch outcome, it stores the valid results and discards the others.
- IA-64 compilers will scan the source code to find upcoming loads from memory, then will add a speculative load instruction and a speculative check instruction. At run time, the first instruction loads the data from memory before the program needs it. The second instruction verifies the load before letting the program use the data. Speculative loading helps hide the long latencies of memory accesses and helps increase parallelism.

One implication of IA-64 is that compilers will have to be a lot smarter about the microarchitectures of the CPUs they target. Existing chips -- even RISC chips with optimized compilers -- do much more optimizing at run time than IA-64 chips will. IA-64 transfers the job of optimizing the instruction stream to the compiler. Successive generations of IA-64 processors will run older IA-64 software, but the software might not run at top speed until it's recompiled. In the IA-64 age, developers might have to ship multiple binaries to get the best performance on a broad installed base of IA-64 systems.

Another impact will be code expansion. IA-64 instructions are longer than 32-bit RISC instructions -- about 40 bits each. Just by recompiling existing code, developers will almost certainly see their programs grow larger. And those programs will probably take longer to compile because IA-64 demands a lot more work from the compiler, as we'll see in a moment. Intel and HP say they're already working with tool vendors to help them revise their products.

Dis appearing Branches

Predication is a prime example of the new burden shifted onto compilers. This technique is central to IA-64's branch elimination and parallel instruction scheduling.

Normally, a compiler turns a source-code branch statement (such as IF-THEN-ELSE) into alternate blocks of machine code arranged in a sequential stream. Depending on the outcome of the branch, the CPU will execute one of those *basic blocks* by jumping over the others. Modern CPUs try to predict the outcome and speculatively execute the target block, paying a heavy penalty in lost cycles if they mispredict. The basic blocks are small, often two or three instructions, and branches occur about every six instructions. The sequential, choppy nature of this code makes parallel execution difficult.

When an IA-64 compiler finds a branch statement in the source code, it analyzes the branch to see if it's a candidate for predication. Compilers can't predicate every branch: Dynamic method calls that the CPU won't discover until run time are one obvious exception. In other cases, predication might cost more cycles than it saves. Compilers will have to be clever about this.

If the compiler determines that predication makes sense, it marks all the instructions that represent each path of the branch with a unique identifier called a *predicate* . For example, the compiler might tag each instruction that follows the TRUE condition with the predicate P1; and it might tag each instruction that follows the FALSE condition with the predicate P2. IA-64 defines a 6-bit field in each instruction to store this predicate. Thus, there are 64 unique predicates available at one time. Any number of instructions that share a particular branch path will share the same predicate.

After tagging the instructions with predicates, the compiler determines which instructions the CPU can execute in parallel. Again, this requires the compiler to know a lot about the CPU's microarchitecture, because different IA-64 chips will have different numbers and types of functional units. Also, of course, the compiler must watch out for data dependencies -- an operation that needs the result of a previous operation cannot execute in parallel with that operation. But the compiler will almost always find some parallelism by pairing instructions from different branch outcomes because they represent independent paths through the program.

Now the compiler can start assembling the machine-code instructions into 128-bit bundles of three instructions each. The bundle's template field not only identifies which instructions in the bundle can execute independently but also which instructions in the following bundles are independent. So if the compiler finds 16 instructions that have no mutual dependencies, it could package them into six different bundles (three in each of the first five bundles, and one in the sixth) and flag them in the templates.

The bundled instructions don't have to be in their original program order, and they can represent entirely different paths of a branch. Also, the compiler can mix dependent and independent instructions together in a bundle, because the template keeps track of which is which. Unlike some previous very-long instruction word (VLIW) architectures, IA-64 does not insert null-operation instructions (NOPS) to fill slots in the bundles.

At run time, the CPU scans the templates, picks out the instructions that do not have mutual dependencies, and then dispatches them in parallel to the functional units. The CPU then schedules instructions that are dependent according to their requirements.

When the CPU finds a predicated branch, it doesn't try to predict which way the branch will fork, and it doesn't jump over blocks of code to speculatively execute a predicted path. Instead, the CPU begins executing the code for every possible branch outcome. In effect, there is no branch at the machine level. There is just one unbroken stream of code that the compiler has rearranged in the most parallel order.

At some point, of course, the CPU will eventually evaluate the compare operation that corresponds to the IF-THEN statement. Now the CPU knows the outcome. Let's say the condition is TRUE, so the valid path is predicate P1, not P2. The 6-bit predicate field in each IA-64 instruction refers to a set of 64 predicate registers (P0-P63), and each register is 1 bit wide. The CPU will store a 1 in predicate register P1 to represent TRUE, and it will store a 0 in predicate register P2 to represent FALSE.

By this time, the CPU has probably executed some instructions from both possible paths. But it hasn't stored the results yet. Before taking that final step, the CPU checks each instruction's predicate register. If the register contains a 1, the instruction is valid, so the CPU retires the instruction and stores the result. If the register contains a 0, the instruction is invalid, so the CPU discards the result. (See the figure "How Predication Works" .)

Predication effectively remove s the negative impact of a branch at the machine level while preserving branch behavior. Again, it can't remove every branch. However, if the compiler cannot predicate a branch, or chooses not to, an IA-64 processor will behave much like a conventional processor: It will try to predict which way the branch will turn, and it may speculatively execute some instructions along the predicted path. Simulations of this strategy indicate that predication can eliminate more than half of the branches in a typical program -- and therefore reduce by half the number of potential mispredictions.

This has several benefits. It reduces code fragmentation at the machine level because the compiler can merge small basic blocks into larger blocks that branches don't chop up. Larger blocks give the compiler more freedom to rearrange instructions for parallel execution. It also drastically reduces the hazard of mispredicted branches because every branch doesn't require the CPU to play fortune-teller. And it keeps the function al units busy because the CPU can dispatch more instructions in parallel.

The downside of predication is that the CPU always executes instructions it's going to throw away. Whether the predicated condition evaluates TRUE or FALSE, the CPU does perform redundant work. The trick, of course, is to make sure the CPU saves more clock cycles than it wastes. Clearly, predication assumes that IA-64 compilers will be smart and that IA-64 processors will be very wide superscalar chips with lots of resources to spare. When you're rich, you can afford to spend lavishly.

He Ain't Heavy, He's My Data

Another key feature of IA-64 is speculative loading. Not only will this allow IA-64 processors to load data from memory before the program needs it, it will also postpone the reporting of exceptions if the load is not legal. In geekspeak, this technique allows the CPU to *hoist* the load operation higher in the instruction stream -- in some cases, even above a branch.

The goal is to separate the loading of data from the use of that data. By paying attention to this, the CPU won't have to twiddle its thumbs while waiting for data in slow memory to show up. Like predication, it's a combination of compile-time and run-time optimizations.

First, the compiler analyzes the program, looking for any operations that will need data from memory. Whenever possible, the compiler inserts a speculative load instruction at an earlier point in the instruction stream, well ahead of the operation that will actually use the data. The compiler also inserts a matching speculative check instruction immediately before the particular operation that will use the data. At the same time, of course, the compiler rearranges the surrounding instructions so that the CPU can dispatch them in parallel.

At run time, the CPU encounters the speculative load instruction first and tries to retrieve the data from memory. Here's where an IA-64 processor differs from a regular processor. Sometimes the load will be invalid -- it might belong to a block of code beyond a branch that has not executed yet. A traditional CPU would immediately trigger an exception. If the program could not handle the exception, it would likely crash.

But an IA-64 processor won't immediately report an exception if the load is invalid. Instead, the CPU postpones the exception until it encounters the speculative check instruction that matches the speculative load. Only then does the CPU report the exception. By then, however, the CPU has resolved the branch that led to the exception in the first place. If the path to which the load belongs turns out to be invalid, then the load is also invalid, so the CPU goes ahead and reports the exception. But if the load is valid, it's as if the exception never happened. (See the figure "How Speculative Loading Works" .)

Speculative loading is similar to the TRY-CATCH structures in some programming languages, except that it works at the machine level. In Java, for instance, a TRY statement will attempt a risky operation, such as opening a file. If TRY succeeds, the program continues normally. If the system can't open the file and throws an exception, CATCH grabs it and stops the program from crashing. IA-64's speculative check is a safety valve for exceptions, like CATCH.

This technique, combined with predication, gives the compiler much more flexibility to reorder instructions and increase parallelism. The ability to hoist loads above branches is particularly powerful. Since branches typically occur about every six instructions, they would severely inhibit IA-64's ability to load data from memory long before it's needed. It would be almost impossible to retrofit an existing architecture with these features because the compiler and the CPU must collaborate to make it happen.

Beyond RISC

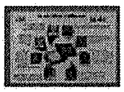
In the heady days of the 1980s, some RISC engineers ridiculed CISC and foretold the doom of the x86 family. Unfortunately for them, the penalty for underestimating Intel is even greater tha n the penalty for mispredicting branches. Business and technology are two different things. RISC might be technically superior to CISC, but Intel's vast resources and the momentum of DOS and Windows have kept the x86 competitive.

Now, Intel says RISC is running out of gas. Could it be that Intel might be making the same mistake that RISC fans made in the 1980s? Will RISC stave off the IA-64 challenge?

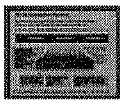
It's too early to tell. However, it's doubtful that RISC vendors can tap the same depth of resources that keeps the x86 alive. The most popular RISC architecture (not counting embedded applications) is the PowerPC. And the only high-volume PowerPC vendor is Apple, a company struggling for survival. Without more business, how long can RISC vendors justify the expensive research and development it takes to battle Intel?

IA-64 chips are still two years away. Intel's competitors -- from both the RISC and the CISC camps -- have that much time to take the offensive.

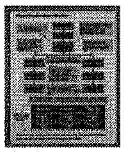
IA-64: What's Different

illustration_link (53 Kbytes) 

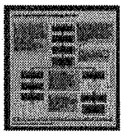
IA-64 Instruction Format

illustration_link (39 Kbytes) 

How Predication Works

illustration_link (37 Kbytes) 

How Speculative Loading Works

illustration_link (39 Kbytes) 

Tom R. Halfhill is a BYTE senior editor based in San Mateo, California. You can reach him at thalfhill@byte.com .

Do you want Integrated Products?
TAPE STORAGE
IBM

Byte
Feedback
Write to Byte

Newsletter
Sign Up Now

BYTE Categories
Previous Weeks
Columns
Features
Audio

Search:
Byte
Research Center

Print Archives
By Issue By Topic

Resources
Downloads

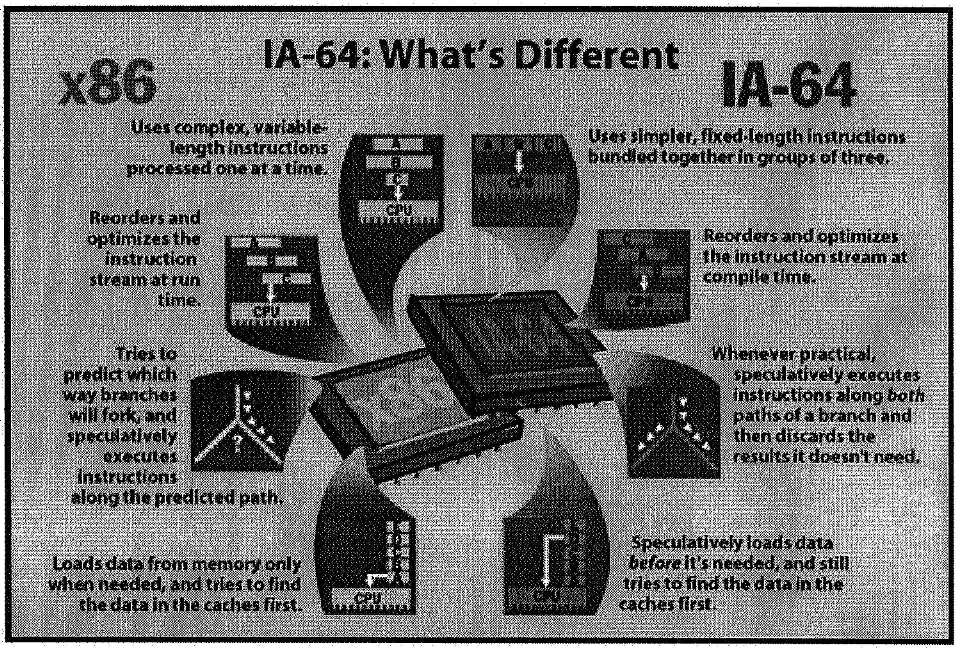
Java:
Columns & Resources
Books
More Books
Java One Audio Report

History Of By
Part I Part I

BYTE HumorSearch
Ian Shoales' Page

About Us
Byte Editorial Staff
Feedback
Sales Staff
Privacy Policy

IA-64: What's Different


IA-64: What's Different
x86
IA-64
Uses complex, variable-length instructions processed one at a time.
Uses simpler, fixed-length instructions bundled together in groups of three.
Reorders and optimizes the instruction stream at run time.
Reorders and optimizes the instruction stream at compile time.
Tries to predict which way branches will fork, and speculatively executes instructions along the predicted path.
Whenever practical, speculatively executes instructions along *both* paths of a branch and then discards the results it doesn't need.
Loads data from memory only when needed, and tries to find the data in the caches first.
Speculatively loads data *before* it's needed, and still tries to find the data in the caches first.
CPU

Up Level Subscribe

Introducing New Disk Storage for [illegible]
IBM

Sponsored by:

redefine your career

TechWeb Sites
Byte.com
CMPmetrics
Data Communications
File Mine
InformationWeek
InternetWeek

CLICK HERE
In the ever changing Communications Industry, tures, news and facts
we have the latest in-depth product news ie telecom front?
Site sponsored by Lucent Technologies ed by Lucent Technologies

Byte

Feedback
Write to Byte

Newsletter
Sign Up Now

BYTE Categories
Previous Weeks
Columns
Features
Audio

Search:
Byte
Research Center

Print Archives
By Issue By Topic

Resources
Downloads

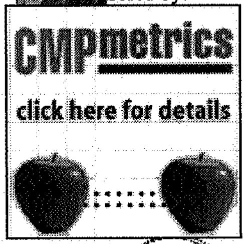
Java:
Columns & Resources
Books
More Books
Java One Audio Report

History Of Byte:
Part I Part II Part III

BYTE Humor
Ian Shoales' Page

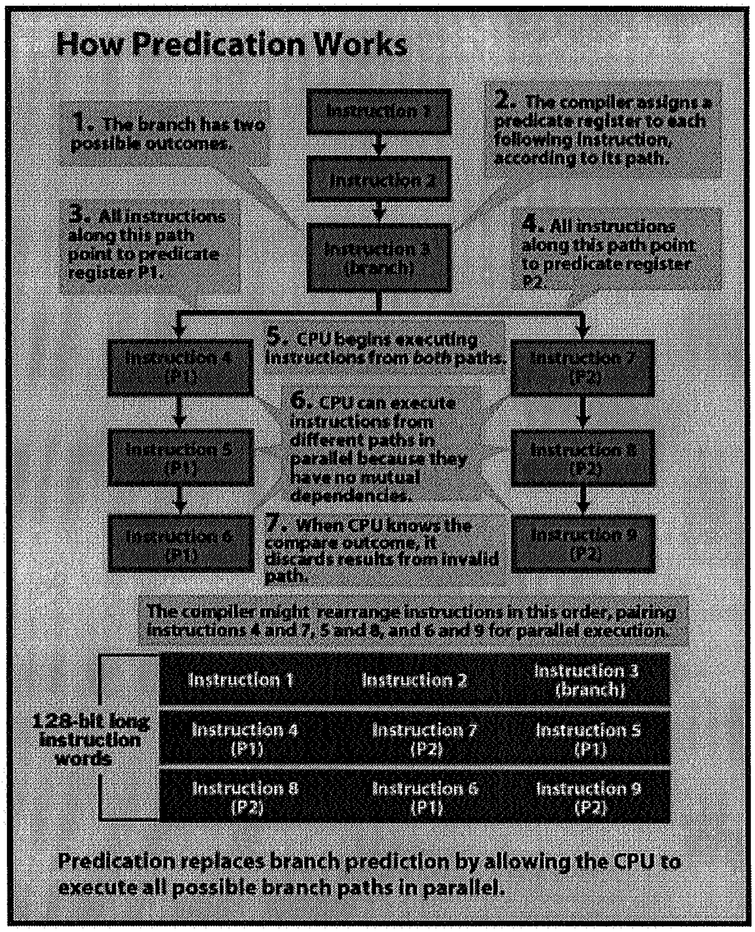
About Us
Byte Editorial Staff
Feedback
Sales Staff
Privacy Policy

sored by:



TechWeb Sites
Byte.com
CMPmetrics
Data Communications
File Mine
InformationWeek
InternetWeek

How Predication Works


How Predication Works
Instruction 1
Instruction 2
Instruction 3 (branch)
1. The branch has two possible outcomes.
2. The compiler assigns a predicate register to each following instruction, according to its path.
3. All instructions along this path point to predicate register P1.
4. All instructions along this path point to predicate register P2.
Instruction 4 (P1)
Instruction 5 (P1)
Instruction 6 (P1)
Instruction 7 (P2)
Instruction 8 (P2)
Instruction 9 (P2)
5. CPU begins executing instructions from both paths.
6. CPU can execute instructions from different paths in parallel because they have no mutual dependencies.
7. When CPU knows the compare outcome, it discards results from invalid path.
The compiler might rearrange instructions in this order, pairing instructions 4 and 7, 5 and 8, and 6 and 9 for parallel execution.
128-bit long instruction words
Instruction 1
Instruction 2
Instruction 3 (branch)
Instruction 4 (P1)
Instruction 7 (P2)
Instruction 5 (P1)
Instruction 8 (P2)
Instruction 6 (P1)
Instruction 9 (P2)
Predication replaces branch prediction by allowing the CPU to execute all possible branch paths in parallel.

Performance Characterization of the Pentium® Pro Processor

Dileep Bhandarkar and Jason Ding
Intel Corporation
Santa Clara, California, USA

Abstract

In this paper, we characterize the performance of several business and technical benchmarks on a Pentium® Pro processor based system. Various architectural data are collected using a performance monitoring counter tool. Results show that the Pentium Pro processor achieves significantly lower cycles per instruction than the Pentium processor due to its out of order and speculative execution, and non-blocking cache and memory system. Its higher clock frequency also contributes to even higher performance.

Keywords: Pentium® Pro processor, computer architecture, performance evaluation, workload characterization, out of order execution, speculative execution, SPEC CPU95, SYSmark/NT.

1. Introduction

The Intel Pentium® Pro processor was disclosed in February 1995 at ISSCC [1] and began shipping later that year. The micro-architecture implements several new features that are not found in previous implementations of the Intel Architecture. This paper analyzes the major performance characteristics of several business and technical benchmarks on a Pentium Pro processor based system. Measurements were performed using the built-in performance counters of the processor. Results are presented for cycles per instruction, cache miss statistics, branch prediction statistics, speculative execution, stall cycles, and other micro-architecture features.

Current literature contains numerous papers that present simulations of various machine structures. Often these simulations do not model the entire machine accurately or only use traces of parts of popular benchmarks. We present measured characteristics of a recent microprocessor to allow researchers to calibrate their theoretical results. The paper presents a lot of raw data and some analysis wherever possible. In a modern superscalar out-of-order processor, it is not always possible to derive precise cause-effect relationships.

Some of the results presented here are consistent with the behavior of SPEC benchmarks on other architectures, e.g., the FP benchmarks have lower Icache misses and higher Dcache misses than the integer benchmarks. Other measurements (branch mispredicts, micro-op statistics, and speculative execution) provide insight into the inner workings of the Pentium Pro processor.

2. Architectural Features of the Pentium® Pro Processor

The Intel Pentium Pro processor implements dynamic execution using an out-of-order, speculative execution engine, with register renaming of integer, floating point and flags variables, multiprocessing bus support, and carefully controlled memory access reordering. The flow of Intel IA-32 Architecture instructions is predicted and these instructions are decoded into micro-operations (uops), or series of uops. These uops are register-renamed, placed into an out-of-order speculative pool of pending operations, executed in dataflow order (when operands are ready), and retired to permanent machine state in source program order. This is accomplished with one general mechanism to handle unexpected asynchronous events such as mispredicted branches, instruction faults and traps, and external interrupts. Dynamic execution, or the combination of branch prediction, speculation and micro-dataflow, is the key to its high performance.

Figure 1 shows a block diagram of the processor. The basic operation of the microarchitecture is as described in the ISSCC paper [1]:

1. The 512 entry Branch Target Buffer (BTB) helps the Instruction Fetch Unit (IFU) choose an instruction cache line for the next instruction fetch. Icache line fetches are pipelined with a new instruction line fetch commencing on every CPU clock cycle.

2. Three parallel decoders (ID) convert multiple Intel Architecture instructions into multiple sets of uops each clock cycle. Instructions that require more than 4 uops are handled by the microinstruction sequencer.

0-8186-7764-3/97 $10.00 © 1997 IEEE

3. The sources and destinations of up to 3 uops are renamed every cycle to a set of 40 physical registers by the Register Alias Table (RAT), which eliminates register re-use artifacts, and are forwarded to the 20-entry Reservation Station (RS) and to the 40-entry ReOrder Buffer (ROB).

4. The renamed uops are queued in the RS where they wait for their source data - this can come from several places, including immediates, data bypassed from just-executed uops, data present in a ROB entry, and data residing in architectural registers (such as EAX).

5. The queued uops are dynamically executed according to their true data dependencies and execution unit availability (integer, FP, address generation, etc.). The order in which uops execute in time has no particular relationship to the order implied by the source program.

6. Memory operations are dispatched from the RS to the Address Generation Unit (AGU) and to the Memory Ordering Buffer (MOB). The MOB ensures that the proper memory access ordering rules are observed.

7. Once a uop has executed, and its destination data has been produced, that result data is forwarded to subsequent uops that need it, and the uop becomes a candidate for "retirement".

8. Retirement hardware in the ROB uses uop timestamps to reimpose the original program order on the uops as their results are committed to permanent architectural machine state in the Retirement Register File (RRF). This retirement process must observe not only the original program order, it must correctly handle interrupts and faults, and flush all or part of its state on detection of a mispredicted branch. When a uop is retired, the ROB writes that uop's result into the appropriate RRF entry and notifies the RAT of that retirement so that subsequent register renaming can be activated. Up to 3 uops can be retired per clock cycle.

The Pentium Pro processor implements a 14-stage pipeline capable of decoding 3 instructions per clock cycle. The in-order front end has 8 stages. The out-of-order core has 3 stages, and the in-order retirement logic has 3 stages. For an integer op, say a register-to-register add, the execute phase is just one cycle. Floating point adds have a latency of 3 cycles, and a throughput of 1 per cycle. FP multiply has a latency of 5 cycles and a repetition rate of 1 every 2 cycles. Integer multiply has a latency of 4 cycles and a throughput of 1 every cycle. Loads have a latency of 3 cycles on a Dcache hit. FDIV is not pipelined; it takes 17 cycles for single, 32 cycles for double, and 37 cycles for extended precision. The processor includes separate data and instruction L1 caches (each of which is 8KB). The instruction cache is 4-way set associative, and the data cache is dual ported, non-blocking, 2-way set associative supporting one load and one store operation per cycle. Both instruction and data cache line sizes are 32 byte wide. More details of the microarchitecture can be found elsewhere [2].

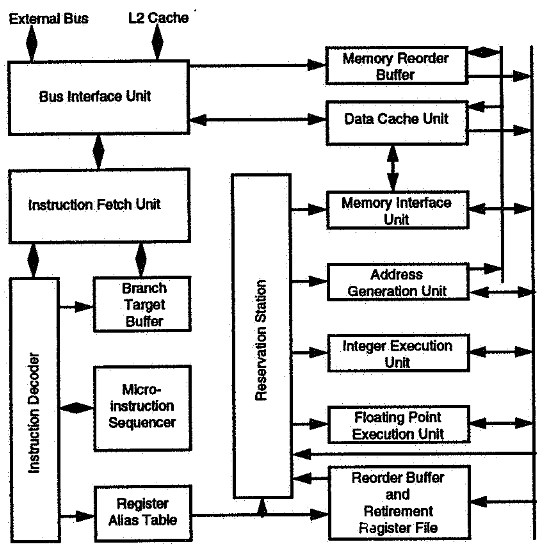


Figure 1 Pentium® Pro Processor Block Diagram

The secondary cache (L2 cache), which can be either 256KB or 512KB in size, is located on a separate die (but within the same package). The L2 cache is 4-way set associative unified non-blocking cache for storage of both instructions and data. It is closely coupled with a dedicated 64-bit full clock-speed backside cache bus. The L2 cache line is also 32 bytes wide. The L2 cache fills the L1 cache in a full frequency 4-1-1-1 cycle transfer burst transaction. The processor connects to I/O and memory via a separate 64-bit bus that operates at either 60 or 66 MHz. The bus implements a pipelined demultiplexed design with up to 8 outstanding bus transactions.

3. Performance Monitoring Facility

The Pentium® Pro processor implements two performance counters[3]. Each performance counter has an associated event select register that controls what is counted. The counters are accessed via the RDMSR and WRMSR instructions. Table 1 shows a partial list of performance metrics that can be measured by selecting the two events to be monitored.

Table 1. Pentium® Pro Processor Counter based Performance Metrics

Performance Metric	Numerator Event	Denominator Event
Data references per instruction	DATA_MEM_REFS	INST_RETIRED
L1 Dcache misses per instruction	DCU_LINES_IN	INST_RETIRED
L1 Icache misses per instruction	L2_IFETCH	INST_RETIRED
ITLB misses per instruction	ITLB_MISS	INST_RETIRED
Istalls cycles per instruction	IFU_MEM_STALL	INST_RETIRED
L1 cache misses per instruction	L2_RQSTS	INST_RETIRED
L2 cache misses per instruction	L2_LINES_IN	INST_RETIRED
L2 Miss ratio	L2_LINES_IN	L2_RQSTS
Memory transactions per instruction	BUS_TRAN_MEM	INST_RETIRED
FLOPS per instruction	FLOPS	INST_RETIRED
UOPS per instruction	UOPS_RETIRED	INST_RETIRED
Speculative execution factor	INST_DECODED	INST_RETIRED
Branch frequency	BR_INST_RETIRED	INST_RETIRED
Branch mispredict ratio	BR_MISS_PRED_RETIRED	BR_INST_RETIRED
Branch taken ratio	BR_TAKEN_RETIRED	BR_INST_RETIRED
BTB miss ratio	BTB_MISSES	BR_INST_DECODED
Branch Speculation factor	BR_INST_DECODED	BR_INST_RETIRED
Resource stalls per instruction	RESOURCE_STALLS	INST_RETIRED
Cycles per instruction	CPU_CLK_UNHALTED	INST_RETIRED

Table 2. Basic Characteristics of Systems

Processor	Intel Pentium® Pro Processor	Intel Pentium® Processor
CPU Core Frequency	150 MHz	120 MHz
Bus Frequency	60 MHz	60 MHz
Data bus	64-bit	64-bit
Address bus	36-bit	32-bit
On-chip L1 cache	8 KB data, 8 KB instruction	8 KB data, 8 KB instruction
Off-chip L2 cache	4-way 256 KB	512 KB (Dell), 256 KB (Gateway)
L2 cache timing	4-1-1-1 @ 150 MHz CPU freq.	3-1-1-1 @ 60 MHz bus frequency
System Chip Set	82450GX/KX	82430FX
Memory timing (bus cycles)	14-1-1-1 (4-way interleaving) 14-2-2-2 (2-way interleaving) 14-4-4-4 (no interleaving)	13-3-3-3 (Fast Page Mode DRAM) 13-2-2-2 (EDO DRAM)
Basic Pipeline	14 stages	5 stages
Superscalar	3-way	2-way
Execution units	5	3
Branch prediction	4-way 512 entry BTB, 4-bit history, 2 level adaptive	4-way 256 entry BTB, 2-bit history
Execution model	Out of order	In order
Speculative Execution	Yes	No
McCalpin Streams Memory Bandwidth	140 MB/sec (4-way interleaving) 128 MB/sec (2-way interleaving) 97 MB/sec (no interleaving)	82 MB/sec (Gateway 2000 P120)
SYSmark/NT rating	497 (Digital Celebris* XL6150)	294 (Gateway 2000 P120)
SPECint95	6.08 (Intel Alder System)	3.53 (Dell Dimension XPS P120)
SPECfp95	5.42 (Intel Alder System)	2.92 (Dell Dimension XPS P120)

4. Comparing the Pentium® and Pentium® Pro Processors

This section compares the basic performance characteristics of the Pentium [4] and Pentium Pro processors. Table 2 compares the basic characteristics of these two processors. We chose the 120 MHz Pentium and the 150 MHz Pentium Pro processors because both are fabricated in the same 0.6μ technology and use a 60 MHz external bus. For the SPEC benchmarks, the Pentium system was a Dell Dimension XPS P120 with a 512KB pipelined burst L2 cache, and the Pentium Pro system was an Intel Alder system with a 150MHz Pentium Pro CPU with 256KB L2 cache and a 4-way interleaved memory.

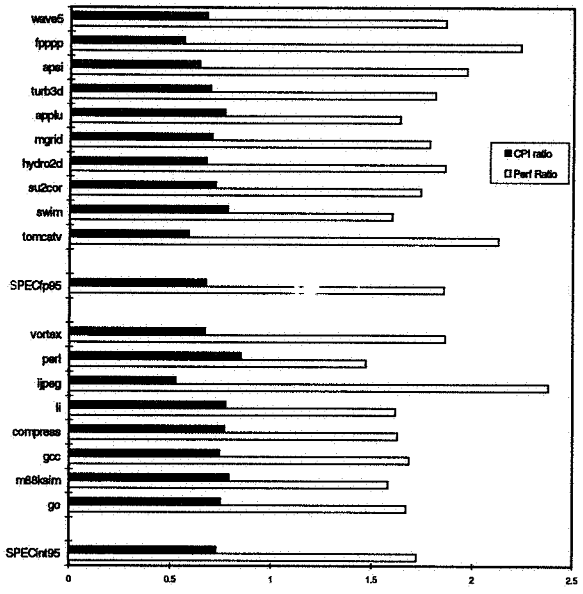


Figure 2 Performance Comparison of Pentium® and Pentium® Pro Processors on SPEC95

Figure 2 shows the SPECratios and the cycles per instruction of the Pentium Pro processor relative to the Pentium processor for the SPEC95 benchmark suite for the two systems. The SPEC* results were obtained using Intel Reference Compiler 2.3 Beta on UnixWare* v2.0 on an Intel Alder system. The Pentium Pro processor achieves CPIs 15% to 50% lower than the Pentium processor, in spite of the fact that it uses a design style that emphasizes a fast clock frequency. Designs that emphasize clock frequency generally result in deeper pipelines and longer CPI. The Pentium Pro processor design attempts to increase frequency while reducing CPI, without being overly focused on optimal CPI or fastest clock [5].

The Pentium Pro processor runs at 1.6 to 2.4 times the performance of the Pentium processor on the SPEC95 suite[6], achieving 70% higher SPECint95 and 85% higher SPECfp95. This performance comes from a 25% faster clock frequency and a 15 to 50% reduction in CPI compared to the Pentium processor. The Pentium Pro processor can issue up to 3 instructions every clock cycle, while the Pentium processor can issue only two. The out of order execution model of the Pentium Pro processor also allows useful work to proceed while prior operations are stalled, thereby lowering the CPI.

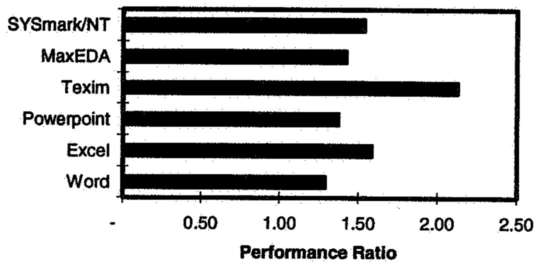


Figure 3 Performance Comparison of Pentium® and Pentium® Pro Processors on SYSmark/NT

Figure 3 shows the performance of a 150 MHz Pentium Pro processor based Digital Celebris 6150 compared to a 120 MHz Pentium processor based Gateway 2000 P120 system on the SYSmark* for Windows NT suite from BAPCO[7], which contains project management software (Welcom Software Technology Texim* Project 2.0e), computer-aided PCB design tool (Orcad MaxEDA* 6.0) and Microsoft Office* applications for word processing (Word* 6.0), presentation graphics (PowerPoint* 4.0), and spreadsheets (Excel 5.0). Both system had a 256KB L2 cache, but the Pentium Pro processor had a faster L2 cache (4-1-1-1 timing at full CPU clock frequency) on its dedicated L2 cache bus. The Pentium Pro processor runs 29% to 113% faster than the Pentium processor, with an overall 54% higher SYSmark score.

These results are slightly lower than the SPEC95 results because the desktop applications in the SYSmark benchmark perform some I/O operations that include wait times that do not scale with CPU performance. The SPEC benchmarks used compilers that generate binaries that are optimized for each target machine. The SYSmark/NT benchmarks use old binaries that are not optimized for the Pentium Pro processor. These benchmarks have large working set sizes for code and data and also contain many context switches. The SYSmark/NT benchmarks also result in higher L2 cache misses as shown in a later section.

5. Detailed Characterization of SPEC CPU95 Benchmarks

This section presents a detailed characterization of Pentium® Pro processor running the SPEC CPU95 suite. The performance counter measurements presented in the rest of this paper were done on a Digital Celebris XL6200 running Microsoft Windows* NT Workstation Version 3.51. The central processor in the Digital Celebris XL6200 is a 200MHz Pentium Pro processor with 256KB L2 cache. The Celebris XL6200 system that we used in our test was configured with 128MB DRAM with 2-way interleaving and 14-2-2-2 memory timing at 66 MHz bus frequency. The SPEC benchmarks were compiled with Intel FORTRAN and C Reference Compilers Version 2.3.

5.1 Cycles per Instruction

Figure 4 shows the cycles per instruction (CPI) for the SPEC95 benchmark suite. Several integer benchmarks achieve less than one cycle per instruction. The CPIs are remarkably low for a processor that implements a 14-stage pipeline. The 'ow CPI is due to the overlapped out-of-order execution that mitigates the effect of the latency of individual operations, fast L2 cache, and adaptive two level branch prediction scheme. The FP benchmarks have higher CPI due to longer execution latencies and higher L2 cache misses.

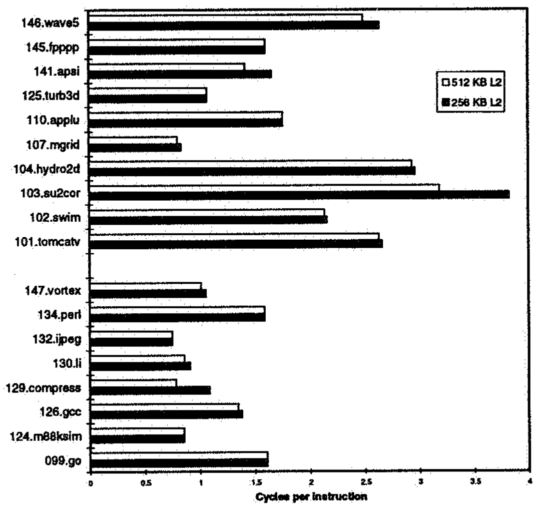


Figure 4 Cycles per Instruction

We measured the CPI for the processor with 512 KB L2 cache too, by replacing the CPU in the Digital Celebris system. Only compress (28%), li (5%) su2cor (17%), apsi (15%), and wave5 (6%) showed more than 5% improvement in CPI with the larger cache. The L2 miss ratio reduction was 85%, 89%, 40%, 48%, and 19% respectively. Simulations show that the CPI would decrease further for su2cor (25%), apsi (14%), and wave5 (5%) if the 4-way L2 cache is doubled again to 1 MB.

5.2 Instruction Decode

The Pentium Pro processor has 3 decoders that can handle up to 3 instructions every cycle (one instruction with up to 4 uops, and two single uop instructions)[5]. The decoder has a 6 uop queue at its output. Only 3 uops can be renamed per cycle, so the decoder has to stall if the queue is too full. Figure 5 shows the percentage of cycles in which 0, 1, 2, or 3 instructions were decoded. Benchmarks with high Icache or L2 misses show many cycles (35% to 51% for integer, 67% to 83% for FP) in which no instructions are decoded. During L2 misses, the CPU can run out of other machine resources causing back pressure on earlier pipe stages. On the integer benchmarks 33% to 54% of the instructions are decoded in cycles in which 3 instructions are decoded; 25% to 64% for FP benchmarks.

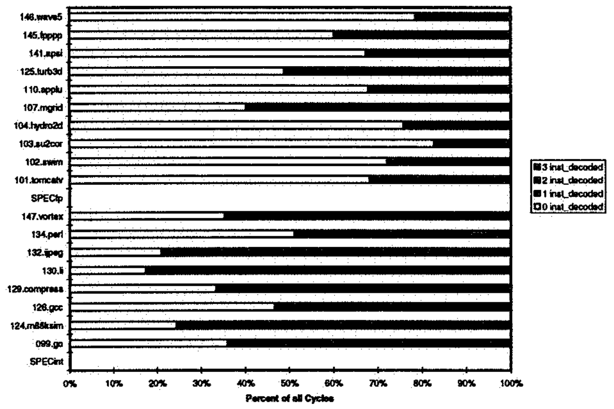


Figure 5 Instruction Decode Profile

5.3 Cache Misses

The L1 data cache can accept a new load or store every cycle and has a latency of three cycles for loads. It can handle as many as four simultaneously outstanding misses. Figure 6 shows the L1 data and instruction cache misses, and L2 cache misses. Except for gcc and m88ksim, the L1 data misses are always higher than the L1 instruction misses. In most cases the L1 instruction misses are so small that they don't even show on the scale used in Figure 6. The integer benchmarks, in general, show much lower L1 data cache and L2 misses than the floating point ones (larger data sets); but higher L1 instruction cache misses (larger code size and fewer loops). The benchmark (wave5) with the highest L1 misses does not have the highest L2 misses. Figure 7 shows a fairly strong correlation between L2 misses and CPI, indicating that the L2 miss latency (about 50 CPU cycles) is not completely overlapped.

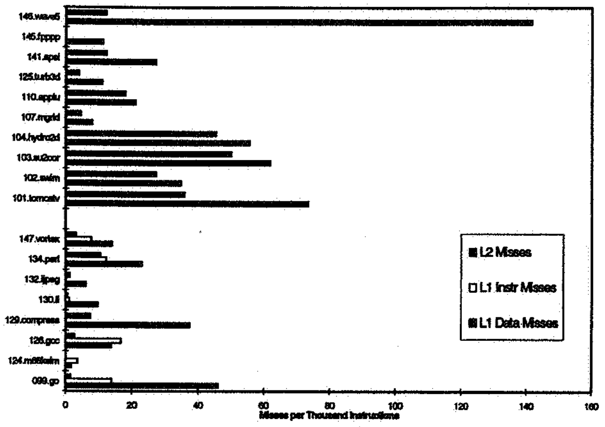


Figure 6 Cache Misses per Thousand Instructions

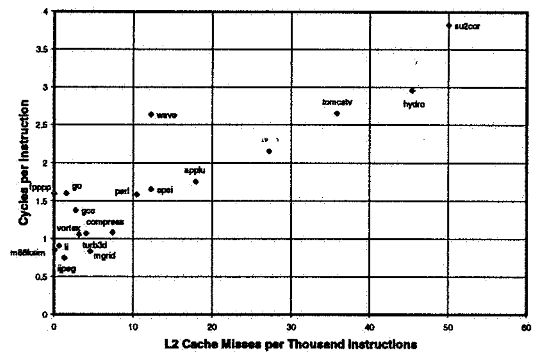


Figure 7 CPI versus L2 Cache Misses

5.4 TLB Misses

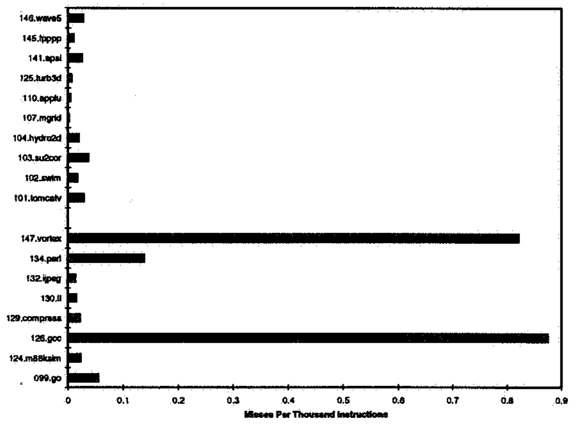


Figure 8 ITLB Statistics

The Pentium® Pro processor has separate TLBs for instructions and data. The processor also has separate TLBs for 4-Kbyte and 4-Mbyte page sizes. The ITLB for 4KB pages has 32 entries. The DTLB for 4KB pages has 64 entries. Both are 4 way set associative. The ITLB for large pages has 4 entries, while the DTLB has 8 entries; both are 4-way set associative. As shown in Figure 8, the ITLB misses are well below 0.1 per thousand instructions, except for a couple of integer benchmarks. The DTLB misses are generally higher than ITLB misses, but they could not be measured accurately. TLB misses do not contribute much to the CPI, as shown later.

5.5 Memory References

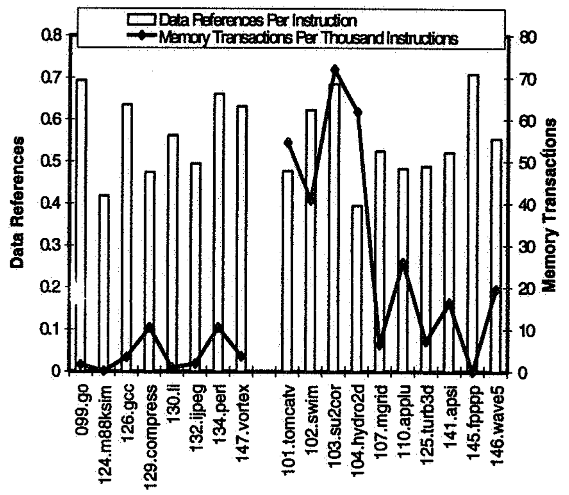


Figure 9 Memory Reference Statistics

Figure 9 shows the number of data references per instruction and the number of memory transactions per thousand instructions. On the average, both the integer and FP benchmarks generate about 1 data reference every two instructions. The IA-32 architecture results in more data references than most RISC architectures because it has fewer registers (8 vs. 32). As might be expected, there is a strong correlation between L2 cache misses and memory transactions. The memory transactions per instruction are higher for the FP benchmarks due to a higher L2 cache miss rate. Note that there can be more than one memory transaction per L2 cache miss if a dirty cache block has to be written back to memory.

5.6 Branch Prediction

The Pentium® Pro processor implements a novel branch prediction scheme, derived from the two-level adaptive scheme of Yeh and Patt[8]. The branch target buffer (BTB) retains both branch history information and the predicted branch target address. The BTB contains 512 entries. If a branch is not found in the BTB, a static prediction (backwards taken, forward not taken) is used. There is no penalty for correctly predicted not-taken branches. Correctly predicted taken branches incur a 1 cycle penalty. Mispredicted branches incur a penalty of about 10-15 cycles, plus additional cycles required to retire the mispredicted branch.

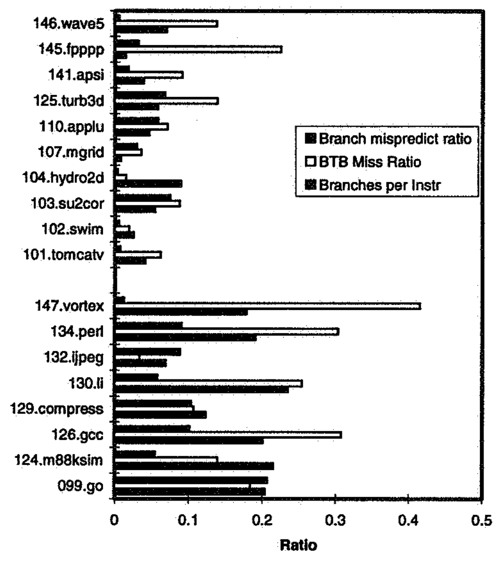


Figure 10 Branch Statistics

Figure 10 shows the frequency of branches, fraction of branches that hit in the BTB, and the accuracy of branch prediction. Even though the BTB miss ratio is fairly high, the branch mispredict ratio is less than 10% for all but one benchmark. The BTB miss ratio is high partly due to the fact that unconditional branches are not stored in the BTB, but are included in the total branch instruction count. As might be expected, the integer benchmarks contain more branches than the FP benchmarks, and they incur a higher branch mispredict ratio (fewer loop branches). The number of mispredicted branches range from about 2 to 40 per thousand instructions for the integer benchmarks, and about 0.1 to 4 for the FP benchmarks. For most of the benchmarks, branch mispredict stalls are not a major contributor to overall CPI.

5.7 Speculative Execution

The Pentium® Pro processor fetches instructions along the predicted path and executes them until the branch is resolved. If a branch is incorrectly predicted, the speculated instructions down the mispredicted path are flushed. Note that there can be other mispredicted branches down a mispredicted branch. Figure 11 shows the average number of instructions issued per retired instruction for the SPEC benchmarks. There are about 13 to 37 speculated instructions per mispredicted branch. Mispredicted branches are not recognized for about 10 to 15 cycles, and the processor can issue up to 3 instructions per cycle. Benchmarks with higher mispredicted branches per instruction have higher speculated instructions.

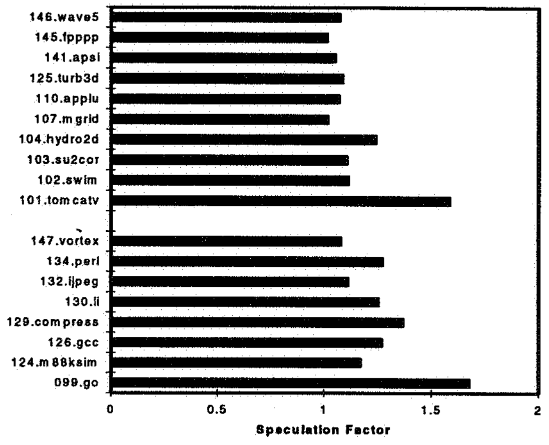


Figure 11 Speculation Factor

5.8 Resource Stalls

Figure 12 shows the I-stream stalls and resource stalls, measured in terms of the cycles in which the stall conditions occur. I-stream stalls are caused mainly by I-cache misses and ITLB misses. Resource stalls show the number of cycles in which resources like register renaming or reorder buffer entries, memory buffer entries, and execution units are full; but these stalls may be overlapped with the execution latency of previously executing instructions. The FP benchmarks, except for fpppp (long basic blocks), incur negligible I-stream stalls. They do incur significantly more resource stalls than integer benchmarks, probably due to long dependency chains.

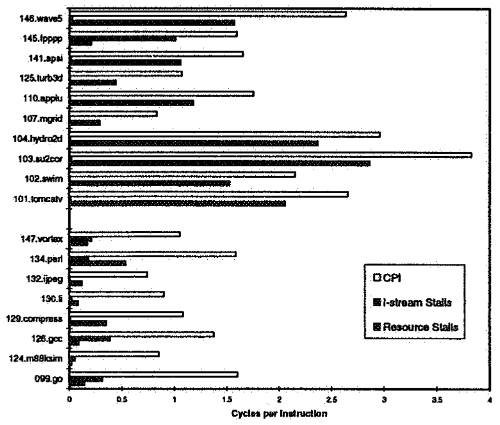


Figure 12 Stall Cycles

5.9 Micro-Operations

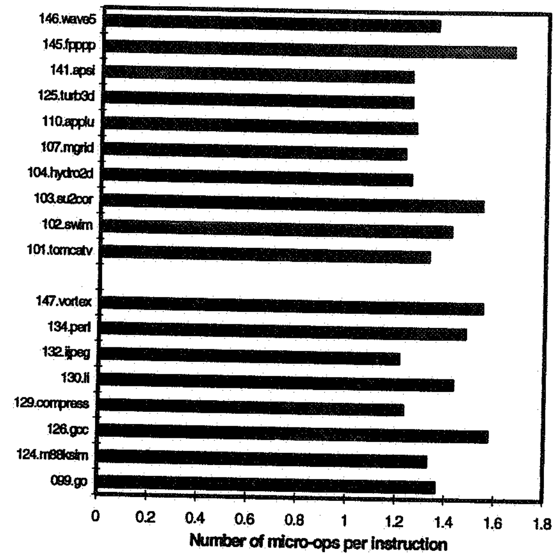


Figure 13 Micro-operations per Instruction

The instruction fetch unit fetches 16 bytes every clock cycle from the I-cache and delivers them to the instruction decoder. Three parallel decoders decode this stream of bytes and convert them into triadic uops. Most instructions are converted directly into single uops, some are decoded into one-to-four uops, and the complex instructions require microcode (sequence of uops). Up to 5 uops can be issued every clock cycle to the various execution units, and up to 3 uops can be retired every cycle. Figure 13 shows the average number of uops executed per instruction for each of the SPEC95 benchmarks. The range is from 1.2 to 1.7, with an average around 1.35.

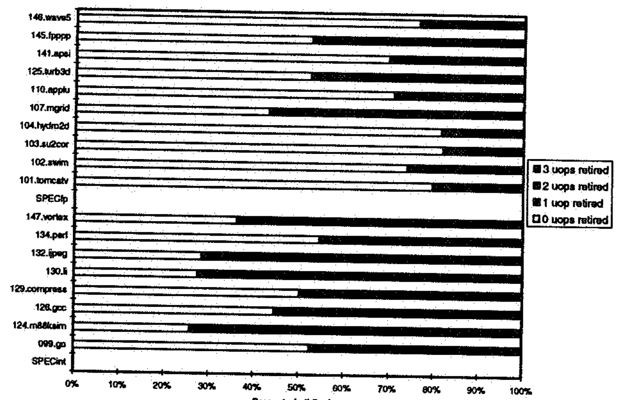


Figure 14 Micro-operations retirement profile

Figure 14 shows that no uops are retired in 25% to 55% of the cycles on the integer benchmarks, and 43% to 82% of the cycles for FP. Benchmarks with low CPI have fewer cycles with no uops retired. Furthermore, about 65% and 80% of the uops are retired in cycles in which 3 uops are retired for the average integer and FP benchmark respectively. This indicates that executed uops often have to wait for uops from previous instructions to be ready for retirement, thereby confirming the value of out of order execution. These younger uops build up more for FP benchmarks because of higher cache misses and longer latencies of FP operations.

5.10 Adding Up the Cycles

Accounting for cycles in an out-of-order machine like the Pentium® Pro processor is difficult due to all the overlapped execution. It is still useful to examine the various components of execution and stalls and compare them to the actual cycles per instruction as shown in Figure 15. The CPI is about 20 to 50% lower than the individual components due to overlapped execution. The figure also shows resource stall cycles in which some resource such as execution unit or buffer entry is not available. Execution can proceed during a resource stall cycle in some other part of the machine. Since more than one uop can be dispatched in a cycle, the figure does not account for execution parallelism. Micro-ops seem to dominate in most integer benchmarks. Resource stalls, and L2 misses contribute the most to the CPI in the FP benchmarks. Branch mispredicts are not a major factor.

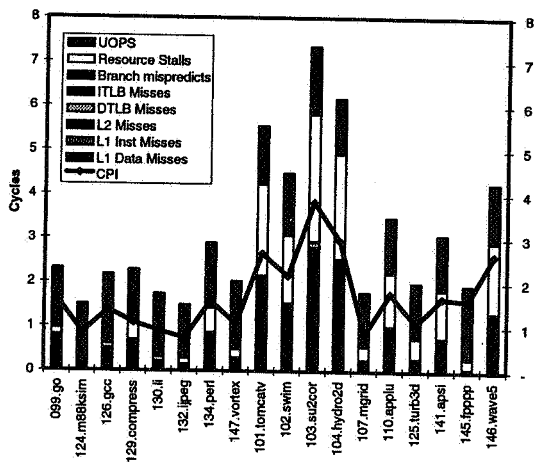


Figure 15 CPI vs. Latency Components

6. Characteristics Across Different Workloads

SPEC95 is a popular CPU intensive benchmark suite. It is widely used to characterize CPU performance. However, the behavior of other workloads can be quite different. This section presents the characteristics of desktop applications running on a Pentium® Pro processor. In particular, we present results for the SYSmark/NT benchmark. These benchmarks are not floating point intensive; Excel contains about 9% FP instructions, MaxEDA has 4%, and the rest less than 0.5%. While the SPEC95 benchmarks were optimized for the Pentium Pro processor using the latest compilers, the SYSmark/NT benchmarks are based on old binaries that have been shipping for many years and were probably not generated with all optimizations turned on.

In this section, we compare the SYSmark/NT benchmark statistics with the minimum, median, or maximum for the SPECint95 and SPECfp95 suites. The data presented here shows that the SPEC integer benchmarks should not be used to predict the performance of real business applications

Figure 16 shows the CPI across different workloads. The business applications (using old binaries with non-optimal code) incur higher CPIs than the median for the SPECint95 benchmarks. Two of the five SYSmark/NT benchmarks incur higher CPI than the median observed for the SPECfp95 suite. The CPI is higher due to higher L2 miss rates, Istream stalls, and resource stalls.

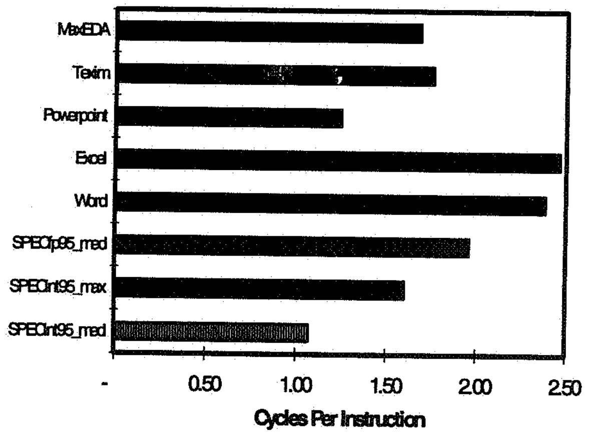


Figure 16 CPI for SYSmark/NT

Figure 17 shows the L2 cache misses. The three Microsoft Office benchmarks incur much higher L2 misses than the SPECint95 median, but well below the SPECfp95 median. The code and data sizes of these business applications are much larger than the SPEC integer benchmarks. Once again, there is fairly strong correlation between L2 misses and CPI. Overall, Word and Excel exhibit the highest L2 misses and stall cycles among the SYSmark/NT benchmarks.

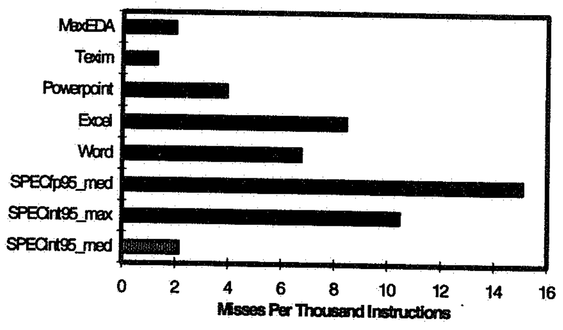


Figure 17 SYSmark/NT L2 Cache Misses

Figure 18 shows the resource stalls. The SYSmark/NT benchmarks incur higher resource stalls than the SPECint95 median, but are well below the SPECfp95 median. The higher resource stalls can be attributed to higher L2 misses during which the internal resources can be consumed by instructions waiting to be retired.

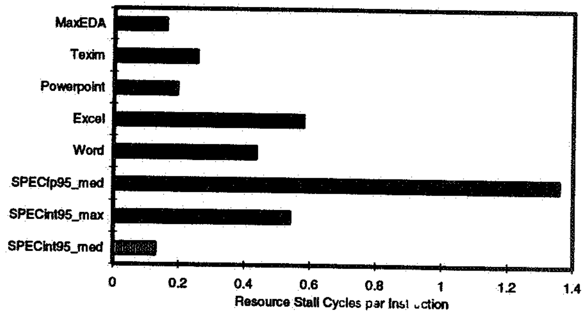


Figure 18 SYSmark/NT Resource Stalls

Figure 19 shows the instruction stalls. They are higher than the SPECint95 median. Once again, this is due to higher occurrence of string instructions in Word and Excel that invoke the microsequencer and require the decoders to stall. These workloads also have high context switch activity resulting in ITLB flushes and Icache misses.

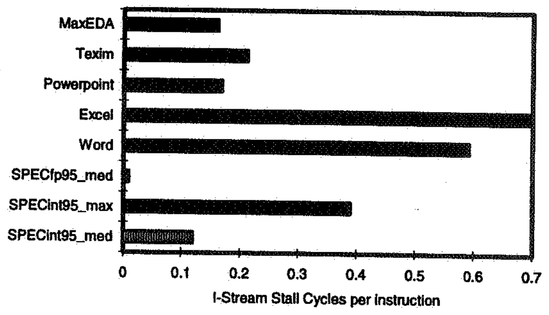


Figure 19 SYSmark/NT Instruction Stalls

Figure 20 shows the uops per instruction. All the SYSmark/NT benchmarks execute more uops than most of the SPEC95 benchmarks. This is probably due to the higher use of character string instructions. There is a strong correlation between uops/instruction and CPI, a trend not observed in the SPEC95 suite.

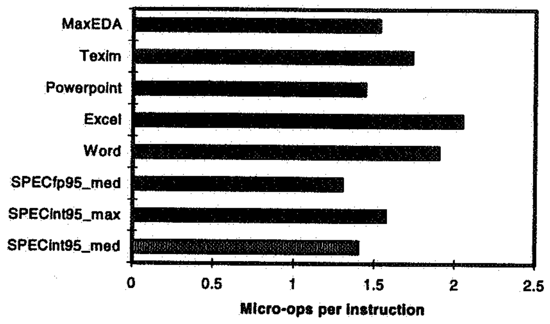


Figure 20 SYSmark/NT Micro-ops Per Instruction

Figure 21 shows the speculation factor. It is in the bottom half of the distribution for SPECint95. The speculation factor is lower because there are fewer mispredicted branches.

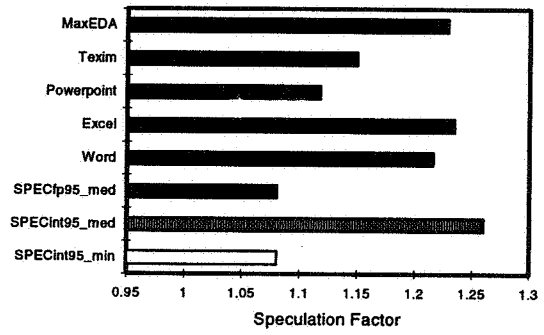


Figure 21 SYSmark/NT Speculation Factor

7. Concluding Remarks

The Pentium® Pro processor was designed to achieve significantly higher performance than the Pentium processor in the same process technology. It achieves this performance through a superpipelined design that yields a 25% faster clock, and with an out of order dynamic execution engine that reduces the CPI. The data presented here shows that the Pentium Pro processor achieves a 15 to 45% reduction in CPI compared to the previous generation design (Pentium processor) in the same process technology, while running at a 25% faster clock frequency. The processor's out-of-order, speculative execution engine does manage to overlap useful work with pending memory accesses to reduce the impact of cache misses. The impact of resource stalls is also reduced by out of order execution. The branch prediction scheme reduces branch mispredictions so as not to make them a significant performance limiter. It performs well even on old binaries that were not optimized for its microarchitecture. Performance counter based measurements show that the overall CPI achieved by the Pentium Pro processor is about 20 to 50% lower than the individual latency components due to overlapped execution.

A detailed comparison of the Pentium Pro processor and Digital's Alpha 21164 RISC processor is reported in another study [9].

8. Acknowledgments

The authors would like to thank the Pentium® Pro processor designers for producing such an interesting microprocessor and for incorporating the performance counting mechanisms that enabled this study. Special thanks to Bob Colwell for his extensive review and valuable comments.

9. References

[1] Robert P. Colwell and Randy L. Steck, "A 0.6um BiCMOS Processor with Dynamic Execution", ISSCC Proceedings, February 1995, pp. 176-177.
[2] Linley Gwennap, "Intel's P6 Uses Decoupled Superscalar Design", Microprocessor Report, Vol. 9, No. 2, 16 February 1995, pp. 9-15.
[3] Intel Corporation, "Pentium Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual", Intel Corporation, Order Number 242692, 1996.
[4] Donald Alpert and Dror Avnon, "Architecture of the Pentium Microprocessor," IEEE Micro, June 1993, pp. 11-21.
[5] David Papworth, "Tuning The Pentium Pro Microarchitecture," IEEE Micro, April 1996, pp. 8-15.
[6] Jeff Reilly, "A Brief Introduction to the SPEC CPU95 Benchmark," IEEE-CS TCCA Newsletter, June 1996. Also, see http://www.specbench.org/osg/cpu95/.
[7] http://www.bapco.com/nt1.htm
[8] Tse-Yu Yeh and Yale Patt, "Two-Level Adaptive Training Branch Prediction," Proc. IEEE Micro-24, Nov 1991, pp. 51-61.
[9] Dileep Bhandarkar, "RISC versus CISC: A Tale of Two Chips," submitted for publication.

* Intel® and Pentium® are registered trademarks of Intel Corporation. Other brands and names are the property of their respective owners.

```
Received: from nsf.ac.uk by dcs.ed.ac.uk id aa20677; 14 Apr 92 13:14 BST
Received: from hermes.intel.com by sun2.nsfnet-relay.ac.uk with SMTP inbound
          id <9319-0@sun2.nsfnet-relay.ac.uk>; Tue, 14 Apr 1992 13:11:56 +0100
Received: from ssduk.intel.com by hermes.intel.com (5.65/10.0i);
          Tue, 14 Apr 92 05:06:24 -0700
Received: from imussd.intel.com by ssduk.intel.com (4.1/SMI-4.1) id AA01282;
          Tue, 14 Apr 92 13:07:45-010
Received: by imussd.intel.com (4.1/SMI-4.1) id AA01254;
          Tue, 14 Apr 92 14:07:21 +0200
From: Thomas Bemmerl <thomas@com.intel.imussd>
Message-Id: <9204141207.AA01254@imussd.intel.com>
Subject: European Supercomputer Development Center (ESDC)
To: stew@ca.utoronto.hub, dmirpt0@es.uib.ps, reijns@nl.tudelft.hdetud11
Date: Tue, 14 Apr 92 14:07:21 MET DST
Cc: rni@uk.ac.ed.dcs, sanders@ch.ethz.inf, NPT@uk.ac.ed.ecsvax
X-Mailer: ELM [version 2.3 PL11]
Sender: thomas@com.intel.imussd
Status: RO
```

Hi,

on March 31st 1992 I have left the Department of Computer Science of Munich University of Technology (TUM). After having finished "Habilitation" last December, I was searching for a new challenging career opportunity, which I received at Intel Corporation. Starting April 1st I took over at Intel Corporation the position to direct the newly founded European Supercomputer Development Center (ESDC). ESDC is a European research and development department of Intel's Supercomputer Systems Division, associated to Intel GmbH Germany located close to Munich in Feldkirchen. Enclosed you find for more detailed information parts of the press release for the announcement of ESDC at Supercomputing Europe '92. Therefore my new address is:

Intel GmbH
European Supercomputer Development Center (ESDC)
Dornacher Str. 1, W-8016 Feldkirchen b. Muenchen, FRG
Tel.: +49-89-90992-0, Fax.: +49-89-9039-142
e-mail: thomas@esdc.intel.com

You see, I will continue to work in the research field known to me, concentrating initially on programming tools and distributed operating systems for parallel supercomputer technology. Intel ESDC will initially start with the following two charters:

1. Active participation in Intel's research and development programs (Touchstone, TriStar, Paragon, iWarp) to achieve affordable TeraFLOPS supercomputer technology by the middle of the decade. Within these R&D programs ESDC initially takes over work in the field of parallel programming tools (performance analysis) and distributed operating systems (dynamic loadbalancing). In the long term ESDC will also become active on research and development projects in the field of parallel applications for TeraFLOPS computers.

2. Enhancement of the Touchstone and TriStar programs by collaborative projects with European research and development groups from industry, government and academia. In cooperation with Intel users and other interested research groups, ESDC is going to be involved in European R&D projects (e.g. ESPRIT) and other European activities on High Performance Computing (HPC).

A challenging long term vision may be a partly combination of HPC activities in Europe and the US, due to the fact that problems addressed by HPC (the so called grand challenges, as global climate modeling and weather forecasting, the human genome project, computational fluid simulation, QCD effects, etc.) are relevant for the US as well as for Europe. Global and international solutions are most appropriate to solve these grand challenges.

For this reason I am very interested to stay in touch with your organization and you personally. Do not hesitate to contact me under my new address, if you have any questions, in particular on opportunities for further scientific collaborations.

Looking forward to fruitful future cooperations, I remain,

with best regards,

Dr. habil. Thomas Bemmerl
Director, European Supercomputer
Development Center (ESDC)

****

News Release

Intel Establishes European Supercomputer Development Centre

Extends Lead in Race to Deliver World's First TeraFLOPS Supercomputer

PARIS, 13. February 1992 -- Intel Corporation, the worlds leading supplier of parallel supercomputers, today announced the establishment of a European Supercomputer Development Centre to be located in Munich, Germany.

Dr. Thomas Bemmerl of the Technical University Munich is moving to Intel to direct the operations of the Centre. Under Bemmerl's direction, Intel's European Supercomputer Development Centre (ESDC) will be collaborating with European Users and European research labs to develop software technologies in support of TeraFLOPS computing.

"The establishment of our European Supercomputer Development Centre is evidence of our commitment to be more actively involved in the High Performance Computing initiatives being discussed in the European Community, such as those being considered by ESPRIT programes, the Rubbia working group and the Euorpean Industry Initiative", said Joseph Mazzarella, international are director for Intel's Supercomputer Systems Division.

According to Justin Rattner, Intel Fellow and director of technology for Intel's Supercomputer Systems Division, ...

Intel's new European Supercomputer Development Centre will initially focus on software tools, environments and operating systems, and their impact on user applications for TeraFLOPS.

According to Steve Nachtsheim, vice president, Intel Products Group - Europe, "Intel's aggressive TeraFLOPS programme, the launch of our powerfull new Paragon XP/S supercomputer and our overall success in Europe has enabled Intel to dramatically extend our worldwide leadership position in supercomputing. Integrating technologies, sofware and applications from Intel, our European partners and customers, is another step in Intel's goal of ensuring the practical usability of TeraFLOPS computers".

....

BYTECH

Bytech Ltd., 2, The Western Centre, Western Road, Bracknell RG12 1RW. Telephone: Sales 0344 482211 Admin/Technical 0344 424222 Telex: 848215

80286 EDUCATIONAL FOCUS

To enable Bona-Fide Educational Establishments to take advantage of Intel's 80286 "Supermicro" technology, Bytech are offering a selected range of Intel's 80286 based products and development kits at a fraction of the normal price.

FOR A VERY LIMITED PERIOD BYTECH WILL GIVE AN EDUCATIONAL DISCOUNT OF UP TO 60% ON THESE "TECHNOLOGY-LEADING" PRODUCTS.

		List(£)	Educational(£)
SBC286/10	- 80286 based CPU board	3057	1832
SBC012CX	- 512KB LBX RAM board	2259	1807
SBC010CX	- 1.0MB LBX RAM board	4769	2861
SBC020CX	- 2.0MB LBX RAM board	8283	6626
SBC286/10 KIT	- System SBC286/10 + SBC012B + System Debug Monitor	7530	3012
SYS310-17MR KIT + SBC286/10 KIT	- RMX 310 Starter Kit	25028	14016
SYS310-17MX KIT + SBC286/10 KIT	- XENIX 310 Starter Kit	24216	13559
SYS310-4 + SYS310-17MR KIT	- 310 Microsystem Hardware Unit	25932	17635
SYS310-4 + SYS310-17MX KIT	- 310 Microsystem Hardware Unit	25120	17083
MDX431B + 111286 + SBC286/10 KIT	- 8 & 16 bit Development System - In-Circuit Emulation for 80286	44050	34603

DON'T DELAY - PHONE 0344-482211 TODAY FOR FURTHER INFORMATION ON THESE SUPER DEALS.

N.B. Offer expires 31st March 1985.

Registered Office: 16 Stratford Place, London W1N 9AF. Reg. No. 1426108 (England) A member of the IBR Electronics Group.

Intel's 32-bit microprocessor unveiled

One of the highlights of the recent ISSCC event was the official unveiling by Intel of its 32-bit microprocessor. Called the iAPX 432, this is a set of three devices which together deliver the kind of computing resources associated with mid-range mainframes. In fact, Intel's designation of 'micromainframe' is a more accurate term than microprocessor—the iAPX 432 bears little relation to any of the microprocessors that have been available to date.

The system is based on a new architecture designed to significantly reduce the cost and time for developing large-scale applications by improving programmer productivity. In addition, the micromainframe includes hardware and software facilities for design of fault-tolerant systems.

The iAPX 432 has been designed to employ Intel's current line of peripheral controller chips, memories, and microprocessors. Thus, designers have a wide variety of field-proven devices available for use in micromainframe-based systems.

Like most mainframes, the 432 is split into functionally specialised data processors and I/O subsystems. Additionally, the designer can add extra data processors or I/O subsystems to improve performance in a software-transparent fashion. This means that a single set of hardware and software components can be applied to many different end-products, or to a single family of products offering multiple levels of performance. Moreover, the performance of a system can be upgraded, if necessary, to meet existing design requirements or to accommodate new application demands.

Because the number of data processors and I/O subsystems is under the control of the user, efficient multiprocessor arbitration and coordination facilities are essential These do not need to be devised by the user, however; they have been built in to the hardware, thereby reducing configuration constraints. The kind of software-transparent multiprocessing that is possible with the 432 allows systems to be designed that have a 1-to-10 performance range.

Two of the micromainframe's three chips make up its general data processor (GDP). They are the instruction decode unit (iAPX 43201) and microexecution unit (iAPX 43202). They act as a pipe-lined pair, one fetching and decoding instructions and the other executing them.

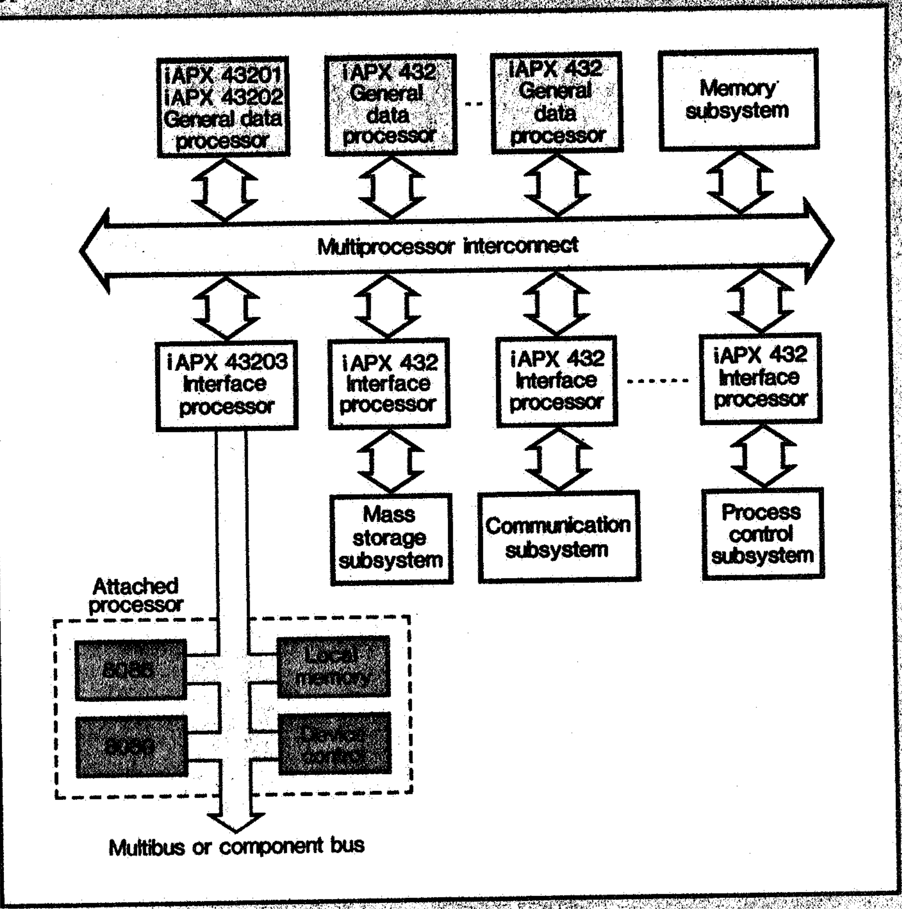
The 32-bit GDP supports a logical address space of 2^{32} or more than 4 billion bytes. What's more, it governs a virtual memory address space of 2^{40} or 1,000,000,000,000 bytes (a terabyte). This address region extends well beyond that of today's mid-range mainframes. It effectively removes all address barriers, allowing large programs and data to be constructed.

Like contemporary mainframes, the micromainframe handles 32-bit data words. But it also operates on floating point values of 32-, 64-, even 80-bits wide. It multiplies 32-bit integers in 6·25 microseconds, and 80-bit floating point numbers in only 26·125 microseconds. This, says Intel, exceeds the performance of several popular mainframe computers.

In addition to floating point data types, the micromainframe handles boolean and character data types. The iAPX 432 instruction set has been specifically designed for high-level languages. It supports from 0 to 3 operand specifiers per instruction, and features symmetric operand addressing modes for scalar, vector, and record elements. This makes high-level languages more efficient and obviates the need for assembly language programming.

Communication with input and output devices (such as mass storage subsystems and high-speed printers) is handled by the third of the three-chip set—the interface processor (43203). It permits a satellite subsystem to act as an 'attached processor' which independently handles all I/O activity. The attached processor is an independent microcom-

Typical system interconnection using multiple processors


iAPX 43201
iAPX 43202
General data processor
iAPX 432 General data processor
iAPX 432 General data processor
Memory subsystem
Multiprocessor interconnect
iAPX 43203 Interface processor
iAPX 432 Interface processor
iAPX 432 Interface processor
iAPX 432 Interface processor
Mass storage subsystem
Communication subsystem
Process control subsystem
Attached processor
8086
8089
Local memory
Device control
Multibus or component bus

puter subsystem configured from the existing line of Intel mid-range microprocessors, peripheral controllers, and Multibus-compatible devices. For example, a single attached processor can be built using the iAPX 86 (8086) as an I/O controller.

The 43203 is connected to the attached processor via a subsystem bus, and communicates with the general data processor (43201 and 43202) via a 'packet bus'. The packet bus is a time-multiplexed interface used for communications between the iAPX 432 processor and memory, and between processors. As many as ten bytes of data can be transferred during a single request or reply.

Micromainframe's attached processor scheme provides nearly unlimited system flexibility. Input/output processing is moved out of the computation system closer to the devices to be controlled. I/O and interrupt burdens are therefore removed from the central system.

A key feature of the 432 is the provision of high level instructions which replace the kind of subroutines normally found in an operating system. For example, there are 'send' and 'receive' instructions which perform, via built in hardware, all the buffering and synchronisation required to transmit data structures between program.

In addition, the silicon operating system automatically distributes the workload among the GDPs. As more processors are added, the workload is automatically and dynamically shared among them. The operating system accounts for some 40% of the 64k bits of microcode—the next heaviest demand is made by the floating point algorithms which take up 18%. In contrast, the basic instruction set uses only 6% of the microcode.

Although the operating system is implemented in hardware, the major scheduling parameters are defined by software so that the improved execution efficiency and reduced software costs have not been attained by sacrificing flexibility.

One of the underlying assumptions behind the development of the 432 was Intel's conviction that current microcomputer programming practice would not be able to cope with the projected increase in microcomputer applications. Perhaps the single most important design goal was the need to increase programmer productivity and software reliability. Thus the 432 instruction set is optimised for the language ADA—most ADA instructions compile to a single 432 instruction—and built-in safeguards prevent incorrect software from corrupting good software.

ADA, a programming language developed with the co-operation of the United States Department of Defense, industry, and universities, is the micromainframe's system programming language. It is a language oriented toward systems programming, numerical problem solving, and real-time applications involving concurrent execution requirements. ADA combines PASCAL's simplicity and elegance with the structure and expressive capability necessary for multi-function software systems. It was selected as the 432's system programming language because it directly supports object-oriented programming methodology and is expected to further improve productivity for software development teams.

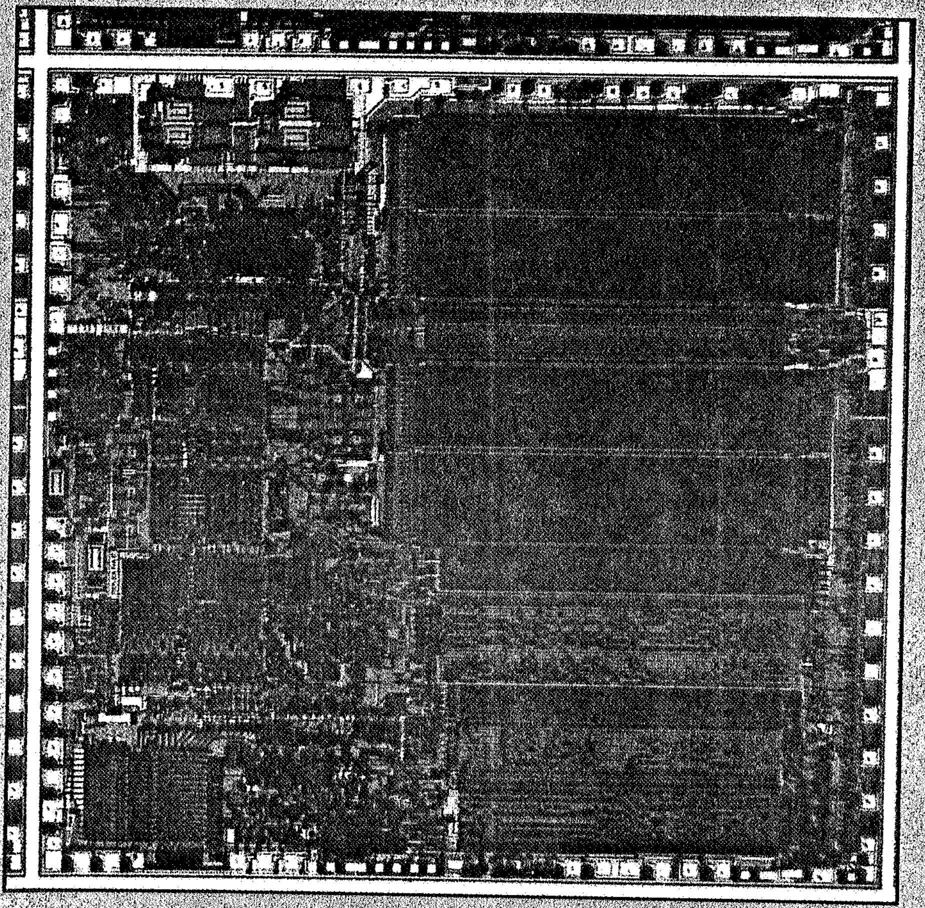
The micromainframe's hardware, operating system, and system programming language all support the concept of 'objects'—a variable length data structure which is of a higher order than data typically recognised and manipulated by contemporary computer hardware.

System programmers write program 'modules' which are based on objects. Each object is typed by a special 'object descriptor' and can only be physically addressed by a special 'access descriptor'. What may gain access to the object, and what operations may be done to it, are inherently controlled.

This facility ensures that *every* data structure in memory is *uniquely* protected in terms of which programs can access it, and to what extent it may be referenced by an authorised program. Programs only have access to the data that they 'need to know'.

Each data structure is also 'typed' to make sure that it is only used in conjunction with operations that make sense for it. For example, it is impossible to 'branch-to-data,' or 'over-write instructions'. These protection facilities guarantee that software errors are detected immediately and confined to offending programs.

The 43201 instruction decode unit, part of the general data processor

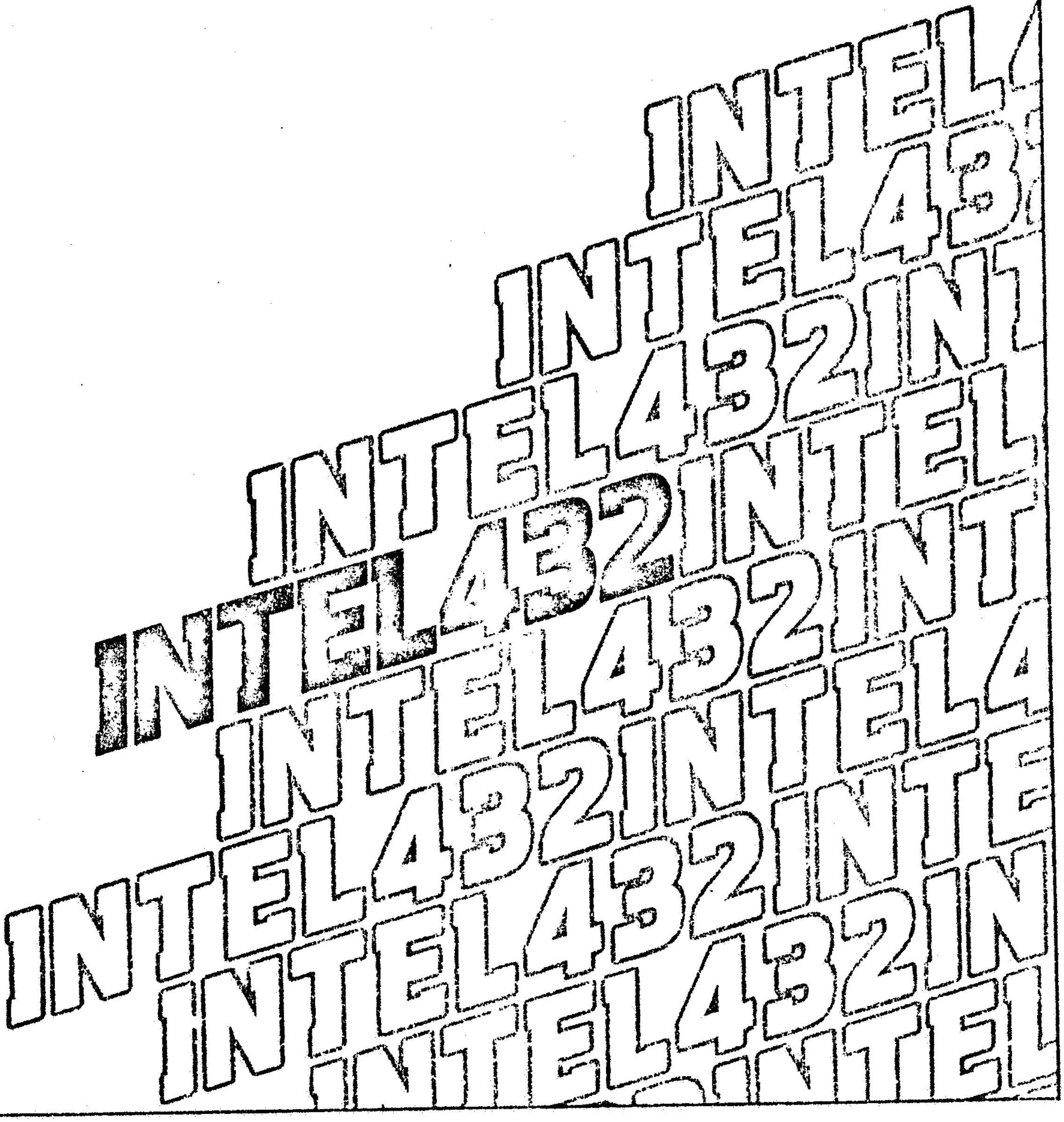


[illegible]

intel

432 Summary Presentation

1980 Intel Technical Symposium



intel

INTEL432

INTEL DELIVERS SOLUTIONS

1

intel

THE INTEL 432

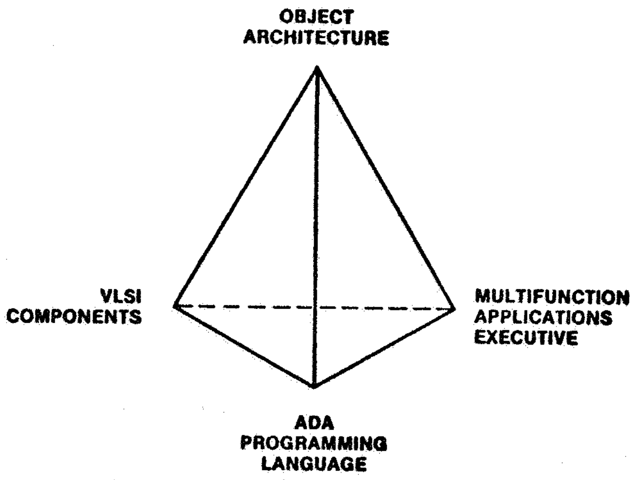
A 32-BIT MICROMAINFRAME:

VLSI IMPLEMENTATION, OVER 200K DEVICES
2^{40} BYTES VIRTUAL ADDRESS SPACE
2^{24} BYTES PHYSICAL ADDRESS SPACE
8, 16, 32, 64, AND 80-BIT DATA TYPES
3 OPERAND ADDRESS MODES
2 MIP PERFORMANCE RANGE
TRANSPARENT MULTIPROCESSING
OBJECT-ORIENTED ARCHITECTURE
ADDRESS CACHING
ORTHOGONAL INSTRUCTION SET
INDEPENDENT AND DECENTRALIZED I/O
SELF-DISPATCHING PROCESSORS
PROTECTION TO THE DATA STRUCTURE LEVEL
ADA ABSTRACTION, TYPING, AND CONCURRENCY

INTEL DELIVERS SOLUTIONS

2

THE 432 MICROMAINFRAME. . .



. . .A TOTAL SYSTEM APPROACH

INTEL DELIVERS SOLUTIONS

3

432 OBJECTIVE:

TO SIGNIFICANTLY REDUCE THE

COST
&
TIME TO MARKET

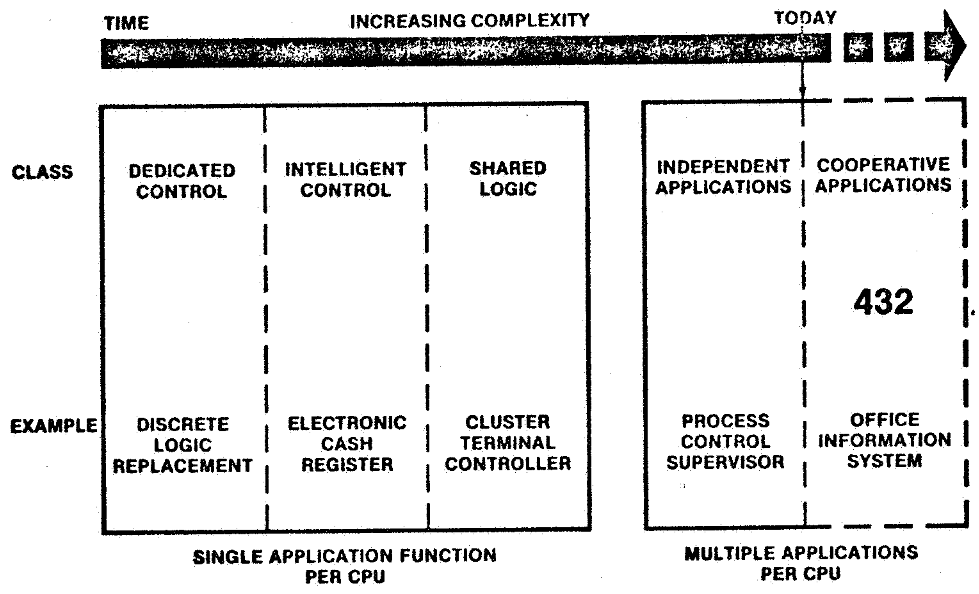
OF SOFTWARE INTENSIVE MICROCOMPUTER APPLICATIONS

INTEL DELIVERS SOLUTIONS

4

intel

MICROCOMPUTER APPLICATIONS ARE GROWING IN COMPLEXITY

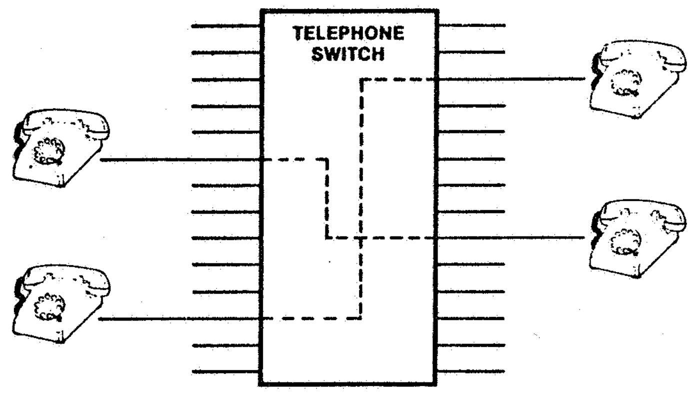


CLASS	DEDICATED CONTROL	INTELLIGENT CONTROL	SHARED LOGIC	INDEPENDENT APPLICATIONS	COOPERATIVE APPLICATIONS
					432
EXAMPLE	DISCRETE LOGIC REPLACEMENT	ELECTRONIC CASH REGISTER	CLUSTER TERMINAL CONTROLLER	PROCESS CONTROL SUPERVISOR	OFFICE INFORMATION SYSTEM
	SINGLE APPLICATION FUNCTION PER CPU			MULTIPLE APPLICATIONS PER CPU	

INTEL DELIVERS SOLUTIONS

intel

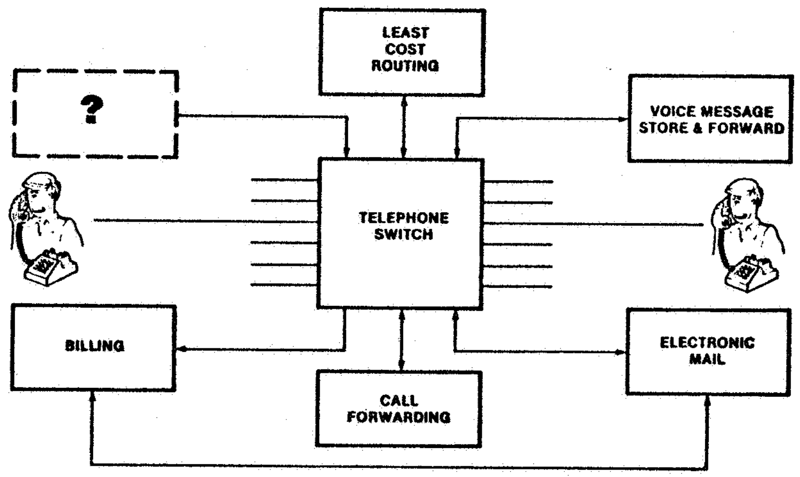
PABX's, ONCE A SIMPLE SWITCHING FUNCTION. . .



- SINGLE FUNCTION
- LINE GROWTH PRE-PLANNED

INTEL DELIVERS SOLUTIONS

...TODAY, A TOTAL MANAGEMENT INFORMATION SYSTEM



- COOPERATING, MULTIPLE APPLICATIONS
- HIGH PERFORMANCE AND FUNCTION
- COMPUTER CRITICAL TO OPERATION OF BUSINESS
- CONTINUOUSLY GROWING

INTEL DELIVERS SOLUTIONS

7

OTHER INCREASINGLY COMPLEX MICROCOMPUTER APPLICATIONS:

- TRANSACTION PROCESSING
- ONLINE OFFICE INFORMATION
- COMPUTER AIDED DESIGN AND SIMULATION
- MULTIFUNCTION BUSINESS SYSTEMS
- FACTORY AUTOMATION AND CONTINUOUS PROCESS CONTROL

INTEL DELIVERS SOLUTIONS

8

intel

ATTRIBUTES OF COMPLEX APPLICATIONS

- ✓ MULTIFUNCTION
- ✓ CONCURRENT
- ✓ LARGE SCALE
- ✓ GROWING AND EVOLVING
- ✓ SOFTWARE INTENSIVE

COMPLEX APPLICATIONS ARE 432 APPLICATIONS

INTEL DELIVERS SOLUTIONS

9

intel

THE 432 MICROMAINFRAME:

- A DIRECT RESPONSE TO THE GROWTH IN APPLICATION COMPLEXITY
- A UNIQUE COMBINATION OF SYSTEM CAPABILITIES OPTIMIZED TO REDUCE THIS COMPLEXITY

INTEL DELIVERS SOLUTIONS

10

intel

THE iAPX 432:

1. LARGE SCALE COMPUTER POWER
2. INCREMENTAL PERFORMANCE CAPACITY
3. HIGHLY DEPENDABLE HARDWARE AND SOFTWARE
4. INCREASED PROGRAMMER PRODUCTIVITY

INTEL DELIVERS SOLUTIONS

11

intel

HOW?

BY COMBINING INTEL's LEADERSHIP

IN VLSI TECHNOLOGY

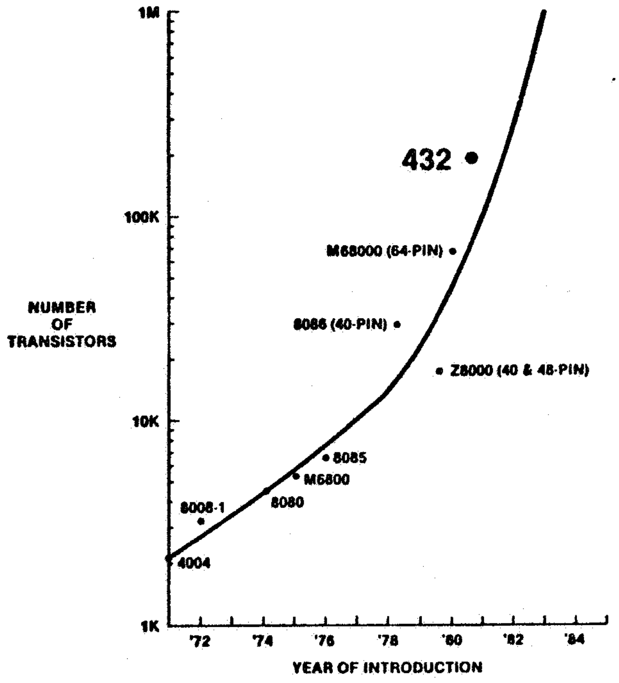
WITH

AN ADVANCED OBJECT-BASED ARCHITECTURE

INTEL DELIVERS SOLUTIONS

12

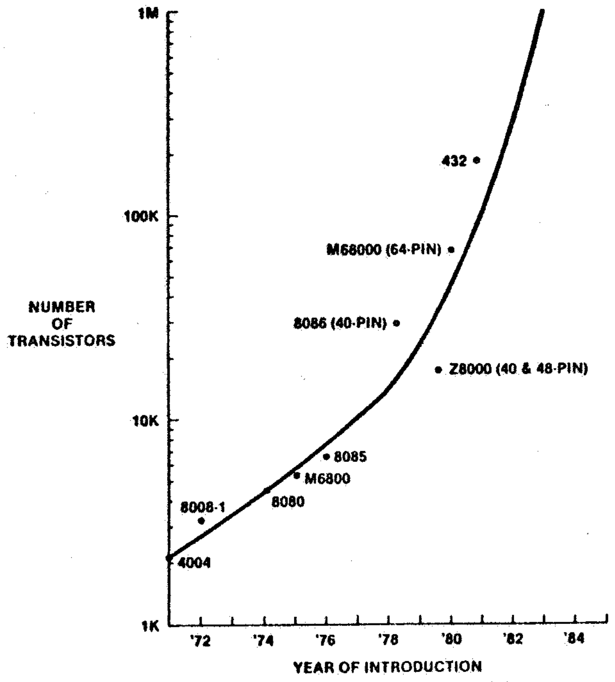
432 PROCESSORS: THE LARGEST SILICON SYSTEMS EVER PRODUCED



INTEL DELIVERS SOLUTIONS

13

432 PROCESSORS: THE LARGEST SILICON SYSTEMS EVER PRODUCED



- IAPX 432 SILICON IS 6 × iAPX 86
- OVER 100 MAN-YEARS IN iAPX 432 SILICON DESIGN, LAYOUT
- 30 MBYTES OF TEST PROGRAMS AND 2 YEARS DEC-10 CPU TIME DEVOTED TO VALIDATION AND CAD

INTEL DELIVERS SOLUTIONS

14

intel

432 LARGE SCALE COMPUTER POWER

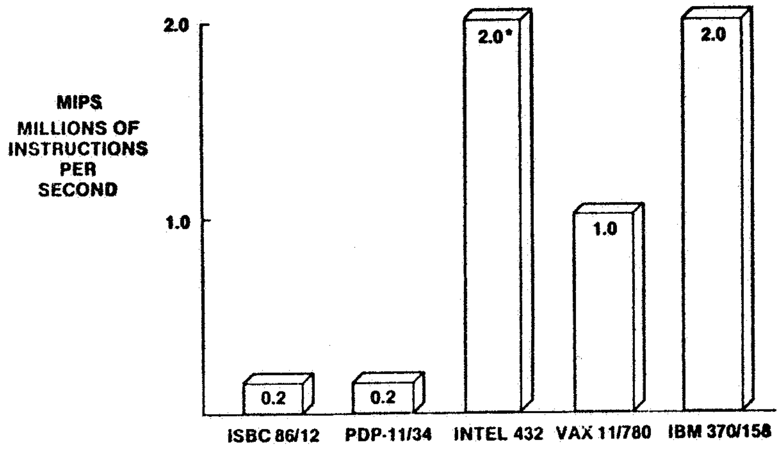
- VIRTUAL ADDRESSING
  - 2^{40} BYTES OF VIRTUAL ADDRESS SPACE
  - 2^{24} BYTES OF PHYSICAL ADDRESS SPACE
- HIGH LEVEL LANGUAGE INSTRUCTION SET
- THE SILICON OPERATING SYSTEM
- INDEPENDENT AND DECENTRALIZED I/O
- PROTECTION TO THE DATA STRUCTURE LEVEL
- MAINFRAME SYSTEM PERFORMANCE

INTEL DELIVERS SOLUTIONS

15

intel

MAINFRAME SYSTEM PERFORMANCE



*THROUGH MULTIPLE PROCESSORS

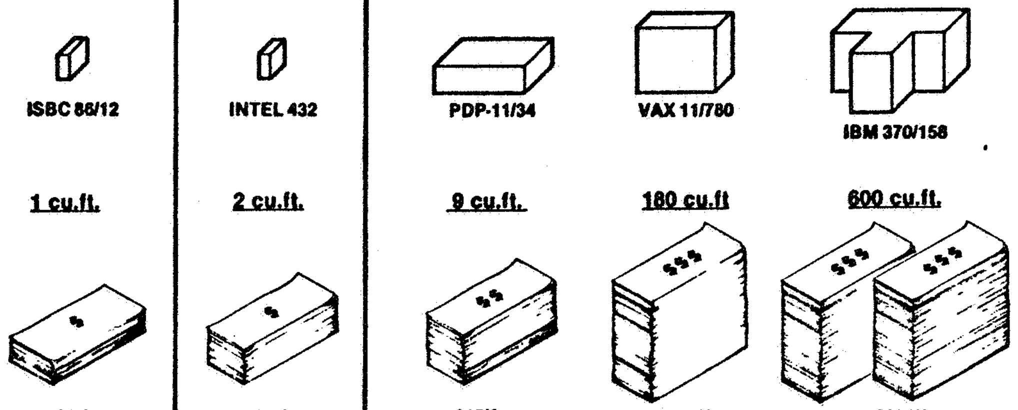
INTEL DELIVERS SOLUTIONS

16

intel

MAINFRAME POWER AT MICROCOMPUTER SIZE AND COST*

MICRO	MICRO MAINFRAME	MINI	SUPER MINI	MAINFRAME
ISBC 86/12	INTEL 432	PDP-11/34	VAX 11/780	IBM 370/158
1 cu.ft.	2 cu.ft.	9 cu.ft.	180 cu.ft	600 cu.ft.
$2K	$5K	$15K	$100K	$300K



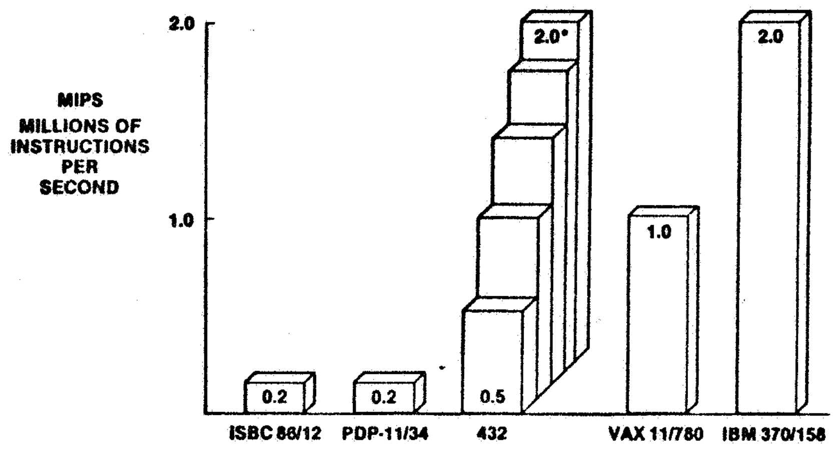
*CENTRAL PROCESSOR SIZE AND COST

INTEL DELIVERS SOLUTIONS

17

intel

THE 432 PROVIDES INCREMENTAL PERFORMANCE CAPACITY

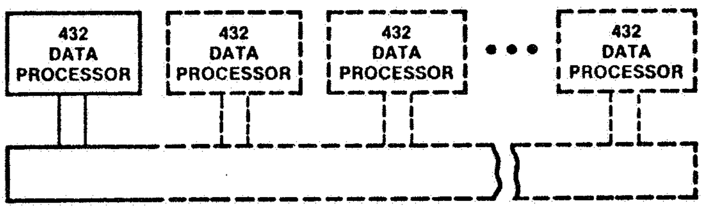


*FOR EXECUTION OF PARALLEL/CONCURRENT PROCESSES

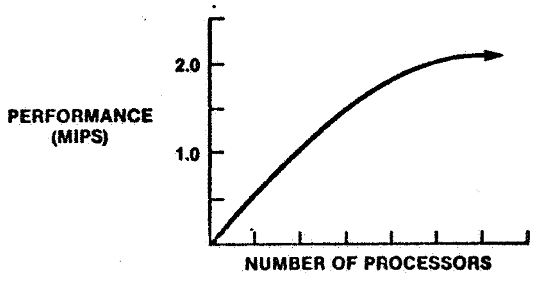
INTEL DELIVERS SOLUTIONS

18

TRANSPARENT MULTIPROCESSING PROVIDES INCREMENTAL PERFORMANCE CAPACITY WITHOUT SOFTWARE CHANGES



Chip type #1 actually 2 types (See 36)



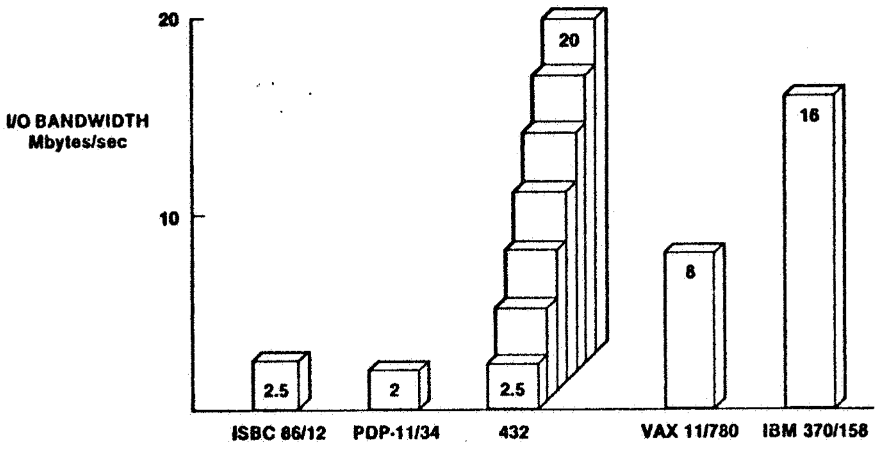
INTEL DELIVERS SOLUTIONS

THE iAPX 432 HAS TRUE 32-BIT MAINFRAME FUNCTIONALITY

	iAPX 432 FUNCTIONALITY	TYPICAL 16-BIT FUNCTIONALITY
ADDRESS SPACE INSTANTANEOUS VIRTUAL	 2^{32} 2^{40}	 2^{16}-2^{18} NO SUPPORT
DATA TYPES	8, 16, 32, 64 and 80-BIT BOOLEAN, CHARACTER ORDINAL, INTEGER, REAL LONG ORDINAL, INTEGER, REAL	8, 16-BIT BOOLEAN, CHARACTER ORDINAL, INTEGER
INSTRUCTION SET	HIGH LEVEL (e.g., A = B + C) MULTI-OPERAND (0-3) VECTOR, RECORD	ASSEMBLY LEVEL TWO OPERAND REGISTER BASED
I/O	INDEPENDENT PARALLEL	DEPENDENT SLAVED
PROTECTION	FINELY GRAINED... PER DATA STRUCTURE	COARSE SUPERVISOR/USER

INTEL DELIVERS SOLUTIONS

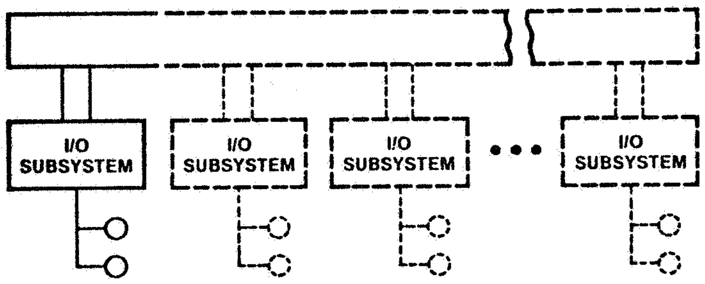
432 PROVIDES INCREMENTAL I/O CAPACITY

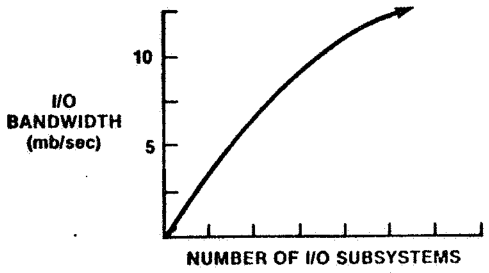


INTEL DELIVERS SOLUTIONS

21

MULTIPLE I/O SUBSYSTEMS PROVIDE INCREMENTAL I/O CAPACITY



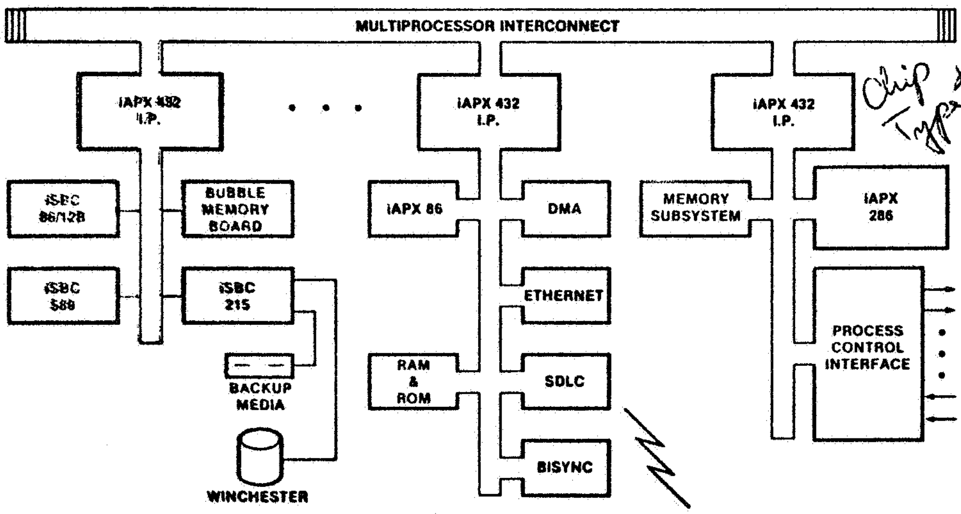


INTEL DELIVERS SOLUTIONS

22

intel

THE INTERFACE PROCESSOR PROVIDES A PATHWAY TO INTEL's ENTIRE PRODUCT LINE OF COMPONENTS AND BOARDS



Chip Type #2

actually 3rd

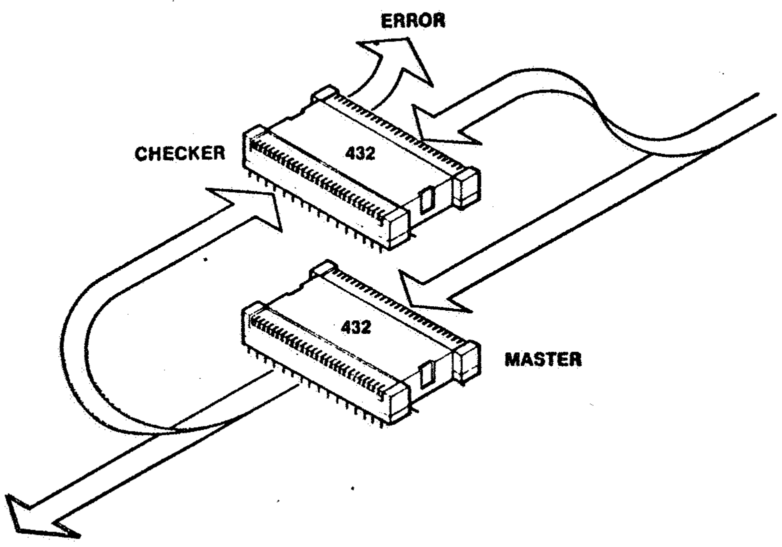
See also #36

INTEL DELIVERS SOLUTIONS

23

intel

FUNCTIONAL REDUNDANCY CHECKING



FOR HARDWARE FAULT DETECTION

INTEL DELIVERS SOLUTIONS

24

ADVANCED ADDRESSING AND PROTECTION

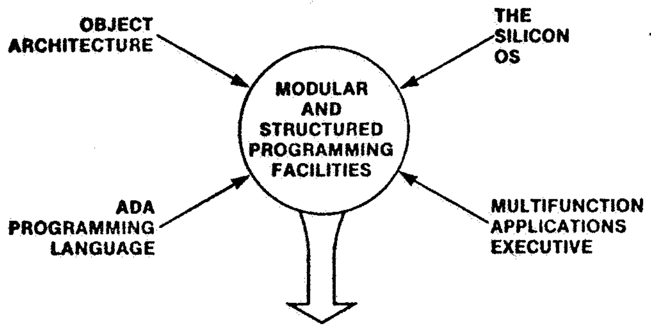
- PROTECTION TO THE DATA STRUCTURE LEVEL
- ENCAPSULATION TO THE LOWEST LEVEL PROGRAM UNIT
- "NEED TO KNOW" ADDRESSING
- DETECTION AND CONFINEMENT OF ACCESS VIOLATIONS

...FOR HIGHLY DEPENDABLE SOFTWARE SYSTEMS

INTEL DELIVERS SOLUTIONS

25

IMPROVED PROGRAMMER PRODUCTIVITY


OBJECT ARCHITECTURE
THE SILICON OS
MODULAR AND STRUCTURED PROGRAMMING FACILITIES
ADA PROGRAMMING LANGUAGE
MULTIFUNCTION APPLICATIONS EXECUTIVE

BUILT-IN SOFTWARE DESIGN METHODOLOGY

INTEL DELIVERS SOLUTIONS

26

intel

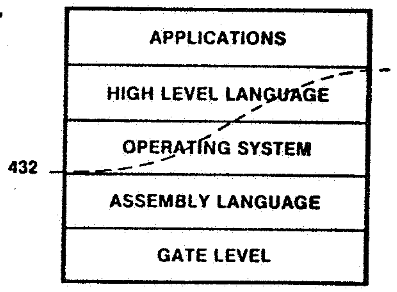
THE 432 ARCHITECTURE

- OVERCOMES FUNCTIONAL BARRIERS
  - TRUE 32-BIT MAINFRAME FUNCTIONALITY
  - OBJECT-BASED SUPPORT FOR STRUCTURED PROGRAMMING
  - HIGH LEVEL LANGUAGE INSTRUCTION SET
  - THE SILICON OS
- IMPROVES PROGRAMMER PRODUCTIVITY

INTEL DELIVERS SOLUTIONS 27

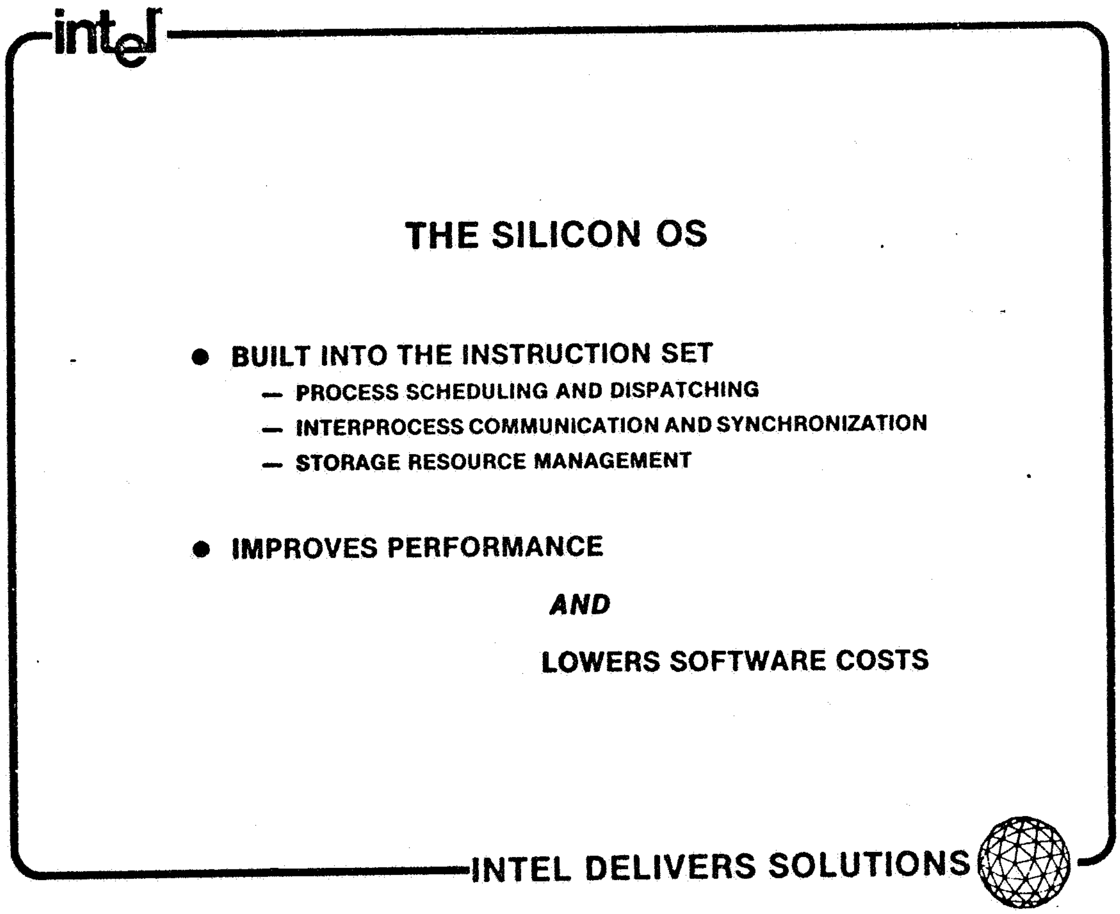
intel

THE 432 RAISES THE HARDWARE/SOFTWARE INTERFACE



THROUGH THE SILICON OS

INTEL DELIVERS SOLUTIONS 28



THE SILICON OS

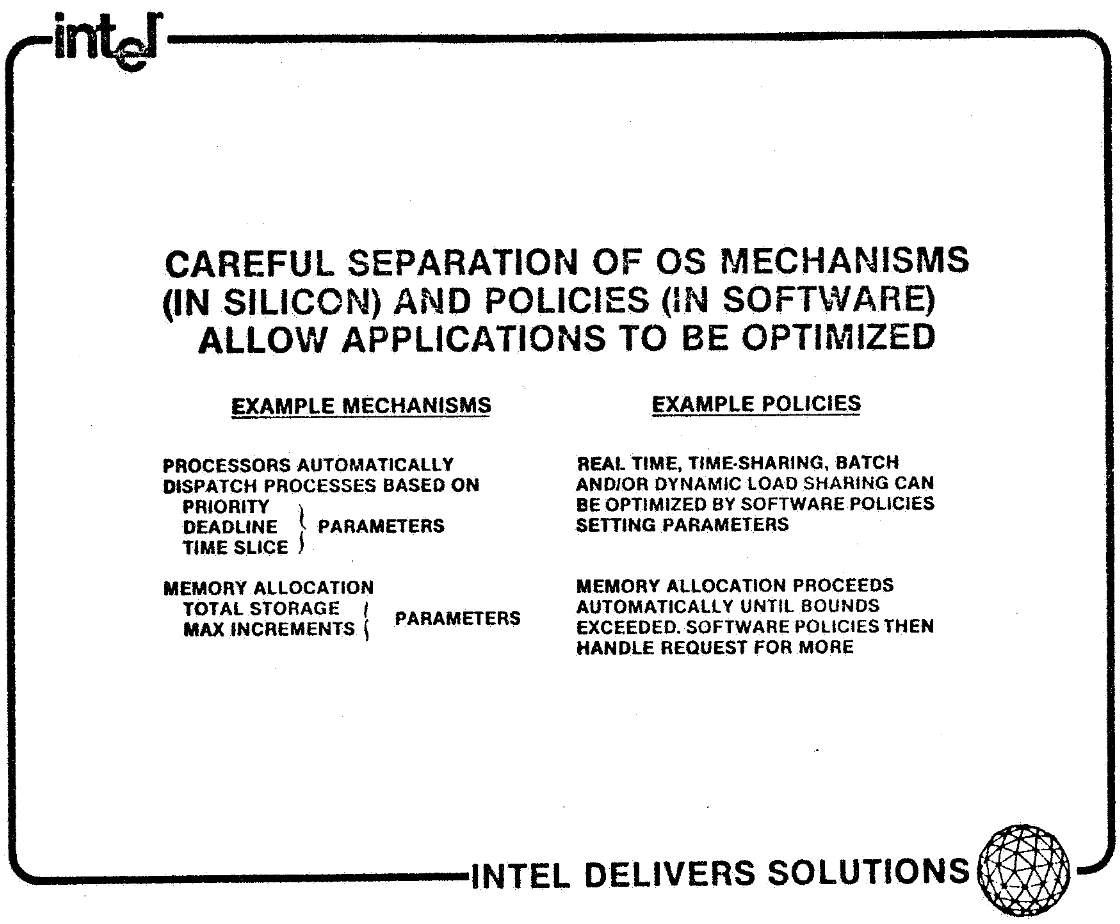
- BUILT INTO THE INSTRUCTION SET
  - PROCESS SCHEDULING AND DISPATCHING
  - INTERPROCESS COMMUNICATION AND SYNCHRONIZATION
  - STORAGE RESOURCE MANAGEMENT
- IMPROVES PERFORMANCE

  AND

  LOWERS SOFTWARE COSTS

INTEL DELIVERS SOLUTIONS

29



CAREFUL SEPARATION OF OS MECHANISMS (IN SILICON) AND POLICIES (IN SOFTWARE) ALLOW APPLICATIONS TO BE OPTIMIZED

EXAMPLE MECHANISMS	EXAMPLE POLICIES
PROCESSORS AUTOMATICALLY DISPATCH PROCESSES BASED ON PRIORITY, DEADLINE, TIME SLICE } PARAMETERS	REAL TIME, TIME-SHARING, BATCH AND/OR DYNAMIC LOAD SHARING CAN BE OPTIMIZED BY SOFTWARE POLICIES SETTING PARAMETERS
MEMORY ALLOCATION TOTAL STORAGE, MAX INCREMENTS } PARAMETERS	MEMORY ALLOCATION PROCEEDS AUTOMATICALLY UNTIL BOUNDS EXCEEDED. SOFTWARE POLICIES THEN HANDLE REQUEST FOR MORE

INTEL DELIVERS SOLUTIONS

30

THE MULTIFUNCTION APPLICATIONS EXECUTIVE

- BUILDS ON THE SILICON OS FOR ADVANCED APPLICATION SUPPORT
- A LIBRARY OF FLEXIBLE TOOLS
- COMPLETE RUNTIME ENVIRONMENT
- EASILY MODIFIED AND/OR EXTENDED

INTEL DELIVERS SOLUTIONS

31

THE ADA PROGRAMMING LANGUAGE

- INTENDED FOR
  - SYSTEMS PROGRAMMING
  - NUMERICAL PROBLEM SOLVING
  - REAL-TIME APPLICATIONS
- PASCAL INSPIRED
- DIRECTLY REFLECTS AND SUPPORTS THE 432 OBJECT-BASED ARCHITECTURE
- IMPROVES PROGRAMMER PRODUCTIVITY THROUGH:
  - STRONG TYPING
  - DATA ABSTRACTION
  - INFORMATION ACCESS CONTROL

INTEL DELIVERS SOLUTIONS

32

intel

A BUILT-IN SOFTWARE DESIGN METHODOLOGY FOR INCREASED PROGRAMMER PRODUCTIVITY

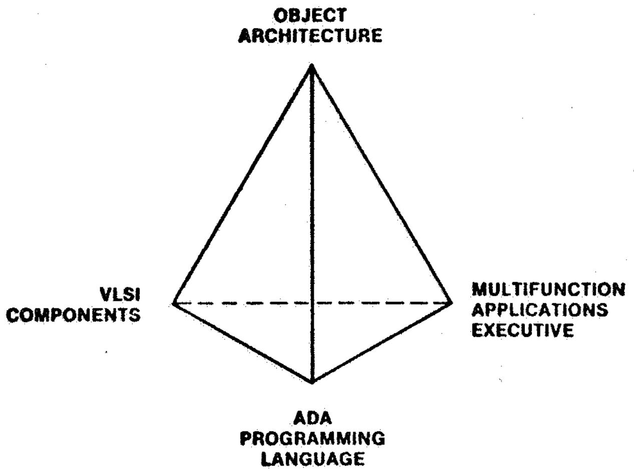
- AN EXTENSION AND GENERALIZATION OF STRUCTURED PROGRAMMING CONCEPTS
- BASED UPON MODULAR PROGRAMS *AND* MODULAR DATA STRUCTURES (OBJECTS)
- A COMPREHENSIVE APPROACH TO ORGANIZING AND BUILDING COMPLEX SOFTWARE SYSTEMS

INTEL DELIVERS SOLUTIONS

33

intel

ALL ELEMENTS OF THE 432 WORK TOGETHER



TO SUPPORT THE SOFTWARE DESIGN METHODOLOGY AND IMPROVE PROGRAMMER PRODUCTIVITY

INTEL DELIVERS SOLUTIONS

34

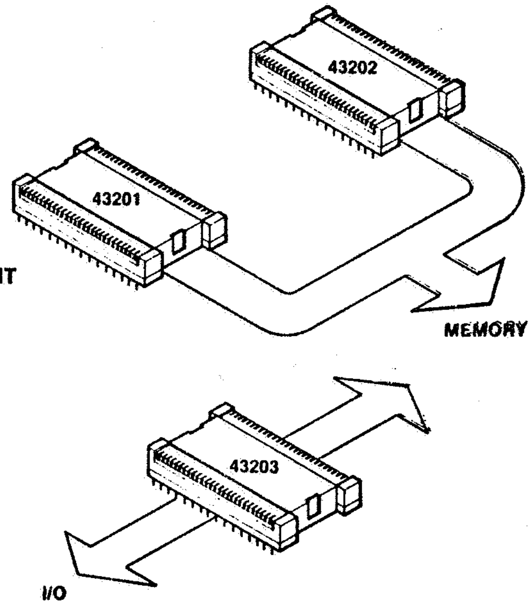
THE INTEL 432 PRODUCT FAMILY

INTEL DELIVERS SOLUTIONS

35

432 COMPONENT PRODUCTS

- GENERAL DATA PROCESSOR
  - — 43201 INSTRUCTION DECODE UNIT
  - — 43202 INSTRUCTION EXECUTION UNIT
- INTERFACE PROCESSOR
  - — 43203 I/O SUBSYSTEM INTERFACE

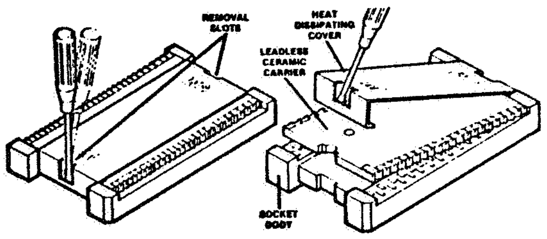


INTEL DELIVERS SOLUTIONS

36

IAPX-432 COMPONENT FAMILY CHARACTERISTICS

- 8 MHz STANDARD CLOCK RATE
- 5 VOLT ONLY OPERATION
- 64-PIN QUAD INLINE PACKAGE (QUIP)



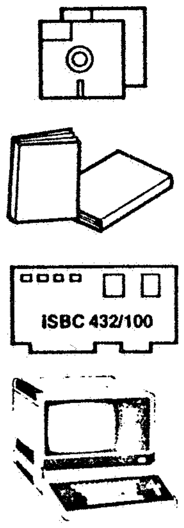
INTEL DELIVERS SOLUTIONS

37

INTELLEC 432/100 EVALUATION SYSTEM

- SOFTWARE
- DOCUMENTATION
- HARDWARE

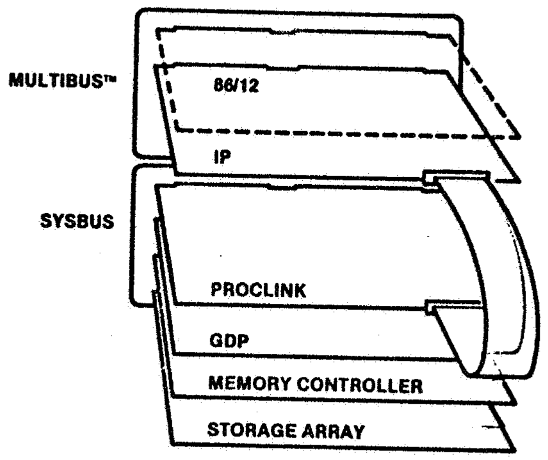
UTILIZES AN INTELLEC



INTEL DELIVERS SOLUTIONS

38

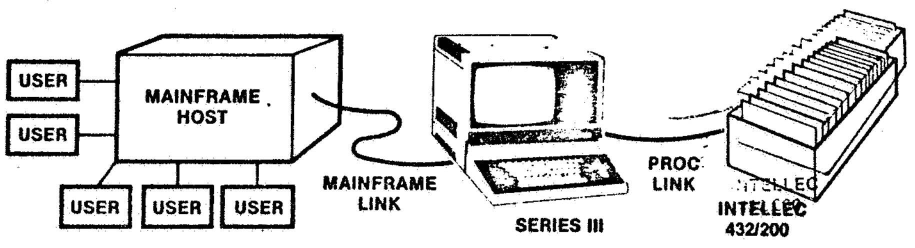
iBCS 432/200 MULTIPROCESSOR BOARD COMPUTER SYSTEM



INTEL DELIVERS SOLUTIONS

39

INTELLEC SERIES III/432 DEVELOPMENT SYSTEM



INTEL DELIVERS SOLUTIONS

40

intel

432 DEVELOPMENT SOFTWARE

- ADA-432 — CROSS COMPILER
- LINK-432 — CROSS LINKER
- DEBUG-432 — SYMBOLIC DEBUGGER
- UTILITIES — LOAD-432, UPDATE-432
- iMAX 432 — MULTIFUNCTION APPLICATIONS EXECUTIVE

INTEL DELIVERS SOLUTIONS

41

intel

THE 432 PRODUCT FAMILY

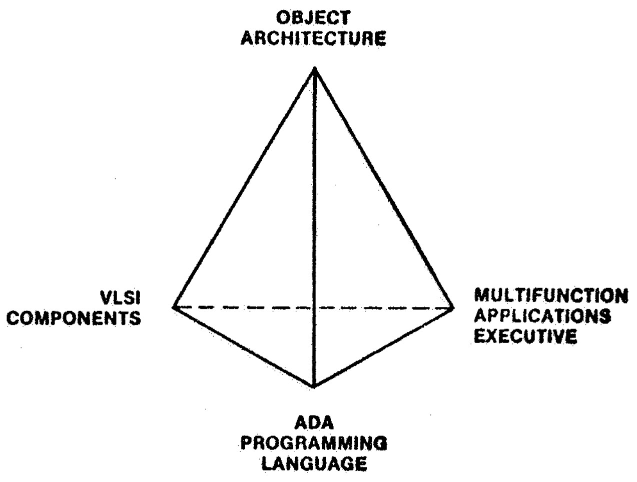
- A COMPLETE SET OF TOOLS
- MULTIPLE LEVELS OF INTEGRATION
- DESIGNED FOR ADVANCED APPLICATIONS

INTEL DELIVERS SOLUTIONS

42

intel

THE 432 TOTAL SYSTEM APPROACH

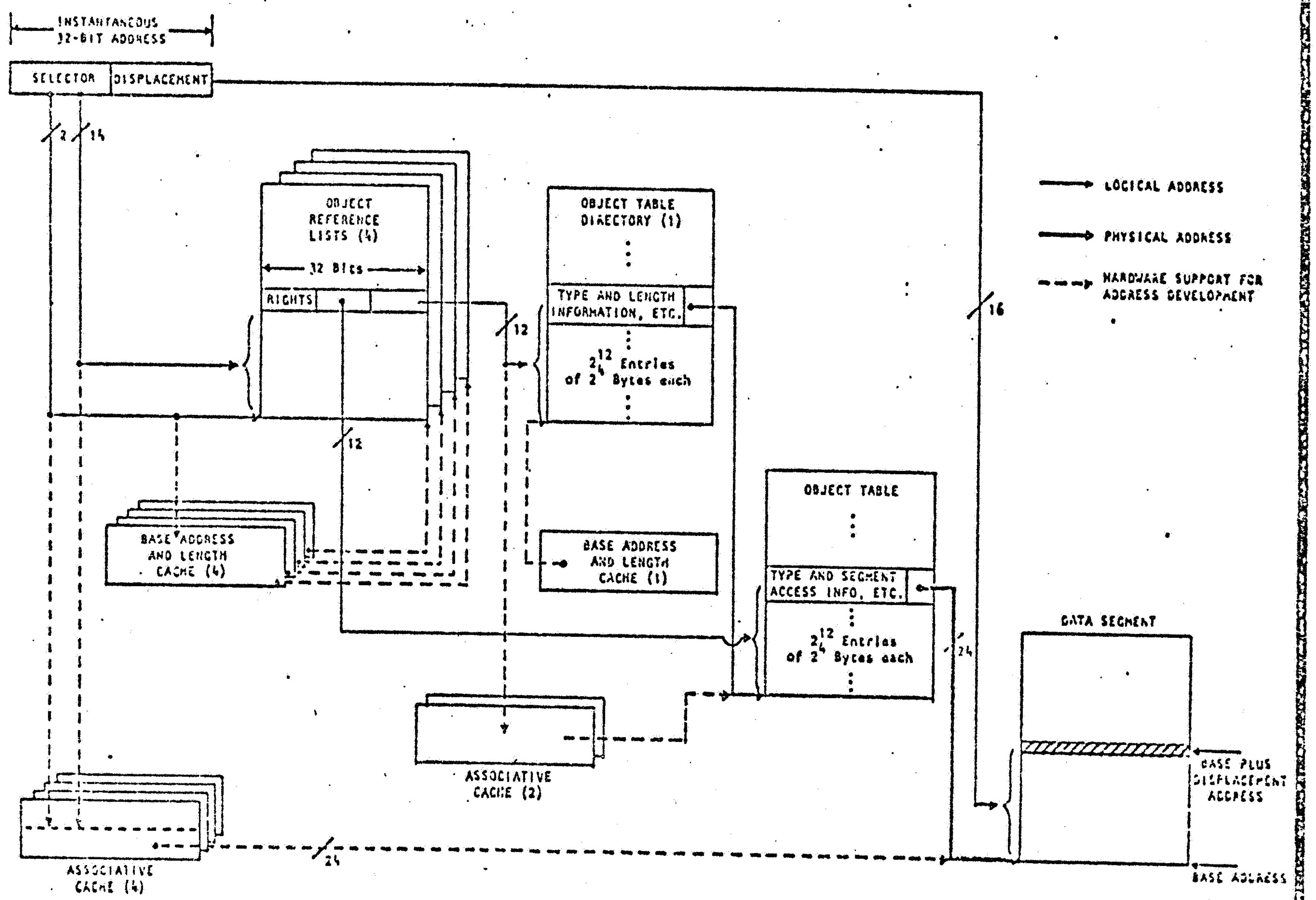


FOR MULTIFUNCTION APPLICATIONS

INTEL DELIVERS SOLUTIONS

43

23



INSTANTANEOUS 32-BIT ADDRESS
SELECTOR
DISPLACEMENT
2
14
OBJECT REFERENCE LISTS (4)
32 Bits
RIGHTS
12
12
OBJECT TABLE DIRECTORY (1)
TYPE AND LENGTH INFORMATION, ETC.
2^{12} Entries of 2^4 Bytes each
BASE ADDRESS AND LENGTH CACHE (4)
BASE ADDRESS AND LENGTH CACHE (1)
OBJECT TABLE
TYPE AND SEGMENT ACCESS INFO, ETC.
2^{12} Entries of 2^4 Bytes each
24
16
ASSOCIATIVE CACHE (2)
ASSOCIATIVE CACHE (4)
24
DATA SEGMENT
BASE PLUS DISPLACEMENT ADDRESS
BASE ADDRESS
LOGICAL ADDRESS
PHYSICAL ADDRESS
HARDWARE SUPPORT FOR ADDRESS DEVELOPMENT

MADE IN U.S.A.

432 ADDRESSING SUMMARY

- LARGE ADDRESS SPACES

INSTANTANEOUS	- 32 BITS or 2^{32} BYTES
SEGMENTS	- 2^{24}
~~SEGMENT SIZE~~	~~2^{16} BYTES~~
VIRTUAL	- 40 BITS or 2^{40} BYTES
PHYSICAL	- 24 BITS or 2^{24} BYTES

- PROTECTION
  - TYPE/LENGTH/RIGHT CHECKING ON A PER REFERENCE BASIS
- MODULARITY AND FAULT ISOLATION
  - LOCALIZED "NEED TO KNOW ADDRESSING"
  - SHORTER INSTRUCTION REFERENCES
- EASES O.S. SOFTWARE ISSUES
  - DYNAMIC RELOCATION
  - ADAPTIVE VIRTUAL MEMORY

- MULTIPLE PROCESSOR OPERATION
- EFFICIENT OPERATION - ON CHIP HARDWARE MAINTAINED CACHES
- BASIS FOR OBJECT ORIENTED ADDRESSING

MADE IN U.S.A.

1.21

iMAX 432 MULTIFUNCTION APPLICATION EXECUTIVE RELEASE 1

- **Run-Time Executive for Intel's 32-bit iAPX 432 Micromainframe**
- **Provides Basic Services For Any 432 Application**
- **Supports multiple iAPX 432 General Data Processors**
- **Library of Ada Packages**
  - **User Configurable**
  - **User Extendable**

Intel's iMAX 432, the Multifunction Applications Executive, is a software product designed for use with computer systems based on the iAPX 432 Micromainframe. iMAX 432 is a collection of software components which provide basic development and run-time services for 432 applications. When coupled with the operating system mechanisms embedded in silicon by the 432 hardware architecture, iMAX 432 provides efficient operating system functions to 432-based systems. The iMAX 432 Executive is appropriate for many applications including communications, EDP, process control, and industrial automation.

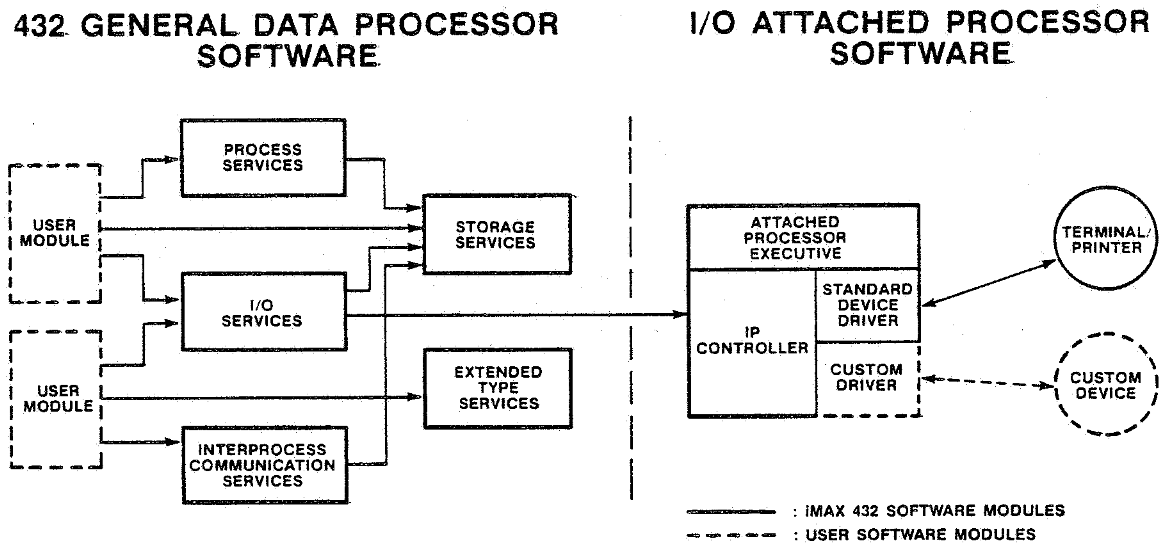


Figure 1. iMAX 432 Operating Environment

FUNCTIONAL DESCRIPTION

The iMAX 432 Executive cooperates with the iAPX 432 hardware to provide full support for the software transparent multiprocessing. This allows 432 systems to offer incremental performance capability through the addition of processor modules. iMAX 432 also manages memory and I/O resources. A device-independent I/O interface is provided, and the system contains drivers to support a terminal or character printer. The executive supports extended types, and allows the same degree of protection to these user-defined data types as to 432 hardware-recognized data types. Services are grouped into modules according to function. These include storage services, process services, I/O services, and initialization services.

iMAX 432 can be partitioned into two parts. The central system portion is coded in Ada and executes on the iAPX 432 General Data Processor(s). The I/O portion is written in PL/M 86 and executes on an attached processor in the I/O Subsystem.

The central system portion of iMAX 432 takes advantage of the compile time package interface checking. Parts of iMAX 432, the specifications to the user visible interfaces, are maintained in files which are accessible to the Ada compiler. When a programmer compiles an Ada module, the compiler checks the module's references to iMAX 432 against its stored iMAX specifications. Module interface errors are discovered easily and early at compile time. As illustrated in Figure 2, the iMAX 432 specifications provide compile-time checking, and the iMAX 432 software components provide run-time services.

FEATURE OVERVIEW

Structured Application Development Environment

iMAX 432 provides a uniform interface to 432 system hardware that remains consistent from application to application. iMAX 432 supports the software transparent multiprocessing environment as well as a range of memory and I/O configurations.

Modular Structure

The user's view of iMAX 432 is that of a library of packages whose interfaces to user-written code are procedure calls. A 432 system does not distinguish between modules of the executive and modules of application code. Protection mechanisms are the same for both. The user can extend system function according to the application's requirement. This modular approach allows quick user familiarity with the services provided.

Operating System Functions in 432 Hardware Architecture

The iMAX 432 Executive cooperates with the 432 hardware architecture, which provides many conventional operating system functions embedded in

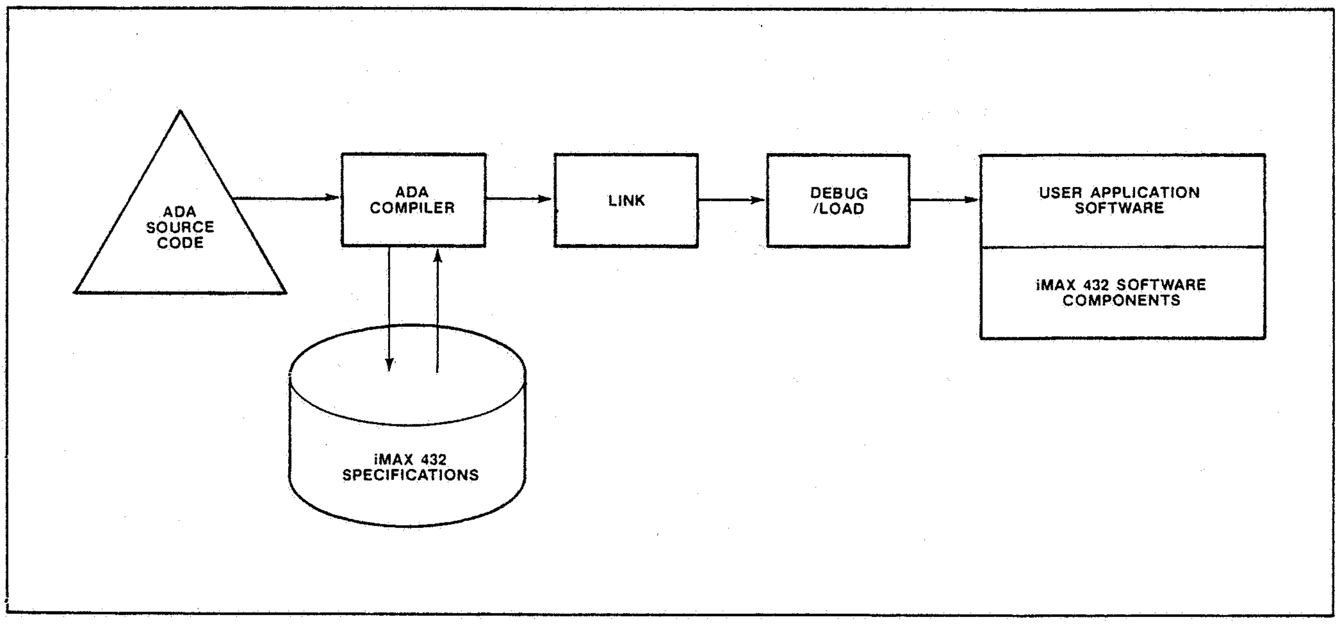


Figure 2. Software Development Environment

silicon. Functions implemented in hardware operate much faster than if left to software, and provides a secure multiprocessing environment. Operating system mechanisms are distinguished from operating system policies by the 432 system architecture. Many of these mechanisms are handled by 432 hardware, including process scheduling and dispatching, processor management, storage allocation, protected data sharing, and exception handling. The policies which control these mechanisms are implemented through parameters set by calls to iMAX 432. This provides flexibility to the user, who can use efficient hardware operations to implement a choice of operating system policies.

Storage Services

Release 1 of iMAX 432 supports dynamic relocation of segments, along with their creation and destruction. Deallocation of memory is provided by a parallel garbage-collection process. Those memory segments which the application program no longer needs are reclaimed automatically. This process executes periodically in parallel with other processes in the system; the system does not stop to invoke garbage collection. Memory compaction is also automatic, and is invoked whenever there is not an adequate block in memory to grant an allocation request.

Process Services

An application for the iAPX 432 is composed of multiple concurrent processes. iMAX 432 manages these processes for user-supplied routines, adding information to make an execution environment recognizable to the hardware. Any number of processes can be created, set running, and destroyed.

iMAX 432 provides scheduling calls to set priorities for processes. Scheduling parameters can be static, or can be updated by a concurrent policy-update process. iMAX 432 provides a means to monitor the flow of control within a process using the trace mode specification.

Process communication and synchronization are accomplished using communication ports built by iMAX 432. Communication ports are queues where processes send and receive messages. Messages can be queued at communication ports in either FIFO or priority/deadline order.

I/O Services

Input/output operations on a 432 system are accomplished through separate, peripheral subsystems. An attached processor (AP) controls device drivers and communicates to the 432 General Data Processors through a 432 Interface Processor (IP). Software controlling the IP is resident on the attached processor, and is supplied with iMAX 432.

Release 1 of iMAX 432 contains facilities to support a character printer or terminal. The user can provide an interface to other I/O devices by adding a device driver to the AP and a 432 GDP interface module.

Initialization Services

The iMAX 432 initialization routine manages system startup. iMAX 432 supplies initial data structures and loads them into 432 system memory. The executive then signals the 432 General Data Processors to start.

Subsequent Releases

The iMAX 432 Executive was engineered for phased release. The second release will contain additional facilities such as virtual memory support, a file system, a human interface, and drivers for additional I/O devices.

intel®

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara CA 95051 • (408) 734-8102 x598

SYSTEM 310 HARDWARE OVERVIEW

	SYSTEM 286/310-41	SYSTEM 286/310-40	SYSTEM 286/310-17	SYSTEM 286/310-4	SYSTEM 86/310-3A, -3	SYSTEM 86/310-2A, -2	SYSTEM 86/310-1
Microprocessor	80286 (6 MHz)	80286 (6 MHz)	80286 (6 MHz)	80286 (6 MHz)	8086 (5 MHz)	8086 (5 MHz)	8086 (5 MHz)
Numeric Coprocessor	80287 (4 MHz)	80287 (4 MHz)	80287 (4 MHz)	80287 (4 MHz)	8087 (5 MHz)	8087 (5 MHz)	N/A
RAM Memory Expandable To	1.0 MB W/ECC 4.0MB W/ECC	1.0 MB W/ECC 4.0MB W/ECC	512KB W/ECC 4.0MB W/ECC	512KB W/ECC 4.0MB W/ECC	640KB 896KB	256KB 896KB	128KB 896KB
Mass Storage	320KB Diskette (Formatted) 42MB Wini (Unformatted)	320KB Diskette (Formatted) 42MB Wini (Unformatted)	320KB Diskette (Formatted) 19MB Wini (Unformatted)	320KB Diskette (Formatted)	320KB Diskette (Formatted) 19MB Wini (12MB in -3) (Unformatted)	320KB Diskette (Formatted) 19MB Wini (12MB in -2) (Unformatted)	320KB Diskette (Formatted)
I/O Ports: Serial Parallel	(10) RS232 (1) Centronics	(2) RS232 (1) Centronics	(2) RS232 (1) Centronics	(2) RS232 (1) Centronics	(1) RS232 (1) Centronics	(1) RS232 (1) Centronics	(1) RS232 (1) Centronics
MULTIBUS Expansion Slots	3 @ 0.65in.	4 @ 0.65in.	4 @ 0.65in.	1 @ 1.20in. 4 @ 0.65in.	4 @ 0.65in.	5 @ 0.65in.	1 @ 1.20in. 5 @ 0.65in.
DC Power Output	270 Watts Maximum +5V@45A +12V@4.7A −12V@4.7A	270 Watts Maximum +5V@45A +12V@4.7A −12V@4.7A	220 Watts Maximum +5V@30A +12V@4.7A −12V@4.7A	220 Watts Maximum +5V@30A +12V@4.7A −12V@4.7A	220 Watts Maximum +5V@30A +12V@4.7A −12V@4.7A	220 Watts Maximum +5V@30A +12V@4.7A −12V@4.7A	220 Watts Maximum +5V@30A +12V@4.7A −12V@4.7A

PERIPHERAL EXPANSION OVERVIEW

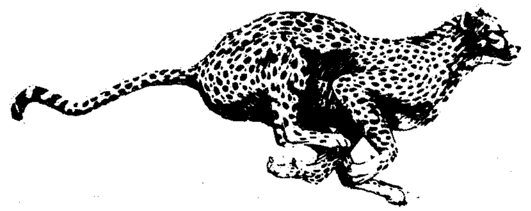
SYS 311 CODE No.	TAPE 45 MB	FLOPPY 320 KB	WINI (1) 42 MB	WINI (2) 42 MB
SYS311-B-01	X	X	X	X
SYS311-B-02	✓	X	X	X
SYS311-B-03	X	X	✓	X
SYS311-B-04	✓	X	✓	X
SYS311-B-05	✓	✓	✓	X
SYS311-B-06	X	X	✓	✓
SYS311-B-07	✓	X	✓	✓
SYS311-B-08	✓	✓	✓	✓

STARTER 310 PRODUCT GUIDE

CPU TYPE	SYSTEM	HARDWARE ONLY	RMX STARTER KIT	XENIX STARTER KIT	WINI	FLOPPY	RAM	EXPANSION I/O (RS232)
–	–	SYP310-90	–	–	–	–	–	–
8086	86/310-1	SYP310-1	–	–	–	320 KB	128 KB	–
	86/310-2	SYP310-2	–	–	12 MB	320 KB	256 KB	–
	86/310-2A	SYP310-2A	–	–	19 MB	320 KB	256 KB	–
	86/310-3	SYP310-3	SYS310-3R	–	12 MB	320 KB	640 KB	–
	86/310-3A	SYP310-3A	SYS310-3AR	–	19 MB	320 KB	640 KB	–
	86/310-3A	SYP310-3A	SYS310-3AR	–	19 MB	320 KB	640 KB	–
80286	286/310-4	SYS310-4A	–	–	–	320 KB	512 KB	–
	286/310-17	SYS310-17A	SYS310-17R	SYS310-17X	19 MB	320 KB	512 KB	–
	286/310-17	SYS310-17M	SYS310-17MR	SYS310-17MX	19 MB	320 KB	512 KB	4-via SXM 544
	286/310-40	SYS310-40A	SYS310-40R	SYS310-40X	42 MB	320 KB	1 MB	–
	286/310-40	SYS310-40M	SYS310-4MR	SYS310-40MX	42 MB	320 KB	1 MB	4-via SXM 544
	286/310-41	SYS310-41A	–	SYS310-41X	42 MB	320 KB	1 MB	8-via SXM 48

For further information, demonstrations, advice on available application software, customisation and a wide range of compatible products

PHONE 0344 482211



intel® + BYTECH™

Bytech Ltd, 2 The Western Centre, Western Road, Bracknell, Berkshire RG12 1RW. Telex: 848128.

Bytech (North), Bahama Road, Haydock, Merseyside WA11 9XB. Telephone: 0942 721171.

'Bytech' is a trademark of Bytech Limited.

INTEL SYSTEM 310 FROM BYTECH

SYSTEM 310

The high performance microsystem family designed to provide the OEM and System Builder with
* FLEXIBILITY—to meet the diverse application needs of today's highly competitive markets.
* EXPANDABILITY—to quickly absorb new advances in VLSI functionality and performance needed to secure decisive product wins in the application and markets of tomorrow.
* GUARANTEES—elimination of 'dead-ends' in the OEM's product future, Multibus architecture provides compatibility and expandability for years to come.

SYSTEM 310 FAMILY

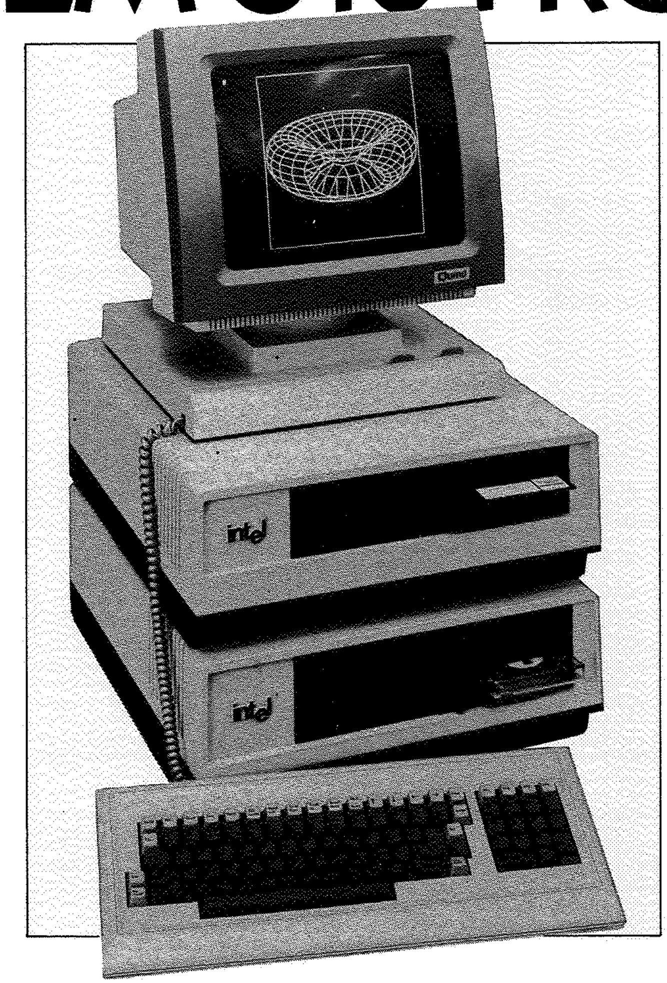
* SYSTEM 310 is built entirely on standards:-
  ● Buses ● Interfaces ● Peripherals ● Software.
* SYSTEM 310 is 'OPEN' to:-
  ● New VLSI expansion ● Application—specific configuration ● Hardware & Software from Independent Vendors ● All levels of integration.
* SYSTEM 310 is compatible with over 1,000 Multibus products available today from over 170 vendors including:-
  ● Disk controllers—SCSI, SASI, SMD, ST506/412 etc. ● Graphics interfaces—GKS, NAPLPS, Virtual Device Driver etc. ● Serial I/O—Sync/Async/Bi-sync, HDLC/SDLC, etc. ● Local area networks—Ethernet, ISO model compatibility ● Global area networks—X25, SNA, 2780/3780, RJE, etc. ● Many other products for configuring unique functionality into System 310.
* SYSTEM 310 is available in over 20 standard configurations off-the-shelf from Bytech (see Configuration Guide overleaf).
* SYSTEM 310 is available as a desk-top, rack-mounting or floor-standing unit.
* Winchester, Diskette and Tape support.
* Choice of 'Micro-engines'—Industry standard 8086 micro and 80286 super-micro.
* SYSTEM 310 offers an unprecedented range of price/performance options.

SYS311—PERIPHERAL EXPANSION SUBSYSTEM

* Designed to complement SYSTEM 310.
* 3 full height 5¼" bays.
* 45Mb Cartridge tape streamer.
* Increased Winchester storage capacity.
* Increased Diskette storage capacity.
* The SYS311 can be configured to any combination of peripherals required by OEM's.

SYSTEM 310 STARTER PACKAGES

Fast, low-risk, easy entry to technology leadership with a complete, comprehensive package for OEM's and system builders, tailored to get application solutions up and running quickly and consequently provide a fast-to-market system level product.
* Choice of hardware units.
* RMX or Xenix packages available.
* Complete Operating System software including:-
  ● Assemblers ● High-level languages ● Utilities and development tools ● 5 day hands-on Intel workshop training course ● 2 days on-site Intel consultancy support ● Software support contract:-
  ● Hot-line technical support service ● Regular software updates ● Software performance reporting service ● Insite technology exchange membership ● Newsletters.
  ● Hardware Warranty ● Systems and software product registration.



SYSTEM 310 SOFTWARE

INTEL'S RMX OPERATING SYSTEM optimises SYSTEM 310 for real-time, multi-tasking, time-critical applications such as:-
● Factory automation ● Distributed access and control functions
● Industrial control ● Communication networks ● Image processing
where fast response to the 'real-world' is required.
* Fortran, PL/M, Pascal, C, Basic languages supported.
* Assemblers, utilities and development tools supplied.
* Industry's most highly configurable operating system.
* Built-in drivers for many of Intel's board level products.
* Off-the-shelf application packages.
* Powerful interactive configuration utility for fast and efficient system configuration, generation and performance tuning.

INTEL'S XENIX OPERATING SYSTEM optimises SYSTEM 310 for interactive, multiuser applications such as:-
● Distributed data processing ● Business data processing
● Distributed software development ● Mainframe communication and interaction ● Scientific, engineering and graphics applications
● Off-the-shelf application packages
where superior human-machine interaction and performance is required.
* Fortran, Cobol, C, Basic languages supported.
* Wealth of development tools and utilities.
* Superior data reliability and integrity—record and file locking, automatic disk recovery.
* Enhanced human interface.
* Interactive configuration utility for fast and efficient system configuration, generation and performance tuning.

INTEL'S MSDOS OPERATING SYSTEM complete and fully licensed version of microsoft's MSDOS 2.11 operating system.
* Complementary operating system for SYSTEM 310 running RMX or XENIX.
* Stand-alone MSDOS also available.
* Wealth of off-the-shelf languages and application packages available from many independent vendors.

SYSTEM 310 FAST FROM BYTECH

Bytech hold large stocks of SYSTEM 310, so we can deliver fast off-the-shelf.

For a comprehensive data pack covering all aspects of SYSTEM 310 hardware and software as well as information and technical advice on all aspects of configuration and customisation, call us now on

PHONE 0344 482211

intel® + BYTECH™

'Bytech' is a trademark of Bytech Limited.

iSCurrents

Published by Intel Scientific Computers

Winter 1989/90

Intel Announces iPSC/860: 7.6 GFlop System Unveiled in London

U.S. Debut at Uniforum in January

Beaverton, OR (December 18, 1989) On January 10, 1989, Intel Scientific Computers will unveil the iPSC/860, the first in a series of supercomputers based on Intel's advanced RISC microprocessor technology. The American debut of the iPSC/860 will be in the Intel booth at Uniforum from January 23-25.

The iPSC/860 is an expandable supercomputer, ranging from 8 to 128 processors. System performance ranges from 480 MFlops to 7.6 Gigaflops.

Random access memory is also expandable, and currently ranges from 64 Megabytes to 2 Gigabytes.

Beta sites include NASA-Ames and Oak Ridge National Laboratories. Customers include Boeing Advanced Systems and others.

The next issue of iSC Currents will carry several feature articles on the iPSC/860. Until then...

Pacific Sierra and Intel Announce Intention to Offer CASE tools for Dusty Deck FORTRAN Programs

FORGE to be the First Set of Programming Tools Designed to Help Developers Port FORTRAN to iPSC Family

Reno, Nevada (November 13, 1989) — Intel and Pacific Sierra Research announced today their intention to develop a series of programming tools aimed at helping developers port existing FORTRAN code to the iPSC family of high performance systems.

The first offering will be FORGE, an interactive set of tools that simplifies the task of developing, supporting, and evolving FORTRAN programs.

The Nature of the Problem

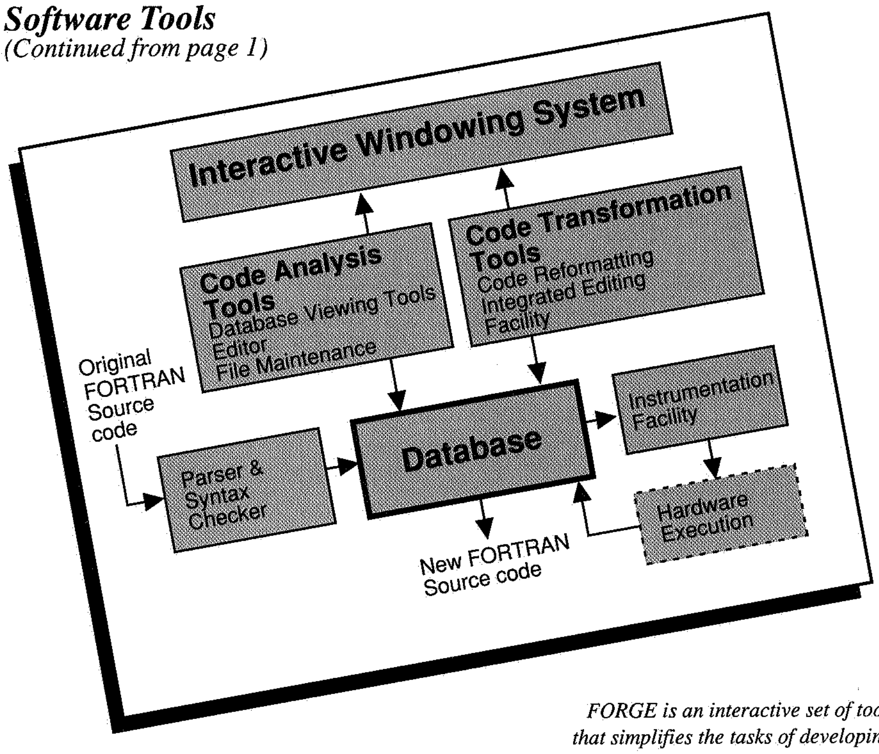
Transforming a serial FORTRAN program into a "parallel" structure requires decomposing the existing code and restructuring it in a more efficient parallel form. To produce the best parallel code, the programmer must understand as much as possible about the existing code before the restructuring process

(Continued on page 2)

IN THIS ISSUE OF iSC

Software Tools

(Continued from page 1)



FORGE is an interactive set of tools that simplifies the tasks of developing, supporting, and evolving FORTRAN programs.

begins. Very often, serial FORTRAN programs being converted have been developed by scientists or engineers, rather than the programmer doing the conversion. The inherent programming difficulties contributed by the original programmer, combined with often poor or non-existent documentation, markedly increase the difficulty of the code transformation process.

Porting serial code to supercomputers or creating highly efficient parallel code for them, whether they are shared memory or distributed memory architectures, requires restructuring codes at a higher level to generate the most efficient parallel algorithms. Current automatic parallelizing compilers are of limited help, because they extract only the parallelism that exists in the low-level DO loops.

While fully automated solutions are not technically feasible, today a comprehensive set of programming tools is available that provides the required high-level approach for software developers. FORGE is an interactive set of tools that can simplify the tasks of developing, supporting, and evolving FORTRAN programs and thus can enhance the effectiveness of programmers in developing or porting code for parallel systems.

The FORGE System

FORGE is built around a FORTRAN database that provides a parsed version of the serial code to assist the programmer in his analysis and understanding of the program content and structure. Rather than having to dig through stacks of listings, FORGE allows the programmer to efficiently examine the code through a Quick-Look tool set designed for easy query of the database information. The graphical interface is a UNIX workstation based on X-windows.

The baseline FORGE system consists of a FORTRAN syntax parser, database generator, program maintenance utility, and database viewing tools that enable the programmer to examine and manipulate an entire FORTRAN program as a single entity. The analysis capabilities of FORGE enable the programmer to understand program flow and variable usage. Run time statistics are provided through an optional instrumentation facility. Code manipulations may be performed by the programmer using an editor of choice. FORGE is well-documented with a HELP facility to assist the programmer in using the analysis tools.

Management Resources

A program database is created from a group of program units, grouped as "packages" by the programmer. A package is a collection of program units that make up the entire program or contribute to a common functionality. After user creation and definition of the program units that comprise a package, FORGE parses each unit and creates the database containing symbol usage and control information on both a global and a local basis. Several packages may be analyzed individually or as a single entity. The database contains information on all symbol usage throughout the program. This enables interprocedural analysis of both control and dataflow.

If FORGE encounters a FORTRAN error, it initiates the system editor to allow the programmer to immediately modify the code and correct the error. Upon exiting the editor, the new code is reparsed. The size of the parsed database code may be from 3 to 5 times larger than the original FORTRAN code. FORGE can support programs of any size, given the availability of adequate disk storage.

Package manipulation tools provide capabilities for selecting, creating, and deleting packages and viewing source files; for analyzing and modifying packages; for defining and editing files in packages; and for maintaining several versions of each routine. Database viewing tools enable the programmer to browse the database and perform such functions as: trace variables, display constants, query database, list all routines, show COMMON block grid, and perform consistency checks on both COMMON blocks and argument passing.

The COMMON block grid illustrates how COMMON variables are used throughout the program to restructure COMMON storage. Equivalencing and COMMON block differences, as well as name changing through argument pass-

(Continued on page 12)

Technical University of Munich

The Technical University (TU) of Munich and Intel have a long history. The school has been using Intel chips, boards, development systems, and programming tools in its classrooms for ten years. More recently, TU Munich purchased an iPSC/2 after examining a transputer-based megacluster machine and a tree-structured parallel machine. The iPSC/2 won out because of its Direct-Connect architecture, integrated memory management, a load balancing feature, and its open architecture, which provided total access to the software.

"The iPSC/2 was the first machine to offer full connection between all the nodes," said Professor Thomas Bemmerl, the director of TU Munich's parallel research group. "It's a big step on the hardware manufacturer's part towards simplifying parallel programming."

Simplifying parallel programming is largely what Bemmerl and his cohorts at TU Munich are engaged in. Three years ago, they started developing a set of programming tools to make parallel programming as easy as sequential programming. To date they have completed a portable parallel operating system, known as the MKK, a method for monitoring the behavior of parallel software, a debugger, a performance analyzer, and a visualization tool.

TU Munich's goal is to come up with application development tools that will make parallel machines easy to program, thus broadening the appeal and usage of parallel machines. Their accomplishments to date have been impressive.

The Three Stages of Parallel Programming

The parallel research group at TU Munich reasoned that parallel programming will pass through three stages. The first and current stage is manual programming, where every line of code must be written by hand. The next is interactive programming, where developers "plug in" values to stylized forms, much like the application generators used today in sequential programming. The final stage, which Professor Bemmerl's group intends to help usher in, will be fully automatic programming.

TU Munich campus

TU Munich identified three problems with today's parallel software development methods: they are too time-consuming and, thus, too expensive. Parallel software development tends to be architecture-specific. Finally, the complexity of the software often slows down the parallel hardware.

TU Munich established three design goals to solve these problems: 1) increase programmer productivity by developing an integrated toolkit, 2) create a portable tool environment (one supporting multiple microprocessors), and 3) optimize performance by creating close cooperation among hardware, operating system, and tools.

The resulting toolkit has the following characteristics:

1) Tight integration between tools (a shared common symbol database and graphic primitives)

2) Support of multiple abstraction levels, allowing developers to work in assembler and high level languages, even at the process level

3) Support of a number of instrumentation techniques, i.e., hardware monitoring, software monitoring, hybrid monitoring, and simulation

4) Portability across various parallel architectures

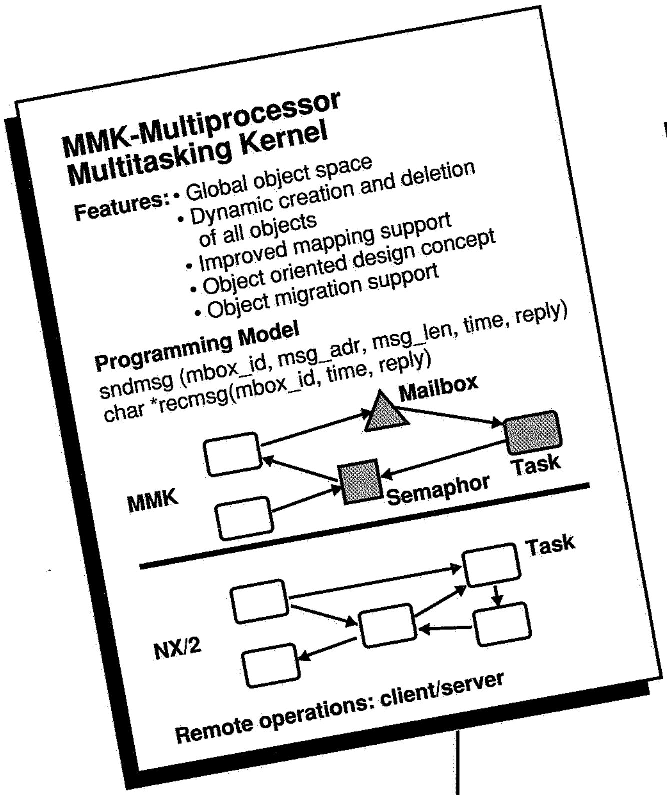
5) Adaptability for different process concepts, programming languages, and compilers

6) Expandability with new tools and functionality

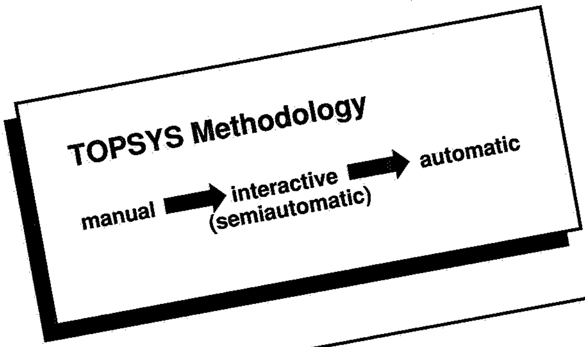
7) Friendly, menu-oriented human interface

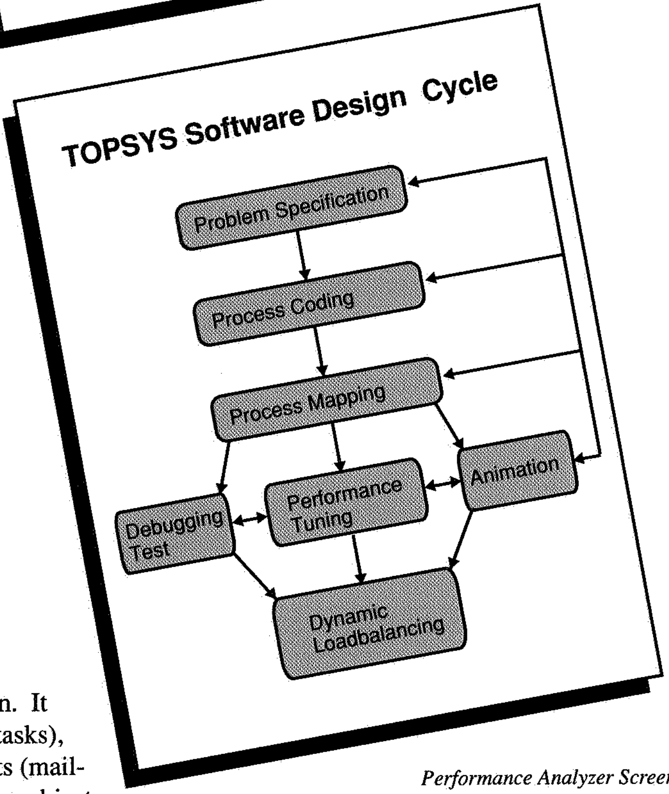
Multiprocessor, Multitasking Kernel (MKK)

TU Munich's approach to software tool design has been to initially create a transparent layer of systems-level software to sit between the tools and the


MMK-Multiprocessor Multitasking Kernel
Features: • Global object space
• Dynamic creation and deletion of all objects
• Improved mapping support
• Object oriented design concept
• Object migration support
Programming Model
sndmsg (mbox_id, msg_adr, msg_len, time, reply)
char *recmsg(mbox_id, time, reply)
Mailbox
Task
Semaphor
MMK
Task
NX/2
Remote operations: client/server

MMK Illustration





Performance Analyzer Screen

hardware. Their strategy is similar to that of the ISO/OSI network model, where a layer at level "i" needs no knowledge of software or hardware installed at levels lower than "i." This layering strategy was the key to portability between architectures and to future expandability.

TU Munich's portable layer is called the Multiprocessor Multitasking Kernel (MMK), which sits atop a multiprocessor operating system. Its main purpose is to balance the load among processors, and it does this by automatically routing tasks based on processor availability.

Few multiprocessor operating systems perform load balancing. Instead, the programmer is required to keep tabs on which processor is running which process. Whenever the programmer sends a message to a specific process, he has to specify in his program the load on which the process is localized.

With MMK, the programmer does not have to keep in mind which process is running; MMK keeps track of it for him. This transparent multitasking process capability is the main feature of MMK.

MMK also allows developers to think about their parallel programs in an object-oriented fashion. It offers active objects (tasks), communication objects (mailboxes), synchronization objects (semaphores) and storage objects (memory). The object-oriented architecture is another step toward freeing the programmer from considerations of specific architectures, letting him concentrate on the application instead.

MMK objects are locally and globally available, meaning it makes no difference to the programmer whether the objects are located on a local or a remote processor node. All MMK objects can be dynamically created and deleted, implying that the number of objects is limited only by the amount of available memory. With the current version of MMK, dynamic object migration is also possible, meaning that objects can be migrated from one processor load to another to improve total processing speed.

Ultimately, the group will implement a dynamic load balancer that will continuously migrate objects between processors to achieve optimum performance throughout runtime.

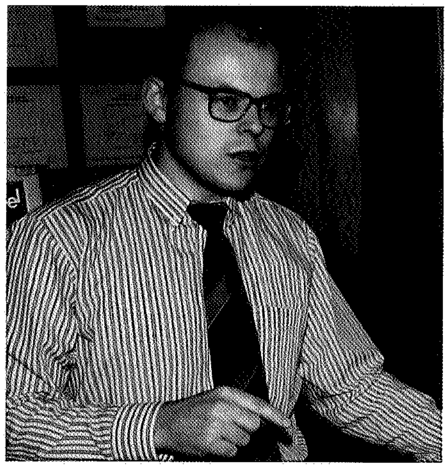
Peeking in on the System Without Slowing Things Down

After getting MMK in place, Bemmerl needed to create one more important element before tackling the tools themselves. That element was some technique for monitoring the execution of parallel programs. Monitoring can be done in hardware, software or a combination of both.

Bemmerl's group implemented one of each kind of monitor. All three support interactive tool use. None require extra recompilation. And, the tools need not know which monitoring technique they are using.

The transparent nature of MMK makes interchangeability possible. For example, the hardware monitor is easily replaced with the software monitor without changing the upper layers of the MMK model. All three monitors offer the same interface to the upper layers; they differ only in the degree of retardation of program execution.

The monitors allow the programmer to specify predicates and conditions (e.g., breakpoints, trace conditions, triggers, etc.) about the program execution at several abstraction levels. He can display and modify contents of memory cells, I/O ports and MMK objects.


Thomas Bemmerl

At the host level, the monitors manage events, actions and symbols. Based on primitive events and action mechanisms, the tools can specify complex predicates about the dynamic behavior of the target system. The specification of events and actions is based on symbols in the monitored program.

The monitoring system is completely distributed, meaning each processor is monitored individually and the data then combined and presented to the programmer under the friendly guise of a graphical user interface. The programmer can opt to look at systems-level activity, at a specific node, or even at a specific process.

The star performer of the monitoring system is the hardware monitor. Non-intrusive monitoring is done in hardware. Each processor is individually monitored through hardware logic. Each hardware monitor is adapted to the signals of the processor it's monitoring, so when a statement is executed, it's recognized by the monitor in parallel with the execution of the load program. As a result, program execution is not affected at all.

The hardware monitor currently consumes an entire board per processor. Bemmerl's team is working with the University's electrical engineering department and Siemens Corporation to shrink the board to a single chip.

Tools With a Friendly Face

So far, Bemmerl's group has completed a debugger, performance analyzer and visualizer under the MMK model. Remaining are a configurator, mapper, load generator, load balancer and test system. All the tools are tightly integrated and all are accessible from a common graphical user interface.

The debugger lets the programmer display and modify the states of programs running on the parallel processor. A powerful debugging language allows specification of complex predicates about the dynamic execution of programs.

The performance analyzer gives the programmer graphical information about the efficiency of the communication between processes or processor elements, about the activation of procedures, access to variables and operating system objects. The programmer can then localize bottlenecks and optimize mapping of processes into processors.

The visualization tool offers another way of displaying the dynamic behavior of multiple processors. This tool shows graphically the flow of communications between processes or processors, as well as control and data flow.

The graphical user interface, implemented on a Sun workstation, makes frequent interactive use of the toolkit easier and more engaging.

The Barriers are Falling

Bemmerl has three goals: to demonstrate the portability of the toolkit to other parallel processor architectures, to shrink the hardware monitor board to a chip and to validate the toolkit on real-life applications.

"We are very proud of the hardware monitor in particular," concludes Bemmerl. "It's a very complex design and represents a major breakthrough in the ability to observe parallel programs in action."

"We hope our work will make a difference in helping parallel machines become more widely accepted by making them easier to program. Many people stand to benefit."

TU Munich is using a 32-node iPSC/2 with 80 Megabytes of memory and a separate pair of processors dedicated to I/O. The I/O system has 2.8 Gigabytes of secondary storage.

Oklahoma State University Processes Images and Sound with Hypercube

Submitted by Professor Keith Teague, School of Electrical and Computer Engineering, Oklahoma State University. Dr. Keith Teague manages the iPSC/2 hypercube parallel processing laboratory at Oklahoma State University.

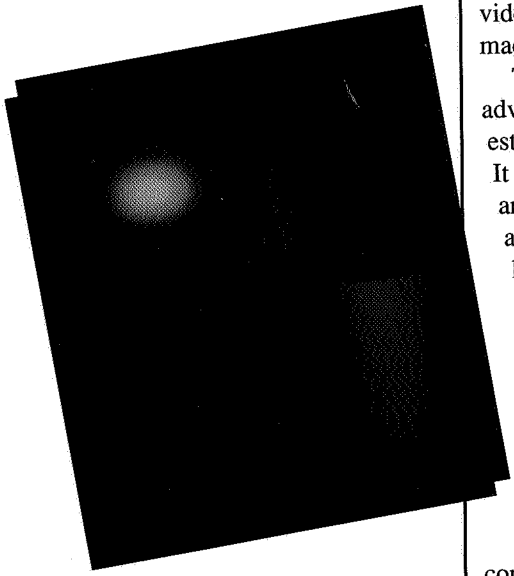
With a grant from DARPA and the State of Oklahoma, Oklahoma State University purchased an Intel iPSC®/2 supercomputer two years ago. It's at the heart of four graduate research projects in signal and image processing:

- Hypercube Image Processor
- Superquadric neural network
- Hypercube Ray Tracer
- Digital coding of high-fidelity audio

Hypercube Image Processor

We developed the Hypercube Image Processor (HIP) here at Oklahoma State University to be both an interactive image processing system and a framework for developing parallel image processing algorithms.

In the area of parallel processing, it's well known that the biggest challenge is developing software. The idea of HIP is to make parallel programming easier. It does this by providing standard methods of data decomposition. Decomposition is an ordered method of breaking the data into finite elements so that it can be parcelled out to various processors. By developing standard methods and providing libraries for common graphical elements, programmers won't have to reinvent the wheel every time.



In its library are decomposition methods for horizontal strips, checkerboards and pyramids, as well as numerous functions to operate on these images. Custom decompositions and functions can also be programmed and incorporated into HIP. The significance of this is that no programming effort is ever wasted; code produced by other programmers can be easily used in new applications.

Another beauty of HIP is that it can be used by both programmers and end users. Programmers access HIP through C and users access it through a UNIX® shell. To a user, HIP looks much like a serial image processor with a very large number of buffers and excellent response time. Its parallelism is hidden.

Ron Daniel, a Ph.D. student, and Michael Carter, a Master's student, are working on this project.

Superquadric Neural Network

Ron Daniel is also using HIP and the iPSC/2 to develop a neural network to model 3D objects for machine vision. The network will model individual components with geometric solids known as superquadrics. These graphics primitives can describe a wide variety of physical shapes with a very compact representation. Furthermore, the decomposition of objects into simpler component parts is a very similar process to the decompositions the human brain makes. These high-quality models should provide a significant step forward in machine vision.

The neural network has several advantages over a serial algorithm for estimating the superquadric parameters. It can work with more complex scenes, and its massive parallelism would allow real-time processing on custom hardware. The network being developed and simulated in the iPSC/2 provides much faster simulations than could be performed on non-parallel machines.

Hypercube Ray Tracer

Ray tracing is a popular technique for rendering realistic computer-generated images. It is, however, extremely costly in terms of computing time. The iPSC/2 hypercube computer is being used to study parallel techniques that reduce the time required to generate complex images.

Research at Oklahoma State University has confirmed that ray tracing is highly "parallelizeable" and is well-suited to implementation on the hypercube. It exhibits a fine-grained parallelism at the pixel level - each pixel is totally independent of pixels around it. Thus, the pixels may be assigned to the iPSC/2 nodes in a way most efficient for intensity calculation. We have demonstrated speedup in excess of 31 times on a 32-node iPSC/2 and expect this speedup to continue linearly to a large number of nodes.

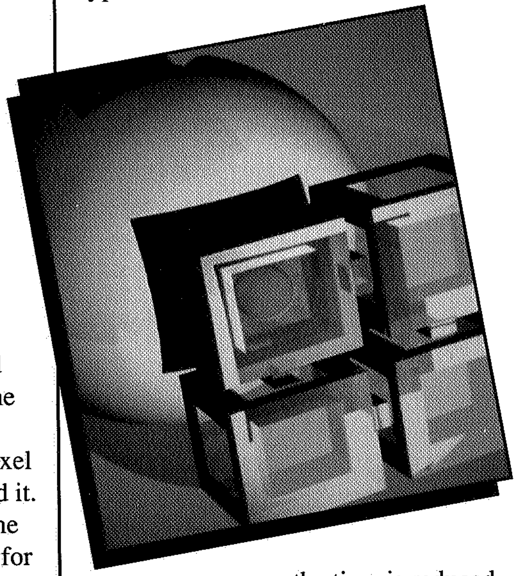
Michael Carter is working on this project as part of his Master's thesis.

Digital Coding of High-Fidelity Audio

The fourth program revolving around the iPSC/2 hypercube computer is an investigation into digital audio code compression. It is our belief that such methods could stimulate the development of dialup music services that would turn tomorrow's telephone into a piece of high-fidelity digital audio equipment, with quality very close to that of compact disc music.

Research into digital audio compression for digital telephone transmission is currently going on in France, Germany, Switzerland, Japan, and other locations in the U.S., such as AT&T Bell Labs. Although the quality of digital music is extremely high, the conversion of analog to digital audio produces files that are huge - much too large to send over digital phone lines.

The iPSC/2 and new signal processing methods are allowing us to shrink digital music signals. One such method is called vector quantization. Although it shows great promise as a code compression scheme, it is very computationally intensive. On a machine like a VAX®/780, it can take days to process only a few minutes of music. On the hypercube,



the time is reduced by a factor of up to 100. Since the techniques need to be developed, tried, tested and then re-developed, again and again, faster turnaround is essential.

Gary Pearson, a Ph.D. candidate at Oklahoma State University, is the primary investigator of audio compression for digital telephone transmission.

The iPSC/2 is also used for instruction in a graduate projects course on parallel processing.

SERC, Daresbury

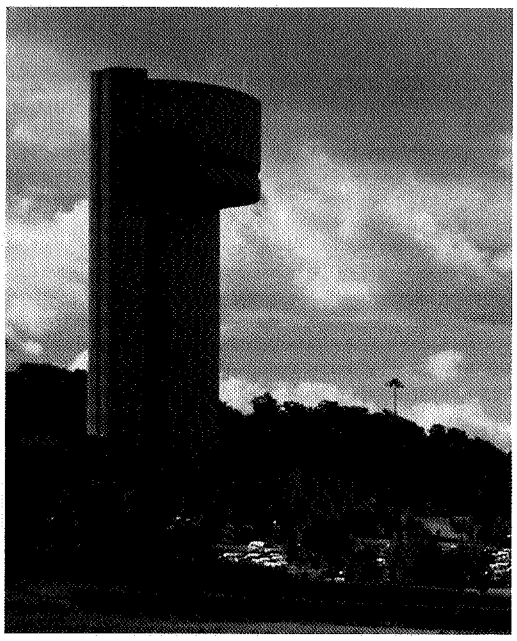
Located approximately150 miles northwest of London, between Manchester and Liverpool, the Daresbury Laboratory (DL) is one of two national laboratories in the United Kingdom under the control of the Science and Engineering Research Council (SERC).

SERC's main responsibility is to guide and fund research in physical science and engineering in U.K. universities. This is achieved in part by SERC running two laboratories and two observatories which provide facilities for academic research that is too large or unsuitable for siting at individual universities.

One of the major areas of scientific computation at the Laboratory is computational chemistry, with the Theory and Computational Science Division the focus of activities. The Computational Science Group at Daresbury is responsible for supporting university research in an increasingly broad range of computational subjects, which is largely accomplished by means of the Collaborative Computational Projects (CCPs). Each project is organized by a working group, which defines the scientific direction of the work, and carries out the agreed program. The general aims of the projects are to encourage basic research in the given areas, to develop and maintain relevant software packages, and to disseminate information among University and other research groups by organizing 'symposia' or 'workshops.'

Nine projects are currently supported by SERC, including those in the areas of Quantum Chemistry, Continuum States of Atoms and Molecules, Computational Studies of Surfaces, Protein Crystallography, Computer Simulation of Condensed Phases, Heavy Particle Dynamics Analysis of Astronomical Spectra, Electronic Structure of Solids and Plasma Physics.

The Theory and Computational Science Division pioneered the exploitation of vector machines in the U.K., and it continues to provide a national focus in the area of parallel processing, through the activities of the Advanced Research Computing Group (ARCG), a group with expertise in the use of multiprocessor systems.



ARCG works in close collaboration with universities and with industry in developing parallel processor algorithms, with major application codes from a variety of disciplines running on MIMD configurations typified by the Intel iPSC/2 and other systems.

The Advanced Research Computing (ARC) project at Daresbury is now centered around the iPSC/2, the second generation systems of hypercube architectures from Intel Scientific Computers and another, transputer based system. Experience with the hypercube since installation suggests key features that have led to a wide range of application software ported.

- The machine has clearly been designed as a FORTRAN engine from the outset, with a wide-ranging software library provided for communications, etc.
- The UNIX environment of the hypercube has led to a natural, rather than a forced, integration into the surroundings typified by the Convex C220, Ardent Titan-2, and Sun workstations. The ability, for example, to run the remote hosting software from any of these machines provides a natural route for scientists to access the machine. Our original belief that the iPSC/2 presents a user environment far superior to any of its contenders has been substantiated by our experience to date.

At installation the iPSC/2 comprised 32 multiple instruction multiple data (MIMD) nodes placed on a hypercube with direct-connect channels. Each node comprises an Intel 80386 processor (4 MIPs in 32-bit arithmetic), 4 Mbyte memory, a Weitek 1167 SX scalar accelerator, plus 64 Kbyte instruction and data cache. Vector processing on each node is achieved through a vector accelerator (6.6 Mflop in 64-bit or 20 Mflop 32-bit).

In May 1989, the iPSC/2 was upgraded through the provision of:

1) a concurrent I/O system, permitting fast parallel random access to 1.5 Gbytes of data from any of the hypercube nodes, and 2) an additional 32 SX nodes.

The resulting 64-node system provides the community (at Daresbury) with a machine that exhibits supercomputer performance over a broad range of application codes.

(Note: On March 6, 1989, SERC will hold a Workshop on the Intel Hypercube, "Applications Details and Results." We will update the SERC story in the next issue of iSCurrents, with a report from that workshop. For more information on the SERC workshop, contact R.J. Allan at SERC (phone 925 603 242. He can be reached via his email address RJA@DL.DLGM).

SERC TO HOLD WORKSHOP ON iPSC/2 APPLICATIONS AND RESULTS

Workshop in Daresbury, U.K. on March 6

(December 18, 1989) The Science and Engineering Research Center (SERC) at Daresbury, U.K., announced it will hold a workshop on March 6, titled "Intel Hypercube — Applications and Algorithms: Details and Results."

The agenda at present is as follows:

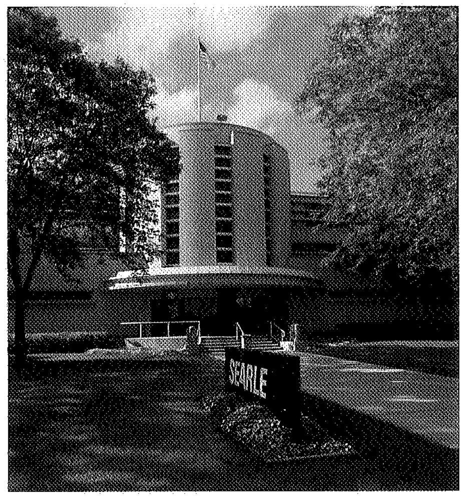
10:30	Coffee
10:45	Opening remarks, Martyn Guest, Computational Chemist
11:00	Overview of the iPSC/2, W.H. Purvis
11:30	Quantum Chemistry, Martyn Guest
12:00	Molecular Dynamics, W. Smith
12:30	Numerical Modeling, R.J. Blake
13:00	Lunch
14:00	Atomic Physics, R.J. Allan
14:30	Electronic Structures of Solics, W.M. Temmerman
15:00	Coffee and Open Discussion
16:00	Closing, Martyn Guest

For more information, contact: R.J. Allan (44) 925 603 242, or reach him by email at RJA@DL.DLGM

(The SERC Workshop is not an Intel-sponsored event.)

Intel Scientific Computers' Distinguished User Focus:

Searle's Drug Design Group



From hypertension to drugs for NSAID-induced ulcers, Searle is a name synonymous with the pharmaceutical industry. Now a wholly owned subsidiary of the Monsanto Company, Searle deploys a six-person drug design group located in the Chicago area. The group models drug-receptor interactions critical to the design of new pharmaceutical products. A 16-node, iPSC/2 parallel supercomputer plays a key role in their success.

"Typically, when Searle undertakes a project to discover novel drugs in certain areas, a discovery project team is formed. It consists of modelers who build a pharmacophore model and who design new chemical structures based on that model, chemists who synthesize these chemical structures, and biologists who develop and perform assays of the potential effects of these proposed drugs," explained Dr. Dale Spangler, of Searle's drug design group. "To develop and refine this model, conformational analysis is crucial. This involves the location of all of the conformations (shapes) for a particular structure that are biologically accessible. The search can require ten thousand energy minimizations, calculations well suited for the Intel parallel environment."

Drug action is usually initiated by the binding of a small molecule to a special protein known as a receptor. The process is similar to finding a key that fits a lock, an analogy suggested by Emil Fischer almost a century ago. When the three-dimensional structure of the receptor is known, the modeling of the complex formed by the drug, receptor, and solvent is known as "receptor fitting."

Unfortunately, in most cases the structures of the receptors are unknown. For such projects, the modeler resorts to an alternative technique known as "pharmacophore modeling" or "receptor mapping." This involves taking structures of compounds with known drug activity, determining their allowed conformations, and matching many combinations of "good" conformations from each of these structures to generate a model consistent with the biological data. As new compounds are tested and their activity becomes known, calculations are performed on these new structures to refine and improve the model.

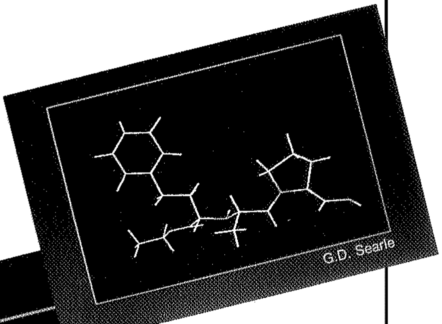
MacroModel & BATCHMIN on the iPSC/2

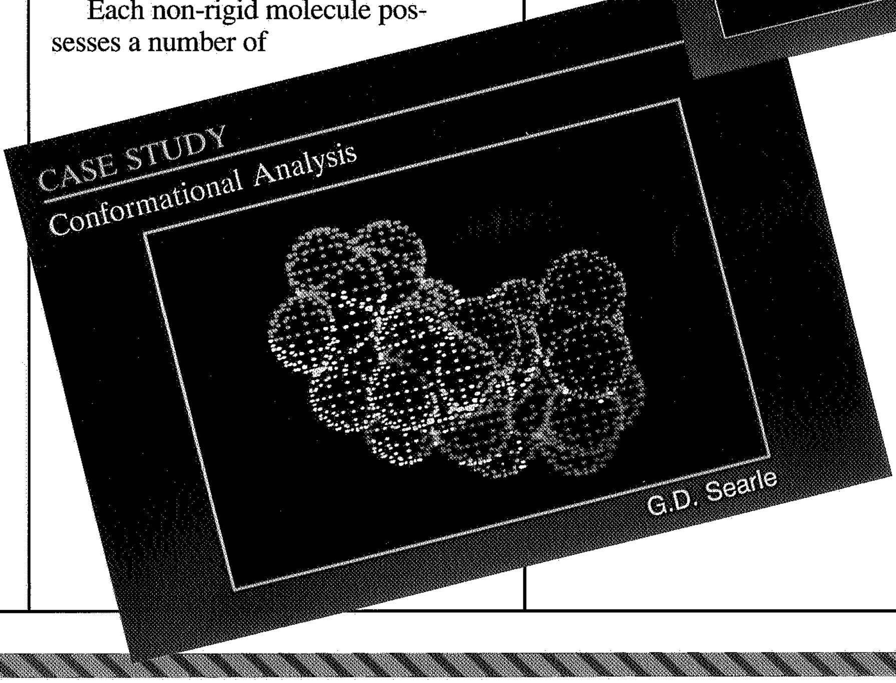
Integrated into a network of VAXes (see figure 1), the Searle iPSC/2 handles the computationally intensive BATCHMIN code (the batch minimizer of MacroModel) used for conformational analysis.

Each non-rigid molecule possesses a number of rotatable chemical bonds (bonds that can be rotated for a minimal energy penalty). By taking 15-30 degree increments of the torsional angle for each of these rotatable bonds, a set of conformations is generated. MacroModel and Sybyl (a general-purpose modeling package from Tripos Associates) can be used to generate the initial starting geometries for this set after removing unreasonable conformations.

The conformations are subsequently passed twice to BATCHMIN. In the first pass, full geometry optimization to locate the energy minima is performed. The MM2 based force-field energy of each conformation is minimized and the global minimum is determined.

In the second pass, constrained optimizations are performed. While holding the torsion angles for the rotatable bonds fixed, all other geometrical parameters are optimized to minimize the MM2-based force field energy. Conformations whose





energy falls within the 5-7 Kcal window above the global minimum are retained. The energy cutoff is conservatively set somewhat higher than the consensus value for a biologically accessible conformation.

At this point the researcher possesses a collection of energy minimized structures defining low energy regions of conformation space for the molecule of interest.

The procedure is performed for each compound used to create or refine the model as well as all proposed structures for synthesis. In many cases these structures possess 5 or more rotatable bonds resulting in tens of thousands of final conformations. The computationally intensive task of optimizing these structures at Searle is executed with BATCHMIN (MacroModel) transported to a 16-node iPSC/2 interfaced seamlessly to a VAX cluster (see figure 1).

"We have carried out calculations that take more than a week on the iPSC/2," Spangler said. "These would require more than 6 months on our shared VAX 8650s." Spangler pointed out that during the month of June 1989, the iPSC/2 logged the equivalent of 15,000 VAX 780 hours. Spangler also indicated that this was only half of the iPSC/2 capabilities.

Goals: Production Environment

Searle's interest in microprocessor-based, high-performance computing was originally spurred by Dr. James Snyder, the head of drug design, and Dr. Errol Sandler, Searle's Director of Scientific MIS. However, it was not until the arrival of the iPSC/2 that Synder and other members of Searle's drug design group saw a commercially available parallel processor that would provide the necessary power for their applications at a reasonable cost.

In April of 1988, Spangler and Dan Volocyk of the Searle's Scientific MIS group sat down with Intel's Tony Anderson and Elliott Swan and strategized what it would take to get the iPSC/2 into a production environment.

The project had three goals: 1) create a seamless interface with the

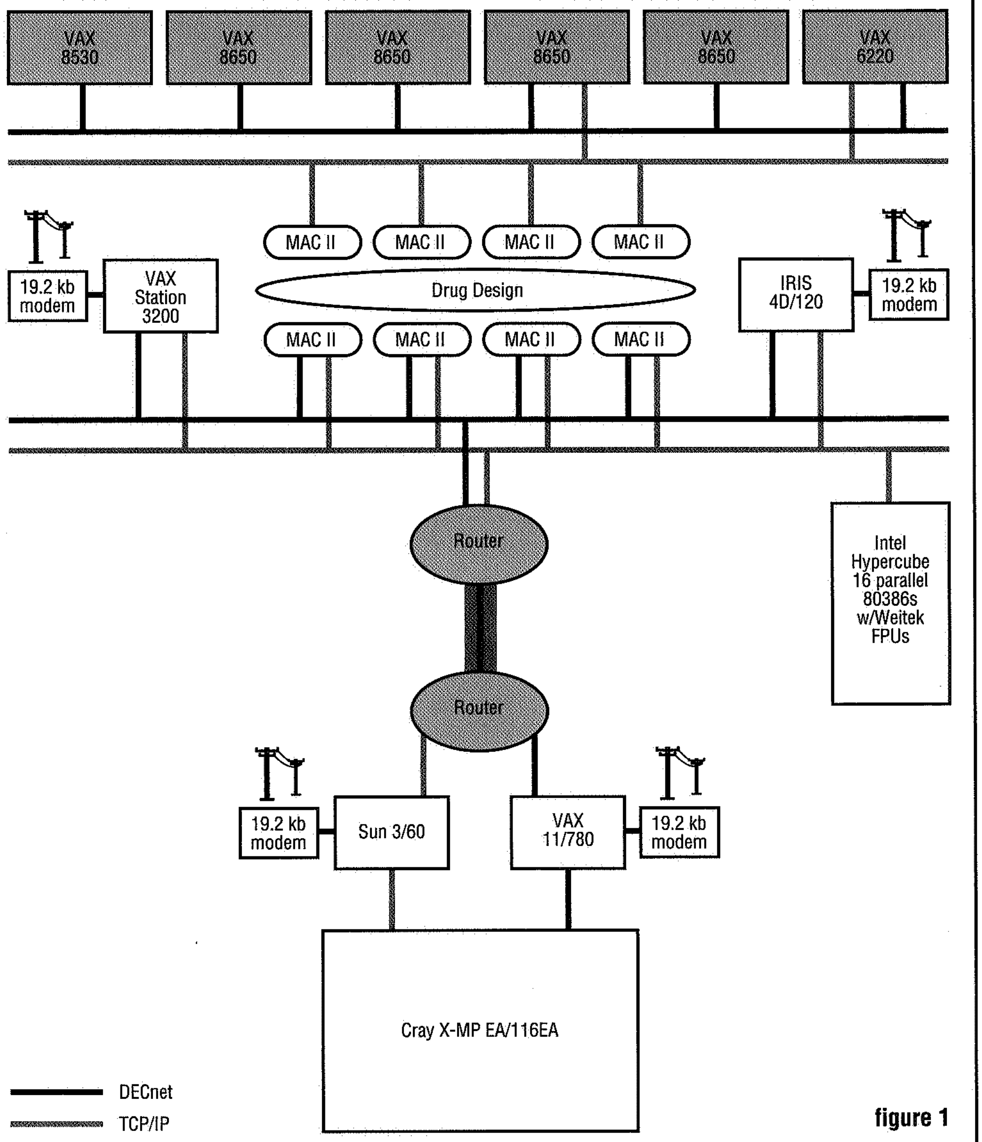


figure 1

VAXes, 2) allow the iPSC/2 to be shared dynamically, and 3) determine a series of programs easy to port.

"Our VAX users weren't too thrilled about having to work in a Unix environment," Volocyk said. "So we wanted to give them transparency of use."

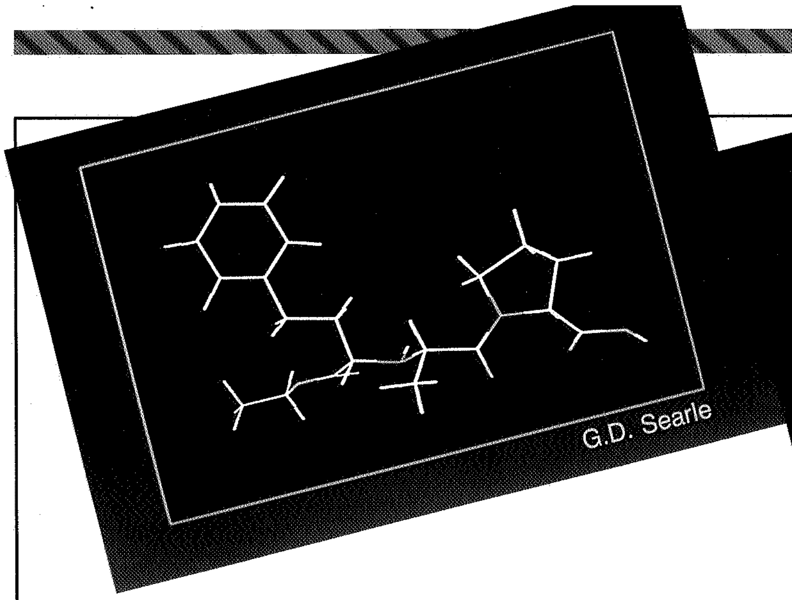
The dynamic sharing of the cube would enable different needs to be satisfied simultaneously.

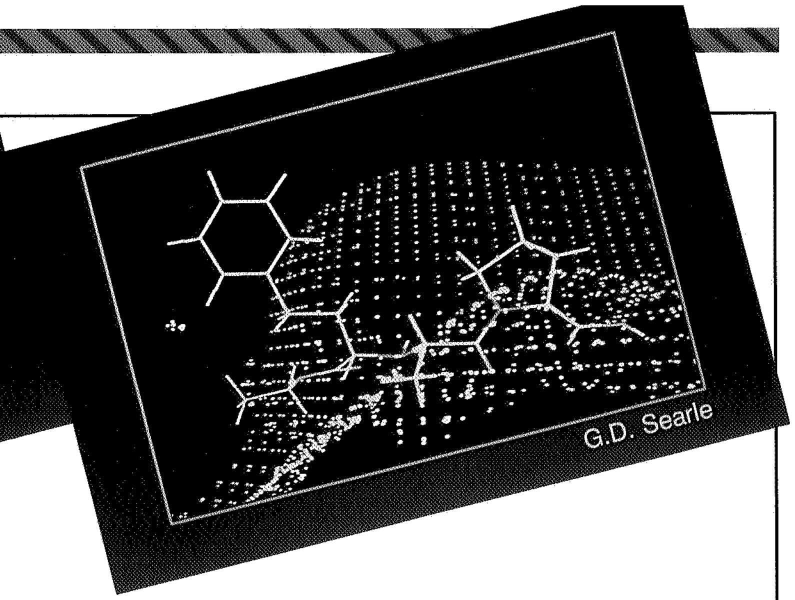
"To allow multiple simultaneous use of the cube, we needed different requirements to be satisfied simultaneously. We needed to allow jobs requiring two weeks to peacefully co-exist with jobs of a half-hour or even interactive jobs," Spangler said. "We needed a system that could give all the nodes to a high-priority job upon demand within a few minutes and split the nodes in an equitable fashion between multiple competing jobs."

BATCHMIN was chosen as the first program to be ported to the Intel parallel environment.

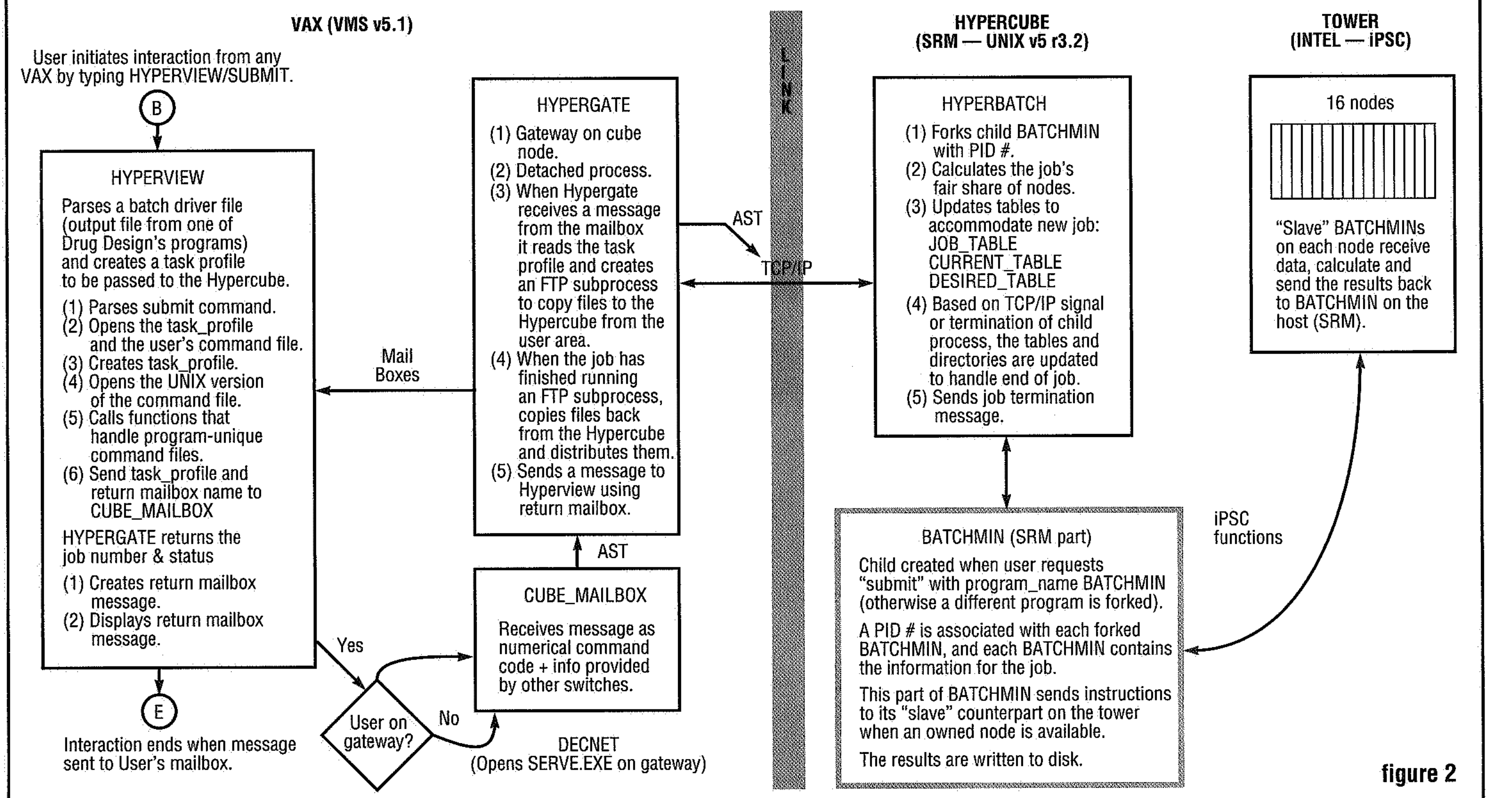
"We needed to port a program whose use would have an immediate impact on our modeling projects," Spangler said.

The development work began in June, and in July the iPSC/2 arrived. In August, the BATCHMIN port began and took one man-month to complete. BATCHMIN was divided into 2 parts: a front-end supervisor running on the SRM to control the collection of data and keep the nodes busy, and a back-end minimizer running on each node to optimize a single conformation.





VAX/HYPERCUBE INTERFACE *SUBMIT*



VAX (VMS v5.1)
User initiates interaction from any VAX by typing HYPERVIEW/SUBMIT.
B
HYPERVIEW
Parses a batch driver file (output file from one of Drug Design's programs) and creates a task profile to be passed to the Hypercube.
(1) Parses submit command.
(2) Opens the task_profile and the user's command file.
(3) Creates task_profile.
(4) Opens the UNIX version of the command file.
(5) Calls functions that handle program-unique command files.
(6) Send task_profile and return mailbox name to CUBE_MAILBOX
HYPERGATE returns the job number & status
(1) Creates return mailbox message.
(2) Displays return mailbox message.
E
Interaction ends when message sent to User's mailbox.
Mail Boxes
Yes
User on gateway?
No
HYPERGATE
(1) Gateway on cube node.
(2) Detached process.
(3) When Hypergate receives a message from the mailbox it reads the task profile and creates an FTP subprocess to copy files to the Hypercube from the user area.
(4) When the job has finished running an FTP subprocess, copies files back from the Hypercube and distributes them.
(5) Sends a message to Hyperview using return mailbox.
AST
CUBE_MAILBOX
Receives message as numerical command code + info provided by other switches.
DECNET (Opens SERVE.EXE on gateway)
AST
TCP/IP
LINK
HYPERCUBE (SRM — UNIX v5 r3.2)
HYPERBATCH
(1) Forks child BATCHMIN with PID #.
(2) Calculates the job's fair share of nodes.
(3) Updates tables to accommodate new job: JOB_TABLE CURRENT_TABLE DESIRED_TABLE
(4) Based on TCP/IP signal or termination of child process, the tables and directories are updated to handle end of job.
(5) Sends job termination message.
BATCHMIN (SRM part)
Child created when user requests "submit" with program_name BATCHMIN (otherwise a different program is forked).
A PID # is associated with each forked BATCHMIN, and each BATCHMIN contains the information for the job.
This part of BATCHMIN sends instructions to its "slave" counterpart on the tower when an owned node is available.
The results are written to disk.
iPSC functions
TOWER (INTEL — iPSC)
16 nodes
"Slave" BATCHMINs on each node receive data, calculate and send the results back to BATCHMIN on the host (SRM).

figure 2

The interface between the users, the VAXes, and the Hypercube was completed after three man-months of work, and in February of 1989, the full system became available to Searle's drug design group.

The net result? The structure of the command files for running on the iPSC/2 are identical to those running on the VAX cluster (see figures 2 and 3). The batch queuing is also similar to batch queuing on the VAX. By changing a single letter in their VMS command, users can send their work either to the VAX or the iPSC/2.

"We are getting eight times the performance of a VAX 8650," Volocyk said. "And, just as important, we're doing work that otherwise couldn't be done on the systems available locally."

Spangler explained that the iPSC/2 has an important place in the heterogeneous computer environment at Searle. This includes a Cray X-MP at Monsanto in St. Louis, which Searle's drug design group gets roughly a 20% share. However, for tasks that parallelize well but do not easily vectorize, the iPSC/2 is the most cost-effective platform.

Price Performance Crucial

"For computational chemistry, we think Intel's going at things the right way with a scalable number of powerful processors," Spangler said. "We weren't really looking at a particular type of parallelization or memory scheme, but the way we get the greatest amount of computing for a fixed

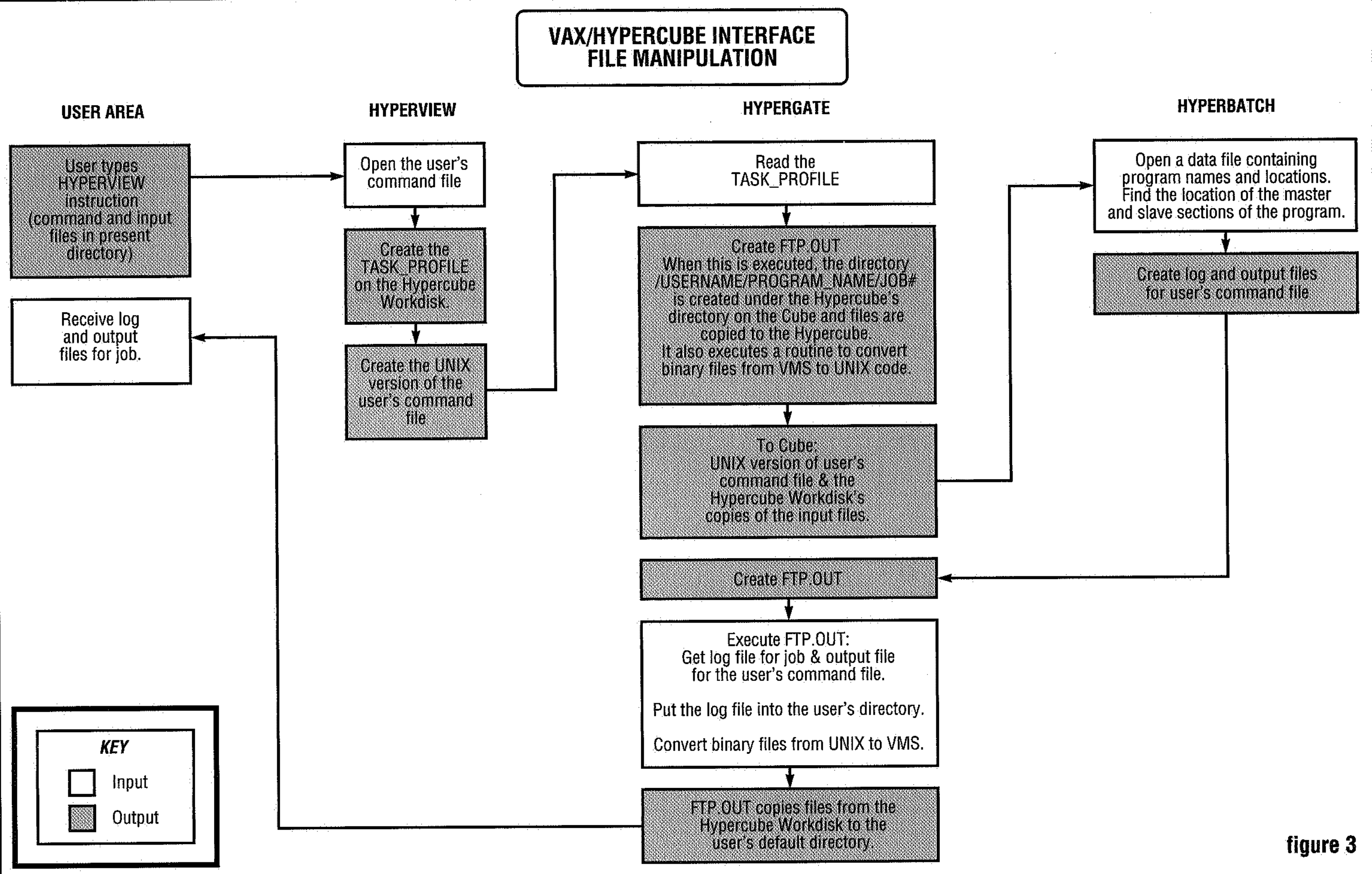

VAX/HYPERCUBE INTERFACE FILE MANIPULATION
USER AREA
HYPERVIEW
HYPERGATE
HYPERBATCH
User types HYPERVIEW instruction (command and input files in present directory)
Receive log and output files for job.
Open the user's command file
Create the TASK_PROFILE on the Hypercube Workdisk.
Create the UNIX version of the user's command file
Read the TASK_PROFILE
Create FTP.OUT When this is executed, the directory /USERNAME/PROGRAM_NAME/JOB# is created under the Hypercube's directory on the Cube and files are copied to the Hypercube. It also executes a routine to convert binary files from VMS to UNIX code.
To Cube: UNIX version of user's command file & the Hypercube Workdisk's copies of the input files.
Create FTP.OUT
Execute FTP.OUT: Get log file for job & output file for the user's command file. Put the log file into the user's directory. Convert binary files from UNIX to VMS.
FTP.OUT copies files from the Hypercube Workdisk to the user's default directory.
Open a data file containing program names and locations. Find the location of the master and slave sections of the program.
Create log and output files for user's command file
KEY
Input
Output

figure 3

number of dollars — happens to be parallel processing. Intel as a company is going to push microprocessor technology primarily from the PC end, and they've already demonstrated that they'll do it in a way to compete with large machines."

Future Plans: Faster Nodes, More Software

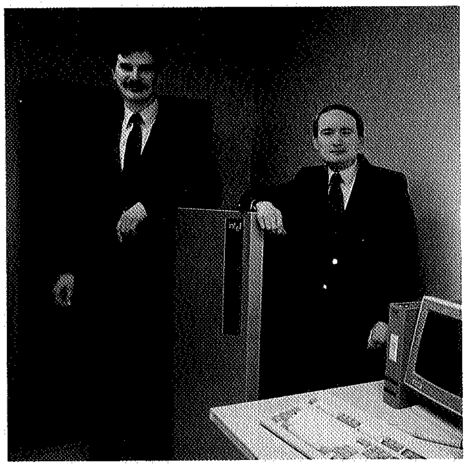
What next?

In the short term, Spangler plans to port the program RMSFIT, which matches the many conformation sets used in receptor mapping, to the iPSC/2. This will involve dividing the program into an SRM supervisor, and a program running on each node that performs many matches.

Over the long haul, Searle hopes to port or locate a version of Amber from Peter Kollman's group at UCSF for the iPSC/2.

"Amber allows us to build prototype receptors to interact with drug molecules as well as with solvents," Spangler said. "We have utilized roughly a thousand hours of Cray time doing prototype calculations. For those calculations that don't need the Cray, we need the most cost-effective computational solution. We're looking at future developments from Intel to help us there."

Note: BATCHMIN for the iPSC/2 will become available to dues paying members of the MacroModel Consortium through Clark Still at Columbia University. Interest in the VAX interface should be directed to G.D. Searle.



intel®

Intel Scientific Computers
15201 N.W. Greenbrier Pkwy.
Beaverton, OR 97006
503-629-7629

Users Group Bits and Bytes: Europe & U.S.

European Users Group

(Rennes, France) On October 3, 1989, approximately 50 iPSC users from 20 sites gathered for the Second European iPSC Users Group Meeting in Rennes, France. Participants came from as far north as Umea, Sweden (near the Arctic Circle) and as far south as Greece.

There were over 25 presentations, most providing a brief overview of activities at each iPSC site. Major emphasis was placed on software and tools, with presentations on STRAND 88, CHORUS, the GMD-Argonne macros, and work on languages and environments at IRISA. The Technical University of Munich described their work on programming tools, including a performance analyzer and a visualizer.

Applications discussed included 3-D Fluid Flow (ONERA), ray tracing (IRISA), and computational chemistry (SERC).

Speakers from the University of Dublin and the University of Umea discussed teaching, using the iPSC/1 and iPSC/2.

The 1990 European Users Group is tentatively scheduled for the week of the CONPAR 90 VAPP IV conference in Zurich. Anyone with alternative suggestions should contact Richard Chamberlain at Intel's Swindon Office (44-793-696 578).

—Richard Chamberlain, iSC European Users Group Coordinator

iSC U.S. USERS' GROUP MEETING: St. Louis, Missouri

(St. Louis, MO) The meeting began on Thursday, October 5, at the Radisson Hotel Clayton in St. Louis, Missouri, with a warm welcome to attendees by JoAnne Wold, newly appointed as iSC User Group Administrator.

The agenda was packed with nine technical presentations on a variety of subjects, ranging in scope from seismic applications to advanced database and information systems programs to graduate level projects in machine vision/image processing. The presentations covered useful ground for all attendees, spanning the gamut of theoretical, research-oriented, and practical applications of parallel processing technology on iPSC machines.

Intel made several announcements. Changes and improvements to the user group and a new University Resource Referral Program called "ACCESS" were described. (More on ACCESS in the next issue of Currents.)

Product announcements were made pertinent to Connectivity, Release 3.1, and More I/O for the iPSC/2.

David Billstrom, Product Marketing Manager for Intel Scientific Computers, gave a description of the current 3.0 software release together with upgrade issues involved in a move to the next release. The features and benefits of the new software release 3.1 were discussed in detail, including the delivery of Ada, VX performance, the repair of Decon, and upgrade issues. Other new product features also discussed included backup tape capability, 9-track tape, VMS link and VME interface.

At dinner Thursday night, Wendy Vittori, iSC Director of Marketing and Strategic Planning brought users up to date on iSC's future directions.

On Friday, October 6, from 9:30 AM to 1:00 PM a University and Industry Forum was held to discuss "Universities as a Resource for Industry in the New World of Parallel Processing." The objective of the forum was to exchange information and ideas on the following issues:

- What does industry need?
- How are universities preparing students to meet the needs of industry in parallel computing?
- Opportunities for industry/ university cooperation in course development and research.
- Evaluation of existing programs (with examples) and proposals for the future.

A panel of representatives from universities and industry facilitated the session. The forum moderator was Dr. Marilyn Livingston, Professor in the Computer Science Department of the University of Illinois. As the questions were posed by Dr. Livingston, the panelists offered their views on the issues to be considered. This was followed by a moderated discussion in which attendees participated. Panelists included:

David Billstrom, Product Marketing Manager, Intel Scientific Computers (Vendor Perspective)

Dr. Akin Ecer, President, Technalysis Incorporated (Industry Perspective)

Dr. Gary B. Lamont, Professor, Department of Electrical and Computer Engineering, Air Force Institute of Technology (University & Government Perspective)

Dr. Charles Mosher, Senior Principle Research Geophysicist, ARCO Oil & Gas Company (Industry Perspective)

Dr. Keith Teague, Associate Professor, Electrical and Computer Engineering, Oklahoma State University (University Perspective)

An edited transcription session was published in the meeting proceedings document for the purpose of sharing the factual information developed by the forum. If you have not received a copy of the proceedings, and would like to receive one, or would like to receive more information about the user group, please contact JoAnne Wold, iSC User Group Administrator, at 503-629-7737.

(To order a copy of the proceedings from iSC's October User Group Meeting, write to JoAnne Wold, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR, 97006, or use the enclosed reply card.

User Requests

Professor Dan McCarthy of the University of Dublin is interested in speaking with iPSC/1 users who are using their systems for teaching. Dr. McCarthy can be reached at the following address:

Dr. Dan McCarthy
Department of Computer Science
School of Engineering
Trinity College
Dublin 2
Eire
Telephone: (35) 31 772941, ext 1783

iPSC/2 Users developing applications in LISP are encouraged to exchange information with Josep Pujol at the University of Blanes in Spain. The address and phone number are:

Josep Pujol Gruart
Centre D'Estudia Avancats de Blanes
Cami De Santa Barbara
17300 Blanes Girona
Spain
Telephone: (34) 72 33 6101/6102/6103

Intel's ACCESS Program

To meet the growing demand by customers for individuals and organizations with iPSC programming experience in a variety of applications, Intel's ACCESS Program was announced at both iSC's European and U.S. User Group Meetings.

The ACCESS Program is designed to match customers who need programming expertise, iPSC consulting, applications counsel, etc. with universities and other organizations that have the personnel and expertise to provide solutions.

As such, the ACCESS Program is designed to serve as a database, allowing iSC to refer customers to universities and organizations listed in the ACCESS files. iSC will not and cannot recommend one university or organization over another; it merely serves as a "bulletin board."

The ACCESS Program is strictly voluntary and will initially be administered through the iSC Users Group. More detailed information on the program will appear in upcoming issues of Currents. Until that time, anyone wishing a detailed description of the ACCESS Program is invited to contact JoAnne Wold, iSC Users Group Coordinator, Intel Scientific Computers, 15201 NW Greenbrier Parkway, Beaverton, OR, 97006. The phone number is (503) 629-7737.

People, Places & Things

On October 9, 1989, at the Society of Petroleum Engineers Annual Conference in San Antonio, Texas, ***Dr. John Wheeler*** of Exxon Production Research, gave a paper on his experiences with an iPSC/2.

At the October 1989 Society of Exploration Geophysicists' Annual Conference in Dallas, Texas, ***Dr. Charles C. Mosher*** of ARCO Oil and Gas Company and ***Dr. David Scott*** of Intel Scientific Computers presented a paper titled "A Parallel Implementation of 3D F-K Migration for Distributed Memory Computers."

At the October 1989 Northcon Conference, held in Portland, Oregon, ***Dr. Sigurd L. Lillevik***, the Engineering Program Coordinator for Intel Scientific Computers, presented a paper titled "Systems Design and Engineering for Large-Scale Parallel Computers."

On November 13, 1989, ***Dr. Mike Barton*** and ***Gary Withers*** presented a paper titled "Computing Performance as A Function of the Speed, Quantity, and Cost of the Processors," at the IEEE-sponsored Supercomputing '89 in Reno, Nevada. Barton is a member of iSC's Applications Science Group. Withers is iSC's Training Manager.

Literature

As a result of the inspiration and perspiration of ***Dr. Horst Simon***, NASA-Ames Research Center, two publications bearing the same title are available: *Are Highly Parallel Systems Ready for Prime Time? A Transcript of a Panel Discussion*, edited by Horst D. Simon, is available from NASA as Report RNR-89-010, October 1989. An edited, shortened version will appear in 1990 in the International Journal of Supercomputer Applications, and will be available through the MIT Press.

The abstract for the International Journal of Supercomputer Applications reads: "This is the edited and abbreviated transcript of a panel discussion held on May 9, 1989 in Los Angeles, California, during the conference, **"Parallel CFD - Implementations and Results Using MIMD Computers."** The purpose of the conference was to discuss recent developments in the use of MIMD parallel computers in high performance computational fluid dynamics. The intent of the panel discussion was to summarize the findings of the meeting, and to give a perspective on the state of the art in using parallel computers for solving large scale engineering and scientific application problems." Panelists included Creon Levit, Numerical Aerodynamic Simulation (NAS) Systems Division, NASA-Ames; Kent Misegades, Cray Research; Gary Montry, Myrias Computers; Ken Neves, Boeing Computer Services; Anthony Patera, MIT; and Justin Rattner, Intel Scientific Computers.

For copies or transcripts of the papers or presentations cited above, please write to Ken Harper, iSC Currents, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006, or use the enclosed reply card. The phone number is (503) 629-7631.

A number of iSC's European Users contributed papers at the First European Workshop on Hypercube and Distributed Computers, held at Rennes, France, 4-6 October, 1989. Those papers and others have been edited by F. Andre and J.P Verjus and published by North-Holland under the title "Hypercube and Distributed Computers."

To obtain copies of "Hypercube and Distributed Computers," please write the following address in Europe:

North-Holland (An imprint of Elsevier Science Publishers B.V.)
Attn: Petra van der Meer
P.O. Box 103
1000 AC Amsterdam,
The Netherlands

In the United States, to obtain copies of "Hypercube and Distributed Computers," please write the following address:

Elsevier Science Publishing Co., Inc.
P.O. Box 882
Madison Square Station
New York, N.Y. 10159

Software Tools

(Continued from page 2)

ing, are considered when tracing a particular symbol. In the query facility, a template-driven filtering capability allows the programmer to specify the characteristics of the variables to be examined. The programmer can easily focus on categories of variable use, such as subscripts, reduction functions, actual or dummy arguments, equivalences, etc., and can combine templates with Venn diagrams.

OTHER Features

Code conversion reformattor supports resequencing of statement labels, code indentation, reordering of declarations, extraction of COMMON blocks into INCLUDE files, conversion of conditioned code blocks into IF ( ) THEN, ELSE, ENDIF structures and typical "tidy" features.

Code instrumentation facility supports the gathering of runtime statistics on CPU usage, DO loop usage, values of critical variables, and input variables. Code size may be from 3 to 5 times larger than the original FORTRAN code. FORGE can support programs of any size, given the availability of adequate disk storage.

Package manipulation tools provide capabilities for selecting, creating, and deleting packages and viewing source files; for analyzing and modifying packages; for defining and editing files in packages; and for maintaining several versions of each routine. Database viewing tools enable the programmer to browse the database and perform such functions as: trace variables, display constants, query database, list all routines, show COMMON block grid, and perform consistency checks on both COMMON blocks and argument passing.

OTHER Features

Code conversion reformattor: supports resequencing of statement labels, code indentation, reordering of declarations, extraction of COMMON blocks into INCLUDE files, conversion of conditioned code blocks into IF () THEN, ELSE, ENDIF structures and typical "tidy" features.

Code instrumentation facility: supports the gathering of runtime statistics on CPU usage, DO loop usage, values of critical variables, and input variables.

The COMMON block grid illustrates how COMMON variables are used throughout the program to restructure COMMON storage. Equivalencing and COMMON block differences, as well as name changing through argument passing, are considered when tracing a particular symbol. In the query facility, a template-driven filtering capability allows the programmer to specify the characteristics of the variables to be examined. The programmer can easily focus on categories of variable use, such as subscripts, reduction functions, actual or dummy arguments, equivalences, etc., and can combine templates with Venn diagrams.

APPLICATION PORTABILITY FOR THE '90'S

In the 1990's, parallel supercomputers will incorporate from a few to several thousand processors. To control development and maintenance costs, application developers will have to be able to create a single application algorithm that can run efficiently across an entire range of parallel system configurations.

A new CASE tool, MIMDizer (mim-dee-izer) will be made available on Intel's iPSC® systems in 1990. MIMDizer is defined to be an interactive software system that will further simplify the task of converting existing sequential FORTRAN programs into efficient parallel code and will improve programmer efficiency in the design and implementation of new parallel algorithms. MIMDizer gets its name from the architectures of Multiple Instruction, Multiple Data Stream systems such as Intel's iPSC family of systems.

MIMDizer will go beyond automatic parallelizing compilers by offering software developers a high-level approach, as opposed to the low-level approach taken by current parallelizing compiler technology. Fully automated solutions are not technically feasible today for coarse-grain restructuring. MIMDizer will optimize programmer participation in the restructuring process, permitting the programmer to decide upon the preferred parallel decomposition strategy while MIMDizer handles most of the implementation detail.

(Continued on page 15)

Customer Training Schedule

Intel Scientific Computers has the following dates tentatively set for customer training in 1990. To confirm these dates and/or enroll, call Gary Withers at (503) 629-7600.

Jan 29	System Administrator Workshop
Jan 30 - Feb 2	Introduction to Parallel and Vector Programming
Mar 26	System Administrator Workshop
Mar 26-30	Introduction to Parallel and Vector Programming
May 21	System Administrator Workshop
May 22-25	Introduction to Parallel and Vector Programming

This Spring, Intel Scientific Computers will also be offering educational seminars for FORTRAN and C programmers to introduce them to programming an iPSC system.

The dates and locales will be determined largely by interested parties. If you are a FORTRAN and/or C programmer interested in learning how to program an iPSC, please use the enclosed reply card or contact one of the following:

David Moody
Intel Scientific Computers
Intel International Ltd
Pipers Way
Swindon
SN3 1RJ
England
(44) - 793-696 578

Kenneth Harper
Intel Scientific Computers
15201 NW Greenbrier Parkway
Beaverton, Oregon 97006
(503) 629-7631

Gary DeLeon
Intel Japan (IJKK)
5-6 Tokadai
Tsukuba City
Ibaraki-Ken 300-26

PARALLEL PROGRAMMING PRIMER AVAILABLE ALONG WITH PARALLEL PROCESSING: A SELF-STUDY INTRODUCTION

Users and interested parties are welcome to receive Intel's Parallel Programming Primer. As the title suggests, the book (roughly 100 pages in length) is intended "as an introduction to developing applications for distributed-memory parallel computers. The model used in the book is based on the Intel iPSC/2 Parallel Supercomputer..."

The Programming Primer contains eight chapters, an appendix with example application codes developed for the iPSC/2, and a Glossary.

The chapter titles are as follows:

Chapter 1:	The Parallel System: Divide and Conquer
Chapter 2:	A Programming Model for the Loosely Coupled Parallel System
Chapter 3:	Working Smart, Not Hard, Parallel Software Engineering Principles
Chapter 4:	Decomposition Strategies
Chapter 5:	Domain Decomposition: Examples and Techniques
Chapter 6:	Control Decomposition: Examples and Techniques
Chapter 7:	Object-Oriented Techniques for Distributed Memory Systems
Chapter 8:	Parallel Programming Tools: Developing a Better Mousetrap

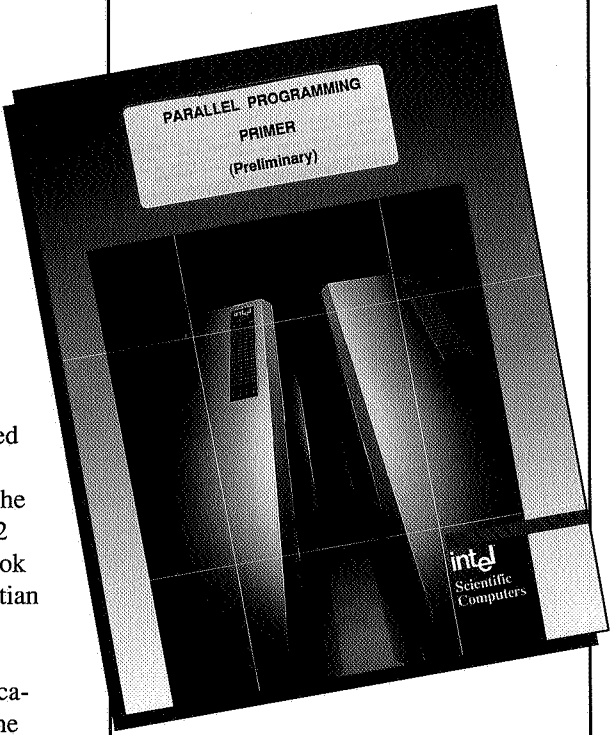
Also available from Intel is Parallel Processing: A Self-Study Introduction: A First Course in Programming the iPSC/2 Hypercube. The Self-Study introduction was developed by Ron Pickering from the Polytechnic Wolverhampton in the U.K., and adapted for the iPSC/2 by Clifford Addison, Jeremy Cook and David Warhurst at the Christian Michelsen Institute in Bergen, Norway.

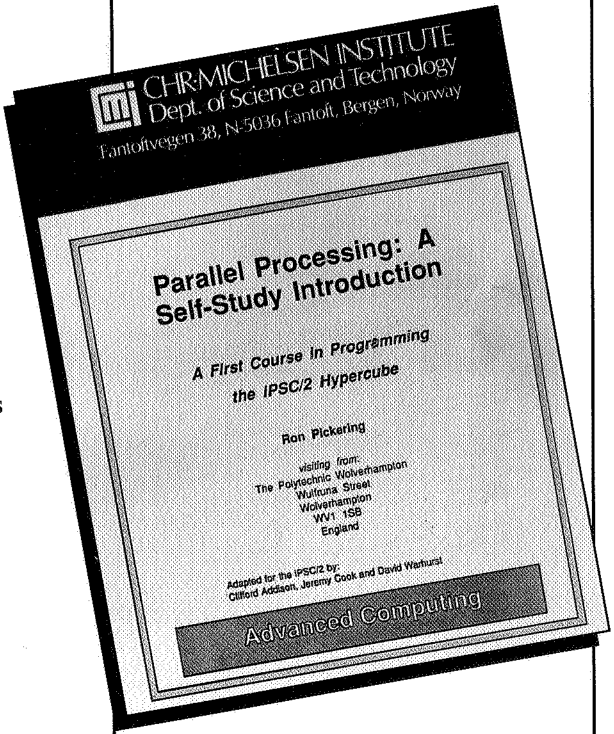
Copies of both of these publications can be obtained by using the enclosed reply card, or contacting either:

David Moody
Intel International Ltd
Intel Scientific Computers
Pipers Way
Swindon
SN3 1RJ
England
(44)- 793-696 578

Kenneth Harper
Intel Scientific Computers
15201 N.W. Greenbrier Parkway
Beaverton, OR 97006
(503) 629-7631

Gary DeLeon
Intel Japan (IJKK)
5-6 Tokadai
Tsukuba City
Ibaraki-Ken 300-26




CHR·MICHELSEN INSTITUTE
Dept. of Science and Technology
Fantoftvegen 38, N-5036 Fantoft, Bergen, Norway

Parallel Processing: A Self-Study Introduction

A First Course in Programming the IPSC/2 Hypercube

Ron Pickering

visiting from:
The Polytechnic Wolverhampton
Wulfruna Street
Wolverhampton
WV1 1SB
England

Adapted for the IPSC/2 by:
Clifford Addison, Jeremy Cook and David Warhurst

Advanced Computing

NASA, INTEL RIACS, & IBM TO HOST PARALLEL CFD '90 in Indianapolis, May 6-8

Indianapolis (November 15, 1989) NASA, Intel, IBM, RIACS, and Indiana University/Purdue University at Indianapolis (UIPUI) will host the Parallel Computational Fluid Dynamics 1990 Conference. Parallel CFD will be held in Indianapolis on the campus of UIPUI on May 6 to May 8.

The organizing committee for Parallel CFD includes: Enrico Clementi, IBM; Justin Rattner, Intel; Richard Blech, NASA-Lewis; Horst Simon, NASA-Ames; Spiro Lekoudis, Office of Naval Research; Richard Sincovec; Research Institute for Advanced Computer Science; Antony Jameson, Princeton University; Anthony Patera, MIT; and Akin Ecer, UIPUI.

F. Ron Bailey, NASA-Ames will be the keynote speaker. Invited speakers include David Caughey, Cornell and NASA-Ames (Block Implicit Multi-grid Solution of the Euler Equations on Parallel Computer, Avi Lin, Temple (Degenerate Parallel Algorithms for Parallel CFD); Wolfgang Schmidt and Bernhard Wagner, Dornier Luftfahrt (Impact on Aircraft Design of Vector and Parallel CFD Applications); Pierre Leca, ONERA (Experiments in Parallel CFD Using Distributed Memory Multi-processors); J. Michael Summa, Analytical Methods (Recent CFD Applications in the Automobile Industry); and William Van Dalsem, and Horst Simon, NASA-Ames (Parallel CFD at NASA-Ames).

There will also be a panel discussion, "Parallel Application Software — Issues and Answers," led by Dan Anderson of Ford Motor Company. Panelists include Dennis Gannon, Indiana University; John Van Rosendale, ICASE Langley; David Gelernter, Yale University; Anthony Patera, MIT; Jacques Perlaux, Avions Marcel Dassault; and Ramesh Agarwal, McDonnell Douglas.

The Parallel CFD organizing committee is soliciting contributed presentations of 15 minutes on subjects that will complement the invited papers.

Because registration for the Parallel CFD conference is limited, advance registration ($200) by check or MasterCard/VISA will be accepted by Pat Fox, Indiana University/Purdue University at Indianapolis, 799 W. Michigan St., Indianapolis IN 46202 (317-274-0806).

A block of rooms has been reserved at the University Hotel and Conference Center. Contact the hotel at 1-800-627-2700 to make hotel reservations. Identify yourself as a Parallel CFD Conference participant to receive the discounted room rate.

Software Tools

(Continued from page 12)

Built upon FORGE, MIMDizer will continue to provide comprehensive analytical information about the existing sequential code, including interprocedural dependencies. Through graphical data flow representation and restructuring tools, the programmer will be able to more efficiently manipulate code blocks into improved parallel structures. Analytical information from the new structures will be provided to ensure application integrity and stability. The MIMDizer code generator will then produce parallel FORTRAN source code for execution on the desired target machine. Instrumented run time code will enable the programmer to gauge the effectiveness of the algorithms and allow iterative optimization of the code. MIMDizer will support rapid application benchmarking for system evaluation purposes, as well as development of highly efficient parallel applications. MIMDizer supports an extremely flexible programming model that will make it practical and cost-effective to develop portable parallel applications.

™FORGE and MIMDizer are trademarks of Pacific-Sierra Research, Inc.
®iPSC and Intel are registered trademarks of Intel Corporation.

LETTER FROM THE EDITOR

As we went to press on the night of December 18, 1989, we received word that Intel Scientific Computers would be unveiling its iPSC/860 at Super-computing Europe in London on January 10. Similarly, we learned that the iPSC/860 would make its American debut at Intel's booth at Uniforum, which is being held January 23 to 25 in Washington, D.C.

Because not all of the pertinent product information was available at the time we went to press, and because we had a commitment from our printer, and because we wanted iSCurrents to continue two quarters in succession as a quarterly publication, we decided to devote much of the Spring Issue to the iPSC/860.

Future Issues will also see articles on "computational finance" as well as reservoir modeling software.

There will also be a full report of the seminar being held March 6 at the Science and Engineering Research Center (SERC) Daresbury, England, on applications and results using the iPSC/2 (see People, Places, & Things).

In addition to the normal product updates, you will be seeing "how-to's" (and perhaps even "how-not-to's") written by Intel applications consultants and customers.

Our primary interest is to make iSCurrents a useful, informative document. Please let us know what your interest is and we will try to accommodate you.

Until next quarter...

Ken Harper

Ken Harper,
Editor

iSCurrents Staff

Executive Editor
Ken Harper
Associate Editor
Mike Cox
Users Group Editor
JoAnne Wold
European Editor
Richard Chamberlain
Asian Editor
Gary DeLeon

PRELIMINARY

INTELLEC® 432/100 EVALUATION AND EDUCATION SYSTEM

- **Evaluation and Education System for the Intel® 432 32-Bit Micromainframe**
- **OBJECT BUILDER Evaluation Software for Execution of 432 Symbolic Machine Instructions**
- **iAPX 432 General Data Processor**
- **Fully Assembled, MULTIBUS™-Compatible iSBC 432/100™ Board**
- **Object Programming Language**
- **Complete Set of Introductory Guides, Reference Manuals, and Educational Texts for the 432**

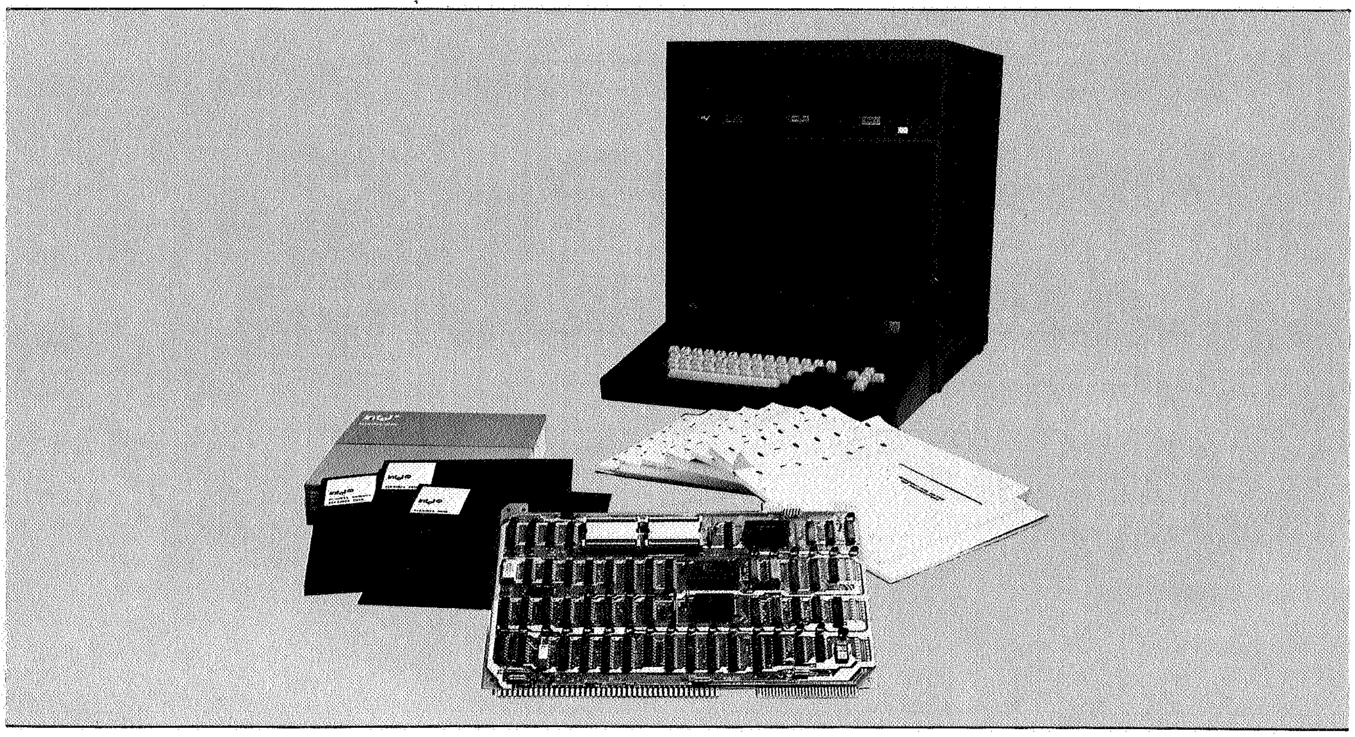
The Intellec 432/100 combines hardware, software, and documentation to provide a complete learning and evaluation environment for the Intel 432 32-bit Micromainframe. The system hardware consists of the iSBC 432/100 board. The fully assembled and tested MULTIBUS-compatible board installs in any Intellec 800, Series II or Series III Development System. The heart of the board is the powerful iAPX 432 32-bit General Data Processor (GDP). The iAPX 432 GDP combines mainframe functionality and performance with a microprocessor form factor and cost. Standard features of the 432 GDP include a high level language instruction set, the Silicon Operating System, and software transparent multiprocessing.

Two software packages are included with the Intellec 432/100 system. They are:

Object Builder — Allows evaluation of the iAPX 432 architecture and provides a means to work with the iAPX 432 at the symbolic machine instruction and system object level. It permits construction of actual iAPX 432 objects, interactive manipulation of them, and execution of iAPX 432 instructions. Through Object Builder, small 432 programs can be created and interpretively executed on the iAPX 432 GDP.

Object Programming Language (OPL) — An object-oriented high level language interpreter designed for teaching and educational program development. OPL facilitates learning of 432 object-oriented design methodology in a user-friendly, interactive, graphical environment.

A complete set of manuals, texts, and introductory guides accompanies the Intellec 432/100 system to assist the user in learning and evaluation.



FUNCTIONAL DESCRIPTION

Hardware

iSBC 432/100™ BOARD

The fully assembled iSBC 432/100 board features Intel's iAPX 432 32-bit GDP. The board is both MULTIBUS and RS232C compatible, and plugs into a MULTIBUS slot in any Intellec 800, Series II or Series III Development System. The memory for the iSBC 432/100 board is provided by and shared with the Intellec Development System (see Figure 1).

Processing Unit

The heart of the iSBC 432/100 board is Intel's powerful iAPX 432 Micromainframe. The iAPX 432 is an advanced 32-bit microprocessor which provides the functionality and performance of a large-scale computer. It is implemented in N-channel depletion load, silicon gate technology (HMOS) and packaged in two 64-pin Quad In-Line Packages (QUIP), integrating over 140,000 transistors. The two components function as a pipelined unit; one fetches and decodes instructions, the other executes them. The two GDP components are linked via a microinstruction data path.

The iAPX 432 features:

- Compiler-oriented Instruction Set
- The Silicon Operating System
- Software transparent multiprocessing for incremental performance capability
- Capability-based ("need to know") addressing and protection
- 2^{40}-byte virtual addressing space
- Symmetrical support of all 8, 16, and 32-bit integer data types, and proposed IEEE standard 32, 64, and 80-bit floating point

The Intel iAPX 432's integration of VLSI and systems technology provides a state-of-the-art means to increase both hardware and software reliability, provide a wide range of performance, and lower the cost of developing large-scale systems.

iSBC 432/100™ Board Logic and Bus Structure

The iSBC 432/100 board communicates and shares memory with the Intellec Development System via the MULTIBUS interface. On-board logic converts iAPX 432 addresses into MULTIBUS I/O and memory commands. iAPX 432 local addresses are transformed into MULTIBUS I/O commands; physical addresses into MULTIBUS memory commands.

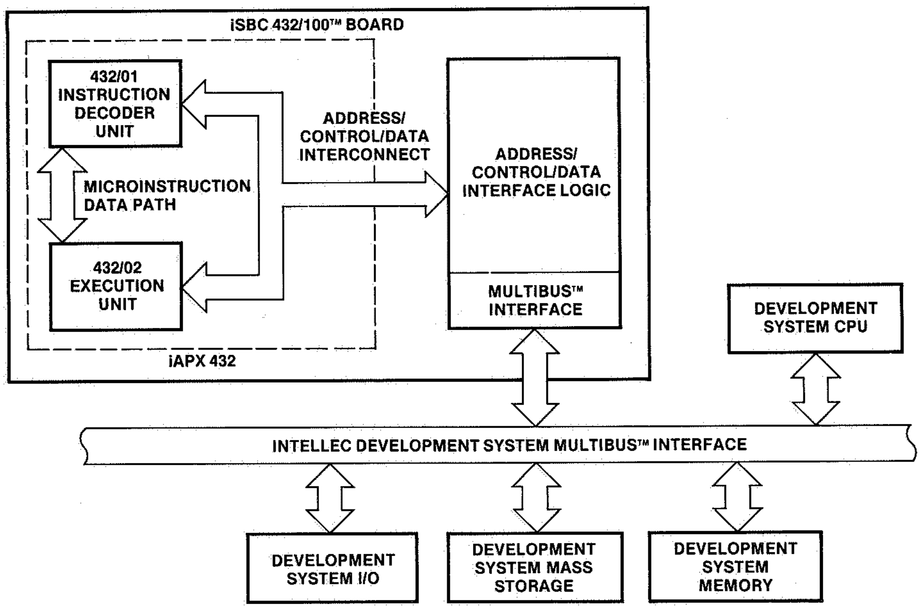
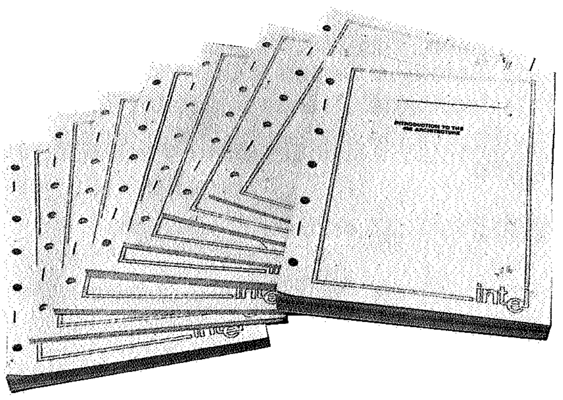

iSBC 432/100™ BOARD
432/01 INSTRUCTION DECODER UNIT
ADDRESS/ CONTROL/DATA INTERCONNECT
MICROINSTRUCTION DATA PATH
432/02 EXECUTION UNIT
iAPX 432
ADDRESS/ CONTROL/DATA INTERFACE LOGIC
MULTIBUS™ INTERFACE
DEVELOPMENT SYSTEM CPU
INTELLEC DEVELOPMENT SYSTEM MULTIBUS™ INTERFACE
DEVELOPMENT SYSTEM I/O
DEVELOPMENT SYSTEM MASS STORAGE
DEVELOPMENT SYSTEM MEMORY

Figure 1. Fully-Configured Intellec® 432 System Block Diagram

Documentation

A comprehensive set of manuals, texts, and introductory guides accompanies the Intellec 432/100 system. Together with the Intellec 432/100 software, they provide a self-contained learning and evaluation environment. Manuals are provided to acquaint the user with the iAPX 432's architecture, to assist in evaluation and design, and to document the Intellec 432/100 hardware and software. The documentation includes:

- **iAPX 432 Object Primer** — Illustrates the use of objects and object-oriented design in the iAPX 432.
- **Introduction to the iAPX 432 Architecture** — Provides a motivation for and an overview of the iAPX 432 architecture.
- **iAPX 432 GDP Architecture Reference Manual** — Describes in detail the system objects and primitive data types and their associated operations, fault handling, and general system concepts and facilities. Serves as the reference text for constructing systems and programs through Object Builder.
- **Getting Started on the Intellec 432/100** — Serves as a guide to the Intellec 432/100 software and documentation.
- **Object Builder User's Guide** — Shows how to use Object Builder to construct sets of 432 objects and execute 432 programs.
- **Object Programming Language User's Manual** — Tutorial on and reference for OPL. Shows how to use OPL for constructing 432 educational programs.
- **iSBC 432/100 Hardware Reference Manual** — Documents principles of operation, board capabilities, installation instructions, and service information.



SPECIFICATIONS

Hardware — iSBC 432/100 Board

CENTRAL PROCESSOR

Intel iAPX 432 General Data Processor (43201 and 43202)

MEMORY ADDRESSING

2^{24} physical, 2^{40} virtual for iAPX 432

iAPX 432 addresses converted to 20-bit MULTIBUS I/O and memory commands

MEMORY CAPACITY

Through Intellec MDS — 1M byte maximum

INTERFACES

MULTIBUS — All signals TTL compatible

Serial I/O — RS232C compatible, one programmable line using 8251A

SERIAL COMMUNICATIONS CHARACTERISTICS

Synchronous — 5–8 bit characters; internal or external character synchronization; automatic sync insertion.

Synchronous Baud Rate — 0 to 64K baud.

Asynchronous — 5–8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection.

Asynchronous Baud Rate — 0 to 19.2K baud.

TIMER

Four Read/Write I/O ports for Intel 8253 Programmable Interval Timer. Used for serial I/O baud rate and processor scheduling clock.

CONTROL I/O PORTS

Processor ID — Read
Processor States — Read
Memory Offset — Write
Processor Control — Write

CONNECTORS

Interface	Pins (qty)	Centers (in.)	Mating Connectors
MULTIBUS	86	0.156	Viking 3KH43/9AMK 12
Serial I/O	26	0.1	3M 3462.000

SOFTWARE SUPPORTED TERMINALS

Intellec
Hazeltine 1500 Series
Lear Seigler ADM 3A

PHYSICAL CHARACTERISTICS

Width — 12.0 in.
Height — 6.75 in.
Depth — 0.70 in.
Weight — 16 oz

DC POWER REQUIREMENTS

4.5A @ +5V ±5%
40 mA @ ±12V ±5%

ENVIRONMENTAL CHARACTERISTICS

Operating Temperature — 0°C to 50°C
Relative Humidity — to 90% (non-condensing)

Software

Object Builder — Requires 192 ~~128~~K of memory
Object Programming Language
Shipped in both single and double density diskettes

Documentation

171858-001 — iAPX 432 Object Primer

171821-001 — Introduction to the iAPX 432 Architecture

171860-001 — iAPX 432 GDP Architecture Reference Manual

171819-001 — Getting Started on the Intellec 432/100

171859-001 — Object Builder User's Guide

171823-001 — Object Programming Language User's Manual

171820-001 — iSBC 432/100 Hardware Reference Manual

ORDERING INFORMATION

Part Number	Description
Intellec 432100	432 Evaluation and Education System

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA • (408) 734-8102 x598

Printed in U.S.A./Y-20/0181/50K/BA/SWW

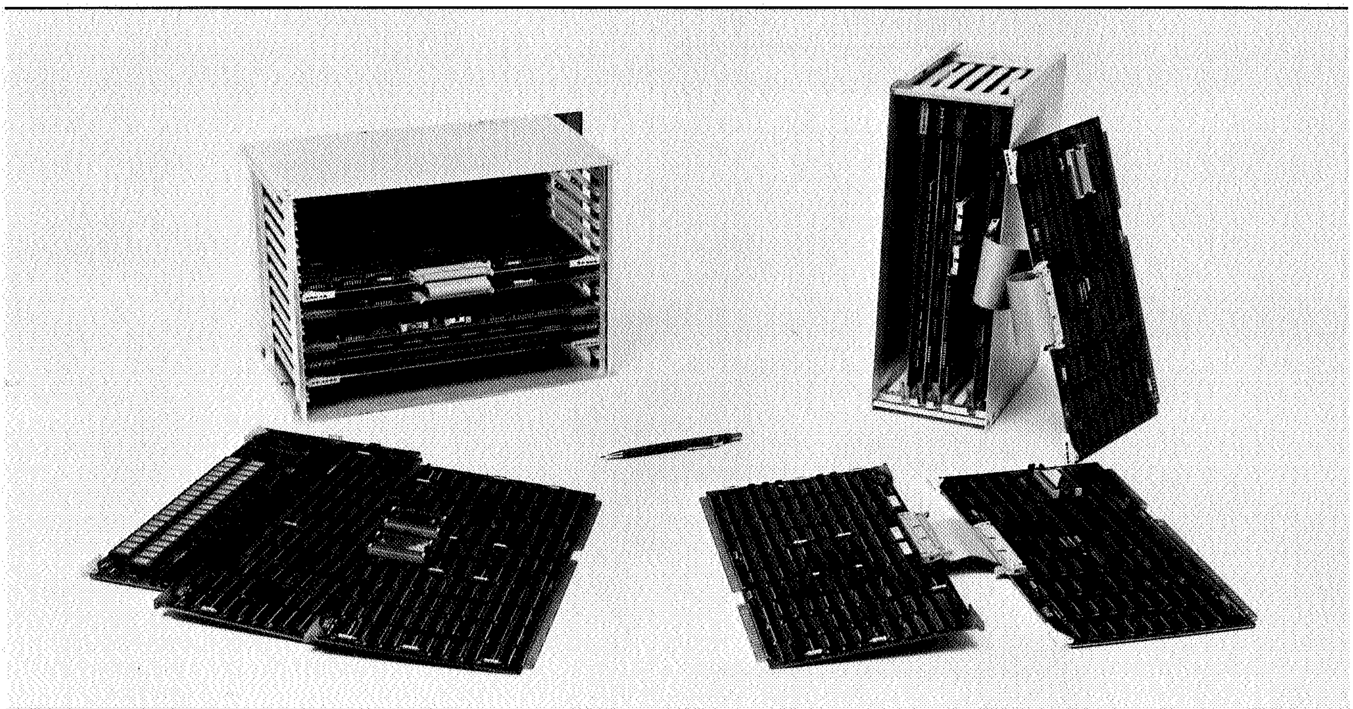
PRELIMINARY

SYSTEM 432/600 32-BIT EXTENSIBLE COMPUTER SYSTEM

- **Modular and Extensible**
  - **— iSBC board form factor**
  - **— Range of CPU performance, memory capacity, I/O capability**
  - **— Field upgradable**
- **Processor Extensibility**
  - **— Up to six processors, any mix of General Data or Interface Processors per system**
- **Memory Features**
  - **— 128K to 4M bytes physical memory**
  - **— Optional memory interleaving**
  - **— ECC protection**
- **Advanced 32-Bit iAPX 432 Architecture**
  - **— Transparent multiprocessing**
  - **— Object oriented addressing**
  - **— High level language instruction set**
  - **— Silicon Operating System**
  - **— 2^{40} bytes virtual memory**
- **Extensible MULTIBUS™ Peripheral Subsystems**
  - **— One to five independent peripheral subsystems**
- **Hardware Floating Point Capability**
- **Diagnostic Software Package**

System 432/600 computers utilize the advanced VLSI architecture of the iAPX 432 Micromainframe. The System 432/600 product line includes circuit boards, backplanes, cardcages, and chassis which are offered individually and as integrated 32-bit computer systems.

Through subsystem extensibility, System 432/600 computers offer a wide range of processing power, memory capacity, and I/O capability. A system including two data processors, 512 kbytes of ECC memory, and a single I/O processor with MULTIBUS subsystem requires approximately one-half of a cubic foot, offering the potential for a desktop 32-bit computer system.



March 1981
171861-001

ARCHITECTURE

The System 432/600 line of 32-bit computers takes full advantage of Intel's 432 architecture. The hardware implemented software transparent multiprocessing provides an immediate range of performance in attacking applications which involve concurrent tasks. Incremental performance can be realized by simply adding additional data processing boards or I/O subsystems.

The object based architecture significantly reduces the cost of developing large software systems, enhances their reliability and security, and facilitates the safe integration of new software modules into existing systems.

The Silicon Operating System improves the performance and reliability of systems using the iAPX 432. Certain time critical, complex, and security sensitive operating system mechanisms are implemented in the hardware of the iAPX 432. However, the user maintains flexibility in system design since the policies that drive the hardware mechanisms are software controlled.

TECHNOLOGY

System 432/600 computers utilize the iAPX 432 General Data Processor (GDP) and Interface Processor (IP) components. The technology which makes these components available in such a compact form is Intel's VLSI HMOS technology — a short channel, high density, semiconductor fabrication process. This process yields a density in excess of 100K devices per chip and allows Intel to offer a highly sophisticated CPU architecture with an unmatched system performance/density ratio.

FUNCTIONAL DESCRIPTION

Functionally, the System 432/600 can be partitioned into three subsystems: processing, memory, and I/O. The processing subsystem consists of one or more General Data Processors (one processor per GDP board). The memory subsystem stores data on modular storage array boards and allows for access by the General Data and I/O processors. The I/O subsystem performs the interface function between the central subsystems (General Data Processors and memory) and external peripheral devices. Multiple and independent I/O subsystems may be implemented simply by adding appropriate boards. Figure 1 graphically presents a block diagram of the partitioning of the minimal System 432/600 computer.

The System 432/600 allows for independent extensibility of its three subsystems. A range of performance and capacitites can be obtained by varying the number of modules of each type in a system.

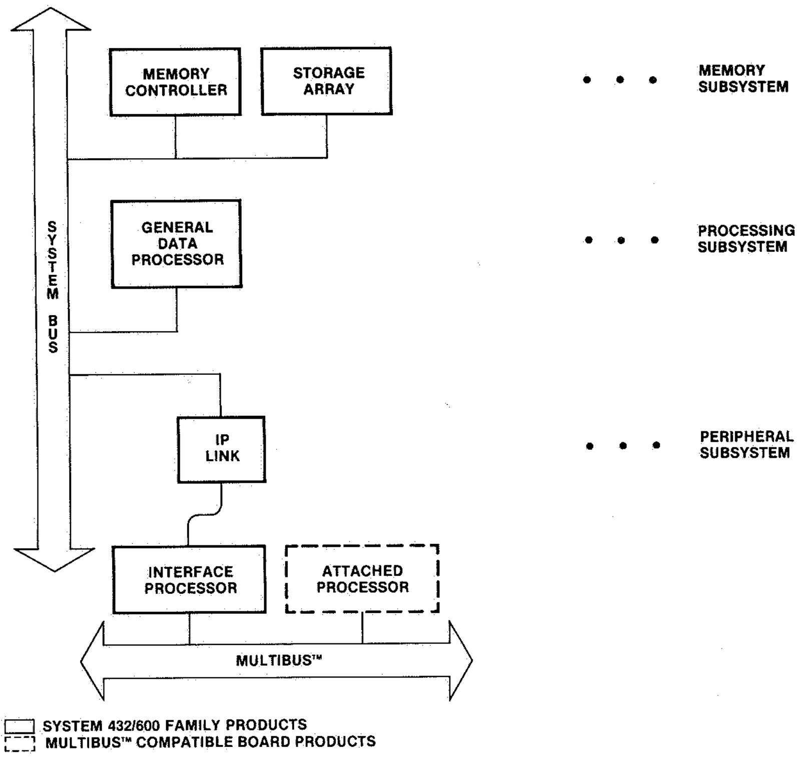


Figure 1. System 432/600 Block Diagram: Minimum Configuration

Transparent multiprocessing allows for an immediate breadth of performance within a product line by simply adding processor modules without any affect on system or applications software. Memory capacity can be expanded in varying increments by simply adding memory boards to the system. I/O capabilities can be expanded by two means, First, any MULTIBUS compatible board can be added to an existing I/O subsystem. And second, parallel and independent I/O subsystems may be added to the system. Figure 2 demonstrates the extensibility of the three subsystems.

The I/O subsystems allow for the off-loading of I/O processing functions from the data processing subsystem. Each I/O subsystem can perform tasks of varying complexity ranging from mass storage control and terminal interface to decentralized data processing.

The I/O subsystem implementation allows for attached processing to be performed by Intel's existing line of peripheral devices. In addition, since the I/O subsystems utilizes the MULTIBUS interface, any compatible single board computer, controller or peripheral may be used with the system. Thus a broad range of devices are immediately available for configuration within the I/O peripheral subsystems of System 432/600 computers.

SYSTEM INTERCONNECT

The System 432/600 utilizes two types of backplane interconnect busses. The System Bus is a 32-bit bus that electrically connects the General Data Processor (GDP), Interface Processor Link (IPLk), Memory Controller (MC), and Storage Array (SA) boards. Parity bits protect each of the 4 bytes. Address and data are time multiplexed on the bus. Dedicated control and status lines coordinate system activity, perform parallel arbitration, and support memory interleaving. Slots on a System Bus backplane are dedicated, in groups, to particular board types. Separate slots accommodate processor boards (GDP or IPLk), storage array boards and the

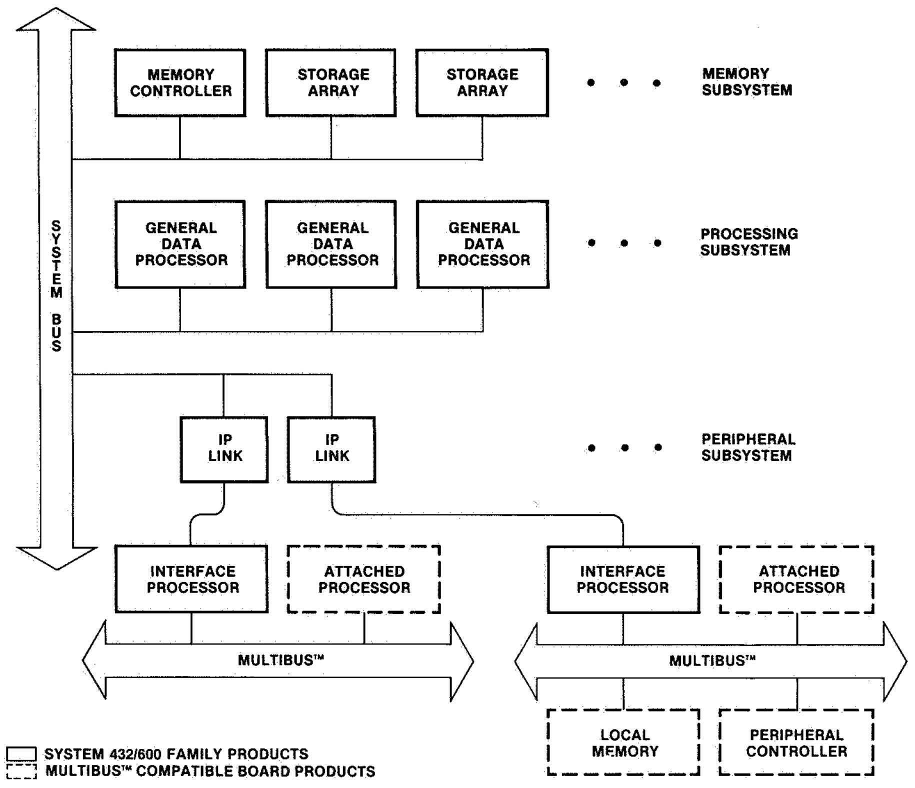


Figure 2. System 432/600 Block Diagram: Extended Configuration

memory controller board. Table 1 summarizes the System Bus backplane options offered by Intel and the partitioning of the slots among the board types.

Table 1. System 432/600 System Bus Backplane Slot Partitioning

Total Backplane Slots	No. of Slots by Group		
	GDP or IPLk	Memory Controller	Memory*
6	3	1	2
12	5	1	6
18**	6	1	10

* Each memory slot accommodates either 128K or 256K byte Storage Array board.

** The eighteenth slot used for test/monitor connector.

The industry standard MULTIBUS interconnect bus connects the Interface Processor (IP) board with an attached processor and other I/O subsystem devices. The MULTIBUS backplane is physically separate from the System Bus backplane.

A flexible cable connects the Interface Processor Link board on the System Bus backplane to the Interface Processor board on the MULTIBUS backplane. This IPLk cable provides a synchronous 16 bit communications path with byte parity and dedicated control lines. Address and data are time multiplexed across the link.

SYSTEM IMPLEMENTATION

Data Processing Subsystem

General Data Processor (GDP) boards perform the data manipulation and computation within the System 432/600. The heart of the GDP board is the 32-bit iAPX 432 General Data Processor. The iAPX 432 GDP features a high level language instruction set, the Silicon Operating System, and software transparent multiprocessing. In support of multiprocessing, the GDP board performs buffering, interconnection between the processor component bus and the System Bus, and arbitration between multiple processors within a system.

Multiple GDP boards may be configured within System 432/600 computers. Architectural features make the addition of processor modules transparent to the software and will enhance the performance of systems executing concurrent tasks. Arbitration between multiple GDP and IPLk boards for access to the System Bus is based on a dynamic priority algorithm. Similar to a round robin scheme, this mechanism assures that GDP and IPLk boards requesting memory access receive equal access to the System Bus.

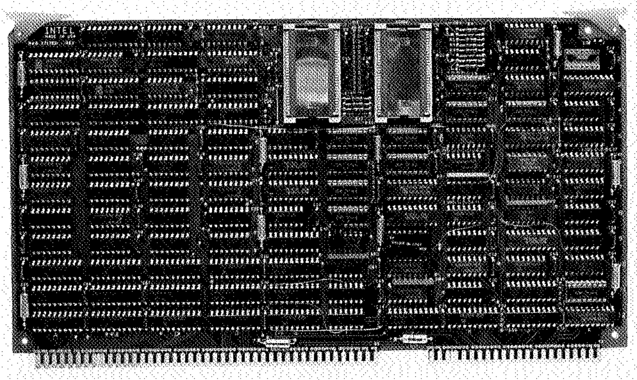


Figure 3. General Data Processor Board

Memory Subsystem

System memory is implemented using modular self-refreshing storage boards. The memory array includes 32 bits of data and seven bits of error correcting code (ECC) per word. The ECC facility allows for single bit error correction and double bit error detection on each module. Two types of storage array boards are available with capacities of 128K and 256K bytes.

Storage array (SA) board types may be mixed within a system providing flexibility in memory size, incremental granularity, and upgradability. The only limitation exists when memory is interleaved, in which case paired boards must be of the same capacity.

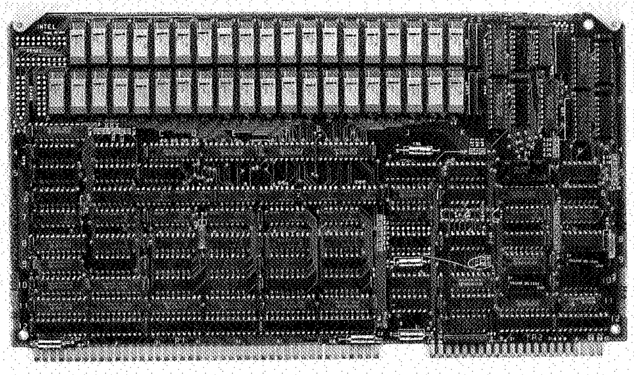


Figure 4. Storage Array Board

The System 432/600 utilizes a single memory controller board. This board services processor memory requests, maps instruction addresses to the proper storage array board(s), supports memory interleaving, and centralizes hardware error logging.

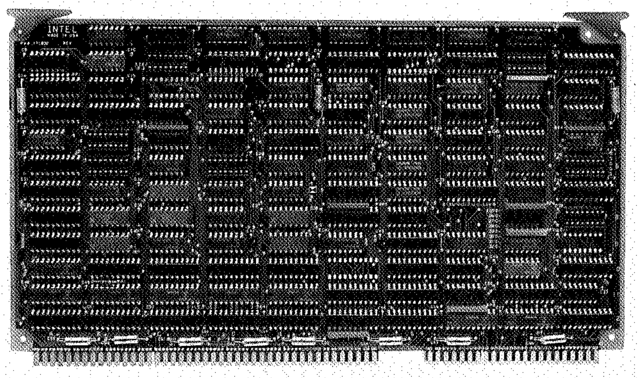


Figure 5. Memory Controller Board

Data can be accessed from memory by either a GDP or an IP in variable length increments, including: 1, 2, 4, 6, 8, and 10 bytes. When requests cross word boundaries memory interleaving significantly improves performance by effectively accessing successive words in parallel. In an interleaved configuration logically sequential words are stored on separate storage array modules. The memory controller initiates accesses to logically sequential words on consecutive clock cycles. Data is returned to the requesting processor on consecutive bus cycles rather than after sequential memory access cycles.

I/O Subsystem

System 432/600 products may be configured with multiple and independent I/O subsystems. Each I/O subsystem requires an Interface Processor Link board connected to the System 432/600 System Bus and an Interface Processor board connected to the MULTIBUS backplane. The Interface Processor board utilizes the iAPX 432 IP component.

In addition to the physical interconnect, the IP-IPLk board pair makes available protected windows through which an attached processor (AP) in the I/O subsystem can access the central system's address space. The IP maps a portion of the AP's address space into the address space of the central system.

The IP extends the AP's instruction set to include the iAPX 432 object oriented instructions. This allows the AP to perform data manipulation functions within the central system. The IP does not fetch instructions on its own but executes instructions, one at a time, as they are received from the AP. After completion of a command the IP sends an interrupt to the AP to signal that it has finished.

The IP facilitates the initialization of the System 432/600 memory through its physical mode addressing capability. Logical addressing within the System 432/600 requires the existence of certain dynamic data structures within the system's address space. These structures must be established before a GDP can begin processing. The AP initializes these structures by utilizing the IP's physical addressing mode.

The attached processor manages the I/O subsystem. The AP is responsible for activities such as polling devices, responding to interrupts, and setting up and monitoring DMA transfers. The AP can take on additional functional capabilities and work as a decentralized data processor.

This system structure improves performance by minimizing the burden of low level I/O tasks on the central processor (GDP), while maintaining the protected and controlled access environment. The functional complexity of a peripheral subsystem is a design parameter. Since multiple and independent I/O subsystems are possible, each subsystem can be optimally configured (hardware and software) for its particular application.

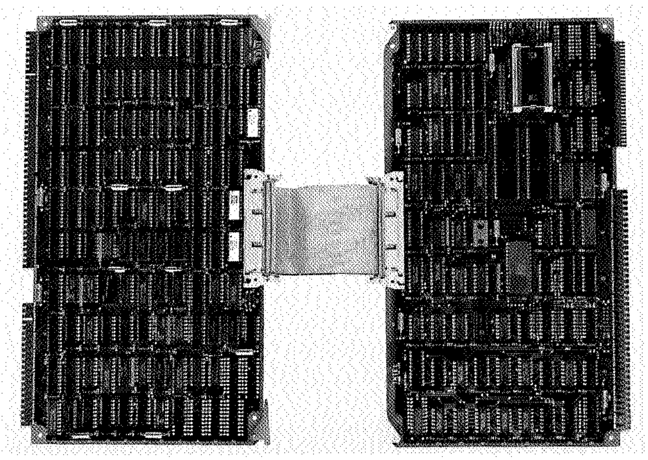


Figure 6. IP-IPLk Board Pair

SYSTEM CONFIGURABILITY AND PACKAGING

Intel offers System 432/600 products at two levels of integration. For maximum configuration flexibility the System 432/600 building blocks, including logic boards, backplanes, cardcages, and chassis are offered individually. Integrated 32-bit computers housed in table top and rack mountable chassis are also available.

Subsystem extensibility within System 432/600 computers is limited by electrical, logical, and performance considerations, as well as the physical size of the backplane. The maximum size of the System Bus is 23 board slots. Logically, a combined total of six GDP and IPLk boards can be configured in a System 432/600 product. Each system requires a single memory controller board which can logically address a maximum of 16 storage array boards. Performance considerations are application dependent and can be met by balancing the mix of GDPs and IPs in a system.

Cardcage size is independent of backplane size for flexibility in system configuration. A cardcage can accommodate any combination of backplanes, limited only by the physical number of board slots in the cardcage. Figure 7 suggests backplane and cardcage combinations.

A stylized, powered and cooled chassis is offered that may be rack or table top mounted. The chassis includes an 18 slot cardcage that can accommodate various backplane configurations.

The System 432/670 is the mid-range member of the integrated 32-bit computer product line. Enclosed in a rack or table top mountable powered and cooled chassis the System 432/670 includes two General Data Processors, one Interface Processor, and 512 kbytes of ECC memory. The System 432/670 includes a 12 slot System Bus backplane that can accommodate subsystem expansion to include two additional GDPs and connections to remote I/O subsystems as well as a total of 1.5 Megabytes of ECC memory. The enclosed MULTIBUS I/O subsystem includes an SBC 86/12A attached processor with a total of 32 kbytes of EPROM/ROM, 64 kbytes of RAM, and three MULTIBUS backplane slots for user configuration.

RELIABILITY AND FIELD MAINTENANCE

Specific hardware design features that enhance the reliability of the System 432/600 include the highly integrated processors; error correction and detection on the RAM storage array; parity checking on all address and data paths; and type checking on accesses to memory data structures.

An Error Correcting Code (ECC) insures the integrity of data stored in memory. A seven bit code is used for each 32-bit memory word. This feature allows for single bit error correction and double bit error detection on each of the storage modules.

Parity bits are generated for each byte of address, data, and access specification on the System Bus as well as the on-board busses and the link cable. This mechanism protects the system from propagation of noise induced errors as well as the failure of discrete logic components.

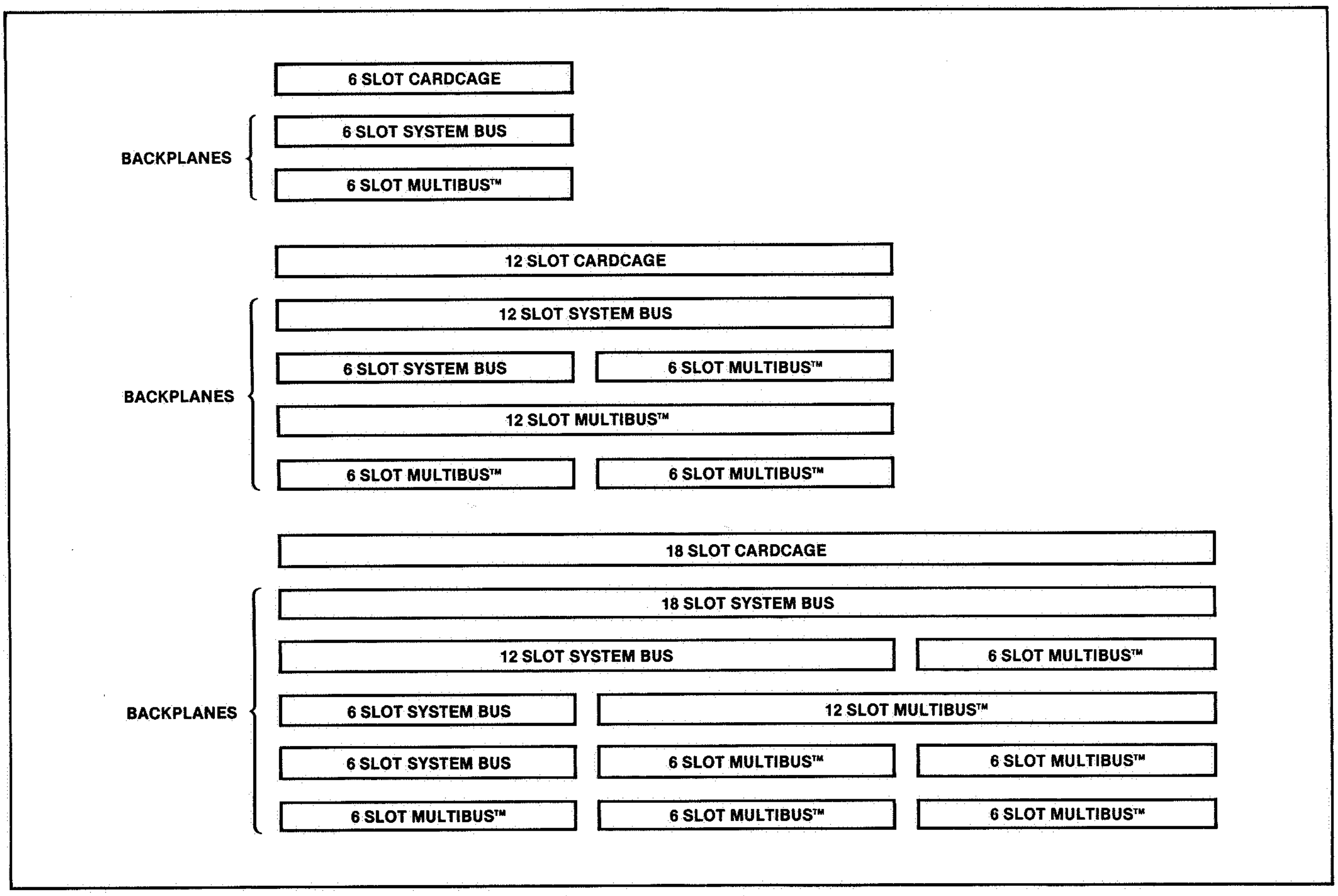


Figure 7. Suggested Cardcage and Backplane Combinations

A series of hardware managed registers located on the Memory Controller board allow for logging and analysis of system errors. A global view of the error state of the system is maintained in the System Error Register.

More detailed error information is stored in the Memory Error Registers and the Processor Error Register. The memory error registers contain information about the particular access that resulted in the error logged in the System Error Register. This register stores the identification of the processor making the request, the word address of the memory failure, and a pointer to the particular bit that failed. The Processor Error Register identifies any processor(s) in a fatal state and any parity error that may have occurred on a data transfer initiated by a processor. Software executing on any processor (GDP or AP) can access all three registers.

A software diagnostic package is offered in the form of a two-level set of programs. At the first level is a system validator designed to provide a basic system confidence test during initialization or maintenance activities. The second level includes a modular set of programs designed to identify the faulty board(s) within a system.

The diagnostic programs begin execution on the attached processor, first testing the Interface Processor and then the IPLk interconnect to the System Bus. The Memory Controller, Storage Array module(s) and General Data Processor(s) are then tested in sequence to verify the proper functioning of those modules. The diagnostic software provides sufficient resolution to allow programmatic board level fault isolation and repair by board replacement.

REAL TIME SOFTWARE

Intel's iMAX 432 Multifunction Applications Executive software, specifically designed for the 432 product line, provides the execution environment to support the advanced features of the architecture. iMAX 432 is a modular collection of software packages that can be configured to meet specific application needs. Packages can be added by the user to extend the capabilities of the executive.

SYSTEM DEVELOPMENT CAPABILITIES

The software development environment for System 432/600 products includes a compiler and a linker operating on a host computer; a loader and debugger running on an Intellec Series III (or II upgrade) Microcomputer Development System; and the System 432/670 as an execution vehicle. A 2780/3780 communications link ties the host to the Intellec Development System. An IP board from the execution vehicle plugs into the MULTIBUS backplane of the Intellec Development System to provide the logical and electrical interconnection between the two.

The systems implementation language for the 432 family of products is based on the Department of Defense standard language Ada. Inspired by Pascal, Ada allows programmers to realize the productivity promised by modern language innovations such as strong typing, data abstraction, and modularity. A true superset of standard Ada, the systems implementation language provides full access to the 432 architecture and supports the more general multitasking and run time facilities. Any standard Ada program will run correctly on a System 432/600 computer.

SPECIFICATIONS

Technology

PROCESSORS — HMOS VLSI General Data Processors and Interface Processors.

MAIN MEMORY — MOS, 150ns-200ns, Dynamic RAM.

BOARDS — Low power Schottky and Schottky TTL interconnect logic.

Data Processing

PROCESSOR — iAPX 432 General Data Processor. 1-5 GDPs per system.

HARDWARE FLOATING POINT — Full proposed IEEE standard floating point.

INSTRUCTIONS — Over 200 instructions; bit variable, frequency encoded.

ADDRESSING — 4 Mbytes physical, 1 terabyte (2**40) virtual.

ADDRESSING MODES — Scalar, static vector, dynamic vector, record.

DATA TYPES — 8-bit characters; 16-, 32-bit integers; 16-, 32-bit ordinals; 32-, 64-, 80-bit reals.

Memory

TYPE — MOS, 150ns-200ns, 64K or 32K Dynamic RAM.

ORGANIZATION — 32 bits data plus 7 bits ECC.

PHYSICAL — Up to 4 Mbytes.

GRANULARITY — 128 and 256 kbyte increments.

ACCESS LENGTHS — 1, 2, 4, 6, 8 and 10 bytes.

MEMORY ACCESS BANDWIDTH — 6.5 Mbytes per second sustained.

I/O

I/O PROCESSOR — iAPX 432 Interface Processor. 1-5 IPs per system.

IP LINK BANDWIDTH — 2.25 Mbytes per second sustained, Multibus to 432 memory.

IP LINK DATA PATH — 16 bits data plus 2 parity bits.

IP LINK LENGTH — Up to 10 ft. @ 8 MHz clock rate.

ATTACHED PROCESSOR — Any Intel SBC or MULTIBUS compatible user device.

System Bus

DATA PATH — 32 bits data plus 4 parity bits.

CLOCK — 8 MHz.

BUS BANDWIDTH — 6.5 Mbytes per second sustained.

Circuit Boards

PHYSICAL DIMENSIONS (Individual Boards)

Width — 12.00 in. (30.48 cm)
Height — 6.75 in. (17.15 cm)
Depth — 0.56 in. (1.43 cm)

ELECTRICAL REQUIREMENTS

Single +5 (±10%) volt supply.

Board	Current @ +5 Volts (Worst Case Max.)
iSBC 432/601 (GDP)	7.0
iSBC 432/602 (IP)	5.5
iSBC 432/603 (IPLk)	6.5
iSBC 432/604 (MC)	6.5
iSBC 432/606 (128kB)	7.0
iSBC 432/607 (256kB)	7.0

ENVIRONMENTAL REQUIREMENTS

Operating Temperature — 0° to 55°C ambient

Relative Humidity — 10%-90% non-condensing

Air Flow — 300 LFPM air flow over boards.

Cardcages

PHYSICAL DIMENSIONS

6 slot cardcage:

Width	13.00 in. (33.02 cm)
Depth	4.88 in. (12.04 cm)
Height	8.25 in. (21.00 cm)

12 slot cardcage:

Width	13.00 in. (33.02 cm)
Depth	9.13 in. (23.19 cm)
Height	8.25 in. (21.00 cm)

18 slot cardcage:

Width	13.00 in. (33.02 cm)
Depth	13.38 in. (33.99 cm)
Height	8.25 in. (21.00 cm)

Chassis

PHYSICAL DIMENSIONS

Width — 16.76 in. (42.57 cm), RETMA compatible
Height — 12.23 in. (31.06 cm)
Depth — 21.75 in. (55.25 cm)

POWER SUPPLY CHARACTERISTICS

Channel Ratings (Max.) — 120 amps @ +5 volts; 10 amps @ +12 volts; 4 amps @ −12 volts; 2 amps @ −5 volts.

Maximum Supply Loadings — 750 watts, total over four channels.

Input Power Requirements (Max.) — 11 amps @ 115 volts ac (+10%, −20%), 62-47 Hz optional: 220V ac, 50 Hz.

CERTIFICATIONS

UL Listed, FCC and CSA certified. Designed to meet VDE EMI requirements.

Reference Manual

System 432/600 Reference Manual (NOT SUPPLIED).

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel Sales Representative, Distributor Office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95037.

Related Documents

Intel 432 System Summary	171867-001
Introduction to the iAPX 432 Architecture	171821-001
iAPX 432 General Data Processor Architecture Reference Manual	171860-001
iAPX 432 Interface Processor Architecture Reference Manual	171863-001
iMAX 432 Advance Information Sheet	171866-001
Intellec Series III/432 Development System Data Sheet	171868-001

ORDERING INFORMATION

Part Number	Description
INTEGRATED SYSTEMS:	
SYS 432/670	Integrated 32-bit computer including: 2 - General data processors (SBC 432/601) 1 - Memory controller (SBC 432/604) 2 - 256 kbyte storage array boards (SBC 432/607) 1 - Interface processor link board (SBC 432/603) 1 - Interface processor board (SBC 432/602) 1 - 12 slot system bus backplane (SBC 432/611) 1 - 6 slot MULTIBUS backplane (SBC 432/615) 1 - Enclosed chassis with 18 slot cardcage (SBC 432/630) 1 - Attached processor (SBC 86/12A) 1 - 16 kbyte Multimodule EPROM (SBC 340) 1 - 32 kbyte Multimodule RAM (SBC 300)

Part Number	Description
BUILDING BLOCKS:	
Data Processor Modules:	
SBC 432/601	General data processor board
Memory Modules:	
SBC 432/604	Memory controller board
SBC 432/606	128 kbyte dynamic RAM storage board with ECC
SBC 432/607	256 kbyte dynamic RAM storage board with ECC
I/O Modules:	
SBC 432/602	Interface processor board
SBC 432/603	Interface processor link board
Backplanes:	
SBC 432/610	6 slot system bus backplane
SBC 432/611	12 slot system bus backplane
SBC 432/612	18 slot system bus backplane
SBC 432/615	6 slot MULTIBUS backplane
SBC 432/616	12 slot MULTIBUS backplane
Cardcages:	
SBC 432/620	6 slot cardcage
SBC 432/621	12 slot cardcage
SBC 432/622	18 slot cardcage
Chassis:	
SBC 432/630	Enclosed, powered, and cooled chassis with 18 slot cardcage
Accessories:	
SBC 432/635	System bus extender card
SBC 432/636	Interface processor link cable kit (external, shielded)
SBC 432/637	Interface processor link cable kit (internal, ribbon)
SBC 432/638	Interface processor link cable (external, 6 ft. shielded)

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA • (408) 734-8102 x598

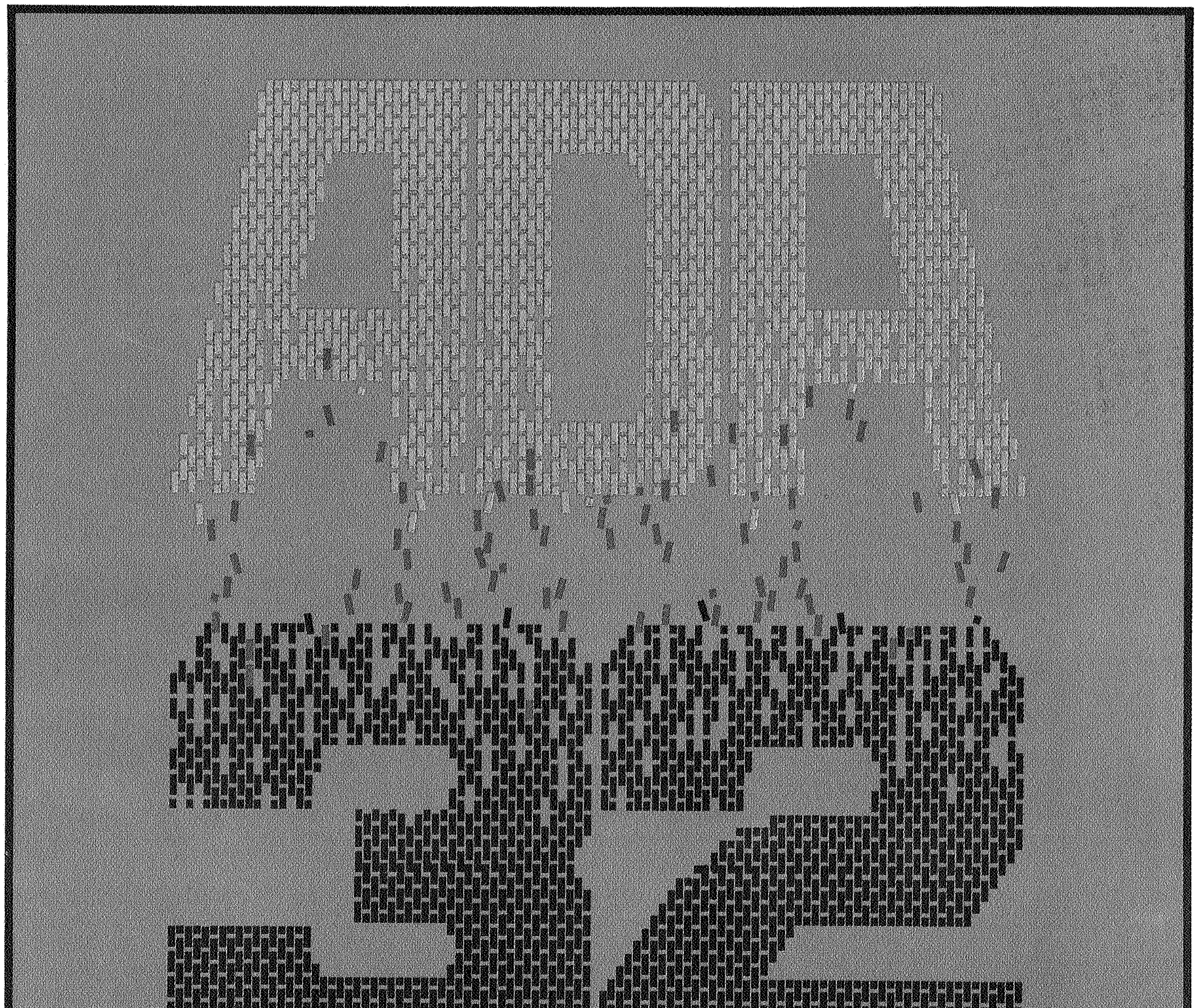
AFN-01871A-4

FEBRUARY 24, 1981

A Mc GRAW-HILL PUBLICATION

Electronics®

SOFTWARE SHAPES VLSI PROCESSOR

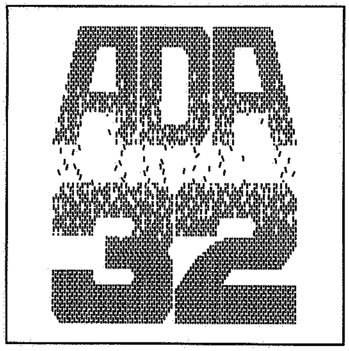


AR 165

Ada determines architecture of 32-bit microprocessor

Its use of the high-level language makes the chip set easy to program for multiuser, multifunction applications like office systems

by Justin Rattner and William W. Lattin *Intel Corp., Aloha, Ore.*



☐ Generation after generation, microprocessors increase in sophistication. And the time and cost of developing new applications for them escalates also, till by now skilled system designers and programmers are in very short supply.

Anticipating this critical situation, Intel Corp. undertook to develop a microcomputer system that can be made to handle complex, software-intensive applications in much less time and at much lower cost than has been usual. The project encompassed nearly all aspects of computer technology and resulted in the development of a new semiconductor process—high-performance MOS, or H-MOS—a new package with a high pin count—the quad in-line package or QUIP—and three of the largest integrated circuits in history (see "A history of the Aloha project," p. 125).

Its crowning achievement thus far is a 32-bit microprocessor—the iAPX 432—that has a major new architecture, a new operating system (iMAX), and one of the first compilers for Ada, the Department of Defense's new standard programming language.

As the first 32-bit microprocessor designed specially for multiuser applications, the two-chip general data processor (GDP) is a significant milestone in computer technology (Figs. 1 and 2). Together with the single-chip interface processor (IP) shown in Fig. 3, it was designed to serve the kind of cooperative, multifunction applications typified by future office information equipment and distributed data-processing systems. Similar systems are also envisioned for use in computer-aided design and factory automation.

Cooperative multifunction applications share four important characteristics. They are large in scale and broad in scope, requiring mainframe computing power. They are software-intensive, each discrete function or service requiring considerable programming. They are expected to evolve over time, so the design must allow for future software enhancement and increments in performance. Finally, they are applications where the failure of the computer system can have serious consequences to human life or the cost of doing business, so that long-term dependability of both hardware and software is essential.

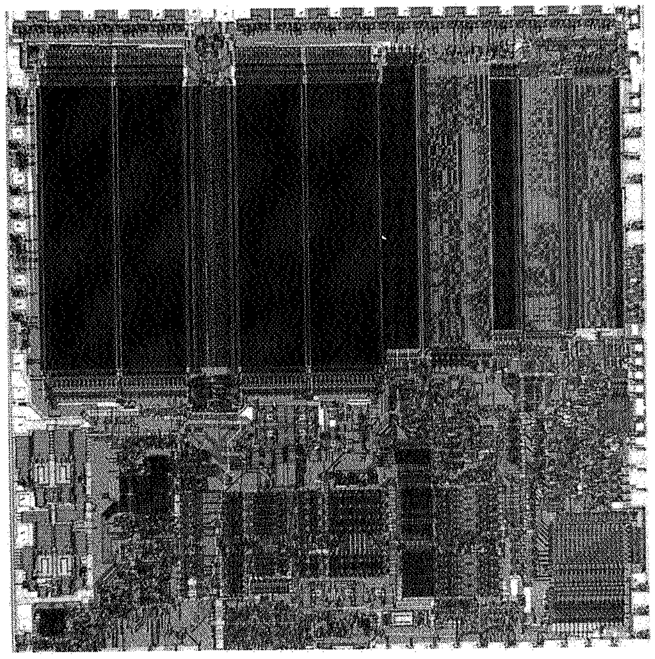
Those characteristics of cooperative multifunction applications guided the 432 designers to its major goals. The breadth of computing power required, in terms of both function and performance, includes support for multiprogrammed and virtual memory operating systems; as such, ultimate performance for fully configured systems was to be that of a mid-range mainframe.

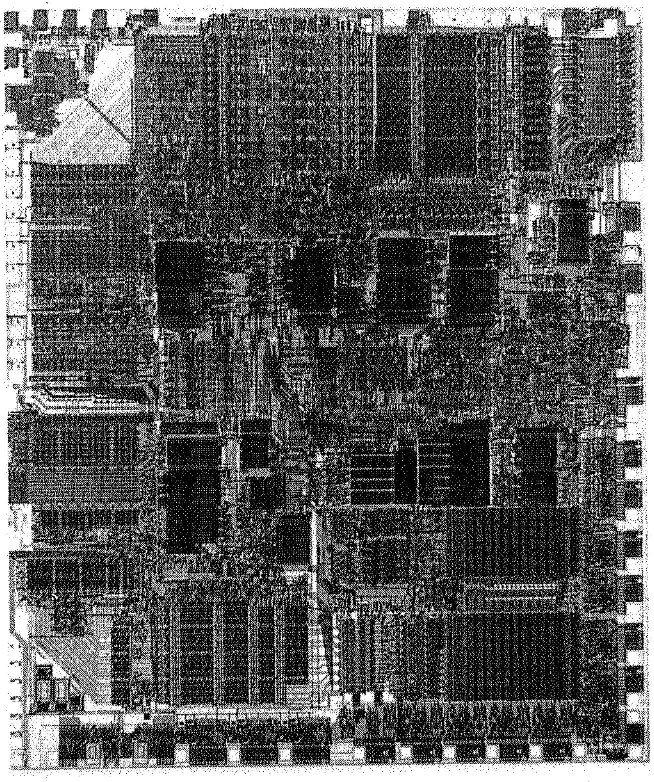
Further, it was decided that the long-dreamed-of incremental performance capability—adding power to a 432-based product already in the field simply by plugging in additional GDPs and IPs—would gracefully accommodate planned or even unplanned growth in computational power over the life of an application. The goal of increased programmer productivity was met by supporting a comprehensive methodology for modular software development, served by using Ada as the 432's native tongue. And finally, to ensure high hardware and software dependability, the 432 includes extensive hardware fault detection and software-protection mechanisms.

Transparent multiprocessing

Through careful attention to multiprocessing issues in the definition of both its system organization and its architecture, the 432 successfully implements the long-sought-after idea of transparent multiprocessing for general-purpose computation. This simple but important concept means that the number of data processors in a 432 system can be increased or decreased without software modification. It is even possible to start or stop a processor at any time without informing, let alone damaging, a single piece of software. More importantly, neither the operating system nor the application programs need rewriting to exploit an increase in the number of processors.

A principal challenge in the development of any multiprocessing system is the design of the interconnection


1. Instruction decoding. One of the two chips that make up the general data processor, the 43201 is the instruction decoder. It contains more than 100,000 devices on a single die, making it one of the densest VLSI circuits to have been fabricated so far.


2. Microexecution unit. Decoded instructions from the 43201 are executed by the 43202, which has over 60,000 devices on chip. Its unusual horseshoe-shaped data bus can be discerned from the photograph. The device is housed in a 64-pin quad in-line package.

structure that ties the processors and memory subsystem together. The 432 approach to this problem is unusual: rather than define a standard bus, the 432 simply defines a standard way for processors to communicate with memory and each other. This frees the designer to choose his own bus structure, optimizing the cost/performance ratio of the application. All 432 processors are compatible with the interconnect protocol.

The main goal of the interconnect protocol is to reduce bus use. For that reason, it puts requests and replies in separate packets, so that the first need not monopolize the bus while waiting for the second. For example, a processor generates a request packet in order to access memory but expects a reply packet (Fig. 4) from the memory system only if the request specified a read cycle. The result is that the processor ties up the bus only long enough to transmit a packet to the interconnect; sometime later a reply packet will be returned, if necessary, to the requesting processor. In the interval, other processors may be active on the interconnection.

For still greater efficiency, the protocol defines packets as variable in length. A single request or reply may transmit from 1 to 16 bytes of information. Fewer individual storage accesses need be made to obtain long operands, and designers using the 432 can improve system performance by widening the interconnect's bus.

Communication between processors is one of several additional functions supported by the interconnect protocol. Because of the packet format, a processor can send an attention signal to one processor or broadcast it to several processors simultaneously. Upon receiving a signal, a processor examines an interprocessor message area defined in memory. The message previously deposited there by the sending processor will instruct the receiving processor as to the desired course of action. Typical interprocessor messages can direct either one or a set of processors to start, stop and redispatch.

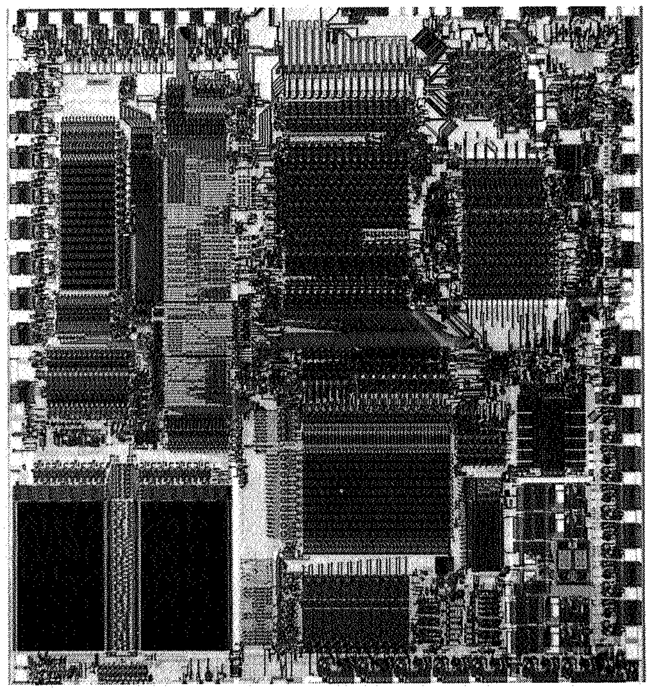
Many bus structures can be designed to meet the 432 packet protocol. A simple, single bus interconnect implemented in discrete logic is shown in Fig. 5. On this bus, the packets are demultiplexed as they leave the processors. The separate address and data lines are interfaced to a static memory subsystem in the conventional way. Up to four processors can be supported without serious contention by this simple interconnect.

A conventional mainframe often offloads onto a minicomputer all responsibility for low-level device control and data transfers. The 432 general data processor (GDP) uses an attached input/output processor in the same way. Device driver execution, device interrupts, direct-memory-access channel initialization are all handled within the I/O subsystem.

Offloading I/O chores

A typical I/O subsystem is built around a Multibus or other standard microprocessor system. A standard microprocessor, such as an 8086, is connected to the bus along with memories and peripheral devices to form a complete, attached I/O processor.

The GDP is connected to the microprocessor-based I/O subsystem by the 432 interface processor (IP). Under software control, a group of programmable, associative memories in the IP called window registers can be programmed to map a subsystem's address space into the 432's. The mapping operation is totally transparent to the attached I/O processor, so that both read and write


3. Input/output support. The single-chip interface processor, 43203, offloads the general data processor of the tasks involved in communicating with input/output devices. It includes 65,000 devices and can emulate many of the general processor's instructions.

cycles on the subsystem bus can proceed normally.

All communication in the 432's central system is based on messages and not interrupts. The IP receives those messages sent to an I/O subsystem and holds them in its window registers while signaling the I/O processor with an interrupt. The I/O processor then fetches the message through the IP window.

Communication in the opposite direction is slightly more complex. Since conventional interrupts do not exist in the central 432 system, the IP must give the attached processor the ability to send messages in the same way as a 432 data processor. The IP looks like a memory-mapped peripheral to the I/O subsystem as the latter writes commands to the IP registers. In passing those messages into the central system the microprogrammed logic of the IP emulates many of the same functions found in the high-level instruction set of the 432 data processor. Among the available commands is one to send message. Consequently a 432 data processor cannot detect any difference between messages sent by an IP and those sent by another data processor.

Multiple I/O subsystems may also be used to incrementally increase I/O processing power much as mulitple data processors increase processing power for general computation. By multiprocessing both I/O and data, the 432 serves many more applications than existing microcomputer systems can and meets the major goal of incremental, field-expandable, processing capability.

On chip

The processing units of the 432 are designed to exploit very large-scale integrated H-MOS technology to the full. The two-chip 43201/2 GDP, whose microphotographs appear in Figs. 1 and 2, contains 160,000 transistors. Over 100,000 devices are on the 43201 alone. The single-chip IP in Fig. 3 holds roughly 65,000 transistors. Each of the three 432 chips is housed in a 64-pin QUIP, as shown in Fig. 6 [*Electronics*, Jan. 4, 1978, p. 130], and each dissipates less than 2.5 watts of power from a single 5-volt supply. Two-phase clocking at 8 megahertz yields the nominal 125-nanosecond microcycle time.

These complex components could not have been developed without several advances in design techniques and tools. Among these was the use of regular logic structures and wiring topologies.

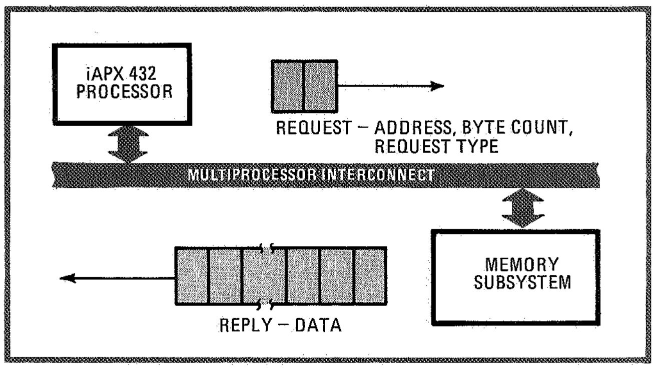
This technique, developed almost simultaneously at Intel by Sam Schwartz and at the California Institute of Technology by Carver Mead, dramatically reduces the number of randomly drawn transistors. Creating a structured integrated circuit design is hard but gets easier with practice. The micrographs reveal a general increase in geometric regularity from the earliest of the chips to be developed, the 43201, to the latest, the 43203.

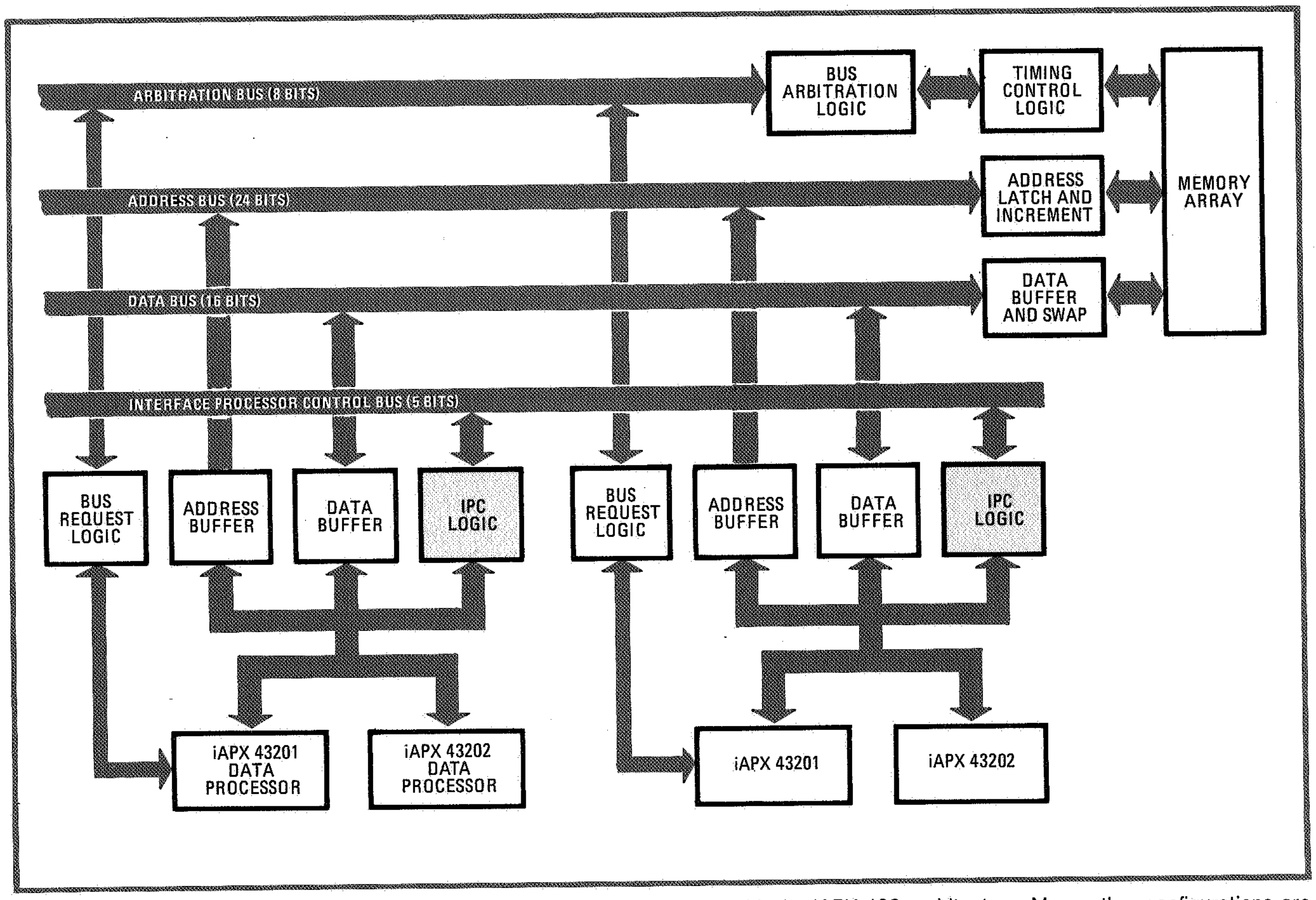
Both the GDP and the IP are microprogrammed and rely on high-performance microarchitectures to minimize microprogram size. The two-chip GDP contains a 4-K-by-16-bit microprogram ROM, and the IP contains a 2-K-by-16-bit ROM. The physical size of the IP's microprogram ROM is further reduced by having 2 bits stored in each ROM cell, a technique developed for the 8087 numeric data processor [*Electronics*, Oct. 9, 1980, p. 39] and improved upon in the 432.

The last two entries of Table 1 mention two of the more interesting functional capabilities of the 43201 and 43202. The address generator on the 43202 is responsible for mapping or translating 432 logical addresses into the physical addresses used to access the memory system. To accelerate the translation process, the address generator maintains a cache of recently used addresses, so that a new entry automatically replaces the one least recently used.

The silicon operating system is largely in microcode with some simple hardware assistance. Consequently the execution time for a typical operation like "send message" is five times faster than for a highly tuned minicomputer operating system and 20 to 30 times faster than for the best mainframe operating system.

Table 2 describes the allocation of microcode in the 43201/2 data processor and provides one of the most


iAPX 432 PROCESSOR
REQUEST – ADDRESS, BYTE COUNT, REQUEST TYPE
MULTIPROCESSOR INTERCONNECT
MEMORY SUBSYSTEM
REPLY – DATA
4. Quantized memory requests. Since memory is shared by several processors, requests to access it are optimized by being quantized into packets. A processor controls the bus just long enough to make a request and receives a reply only if the request was to read.



5. Many bus structures. Illustrated is the minimal bus structure consistent with the iAPX 432 architecture. Many other configurations are possible as long as they adhere to the quantized packet protocol of the 432. For example, a faster bus might use wider data paths.

important clues to the functional power of its microarchitecture: only 6% of the total microprogram is required to implement the basic instruction set. The low percentage is due to the close match between the basic instruction set and the microinstructions. Many instructions can be executed by just a single microinstruction. These microinstructions emerge directly from the instruction decoder located on the 43201 and therefore do not take up space in the main microprogram ROM.

Virtual addresses

Virtual addressing, another important 432 function, uses only 7% of microprogram space on the 43201. This percentage is kept small by a little extra hardware, in the form of back-up copies of certain key registers. When a requested memory segment is not in RAM, microcode can restore the machine to the state it was in when the requesting instruction started. System software can then bring the missing segment in from secondary storage and execute the instruction afresh. The entire operation is transparent to the executing program.

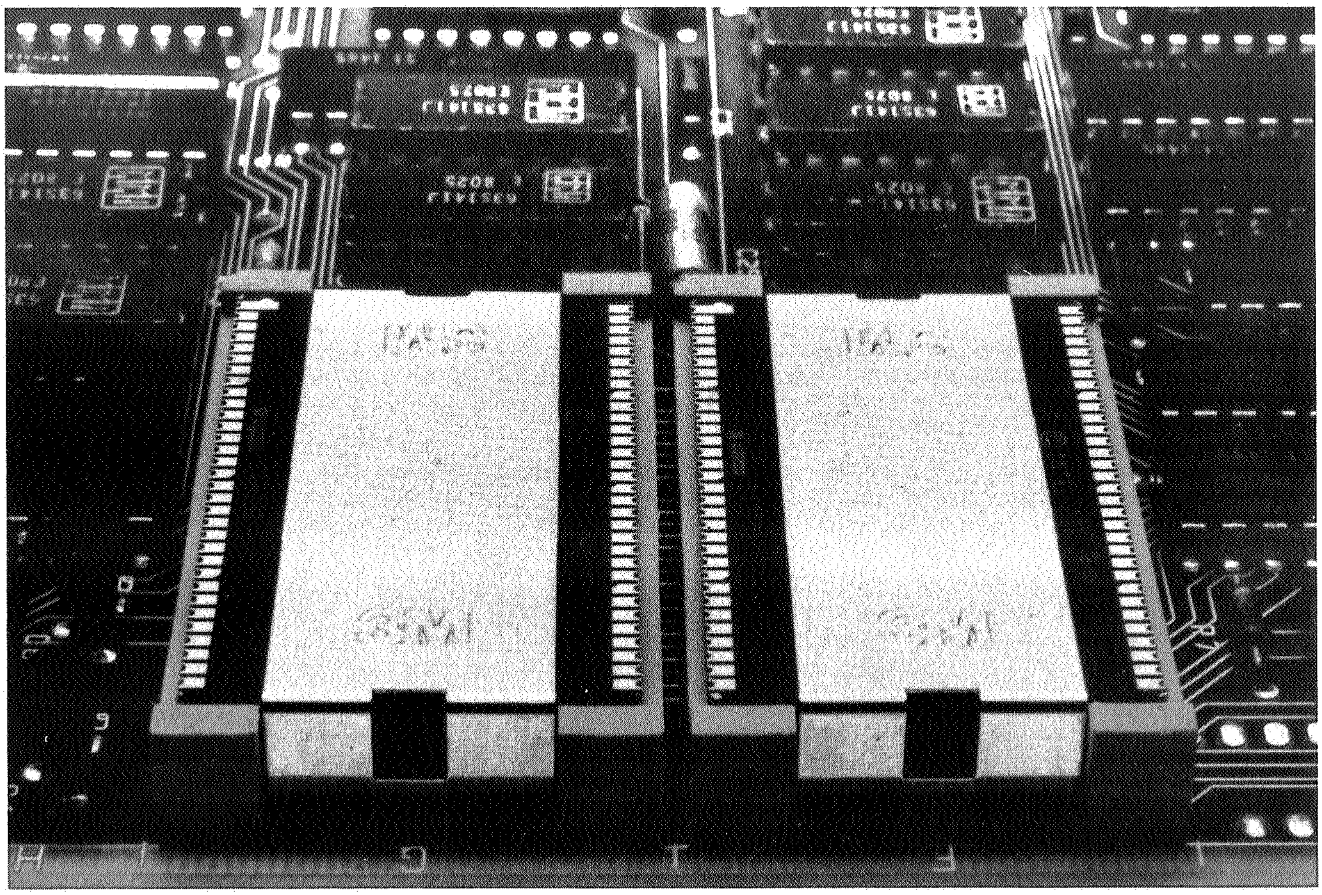
Table 2 also shows that a large amount of microcode is devoted to the silicon operating system. Many of the high-level 432 instructions, such as send message, are included in this total. Equivalent functions, programmed in the instruction set of a typical microprocessor, would take four to eight times as many bits.

All 432 components are designed to operate in two modes so that highly fault-sensitive systems can be built from them. In the master mode, a component operates normally. But in the checker mode—a feature never before found in a microprocessor—all the pins that would normally operate as outputs reverse themselves to function in a special input mode. Instead of asserting output data, they sample the states of their signal lines. The sampled data is compared internally, by an exclusive-OR gate built into each output stage, with the data that would have been asserted in master mode. A mismatch on any pin indicates an error.

A fault-sensitive unit is formed, as shown in Fig. 7, by simply wiring together two identical 432 components. One chip is placed in the master mode and the other in the checker mode by asserting the checker-mode pin. Any error signal asserted by the checker is routed to a special input on the master. In operation, the master and checker stay in lock-step synchrony. Any disagreement apparent at the output pins of the master is flagged immediately by the checker and freezes the operation of both units.

Longer and shorter

The instruction formats are designed to simplify and to reduce the size of code. Consequently, instructions vary in length and have from zero- to three-operand references. The operand addressing modes are modeled after the structures found in high-level languages like Ada to support scalar, vector, and record data types. These correspond roughly to the base-plus-displacement,


6. Quad in-line package. To house the 432 devices, a more reliable package was developed. The QUIP combines a leadless chip-carrier with a socket that has four staggered rows of pins on 100-mil centers. A metal clip helps dissipate heat, and test points are easily accessible.

base-plus-index, and base-plus-displacement-plus-index addressing modes, respectively, found in conventional machines. Instructions also never refer to a register, since registers can be hard to manage during compilation. Instead, operands may come either from memory or from the hardware-supported expression-evaluation stack. Any mix of memory or stack-based operands is allowed. Lastly, the instructions may start and end on any bit boundaries. Naturally, branch instructions are designed to branch to a bit location, too.

With instruction formats such as these, the most frequent statements of a high-level language like Ada compile to single 432 instructions. Some examples are shown in Fig. 8, along with the corresponding instruction lengths (in bits).

An object orientation

The 432 has an object-oriented architecture. Objects provide an identical framework for everything from a simple piece of data, like a byte, to a message being sent to another processor. They are responsible for most of the facilities found in the 432 architecture, including basic computation, language run-time environment, resource management, interprocess communication, and protected addressing.

A data object supports basic computation. It is simply a linear, logical address space from 1-K to 64-K bytes in length. Any type of binary data can be stored in a data object, and a given element within the object is accessed by specifying its byte displacement from the starting address of the object. The complete logical address of a single data item, as found in the operand fields of a typical instruction, contains both the displacement and the program's local name (or nickname) for the data object. This short, local object name selects the much longer access descriptor, which indicates the location of the object's full name (or absolute address). The local name often runs as few as 6 bits in an operand reference.

More than ordinary

Just as data objects support basic computation in the 432, more complex objects are used to support higher-level functions. The hardware knows from their descriptor-type code that they contain more than just ordinary

TABLE 1: EXECUTION TIMES OF THE iAPX 432 32-BIT MICROPROCESSOR

Functional unit	Typical operation	Execution time at 8 MHz (μs)
Variable-precision integer arithmetic unit	32-bit integer multiply	6.25 (16 μs on IBM 370/148)
Microprogrammed floating-point arithmetic unit	80-bit floating multiply	26.125 (38.5 μs on IBM 370/148)
Barrel-shift unit	32-bit field extract	1.875
Address generator with associative cache of least recently used addresses	32-bit memory access	0.75
Silicon operating system	send message	80.875

TABLE 2: ALLOCATION OF MICROCODE IN THE iAPX-43201/2 DATA MICROPROCESSOR

Function	Bits	Percentage
Basic instruction set	3,680	6
Floating-point arithmetic	11,680	18
Run-time environment	6,400	10
Virtual addressing	4,800	7
Fault handling	2,640	4
Silicon operating system	26,400	40
Multiprocessor control	8,640	13
Debug services	1,280	2
Total	64-K	100

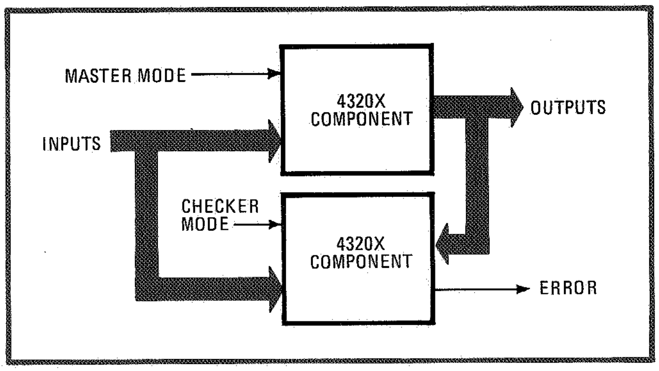
data and uses that knowledge to implement many functions carried out by software on conventional machines. The hardware-recognized objects are referred to generically as system objects.

System objects include domain objects and context objects. Both of these primarily support the run-time environments of high-level languages.

A domain object represents the addressing environment of a program module. Contained within the domain object are all the access descriptors for both the module's instruction objects and its data objects. The domain also contains links to other domains and thus is part of a network of domains representing a completely linked but still modular program.

A domain is actually composed of two parts—public and private—that are analogous to the interface and body portions of an Ada package [*Electronics*, Feb. 10, 1981, p. 127]. The public part contains the links to the objects defined by a module's interface specification, while the links to other objects not defined in the interface but used to implement the module are located in the private part. Only objects whose links are found in the public part of a domain object can be accessed from other connected domains.

Context objects support the dynamic allocation of memory every time they are activated. A context is created dynamically when a procedure is called and is deleted dynamically when the procedure returns. Since a new context is created for every activation, contexts directly support shared, recursive, and re-entrant procedures. Their second major function is to provide each procedure activation with a data object for its local data and an operand stack for expression evaluation.



7. Error-checking mode. Each 432 chip has a checker mode as well as a master mode. Two devices can be wired in parallel and one put in the checker mode to duplicate all the operations of the master and signal an error when it detects a discrepancy.

The remaining 432 system objects support the hardware-based operating system services called the silicon operating system. A data structure representing an individual GDP is called a processor object. There is one processor object for each physical GDP in a 432 system.

Some more objects

An object representing an independent concurrent program or task is called a process object. Processes may be scheduled to run on a processor and thus represent a claim on some part of the system's total processing resources. A process object contains, among other things, the priority of that process.

An object representing a portion of the allocatable or free storage in the system is called a storage resource object. Many such objects may exist in a system to partition storage in accordance with claims and grants. Through storage resource objects, new objects are created dynamically for software by the hardware.

A very flexible object that is used to support the buffered transmission of messages between processes, or programs, is called a port object. Port objects also support the scheduling and dispatching of processes on multiple processors in a multiprogrammed fashion. Ports are able to serve both functions because scheduling and dispatching is modeled as sending a message (the process object) to a process (the processor). In practice, the message is nothing more than an access descriptor. Since an access descriptor can reference any object, the send instruction can be used to send any object and, hence, any complex message. A message might be a data object containing a string of text or a complex object including executable code and perhaps representing an important system resource. Objects, therefore, present a consistent framework in which both processes and processors may communicate conveniently.

Object addressing

Objects are stored in pieces of the address space called segments. Simple objects can be stored by a single segment, but complex ones may occupy many of them. Important information about each object, including its type and location in physical memory, is found in its object descriptor. An object is always addressed via this descriptor, the location of which is indicated by an access descriptor (Fig. 9).

Length information is used to protect the object from out-of-range addresses, and presence information, along with other data in the object descriptor, is used to implement virtual memory. To simplify storage management, all object descriptors are grouped in a central table. Naturally, the object table is an object, too, and its object descriptor is contained within itself.

To select or refer to an object requires a 32-bit access descriptor that contains the identity of the object it references. Each access descriptor also contains other information to help control access to the object to which it refers. Different access rights can be represented by different access descriptors for the same object. This fact

A history of the Aloha project

The iAPX 432 32-bit microprocessor has been in gestation for over six years, a third of that time in Santa Clara, Calif., and the remainder in Aloha, Ore. There its development eventually became known as the special systems operation, or SSO, with Jean-Claude Cornet as director. But its shroud of privacy led some to think SSO stood for "secret systems operation."

In the beginning the 432 was called the 8816, then the 8800. It had to be given a number because at Intel, "as soon as you give something a number, it is instantly perceived as this little thing with side-brazed connections coming out of it," jokes principal engineer Justin R. Rattner (see photo). By November of 1975 the endeavor had coalesced into a working unit under William W. Lattin. He remained 432 program manager until April of last year, when he moved over to another Intel division.

The original idea was to "do something interesting" with very large-scale integration, but with Schottky TTL performance. The team looked at and then discarded a double-diffused MOS process and a modified charge-coupled-device structure, before it finally came up with a short-channel technology that could squeeze 100,000 transistors onto a single chip yet still support future geometry reductions. It was the birth of the high-performance MOS, or H-MOS, process.

After some preliminary design work, the group presented prospective users with a specification. "They were responsible for our emphasis on multiprocessing," says Rattner. "It was the No. 1 thing they wanted." Back then, the notion of software objects as a way of simplifying programming barely existed. "But suggested mechanisms began to get very ad hoc," he adds. "Every new feature seemed to involve a different machine facility, a different hardware unit."

Early one Saturday morning Rattner woke up convinced software objects were the solution. "I wrote for about six hours and called George Cox [a staff scientist], and that Sunday morning we met at Intel to start working out the details," he recalls.

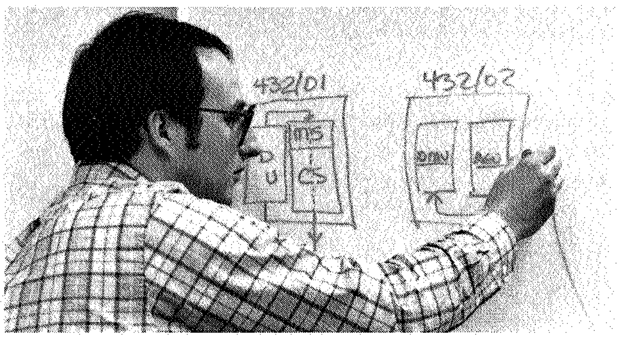
Then came the hardware design, which "made extensive use of regular logic cells and wiring topologies, very much along the lines of the work done at Cal Tech" by Carver Mead, who had begun consulting on the project in 1975. "We knew we couldn't just randomly wire 100,000 transistors," so computer-aided design and simulation tools had to be designed.

Portions of the chips' architecture were implemented directly in silicon—so intimately that they have no logic-gate equivalent. "We looked at each function, but instead of drawing a logic diagram and figuring out a gate implementation, we asked if there was a way to do it with MOS transistors directly," says Rattner. For the processor's control store, the team adapted a read-only memory cell designed for the 8087 numeric processor that stores 2 bits [*Electronics*, Oct. 9, 1980, p. 39]. Without tricks like the ROM cell, "the chips couldn't have been built."

Rattner feels the modular design methodology was "a spectacular success." At least one of the chips, the complex execution unit, "could have been shipped in sample quantities the first day out of fab," he says, adding that all in all, "productivity was five, six, seven times that of some other Intel projects."

The SSO now employs over 100 engineers designing follow-on board computers and design aids for the 432 family. Many of them have worked with minicomputer or larger machines in the past, as also has marketing manager Dave Best.

The project "cost a bundle," says Best, adding that "it was the largest investment in a single program that Intel has ever made—larger even than the magnetic stuff," the bubble memories. **-John G. Posa**



is the basis of the 432's need-to-know protection system: in order to refer to an object, a program must contain an access descriptor for it; a program may only access an object according to privilege rights encoded in the access descriptor it holds.

Access descriptors are found only in a special type of segment called an access segment. This protects the integrity of access descriptors by preventing them from being treated as ordinary data. Only certain instructions are permitted to move or manipulate access descriptors.

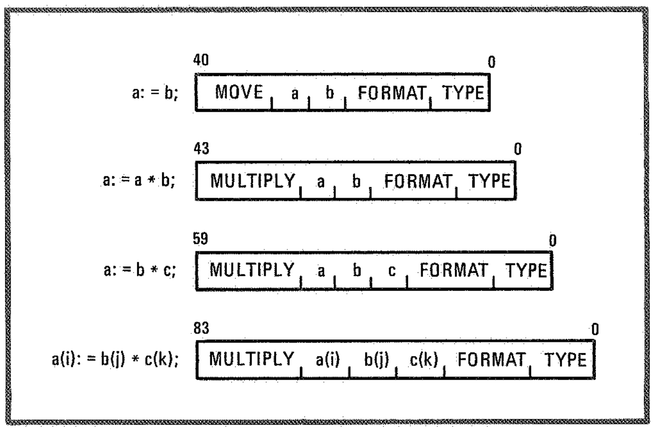
Selective entry

An access descriptor selects one of the 2^{24} entries in the object table, and each entry can specify a single-segment object of up to 2^{16} bytes. That gives a system-wide logical address space of 2^{40}—1 trillion—bytes of information. At any instant, however, the logical addressing environment of a program is restricted to 2^{16} objects of up to 2^{16} bytes, or 4 gigabytes. The instantaneous addressing environment is represented by four access segments, each of which is limited to 2^{14} access descriptors.

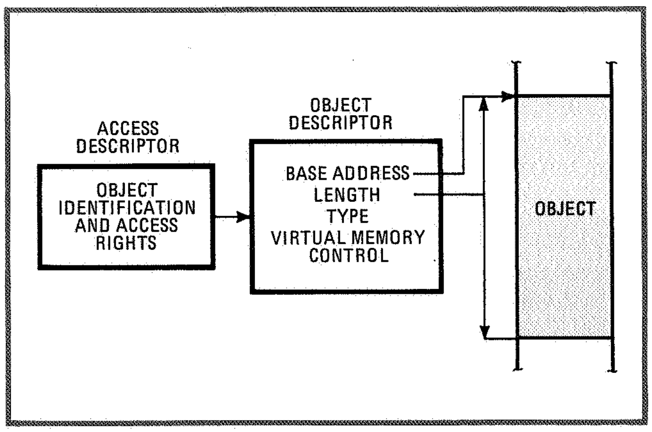
To implement the 432's two-level addressing architecture efficiently, each processor contains a buffer or cache of the most recently used object addresses. Cache data made stale by a software alteration of an object descriptor (which is a relatively infrequent occurrence) can be flushed by the operating system through interprocessor communication.

The key computing resources of the 432, unlike conventional systems, are controlled by hardware-defined system objects rather than by user-supplied software. This difference dramatically changes the way resources are managed by, and ultimately the structure of, the entire system.

For each type of system object, the hardware automatically handles some part of the operations that can be carried out on an object. Some of the operations are



8. Variable-size instructions. Since users are not expected to program the 432 in assembly language, its instructions are not aligned on convenient byte boundaries but instead can be any length of bits. The statements illustrated compile to single 432 instructions.



9. Address calculations. All memory requests are calculated via a two-level operation in which an access descriptor points to an object descriptor and the object descriptor in turn indicates the location and size of the desired memory contents (called an object).

available as instructions, while others are involved only when the hardware determines, independently of any particular instruction, that the operation is needed, such as fetching an absent memory segment. Software is responsible for providing the remainder of the operations defining the complete interface to the object.

This organization presents an important hardware-software tradeoff, involving just which operations should be put into hardware and which would be better left to software. In the 432, the decision to put an operation in hardware is based on one of three factors. First, the timing of an operation may be critical, affecting overall system performance in a fundamental way. Second, an operation may be security-critical, affecting the integrity of the protection system and the isolation of important information. Third, its reliability may be critical, affecting the ability of the system to function correctly in delicate programming situations.

The software part of the object management function is still by no means trivial. Software is largely responsible for creating new objects and disposing of old ones.

Generally speaking, objects work to remove the traditional barriers between the operating system and the application environment. The packages that make up iMAX are tools with which the user may build an application. If the iMAX package for a particular service is not quite right, the user is free to replace it with a package of different design.

Ada: implementation language

Ada, the Department of Defense's new standard programming language, is an ideal systems implementation language for the 432. The goals established for Ada's design were very like those set for the 432 since the language's designers drew upon the same body of research. The ultimate goals of both Ada and the 432 are increased programmer productivity, increased software reliability, and low software life-cycle costs.

Ada is inspired by the Pascal language and has much in common with it. But it differs from Pascal, and moves far beyond the older high-level language, in its support for large-scale, modular software. While several attempts have been made to repair this deficiency in Pascal, none has been widely accepted or used. Ada makes these efforts obsolete.

Through its "package" construct, Ada provides a natural way to put together large programs based on object-oriented modularization. A package defines an object and the operations that can be done on it. Following the object-oriented view further, a package restricts access to an object as specified in a separate interface portion and thus succeeds in hiding the details of its implementation.

Ada turns out to be an ideal language for the 432 not only because they embody the same idea of modularization but because many Ada constructs map directly onto the hardware. For example, the Ada package construct is directly supported in the 432 architecture by the concept of a domain object and Ada subprogram activations become contexts in the 432 architecture.

In those cases where the hardware of the 432 goes beyond Ada's built-in constructs, Ada's definition allows for a special machine-access package. Any 432 instruction or feature can be accessed via this package, eliminating the need for a 432 assembler. Ada is thus the only language used to write the iMAX executive and programmers are not ordinarily expected to require machine access. Hence even systems programmers will use Ada, resorting to direct machine access only rarely.

Ada and the 432 architecture cooperate to provide complementary checks on a program's design. Ada checks data types and interfaces during compilation, and the 432 subsequently rechecks them at run time in order to catch errors missed or possibly caused by the compiler. The 432 architecture also provides those checks on interfaces and data types that in Ada can only be made during execution.

Finally, Ada is the basis of the 432's integrated programming system. This software development system is built around Ada to provide separate compilation of programs with fully checked interfaces as well as link-time checking of module version numbers. The latter capability ensures that old versions of programs will not sneak back into a system. A symbolic source-level debugger lets the application programmer debug programs in Ada rather than 432 machine language. □

intel®

PRELIMINARY

iAPX 43203 VLSI INTERFACE PROCESSOR

- **Fully Independent and Decentralized I/O**
- **Buffered Data Path for High-Speed Burst-Mode Transfers**
- **Initialization/Diagnostic Interface to 432 Systems**
- **Multiple 43203's per System Provide Incremental I/O Capacity**
- **Protected I/O Interface to 432 Memory**
- **Silicon Operating System Instruction Set Extensions for Attached iAPX Processors**
- **Multibus™ System Compatible Interface**
- **Functional Redundancy Checking Mode for Hardware Error Detection**

The Intel 43203 Interface Processor (IP) provides I/O facilities in iAPX 432 micromainframe systems employing peripheral subsystems. An IP maps a portion of the peripheral subsystem address space into iAPX 432 system memory. As any iAPX 432 processor, the IP operates in an object-oriented, capability-based, 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). Refer to Figure 1 for the QUIP representation of the 43203 pin configuration.

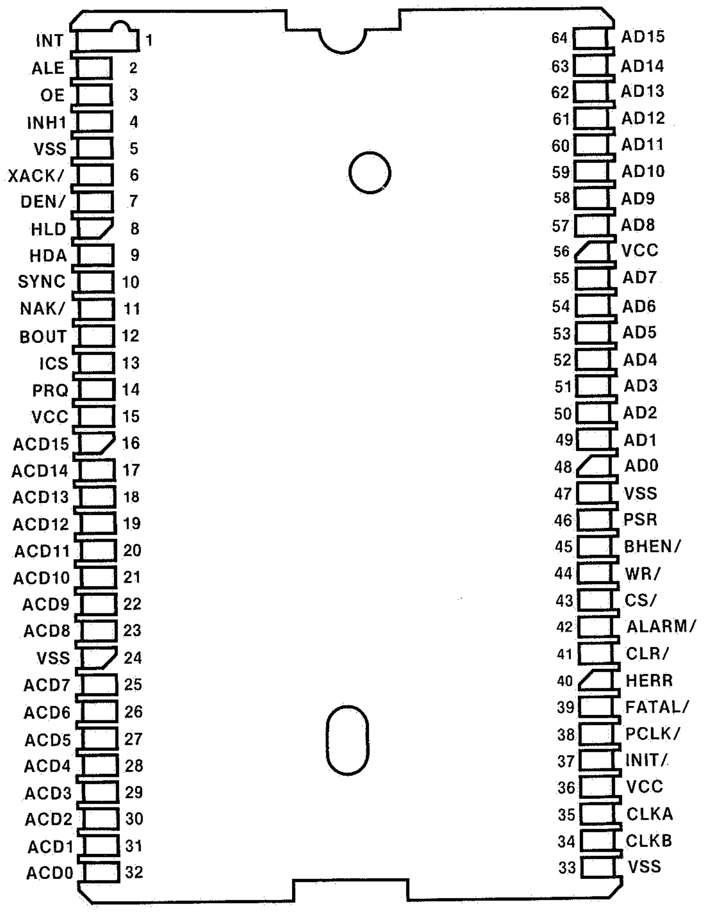


Figure 1. iAPX 43203 Interface Processor Pin Configuration

FUNCTIONAL DESCRIPTION

The block diagram shown in Figure 2 illustrates the internal architecture of the Interface Processor. Figure 3 represents the Interface Processor as a logical device and illustrates the signal interface to the Processor Packet bus (left side) and the peripheral subsystem (right side). The Interface Processor (IP) operates in conjunction with an Attached Processor (AP) to form the logical I/O processor of an iAPX 432 system.

The IP acts as a slave to the AP, and maps 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.

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 4 illustrates a hypothetical configuration 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).

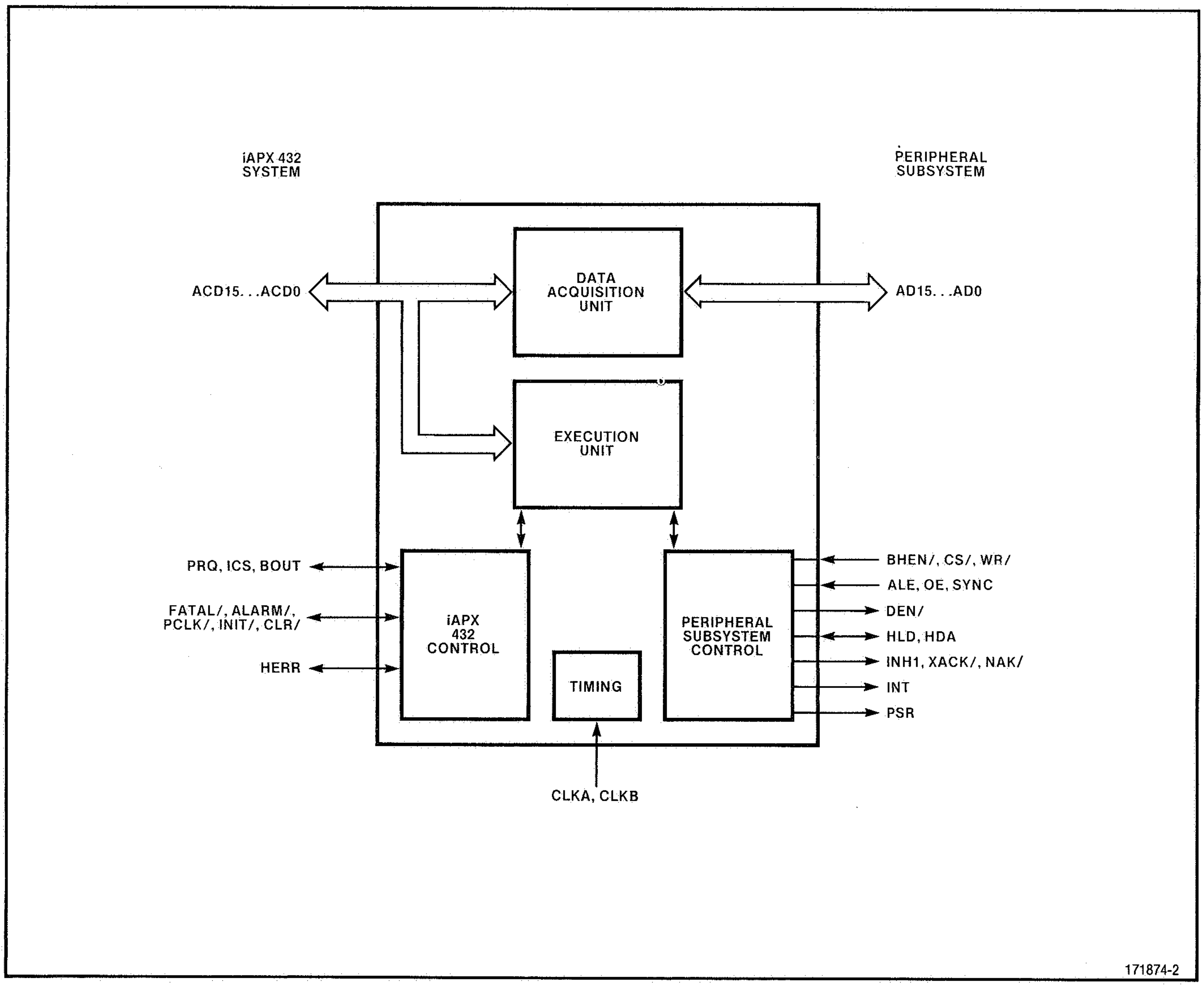


Figure 2. iAPX 43203 IP Functional Block Diagram

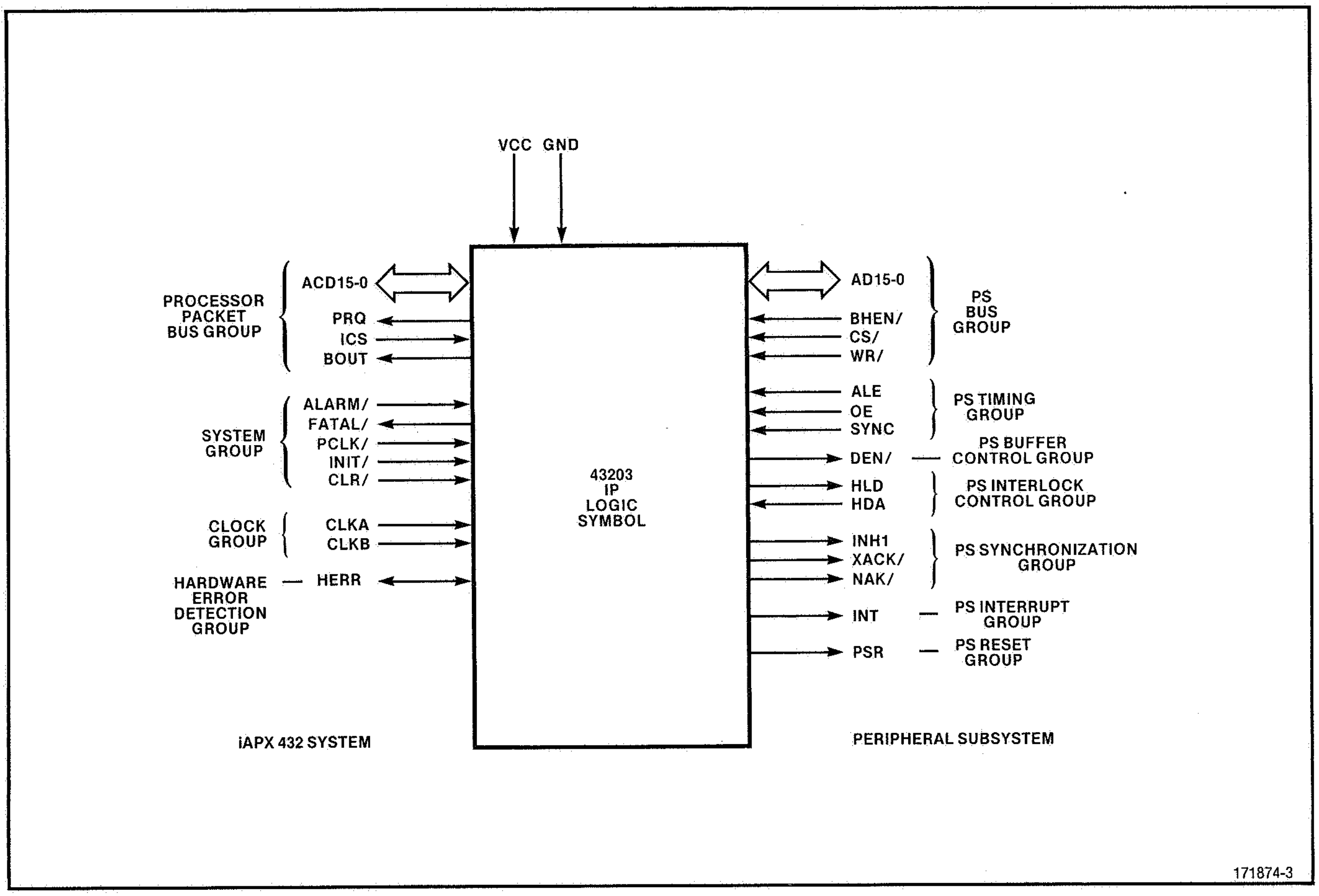


Figure 3. iAPX 43203 IP Logic Symbol

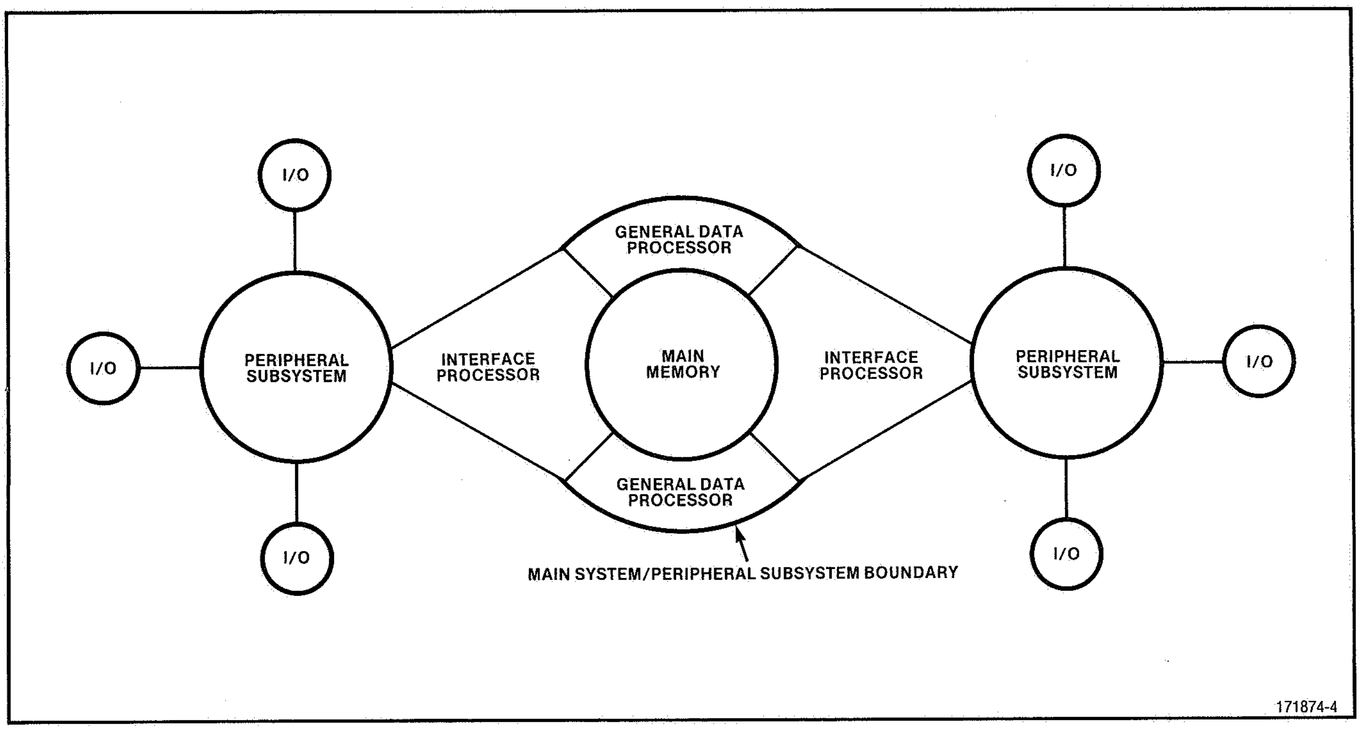


Figure 4. Main System and Peripheral Subsystem

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.

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 Processor Packet bus and the other is a very general interface.

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 5 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 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 6).

iAPX 432 SYSTEM INTERFACE

The IP exists in both the protected environment of the iAPX 432, and the conventional environment of the peripheral 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 attached processor.

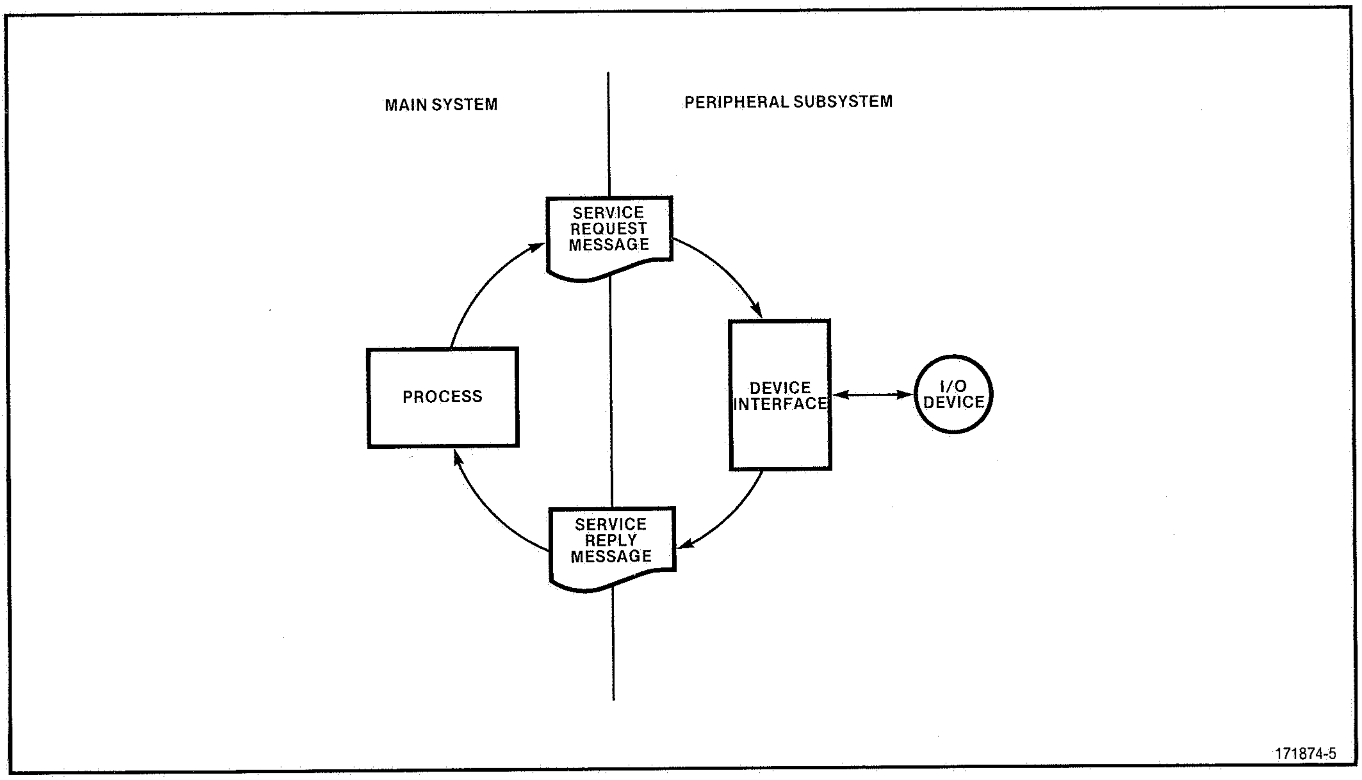


Figure 5. Basic IO Service Cycle

From the iAPX 432 side, an IP looks and behaves very much like any other processor. It attaches to the Processor Packet bus 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 principal 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 attached 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 6). 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 7). 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.

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.

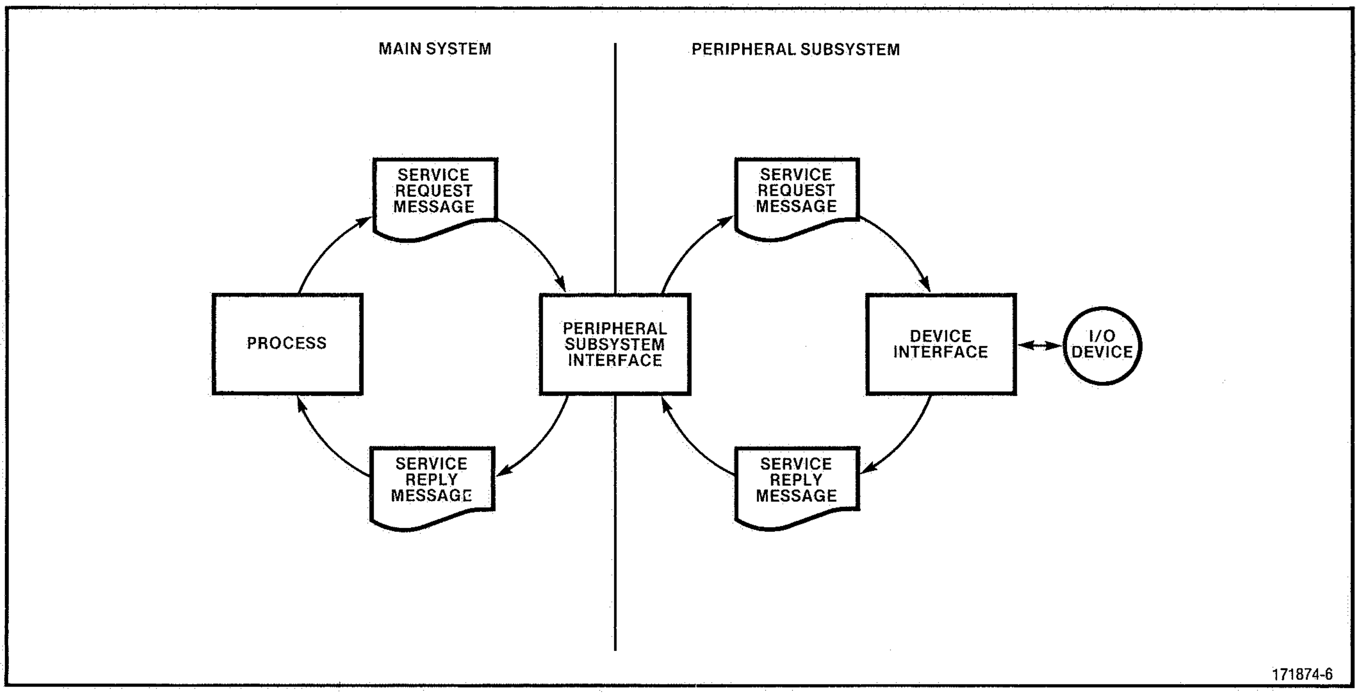


Figure 6. Peripheral Subsystem Interface

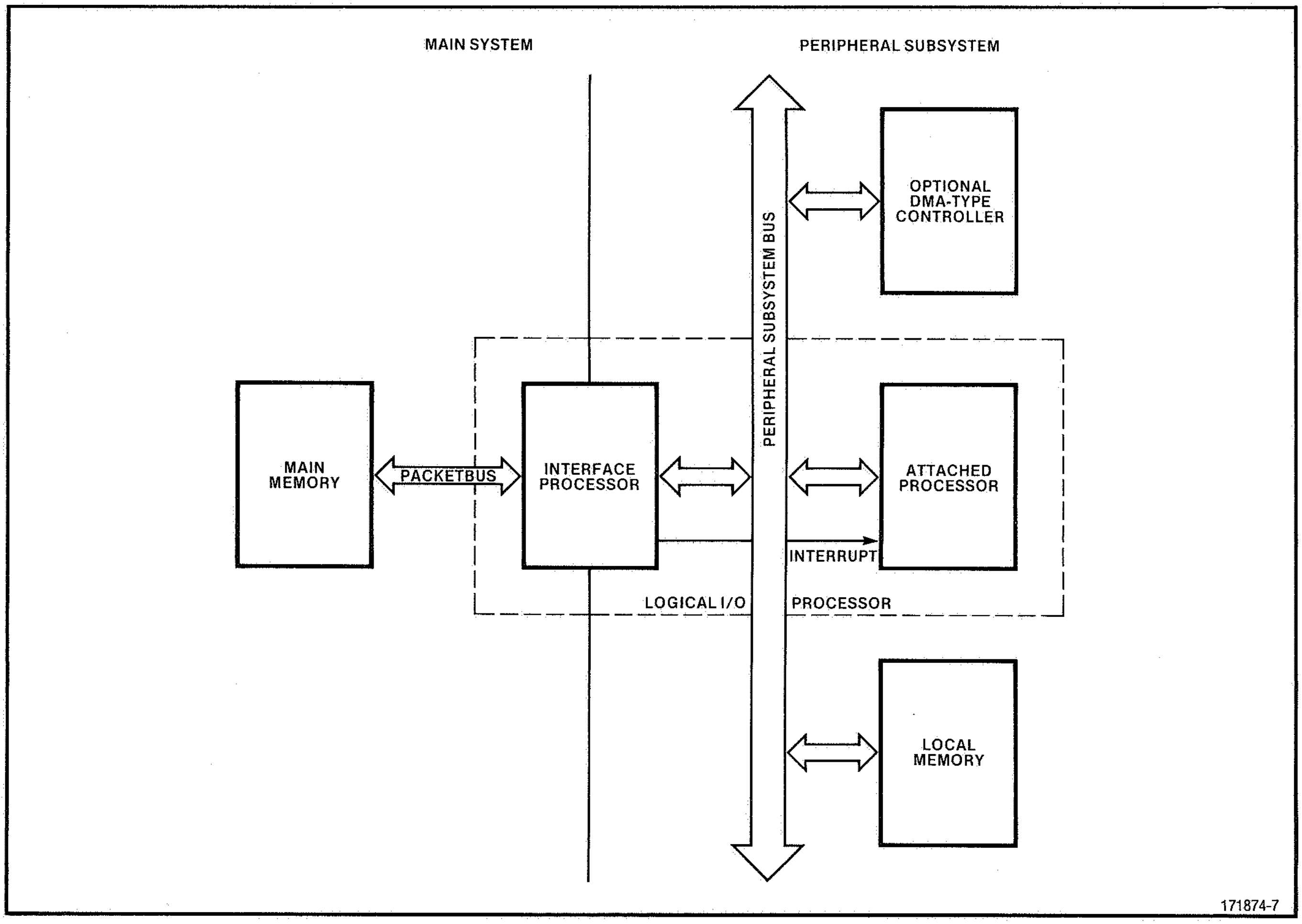


Figure 7. Peripheral Subsystem Interface Hardware

As Figure 7 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.

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 7 shows, the IP presents both a peripheral subsystem bus interface and a standard iAPX 432 Processor Packet bus 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.

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 an RMX-80 or iRMX-86 operating system) 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 8). 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.

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

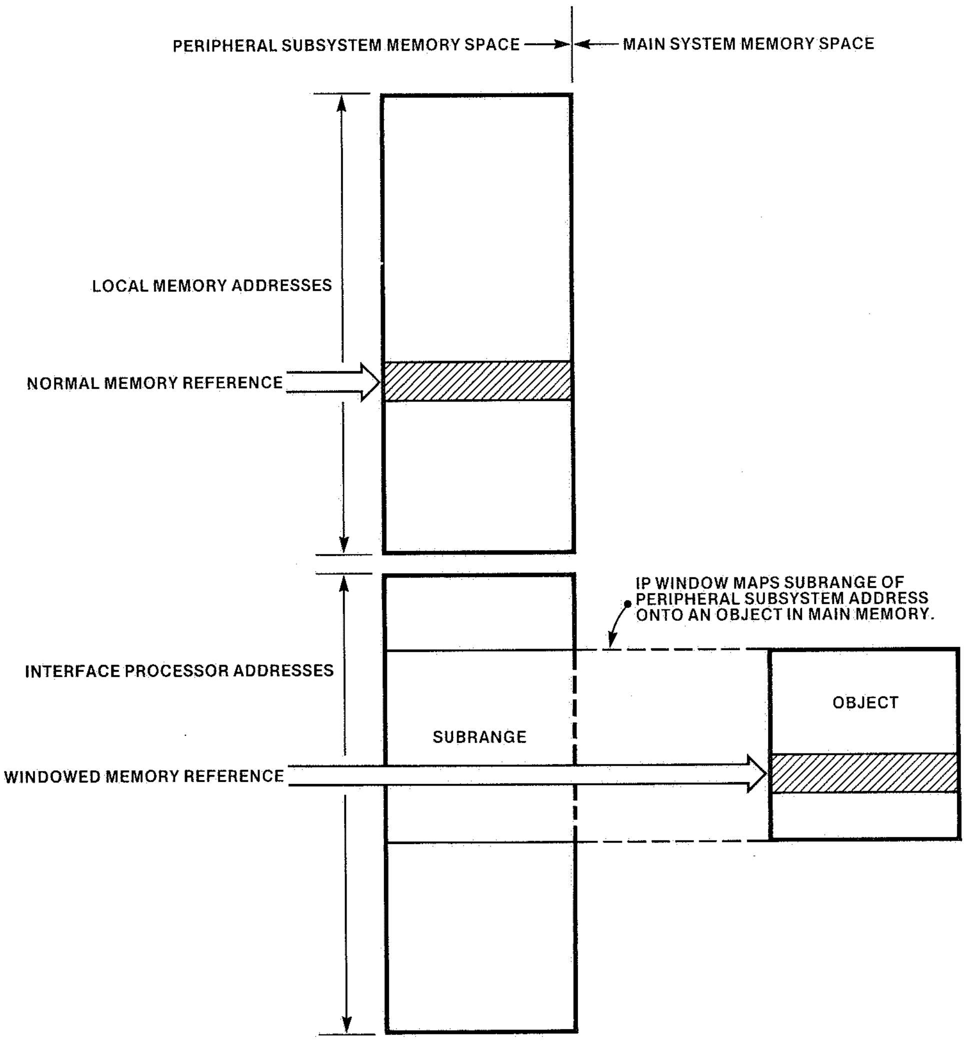

PERIPHERAL SUBSYSTEM MEMORY SPACE
MAIN SYSTEM MEMORY SPACE
LOCAL MEMORY ADDRESSES
NORMAL MEMORY REFERENCE
IP WINDOW MAPS SUBRANGE OF PERIPHERAL SUBSYSTEM ADDRESS ONTO AN OBJECT IN MAIN MEMORY.
INTERFACE PROCESSOR ADDRESSES
OBJECT
SUBRANGE
WINDOWED MEMORY REFERENCE
171874-8

Figure 8. Interface Processor Window

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:

- alter windows;
- exchange messages with GDP processes via the standard 432 communication facility;
- 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 address space. 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.

iAPX 432 INFORMATION STRUCTURE

The following information describes the requirements placed on the logical structure of the iAPX 432 hardware environment. These requirements are concerned directly with the constraints of physical memory, the type of data transferred, and the structure of the data types. These requirements are common to the iAPX 432 family of processors. (Any pin notations called out in this information are described in the 43203 Pin Description section of this data sheet).

Any 432 processor 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 maximum physical memory of 16 Megabytes.

Data Formats

When a processor executes the instructions of an operation within a context, operands found in the logical address space 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 operands 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 9.

DATA POSITIONING

The data operand types shown in Figure 9 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 (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 awaiting 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

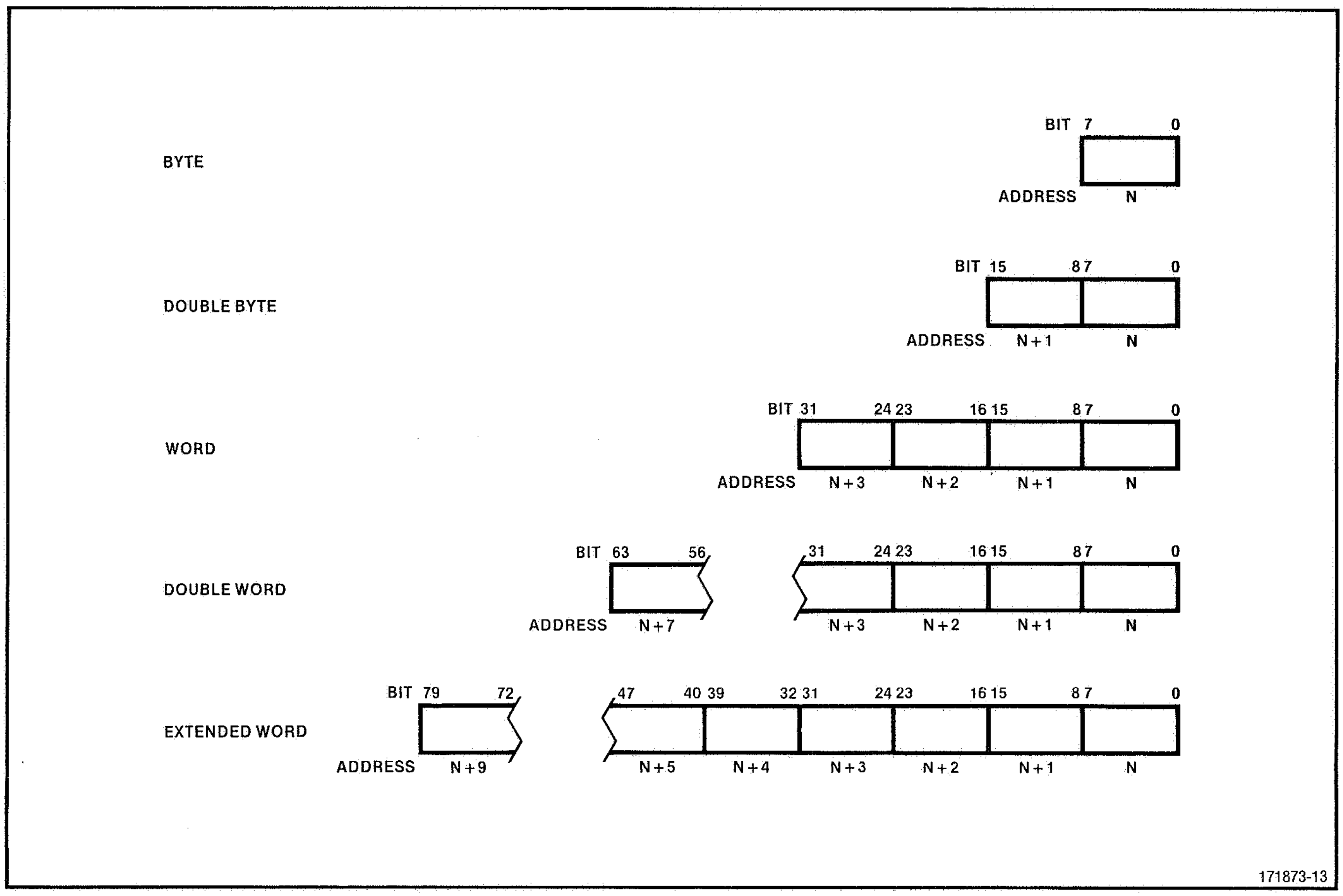


Figure 9. Basic iAPX 432 Data Lengths

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 occur, the memory system must ensure that the set of locations used to store the operand does not get changed to 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 errors by functional redundancy checking (FRC). At initialization time, each iAPX 432 processor is configured to operate as either a master or a checker processor (Figure 10). A master operates in the normal manner. A checker places all output pins that are being checked into a high-impedance state. Thus, those pins which are to be checked on a master and checker are parallel-connected, pin for pin, such that the checker is able to compare its master's output pin values with its own. Any comparison error causes the checker to assert HERR.

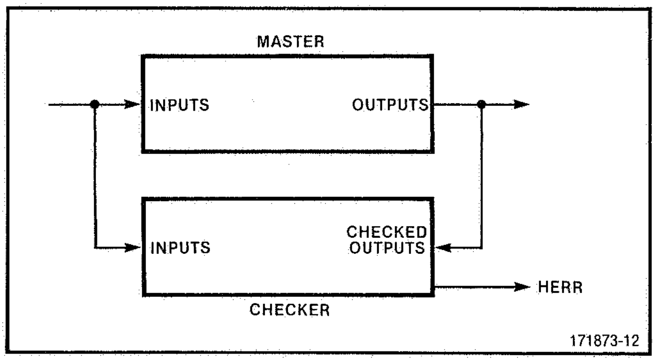


Figure 10. Hardware Error Detection

PROCESSOR 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. Slaves to the Processor Packet bus must support all state transitions to ensure compatibility.

The Processor Packet bus consists of 3 control lines:

- Processor Packet bus Request (PRQ),
- Enable Buffers for Output (BOUT),
- Interconnect Status (ICS).

This bus also includes sixteen 3-state Address-Control-Data lines (ACD15 through ACD0). PRQ has two functions whose use depends upon the application; i.e., PRQ either indicates the first cycle of a transaction on the Processor Packet 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 Packet bus transaction. The processor interprets the ICS input as an indication of one of the following:

- Whether or not an interprocessor communication (IPC) is waiting,
- Whether or not the slave requires more time to service the processor's request,
- 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-ACD0 (Address/Control/Data)

As shown in Figure 11, the first cycle, (T1 or Tvo) of a Processor Packet 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 (T2), 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 T2, the access is cancelled and the ACD lines are not defined.

During the third cycle (T3 or Tw) of a Processor Packet bus transaction the processor presents a high impedance to the ACD lines for read transactions and asserts write data for write transactions.

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 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. The processor may simply enter T3 or it may first enter Tw for any number of cycles (as dictated by ICS).

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 access). 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 access, the bus state moves to Tv (the ACD lines will be high impedance). 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 transaction (see Table 2). 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 or reconfiguration request has been received. An iAPX 432 processor is required to record and service only one IPC or reconfiguration request at a time. Logic in the interconnect system must record and sequence multiple (possibly simultaneous) IPC occurrences and reconfiguration requests to the processor. Thus the logic that forms ICS must accomodate global and local IPC arrivals and requests for reconfiguration as individual events:

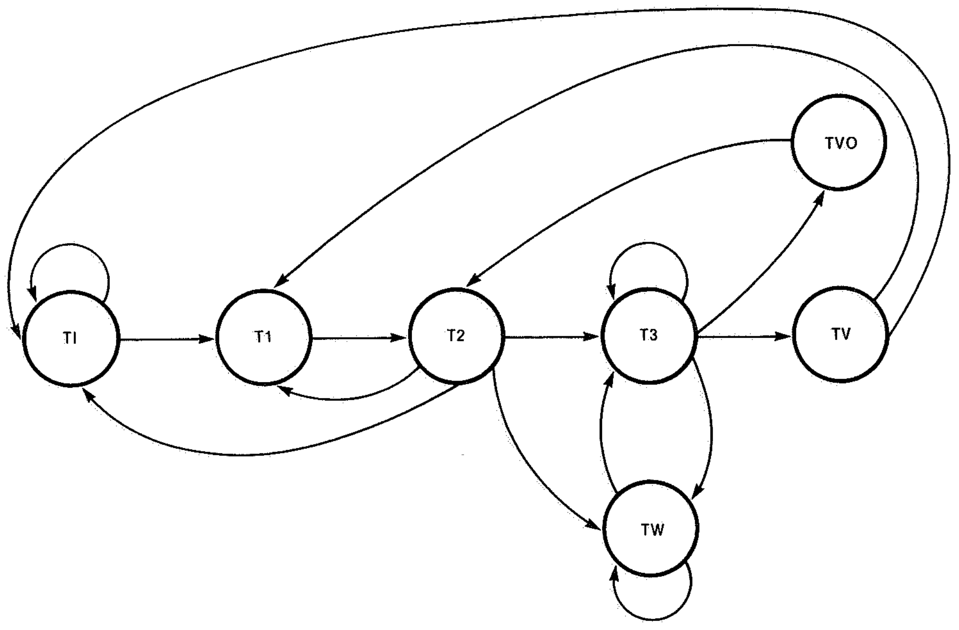
1. Assert IPC significance on ICS for the arrival of an IPC or reconfiguration request.

2. When the iAPX 432 processor reads interconnect address register 2, it will respond to one of the status bits for the IPC or reconfiguration request 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)



Initial State	Next State	Trigger
Ti	Tl	Bus cycle desired
	Ti	No bus cycle desired
Tl	T2	Unconditional
T2	T3	ICS high
	Tw	ICS low
	Tl	Cancelled, Access Pending
	Ti	Cancelled, No Access Pending
T3	T3	Additional transfer required, ICS high
	Tw	Additional transfer required, ICS low
	Tv	All transfers completed, no overlapped access
	Tvo	Current write with overlapped access
Tv	Ti	No access pending
	Tl	Access pending
Tvo	T2	Unconditional
Tw	Tw	ICS low
	T3	ICS high

Figure 11. Processor Packet Bus State Diagram

Table 1. ACD Specification Encoding

ACD 15	ACD 14	ACD 13	ACD 12 ACD 11 ACD 10	ACD 9 ACD 8
Access	Op	RMW	Length	Modifiers
0 - Memory 1 - Other	0 - Read 1 - Write	0 - Nominal 1 - RMW	000 - 1 Byte 001 - 2 Bytes 010 - 4 Bytes 011 - 6 Bytes 100 - 8 Bytes 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

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.

   The interconnect system must signal the second IPC by raising ICS high for at least one cycle and then setting ICS low for at least one cycle during IPC significance time.

Table 2. ICS Interpretation

	Level		State
	High	Low	
IPC	None	Waiting	Ti, T1, T2*
Stretch	Don't	Stretch	T3, Tw
Err	Bus Error	No Error	Tv, Tvo

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

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 Table 3.

Table 3. PRQ Interpretation

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

BOUT (Enable Buffers for Output)

BOUT is provided to control external buffers when they are present. Table 4 and Figures 12 through 16 show its state under various conditions.

Processor Packet Bus Timing Relationships

All timing relationships on the processor packet bus are derived from a simple scheme and related to Table 5. Each timing diagram shown in the following pages (Figures 12 through 17) 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. The solid lines in Figure 18 show the state transitions initiated by the IP.

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.

In all tranfers between a processor and a slave, the data to be driven are clocked three-quarters of a cycle before they are to be sampled. This allows adequate time for the transfer and ensures sufficient hold time after sampling. The BOUT timing is unique because BOUT is intended as a direction control for external buffers.

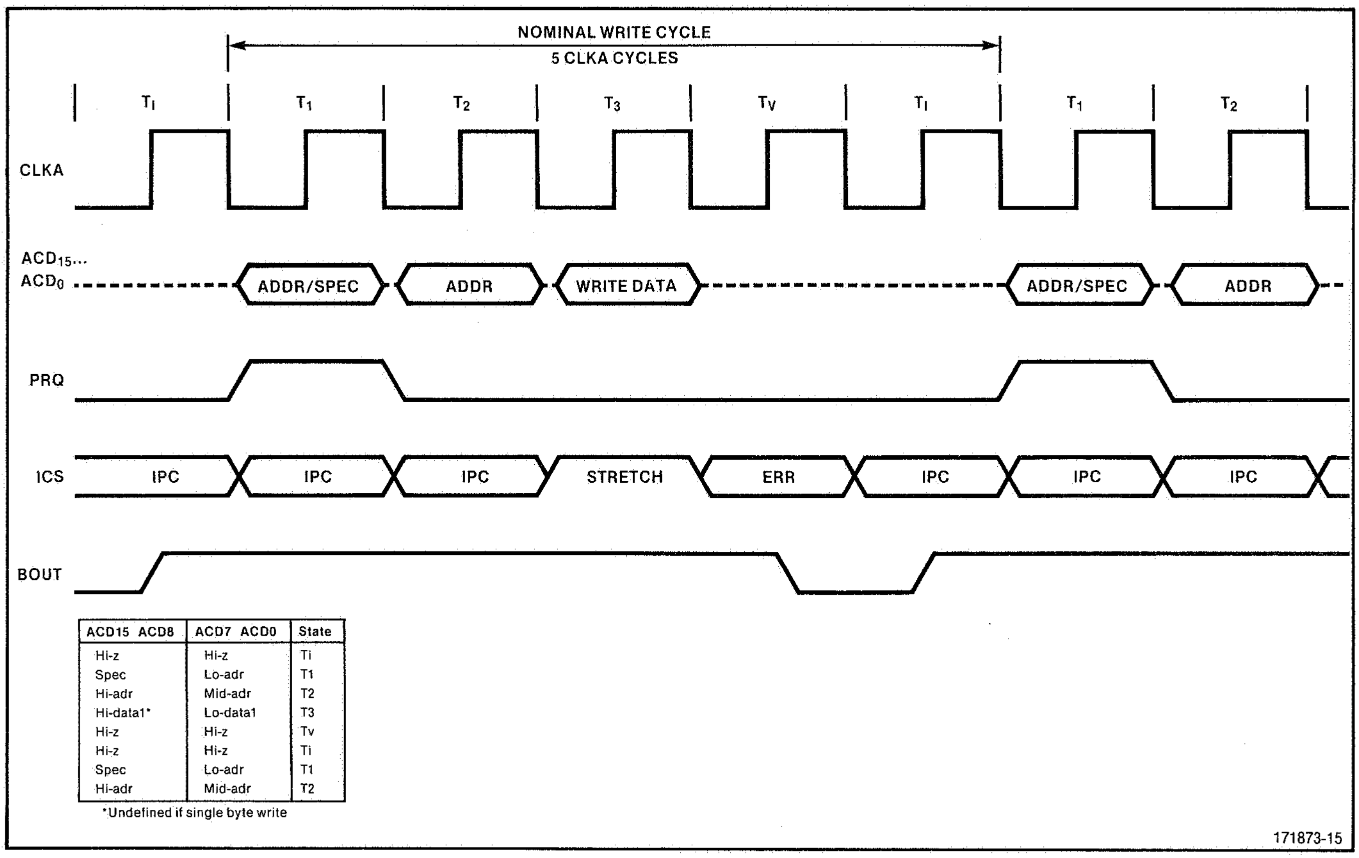
Detailed set-up and hold times depend on the processor implementation and can be found in the ac characteristics section.

Table 4. BOUT Interpretation

BOUT	Always High	Low-to-High Transition or Low	High-to-Low Transition or Low	High-to-Low Transition or High
Write	T1, T2, T3, Tw, Tvo	Ti	None	Tv
Read	T1, T2	Ti, Tv	T3, Tw	None

Table 5. iAPX 432 Component Signaling Scheme

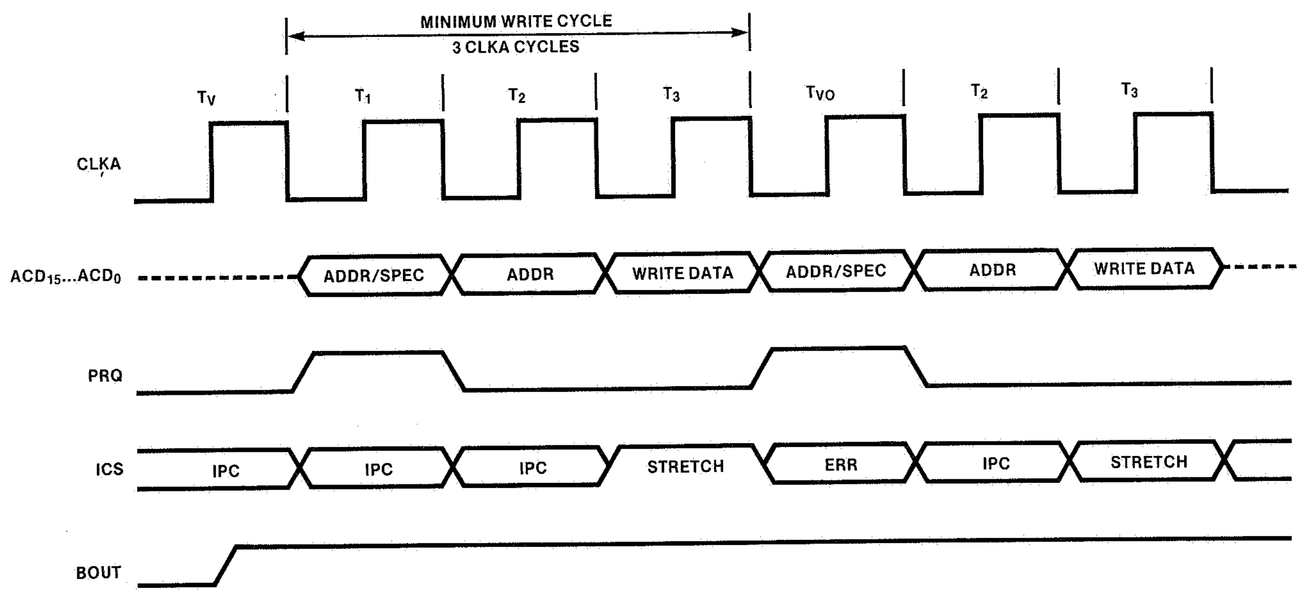
	Processor		Slave	
Inputs Sampled	ACD:	↓CLKA	All:	↑CLKB
	Others:	↑CLKA		
Outputs Driven	All (except BOUT):	↓CLKA	ACD:	↓CLKB
			Others:	↑CLKB
	BOUT:	↑CLKA		



ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data1*	Lo-data1	T3
Hi-z	Hi-z	Tv
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2

*Undefined if single byte write

Figure 12. Nominal Write Cycle Timing

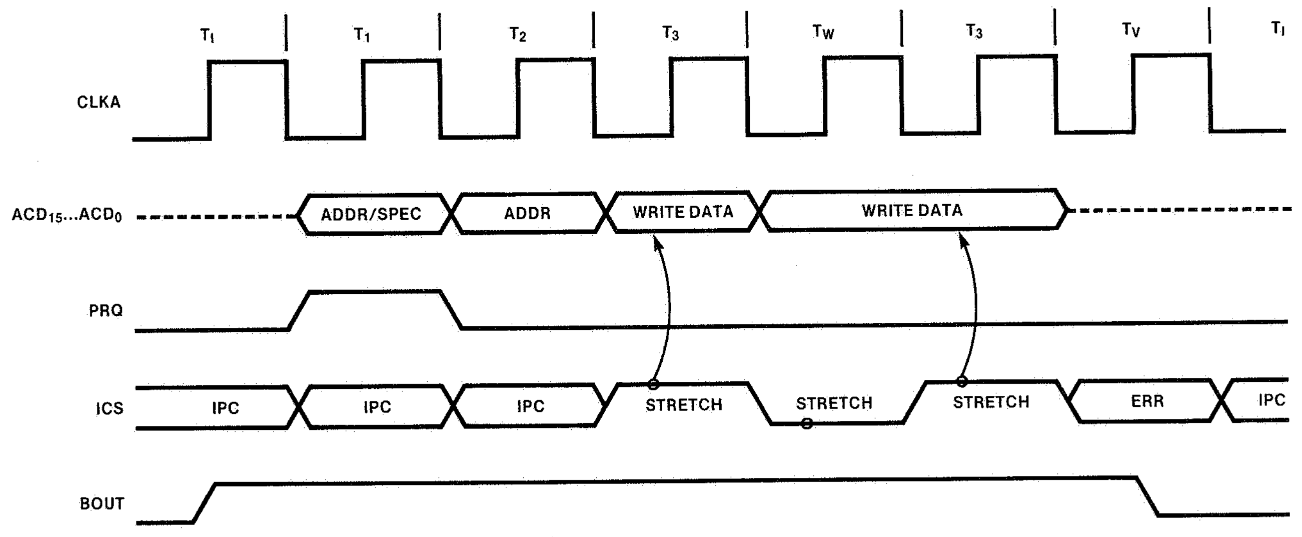


ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Tv**
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data1*	Lo-data1	T3
Spec	Lo-adr	Tvo
Hi-adr	Mid-adr	T2
Hi-data1	Lo-data1	T3

*Undefined if single byte write
**(Preceded by read cycle)

171873-16

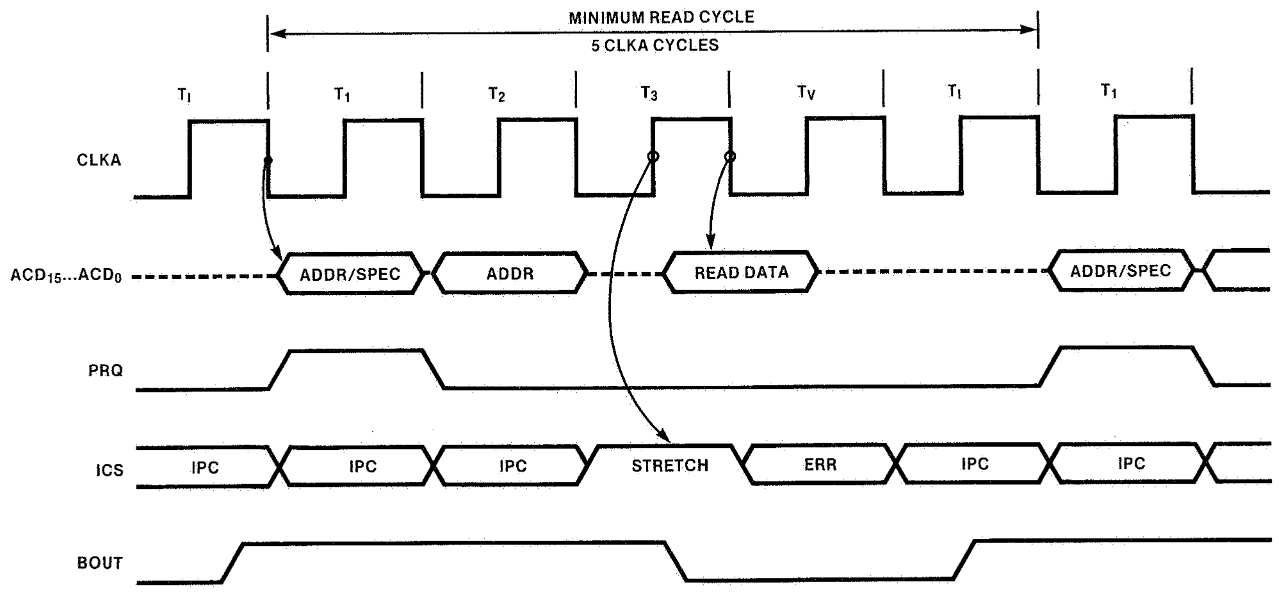
Figure 13. Minimum Write Cycle Timing



ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data1	Lo-data1	T3
Hi-data2	Lo-data2	Tw
Hi-data2	Lo-data2	T3
Hi-z	Hi-z	Tv
Hi-z	Hi-z	Ti

171873-17

Figure 14. Stretched Write Cycle Timing

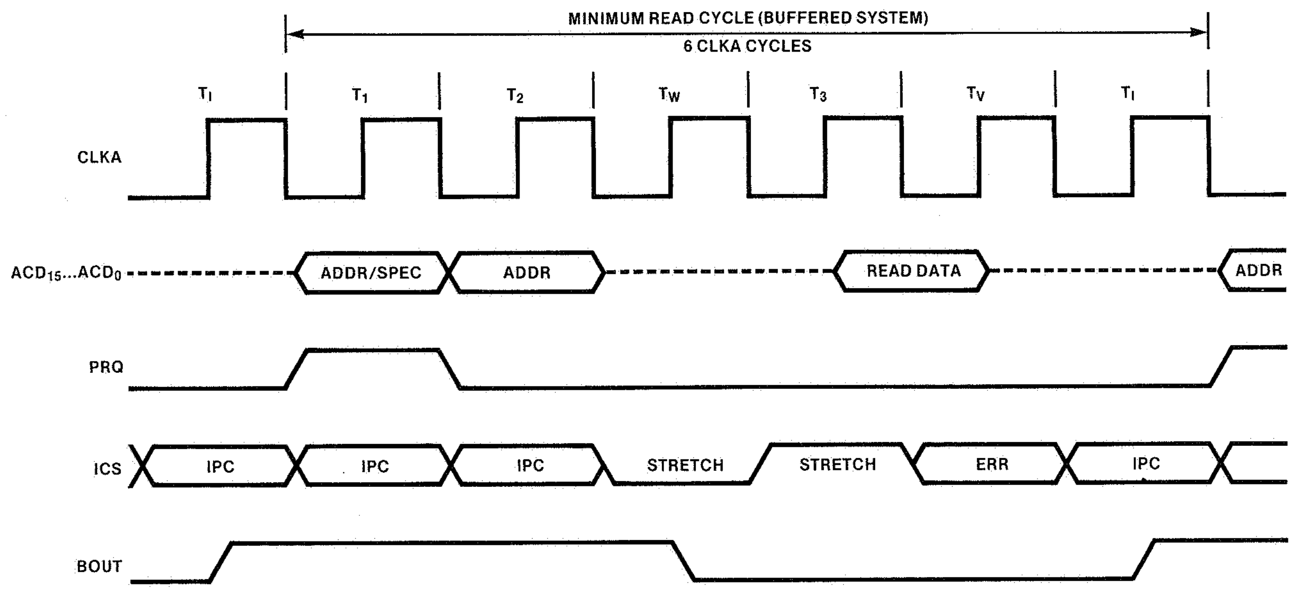


ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data*	Lo-data1	T3
Hi-z	Hi-z	Tv
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1

*Undefined if single byte read

171873-18

Figure 15. Minimum Read Cycle (Not Buffered)

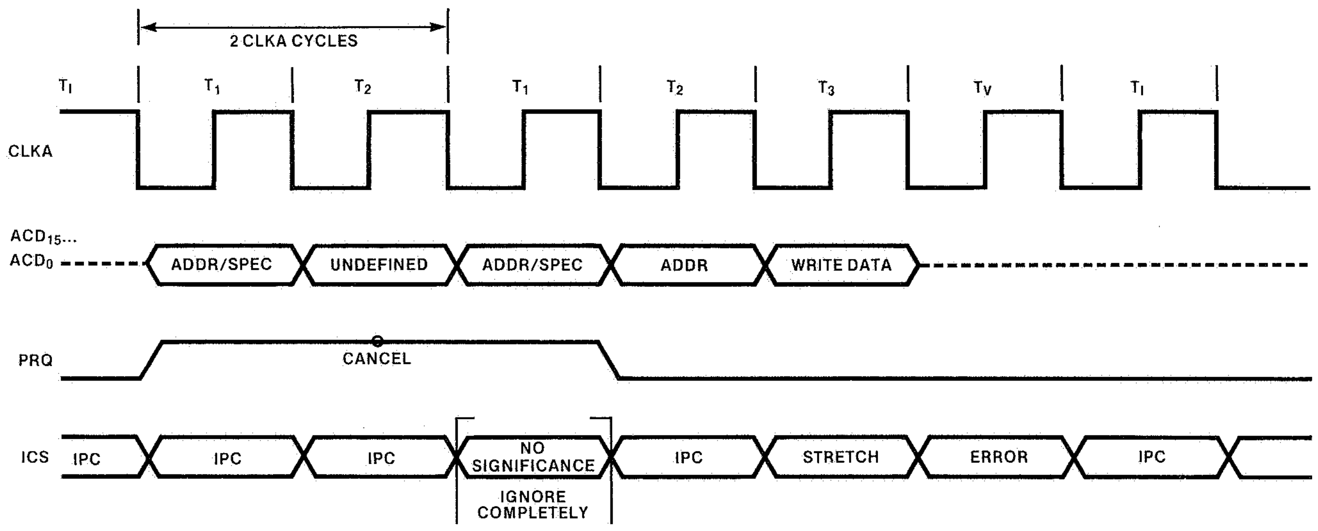


ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-z	Hi-z	Tw
Hi-data1*	Lo-data1	T3
Hi-z	Hi-z	Tv
Hi-z	Hi-z	Ti

*Undefined if single byte read

171873-19

Figure 16. Minimum Read Cycle (Buffered System)

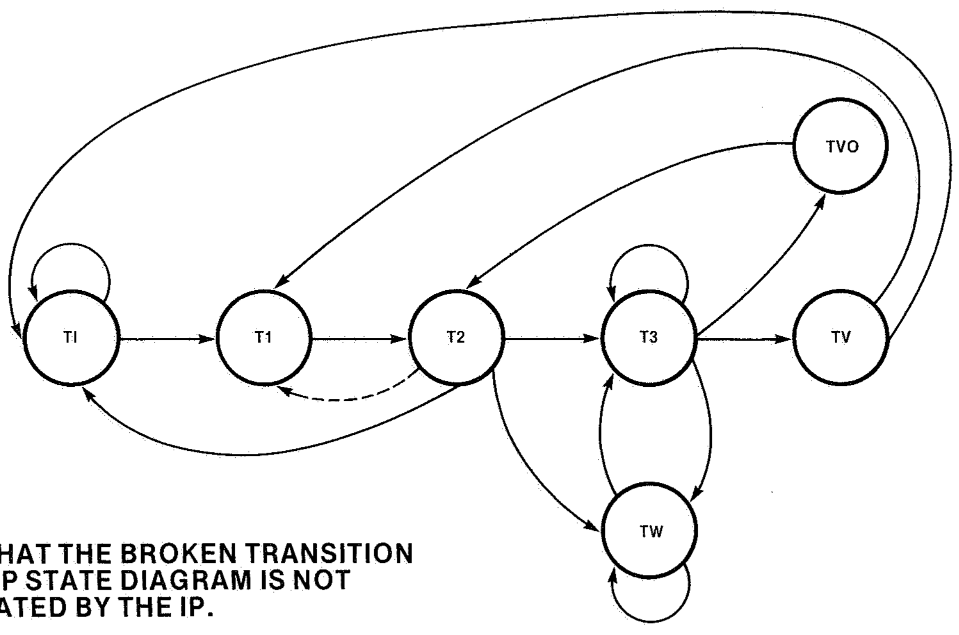


ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1
Undefined	Undefined	T2**
Spec	Lo-adr	T1***
Hi-adr	Mid-adr	T2
Hi-data*	Lo-data	T3
Hi-z	Hi-z	Tv
Hi-z	Hi-z	Ti

*Undefined if single byte write
**Access Cancelled
***New Access Started (Slave must support this subsequent access even though all processors may not implement it.)

171873-20

Figure 17. Minimum Faulted Access Cycle



*NOTE THAT THE BROKEN TRANSITION IN THE IP STATE DIAGRAM IS NOT GENERATED BY THE IP.

171874-9

Figure 18. IP State Diagram

Table 6. iAPX 43203 Interface Processor Pin Summary

432 System Side			
Pin Group	**Pin Name**	**Direction**	**Hardware Error Detection**
PROCESSOR PACKET BUS GROUP	ACD15...ACD0	I/O	X
	PRQ	O	X
	ICS	I	
	BOUT	O	
SYSTEM GROUP	ALARM /	I	
	FATAL/	O (I at Initialization)	
	CLR/	I	
	PCLK/	I	
	INIT/	I	
CLOCK GROUP	CLKA	I	
	CLKB	I	
HARDWARE ERROR DETECTION GROUP	HERR	O (I at Initialization)	

Peripheral Subsystem Side			
Pin Group	**Pin Name**	**Direction**	**Hardware Error Detection**
PERIPHERAL SUBSYSTEM BUS GROUP	AD15... AD0	I/O	X
	BHEN/	I	
	CS/	I	
	WR/	I	
PS TIMING GROUP	ALE	I	
	OE	I	
	SYNC	I	
PS BUFFER CONTROL GROUP	DEN/	O	X
PS INTERLOCK GROUP	HLD	O	X
	HDA	I	
PS SYNCHRONIZATION GROUP	XACK/	O	X
	NAK/	O	X
	INH1	O	X
PS INTERRUPT GROUP	INT	O	X
PS RESET GROUP	PSR	O	X

43203 PIN DESCRIPTION

The following section provides detailed information concerning the 43203 pin description. Table 6 lists a summary of all signal groups, signal names and their active states, and whether or not they are monitored by the Hardware Error Detection circuitry.

Processor Packet Bus Group

ACD_{15}—ACD_0 (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 IP and its environment. These pins are used three ways:

- They may indicate control information for bus transactions,
- They may issue physical addresses generated by the IP for an access, or
- They may transfer data (either direction).

When the 43203 is in checker mode, the ACD pins are monitored by the hardware error detection logic and are in the high impedance mode.

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

PRQ is used to indicate the presence of a transaction between the IP 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 43203 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 detection logic. PRQ is in a high impedance state when the 43203 is in checker mode.

ICS (Interconnect Status, Input, high asserted)

ICS is an indication to the 43203 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:

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

System Group

ALARM/ (Alarm, Input, low asserted)

The ALARM/ input monitors the occurrence of an unusual, system-wide condition such as power failure. ALARM/ is sampled on the rising edge of CLKA.

FATAL/ (Fatal, Output, low asserted) (Master, Input, 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/. FATAL/ is not checked by the hardware error detection logic.

When INIT/ is asserted, the FATAL/ pin assumes an input role. Please refer to the INIT/ pin description for a discussion of this function.

Hardware Error Detection Group

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

HERR is used to signal a discrepancy between a master and a checker (difference between the value internally computed in the checker and that output by the master). The sampling of errors occurs at the most appropriate time for the pin(s) being checked.

HERR is an open drain output which requires an external pullup resistor. Nominally the output is held low. HERR is released upon the detection of discrepancy. The timing of HERR depends on the source of the error. Once HERR is high it will remain high until external logic forces it to go low again. When HERR goes low again, the present HERR error condition is cleared and HERR is immediately capable of detecting and signaling another error.

When INIT/ is asserted, the HERR pin assumes an input role. Please refer to the INIT/ pin description for a discussion of this function.

PCLK/ (Processor Clock, Input, low asserted)

Assertion of PCLK/ for one clock cycle causes the system timer in the IP to decrement. Assertion of PCLK/ for two or more cycles causes the system timer to be reset. PCLK/ must be unasserted for at least 10 clock cycles before being asserted again.

CLR/ (Clear, Input, low asserted)

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, reset to a known state, assert FATAL/, and await an IPC (which resets the IP to the same state as INIT/ assertion does). The IPC will not be serviced for at least four clock cycles following CLR/ assertion.

Response to CLR/ is disabled by the first CLR/ assertion and is reenabled when the IP receives the first IPC (or INIT/ assertion).

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

INIT/ (Initialize, Input, low asserted)

Assertion of INIT/ causes the internal state of the IP to be reset and starts execution of the initialization microcode. INIT/ must be asserted for a minimum of 10 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 INIT/ assertion, the FATAL/ and HERR pins are sampled by the IP to establish the mode in which the two bus interfaces of the IP are to participate in hardware error detection. Table 7 specifies the encoding of the master/checker modes.

Table 7. Representation of MASTER/CHECKER Modes 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

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.

Peripheral Subsystem Bus Group

AD_{15}—AD_0 (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_{15}—AD_0 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.

The address is always a 16-bit, unsigned number. Data may be either 8 bits or 16 bits as defined by BHEN/ and AD_0. The 8-bit data may be transferred on either the high (AD_{15}—AD_8) or the low (AD_7—AD_0) 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 are incorporated in the external CS/ logic.

During the clock 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_0, 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_0 decode as shown in Table 8.

Table 8. Bus Data Controls

BHEN/	AD_0	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_{15}-AD_0 pins and latches WR/, BHEN/ and CS/. Figure 19 shows two styles of ALE.

OE (Data Output Enable, Input, high asserted)

During a read cycle the OE pin enables read data on to the AD_{15}-AD_0 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/.

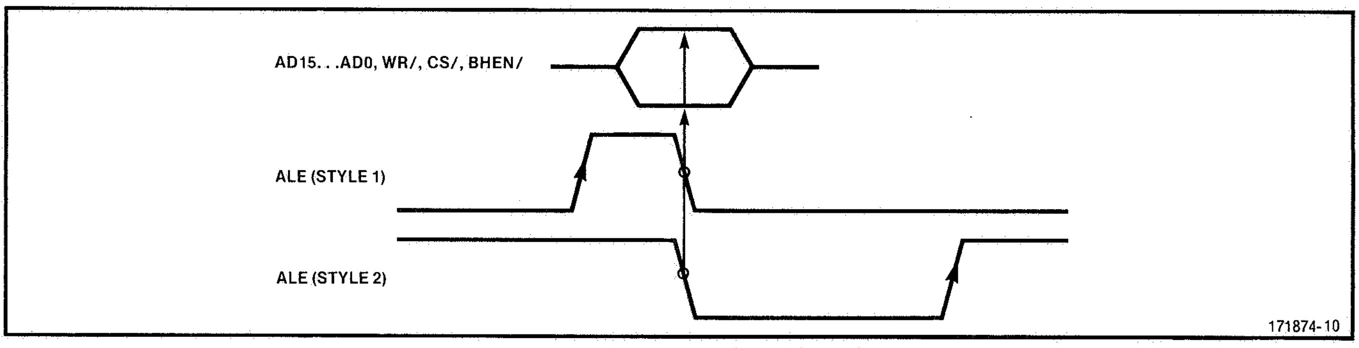


Figure 19. Two styles of ALE

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/, CS/ 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 to the slowest operation (two CLKA's after SYNC). 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 (e.g., 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.

Hardware error detection occurs during the first clock of SYNC 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 been 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_{15}-AD_0 bus.

Five conditions affecting XACK/ behavior are:

- XACK-Delay, user programmable through an IP function request. This parameter establishes the minimum operating XACK-delay with respect to the SYNC signal. Table 9 displays the representation of the XACK-delay codes.
- XACK-enable-flip-flop, set by the rising edge of the ALE signal and reset by the falling edge of the OE signal.
- Internal IP Registers. These are used to determine validity of the peripheral subsystem access and establish access modes.
- Type of access behavior: Random or Buffered, Memory or Interconnect.
- Bus Faults, nonexistent 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 an interconnect access, negative acknowledge may be used to indicate that the access was made to a nonexistent interconnect 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 time.

INH1 (Inhibit, Output, high asserted)

This pin is asynchronously asserted by nonclocked 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 INH1 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 using INH1, as they 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.

Table 9. XACK/ Timing Parameters

Inhibit Mode	WR/	XD_1	XD_0	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

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 time.

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 occurs during CLKA low time.

43203 ELECTRICAL CHARACTERISTICS

Tables 10 through 12 and Figures 20 through 25 provide electrical specification information and include input/output timing, read and write timing, and component maximum ratings.

Instruction Set Comparison

Refer to Table 13 for a GDP/IP operator comparison.

Table 10. 43203 Absolute Maximum Ratings

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 11. iAPX 43203 DC 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...AD0	−	4	mA
	OTHER	−	2	mA
Ioh	@2.4 Voh	−	−0.1	mA

* For operation at 5 MHz or slower, the 43203 may be operated with a Vihc minimum of 2.7 volts.

Table 12. iAPX 43203 AC Characteristics

VCC = 5 ± 10% Ta = 0°C to 70°C Loading: AD15...AD0 20 to 100pf; OTHER 20 to 70pf

Symbol	Description	8 MHz.		5MHz.		Unit
		Min	Max	Min	Max	
	GLOBAL TIMING REQUIREMENTS					
tcy	Clock Cycle Time	125	1000	200	1000	nsec.
tr,tf	Clock Rise and Fall Time	–	10	–	10	nsec.
t1,t2 t3,t4	Clock Pulse Widths	26	250	45	250	nsec.
tis	INIT/ to Signal Hold Time	15	–	20	–	nsec.
tsi	Signal to INIT/ Setup Time	10	–	10	–	nsec.
tie	INIT/ Enable Time	10	–	10	–	tcy
	SYSTEM SIDE TIMING REQUIREMENTS					
tdc	Signal to CLOCK Setup Time	5	–	5	–	nsec.
tcd	Clock to Signal Delay Time	–	55	–	85	nsec.
tdh	Clock to Signal Hold Time	25	–	35	–	nsec.
toh	Clock to Signal Output Hold Time	15	–	20	–	nsec.
ten	Clock to Signal Output Enable Time	15	–	20	–	nsec.
tdf	Clock to Signal Data Float Time	–	55	–	75	nsec.
	PERIPHERAL SUBSYSTEM SIDE TIMING REQUIREMENTS					
tas	AD15...AD0,CS/,WR/,BHEN/ Setup Time to ALE Low	0	–	0	–	nsec.
tah	AD15...AD0,CS/,WR/,BHEN/ Hold Time to ALE Low	32	–	35	–	nsec.
tss	SYNC High Setup Time to CLKA High	50	–	60	–	nsec.
tsh	SYNC Low Hold Time to CLKA High	30	–	40	–	nsec.
tsw	SYNC High Pulse Width	50	tss + 1.5tcy	60	tss + 1.5tcy	nsec.
tds	Write Data Setup to Sampling CLKA High	10	–	20	–	nsec.
tdx*	Write Data Hold to Sampling CLKA Low (Advanced XACK/)	10	–	20	–	nsec.
tdhx	Write Data Hold to XACK/	5	–	5	–	nsec.
tasy	AD15...AD0,CS/,WR/,BHEN/ Setup to SYNC	120	–	160	–	nsec.
	PERIPHERAL SUBSYSTEM TIMING RESPONSES					
tsdh	CLKA High to HLD,INT,PSR	–	75	–	90	nsec.
taih	Valid AD15...AD0,CS/ to Chip INH1 Valid Delay	–	80	–	85	nsec.
tede	OE to DEN/ Delay	–	65	–	70	nsec.
tead	OE to Enable AD15...AD0 Buffers Delay (Read Cycle)	–	70	–	75	nsec.
tdad	OE to Disable AD15...AD0 Buffers Delay (Read Cycle)	–	52	–	52	nsec.
tced	CLKA High to Enable AD15..AD0 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...AD0 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.

Table 12. iAPX 43203 AC Characteristics (Cont'd.)

Symbol	Description	8 MHz.		5MHz.		Unit
		Min	Max	Min	Max	
	XACK/ TIMING CHARACTERISTICS					
	Buffered Accesses with XD= 0					
tax	ALE High to XACK/ Valid	0	65	0	70	nsec.
tdsx	AD15...AD0 Read Data Valid Setup to XACK/ Valid (When internal state does not allow XACK/ before SYNC)	20	–	20	–	nsec.
tadx	Valid AD15...AD0 to XACK/ Valid (When internal state allows XACK/ before SYNC)	–	120	–	140	nsec.
	Buffered Accesses (With XD=1 or XD=2) or Random Accesses					
tdsx	AD15...AD0 Read Data Valid Setup to XACK/	20	–	20	–	nsec.
	Faulted Accesses					
tsdl	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.

* Write data is sampled for only one clock cycle. The PS must meet the t_{DHX} specification thereby guaranteeing t_{DX}.

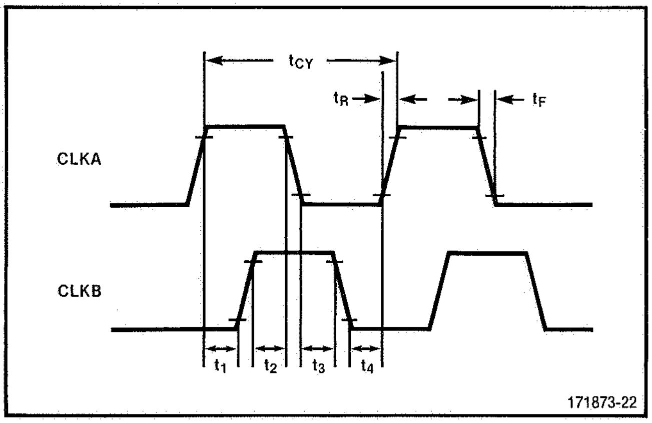
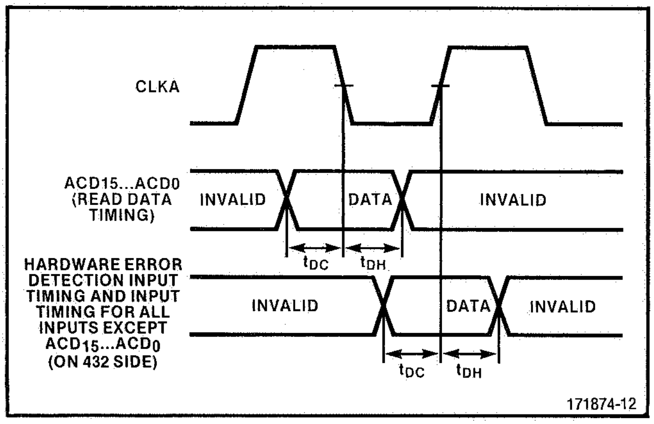


Figure 20. 43203 Clock Input Specification



CLKA
ACD15...ACD0 (READ DATA TIMING)
INVALID
DATA
INVALID
tDC
tDH
HARDWARE ERROR DETECTION INPUT TIMING AND INPUT TIMING FOR ALL INPUTS EXCEPT ACD15...ACD0 (ON 432 SIDE)
INVALID
DATA
INVALID
tDC
tDH
171874-12

Figure 22. 43203 Input Timing Specification

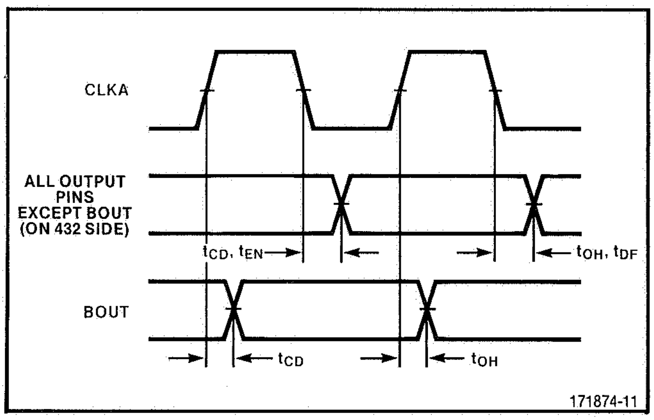


Figure 21. 43203 Output Timing Specification

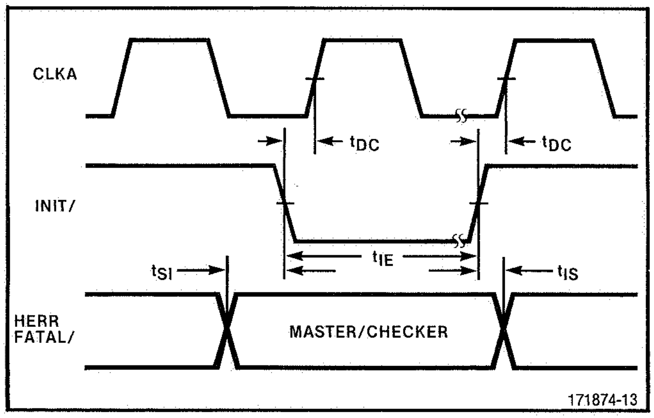


Figure 23. 43203 Initialization Timing

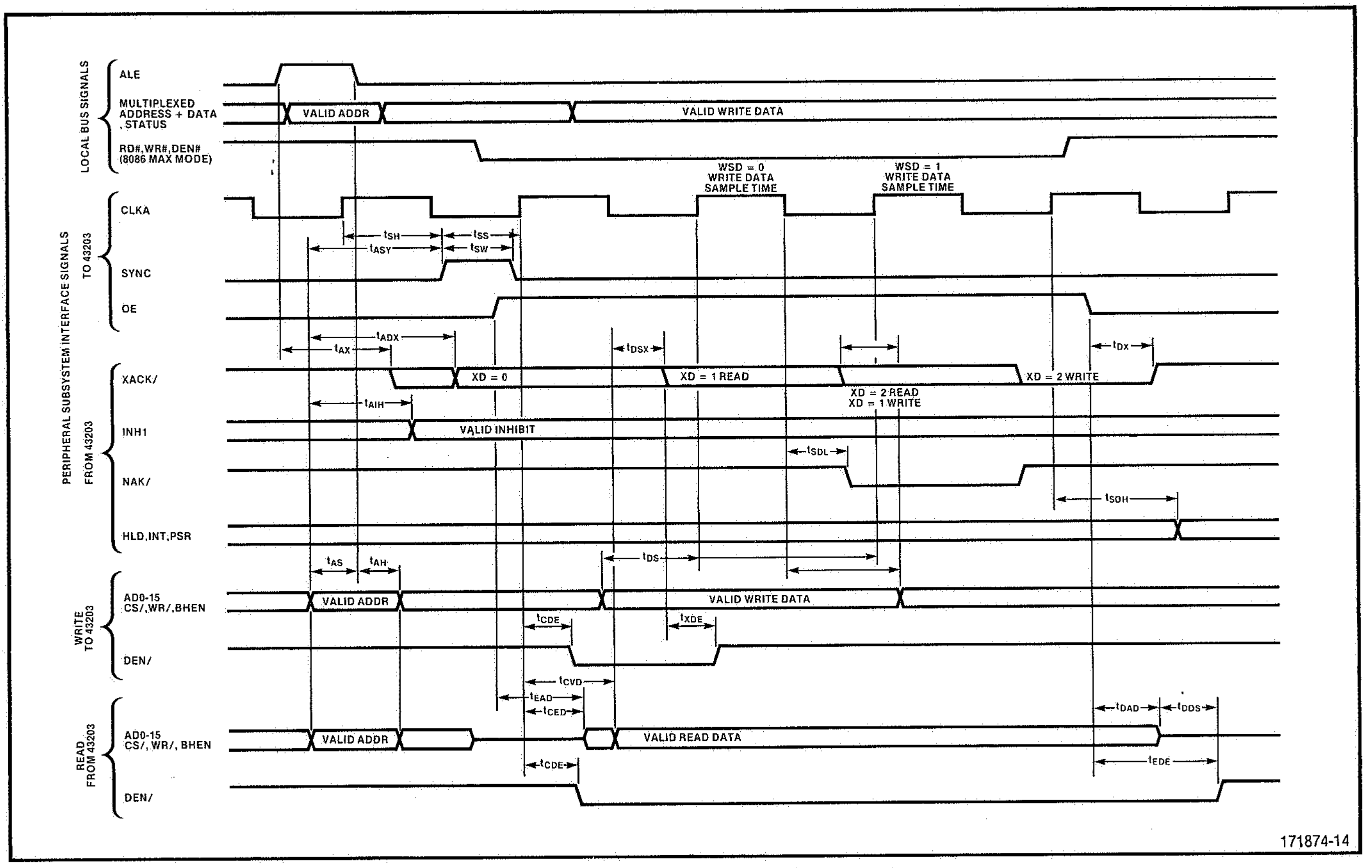


Figure 24. Local Processor Bus Timing

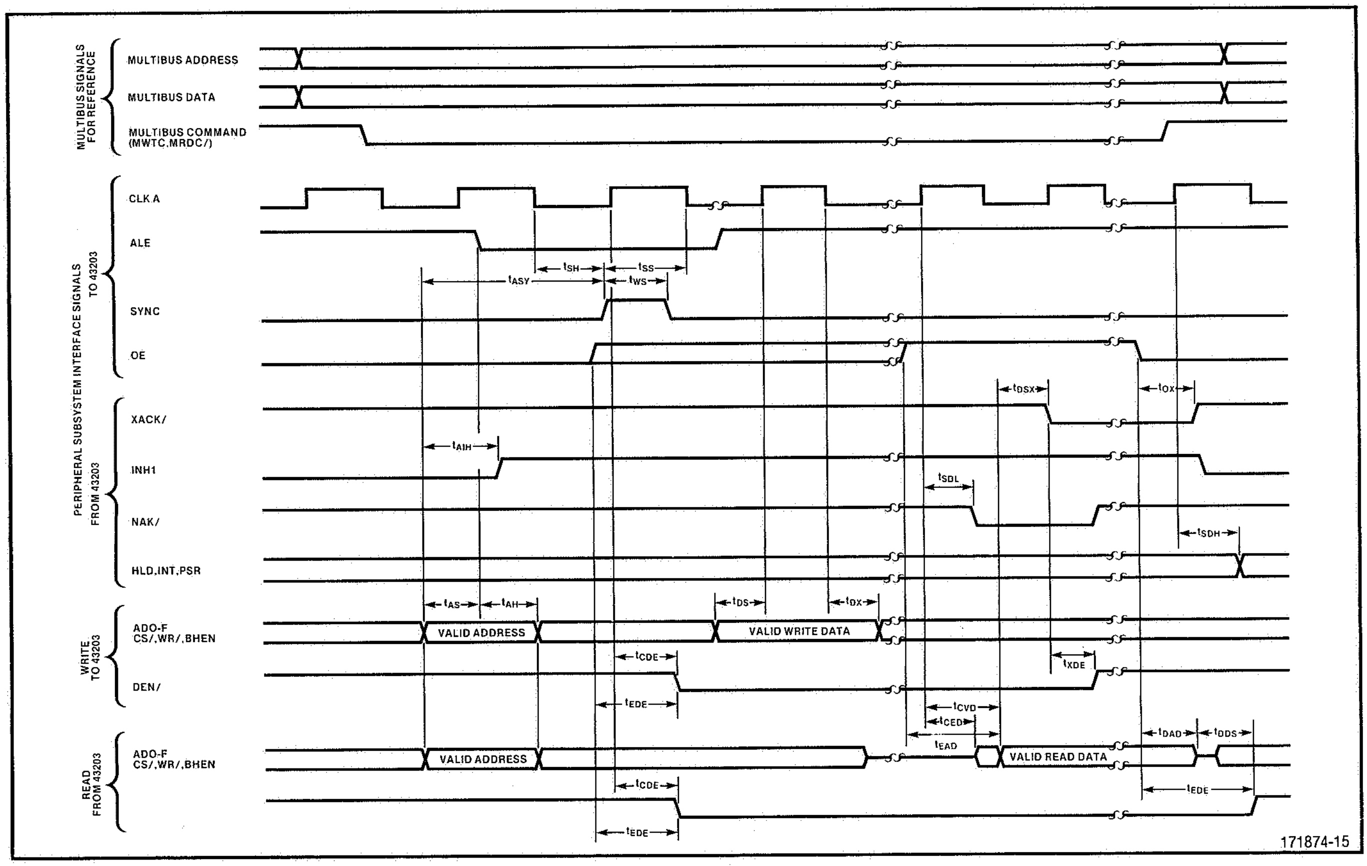


Figure 25. Multibus™ Interface Timing

Table 13. GDP/IP Operator Comparison

OPERATOR	IP IMPLEMENTATION
Window Definition Operator	
Update Window	+
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	=
Retrieve Type	=
Retrieve Type Definition	=
Refinement Operators	
Create Generic Refinement	=
Create Typed Refinement	=
Retrieve Refinement	=
Segment Creation Operators	
Create Data Segment	–
Create Access Segment	–
Create Typed Segment	–
Create Access Descriptor	–
Access Path Inspection Operators	
Inspect Access Descriptor	=
Inspect Access Path	=
Object Interlock Operators	
Lock Object	=
Unlock Object	=
Indivisibly Add Short Ordinal	–
Indivisibly Add Ordinial	–
Indivisibly Insert Short Ordinal	–
Indivisibly Insert Ordinal	–
Context Communication Operators	
Enter Access Segment	=
Enter Process Globals Access Segment	=
Set Context Mode	/
Call	–
Call Context with Message	–
Return	–

Table 13. GDP/IP Operator Comparison (Cont'd.)

OPERATOR	IP IMPLEMENTATION
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	=
Move to Interconnect	–
Move from Interconnect	–
Branch Operators	–
Character Operators	–
Short-Ordinal Operators	–
Short-Integer Operators	–
Ordinal Operators	–
Integer Operators	–
Short-Real Operators	–
Real Operators	–
Temporary-Real Operators	–

Legend:
= IP and GDP identical implementation
+ GDP does not implement operator
– IP does not implement operator
/ While conceptually similar, IP implements operator differently than GDP

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara 95051 (408) 734-8102 x598
Printed in U.S.A./Y-33/0281/50K/PS/GFH

iAPX 43201 iAPX 43202 VLSI GENERAL DATA PROCESSOR

PRELIMINARY

- **Self-Dispatching Processors for Software-Transparent Multiprocessing**
- **Hardware Implemented Inter-Process Communication and Dynamic Storage Allocation**
- **High-Level Language Directed Instruction Set with 0-3 Operand References**
- **Functional Redundancy Checking Mode for Hardware Error Detection**
- **Capability-Based Addressing and Protection**
- **2^{40} Bytes of Virtual Address Space**
- **Object-Based Architecture for Improved Programmer Productivity**
- **Symmetrical Support of All 8-, 16-, and 32-Bit Scalar Data Types and Proposed IEEE Standard 32-, 64-, and 80-Bit Floating Point**

The Intel iAPX 432 General Data Processor (GDP) is a 32-bit microprocessor that 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 system 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 eases their implementation.

The GDP provides 2^{40} bytes of virtual address space 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 each is packaged in a 64-pin Quad In-Line Package (QUIP).

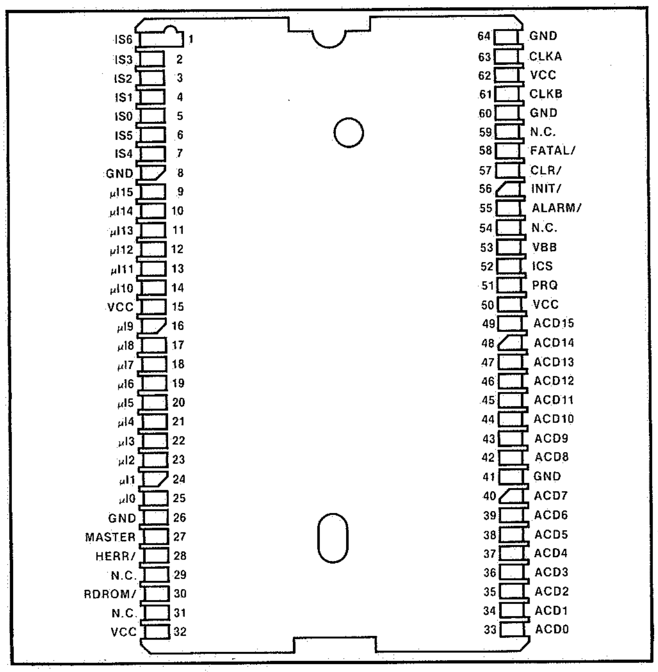


Figure 1. 43201 Pin Assignment 171873-1
Instruction Decoder/Microinstruction Sequencer

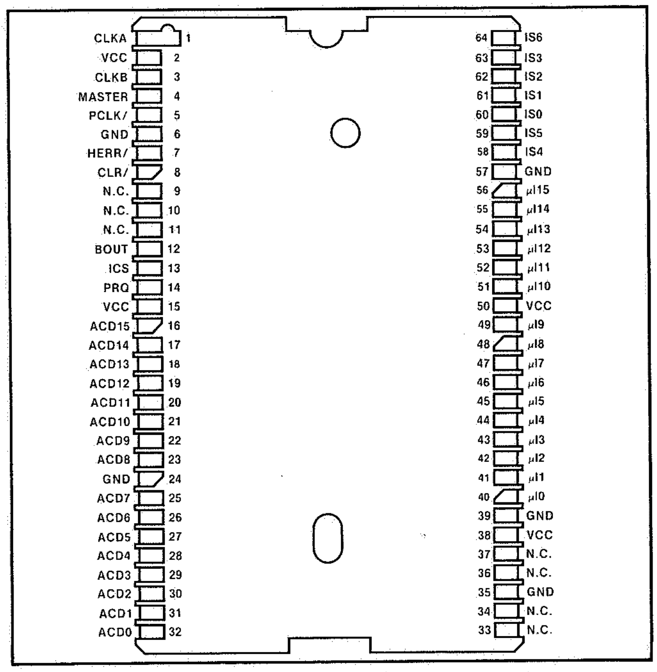


Figure 2. 43202 Pin Assignment
Execution Unit 171873-2

iAPX 432 GDP FUNCTIONAL DESCRIPTION

The general 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:

- Receives macroinstructions
- Processes variable-length fields
- Extracts logical addresses
- Generates starting addresses for the microinstruction procedures
- Generates microinstructions for simple operations

The general task facing the Instruction Decoder is to interpret the macro-instruction stream to determine which micro-instruction sequence should be initiated next and to extract logical address data. The major sub-tasks involved in accomplishing the larger goal are:

- iAPX 432 instructions contain a variable number of bits, depending upon the complexity of the instruction. Instructions may range from a few bits long to several hundred bits long. Instructions may extend over many words in memory. The Instruction Decoder requests words from memory as they are needed.
- A GDP instruction is composed of a variable number of fields and each field may contain a variable number of bits. In most cases, the

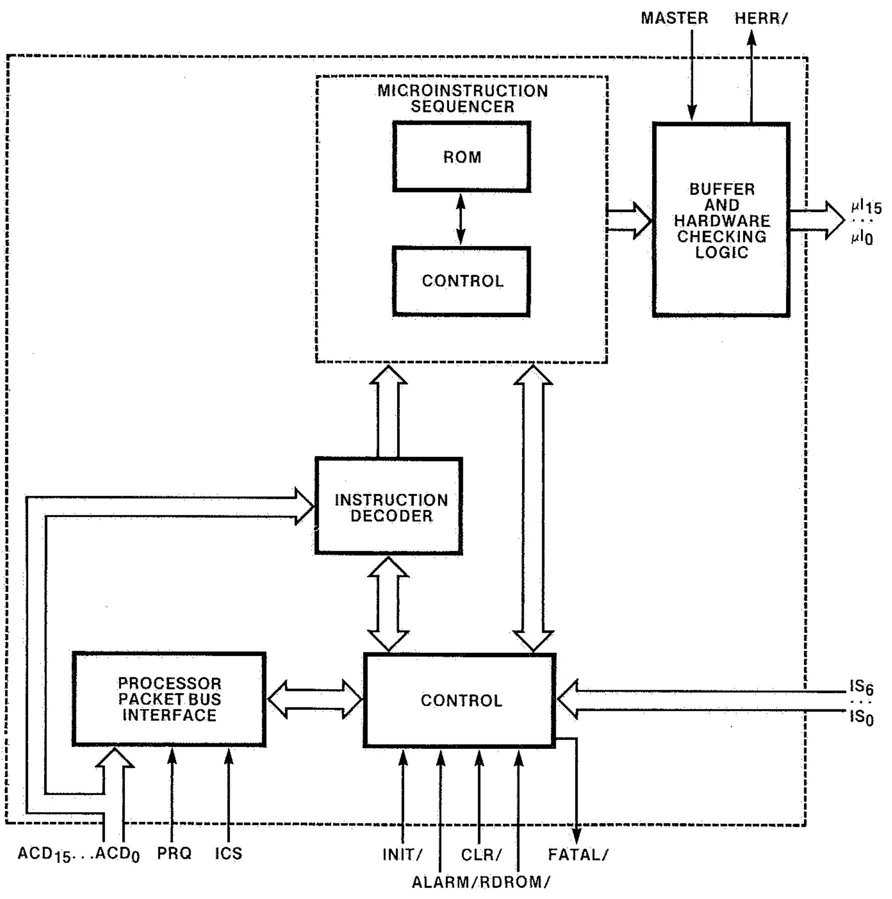


Figure 3. 43201 Block Diagram

171873-3

encoding of a field specifies its length. The Instruction Decoder interprets a field to find its length.

- The Instruction Decoder determines when an instruction boundary has been reached so that it may properly begin decoding the next instruction.
- In some cases, the interpretation of one field may depend upon the value of some previous field. In particular, the interpretation of the opcode (the last field of an instruction) depends on the value of the class field of that instruction (the first field). The Instruction Decoder saves enough information about the instruction to properly interpret every field in the instruction.
- A GDP instruction may contain an explicit reference to some location in memory. This logical address information must be transferred to the Reference Generation Unit so that the correct physical address of the operand may be generated. As with all fields of a GDP instruction, the logical address fields are also variable length fields. The Instruction Decoder has provisions for formatting the logical address information and storing it until needed by the Reference Generation Unit.
- The iAPX 432 instruction set contains several branch instructions. Since iAPX 432 instructions may start at any bit in the segment, the Instruction Decoder is able to start decoding at any point in the segment. Since branches occur fairly often in a typical instruction stream, it is also desirable to minimize the start-up time of the GDP after a branch has occurred.
- The Instruction Decoder provides a mechanism to recover from an instruction that faults. The information necessary for fault recovery will be retained by the Instruction Decoder until the instruction is successfully completed.

Microinstruction Sequencer

The second subprocessor in the pipeline is the Microinstruction Sequencer (MS) which performs the following functions:

- Issues microinstructions to the Execution Unit (EU) (43202)
- Executes microcode sequences out of an on-chip, 4.0K x 16-bit microcode ROM
- Responds to the bus control signals
- Invokes macroinstruction fetches
- Initiates interprocessor communication and fault handling sequences

The role of the Microinstruction Sequencer is to decide which microinstruction should be sent to the Execution Unit for each cycle. The Microinstruction Sequencer must consider each of the following when generating microinstructions:

- There are two sources of microinstructions. They may come from either the Instruction Decoder or from a ROM contained in the Microinstruction Sequencer (MS). The MS must choose the appropriate source.
- The Microinstruction Sequencer must compute the address in ROM (if any) of the next microinstruction.
- The Execution Unit may require variable lengths of time to complete some microinstructions. The Microinstruction Sequencer waits for the Execution Unit to finish the requested operation.

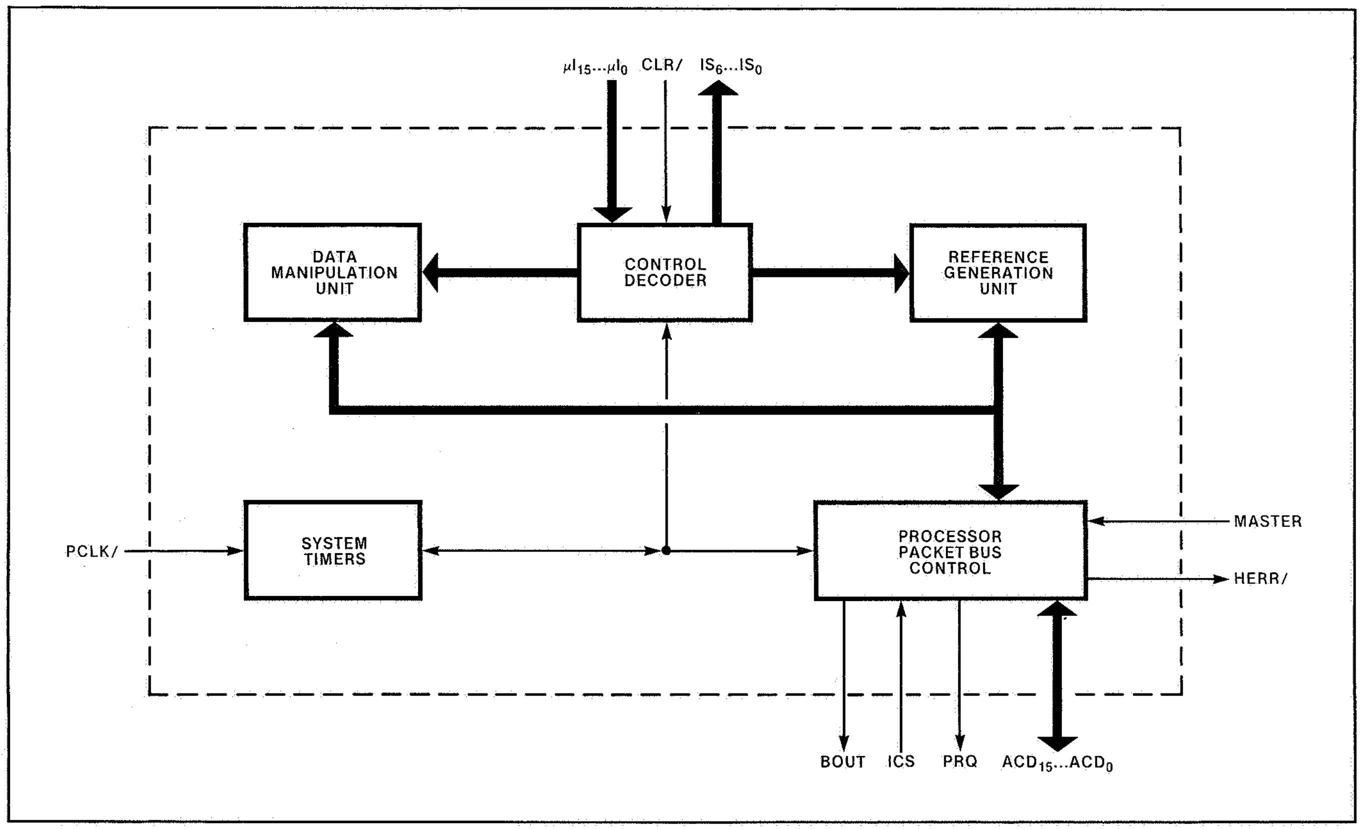
Execution Unit

The 43202 contains the third stage of the GDP pipeline—the Execution Unit (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).

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 Data Manipulation Unit) and Processor Packet bus transactions (in the Reference Generation Unit).

The Data Manipulation Unit contains the registers and arithmetic capabilities to perform the following functions:

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



$\mu I_{15} \ldots \mu I_0$ CLR/ $IS_6 \ldots IS_0$
DATA MANIPULATION UNIT
CONTROL DECODER
REFERENCE GENERATION UNIT
PCLK/
SYSTEM TIMERS
PROCESSOR PACKET BUS CONTROL
MASTER
HERR/
BOUT ICS PRQ $ACD_{15} \ldots ACD_0$

Figure 4. 43202 Block Diagram 171873-4

The Reference Generation Unit performs the following functions:

- Provides the translation of a 40-bit virtual address into a 24-bit physical address
- Provides for a hardware-enforced domain protection system (read, write, alter, accessed)
- Handles sequencing for 8-, 16-, 32-, 64-, and 80-bit memory accesses
- Controls on-chip top-of-stack register

The Execution Unit manipulates data and translates the logical addresses into physical addresses. The efficient performance of these tasks requires:

- While most microinstructions require only a single cycle to complete, there are some that require multiple and even variable numbers of cycles. As a result there are two sequences in the Execution Unit. One sequence is associated with the Data Manipulation Unit and is responsible for controlling multiple-cycle arithmetic operations. The other sequencer works in conjunction with the Reference Generation Unit and is responsible for running cycles on the Processor Packet bus.
- When a reference to a given memory segment has been translated from its logical representation to a physical address, there is a cache in the Reference Generation Unit that maintains the physical base address as well as the length of the segment. Future references to the same segment can use this cached information as the basis for logical to physical address translation.
- There is a hardware-implemented feature which uses least-recently-used algorithms for deciding which cached segment base-length pair to replace when a new segment is referenced.
- The top 16-bit element of the operand stack can be stored in a register in the Data Manipulation Unit.
- A circuit in the Reference Generation Unit checks every memory reference to see if it is within the length of its segment. Since the iAPX 432 architecture controls the type of access (read, write) as well, whether or not the access is allowed at all, this hardware also verifies that the reference is of the proper type.

The 43201 and 43202 components, described above, together form the GDP. Figure 5 is a block diagram that shows both units interfacing to the Packet bus as a single processor.

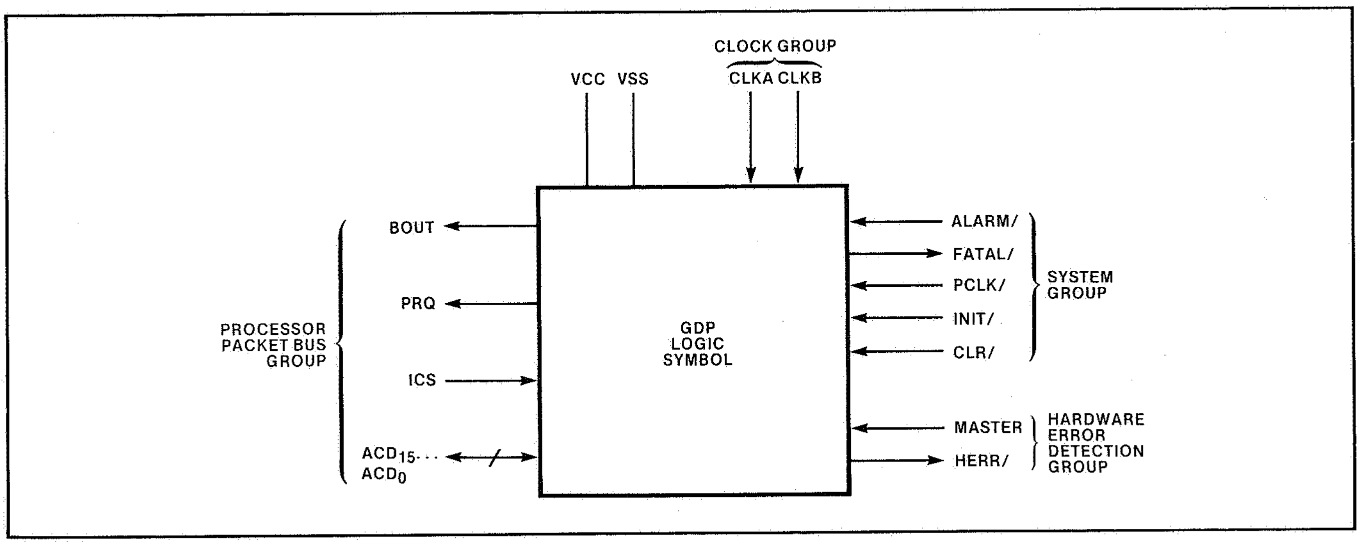


Figure 5. GDP Block Diagram 171873-5

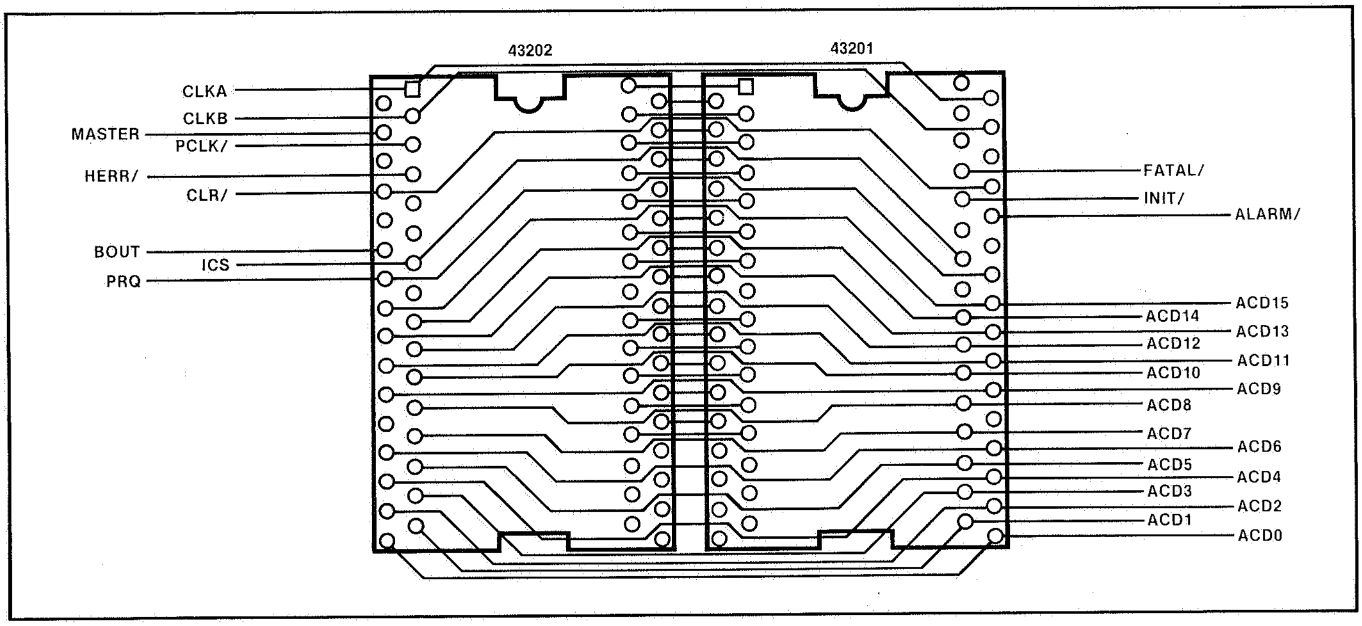


Figure 6. GDP Layout 171873-6

iAPX 43201/43202 PHYSICAL INTERCONNECT

Figure 6 illustrates how the 43201 and 43202 are layed-out to form a GDP.

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 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 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:

- Short Direct
- Long Direct
- Stack Indirect
- General Indirect

The modes of Displacement Generation specify the physical location and displacement of objects within a given segment or segment. The four modes of Displacement Generation are:

- Scalar Data Reference Mode
- Record Item Reference Mode
- Static Vector Element Reference Mode
- Dynamic Vector Element Reference Mode

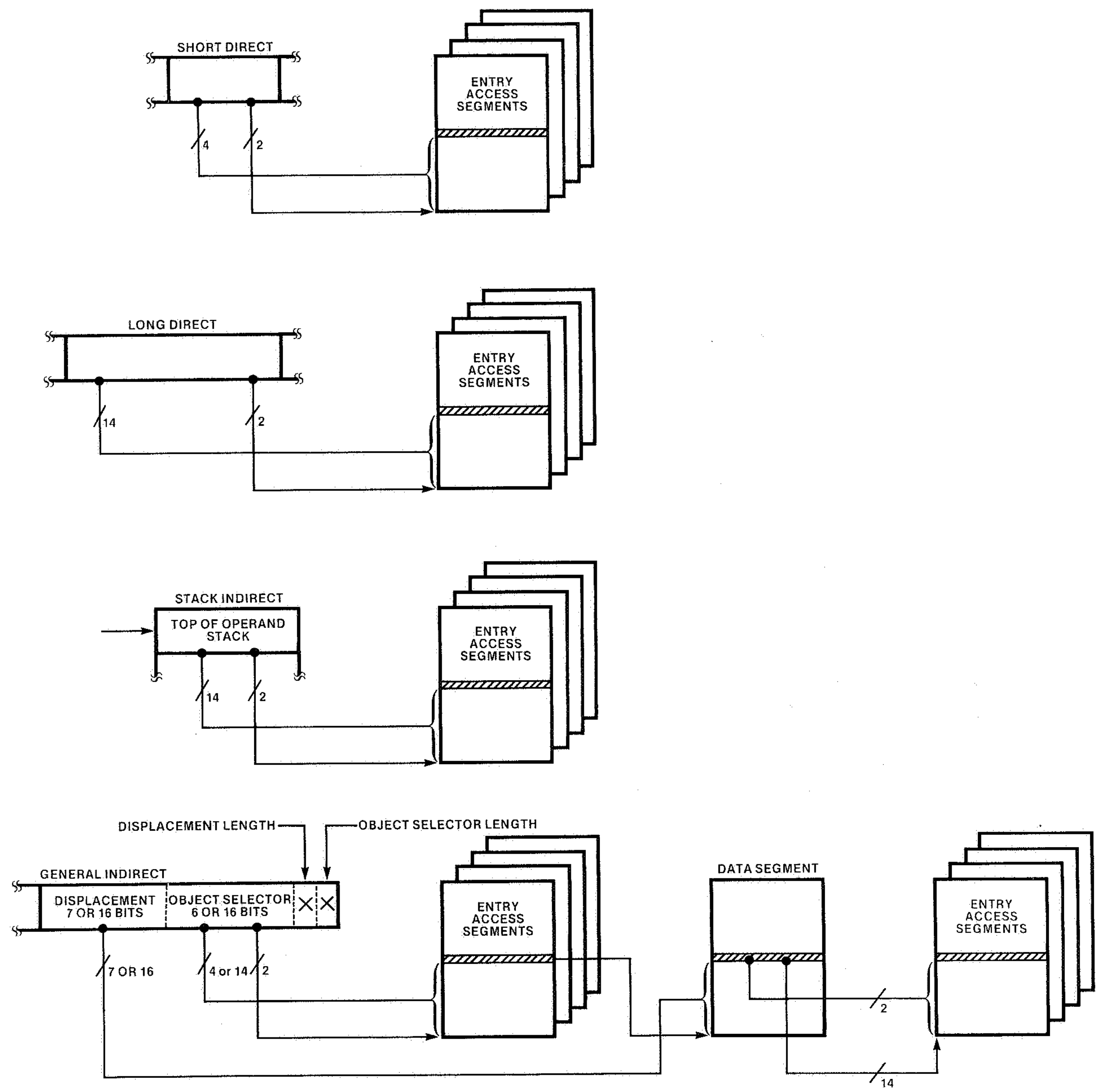
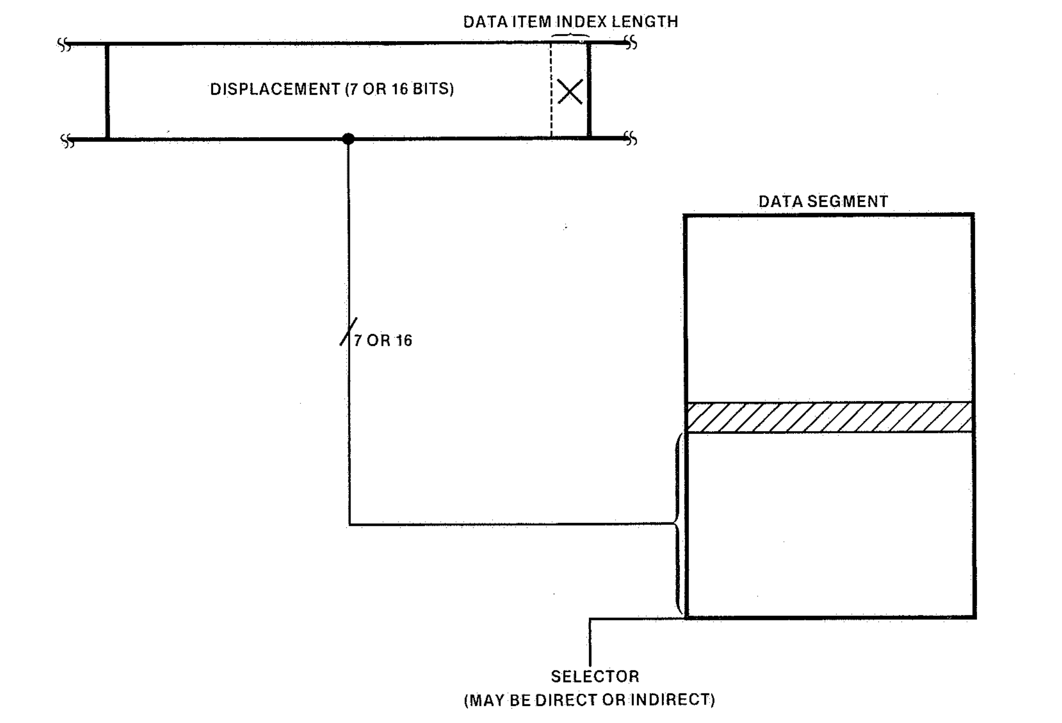


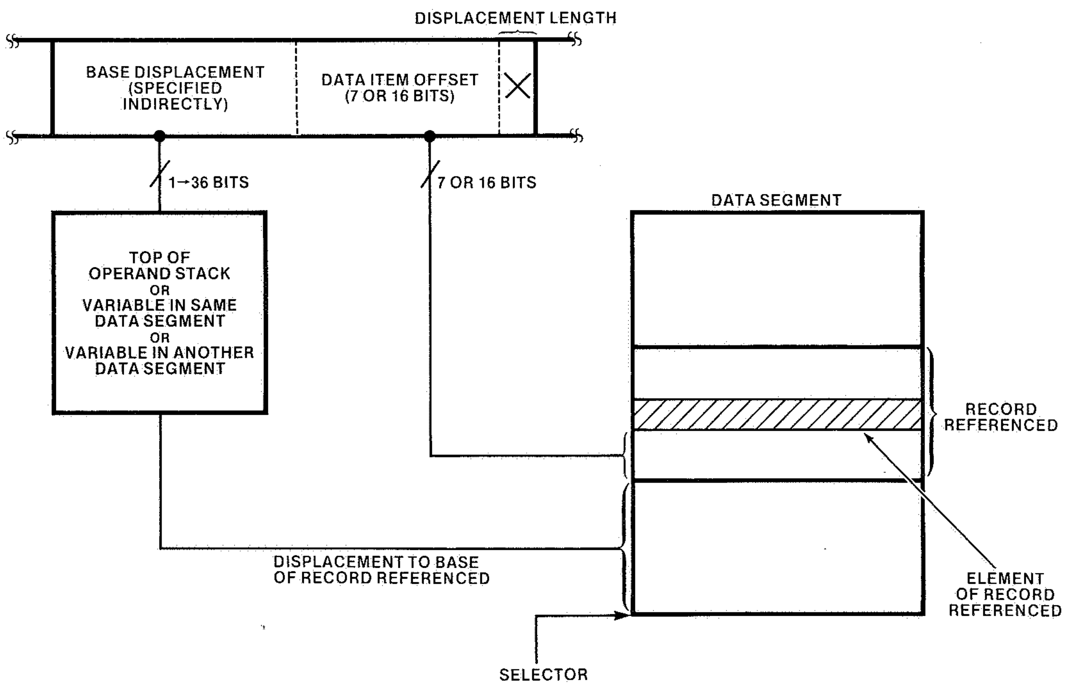
Figure 7. Modes of Selector Generation

171873-7


DATA ITEM INDEX LENGTH
DISPLACEMENT (7 OR 16 BITS)
7 OR 16
DATA SEGMENT
SELECTOR (MAY BE DIRECT OR INDIRECT)

A. SCALAR DATA REFERENCE MODE

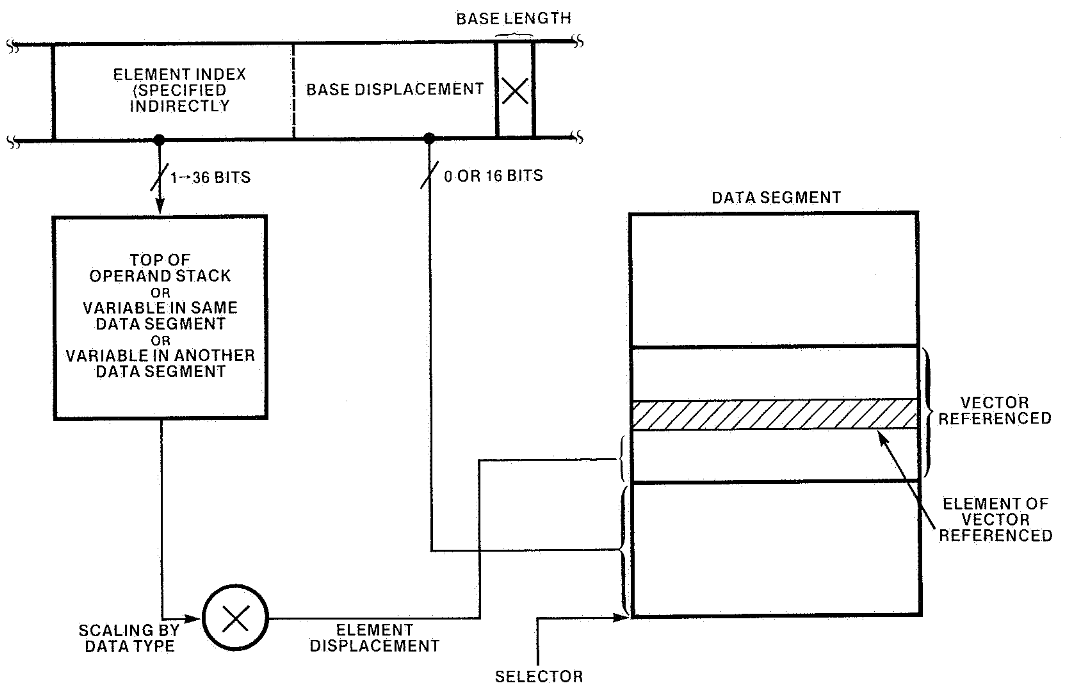
171873-8


DISPLACEMENT LENGTH
BASE DISPLACEMENT (SPECIFIED INDIRECTLY)
DATA ITEM OFFSET (7 OR 16 BITS)
1→36 BITS
7 OR 16 BITS
TOP OF OPERAND STACK OR VARIABLE IN SAME DATA SEGMENT OR VARIABLE IN ANOTHER DATA SEGMENT
DATA SEGMENT
RECORD REFERENCED
DISPLACEMENT TO BASE OF RECORD REFERENCED
ELEMENT OF RECORD REFERENCED
SELECTOR

B. RECORD ITEM REFERENCE MODE

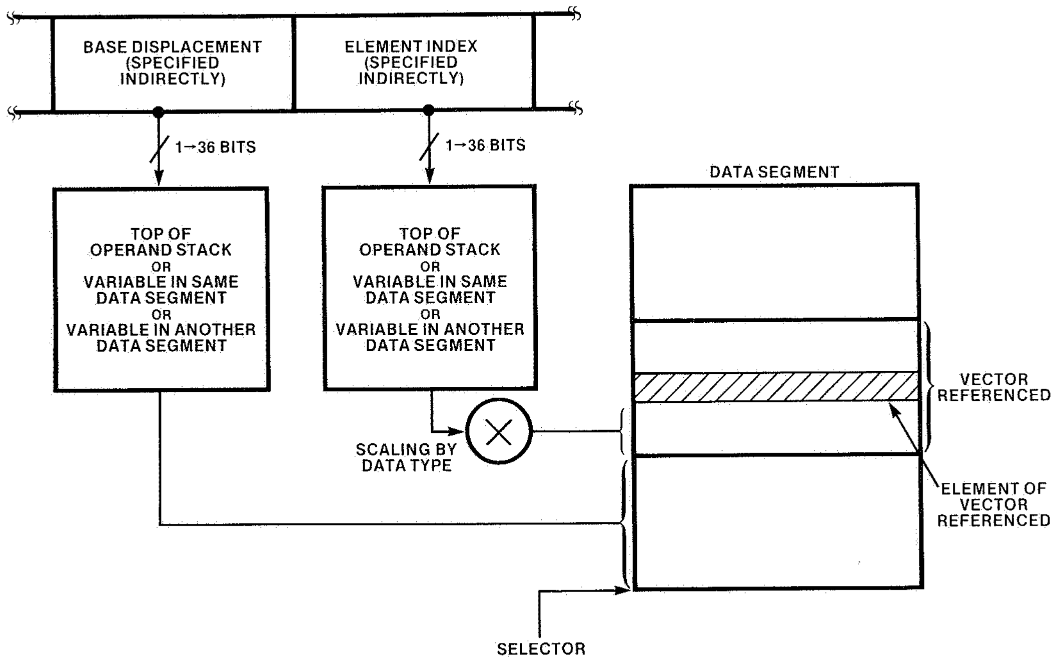
171873-9

Figure 8. Modes of Displacement Generation


BASE LENGTH
ELEMENT INDEX (SPECIFIED INDIRECTLY
BASE DISPLACEMENT
1→36 BITS
0 OR 16 BITS
DATA SEGMENT
TOP OF OPERAND STACK OR VARIABLE IN SAME DATA SEGMENT OR VARIABLE IN ANOTHER DATA SEGMENT
VECTOR REFERENCED
ELEMENT OF VECTOR REFERENCED
SCALING BY DATA TYPE
ELEMENT DISPLACEMENT
SELECTOR

C. STATIC VECTOR ELEMENT REFERENCE MODE

171873-10


BASE DISPLACEMENT (SPECIFIED INDIRECTLY)
ELEMENT INDEX (SPECIFIED INDIRECTLY)
1→36 BITS
1→36 BITS
DATA SEGMENT
TOP OF OPERAND STACK OR VARIABLE IN SAME DATA SEGMENT OR VARIABLE IN ANOTHER DATA SEGMENT
TOP OF OPERAND STACK OR VARIABLE IN SAME DATA SEGMENT OR VARIABLE IN ANOTHER DATA SEGMENT
SCALING BY DATA TYPE
VECTOR REFERENCED
ELEMENT OF VECTOR REFERENCED
SELECTOR

D. DYNAMIC VECTOR ELEMENT REFERENCE MODE

171873-11

Figure 8. Modes of Displacement Generation (Cont'd.)

HARDWARE ERROR DETECTION FOR iAPX 432 PROCESSORS

iAPX 432 processors include a facility to support the hardware detection of errors by functional redundancy checking (FRC). At initialization 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, those pins which are to be checked on 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/ (refer to Figure 9).

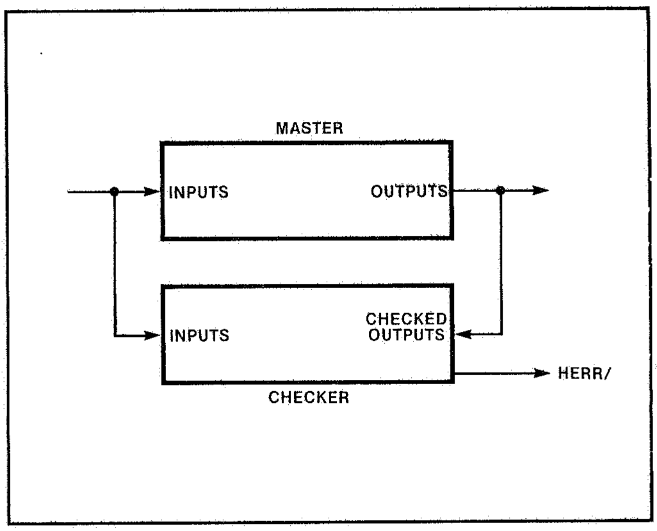


Figure 9. Hardware Error Detection 171873-12

iAPX 432 INFORMATION STRUCTURE

The following section presents the information structure for an iAPX 432 system and includes a discussion of memory system requirements, physical addressing, data formats, and data representation. Any 432 processor 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 maximum physical memory of 16 megabytes.

Data Formats

When a processor executes the instructions of an operation within a context, operands found in the logical address space 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 operands 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 10.

Data Positioning

The data operand types shown in Figure 10 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 (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

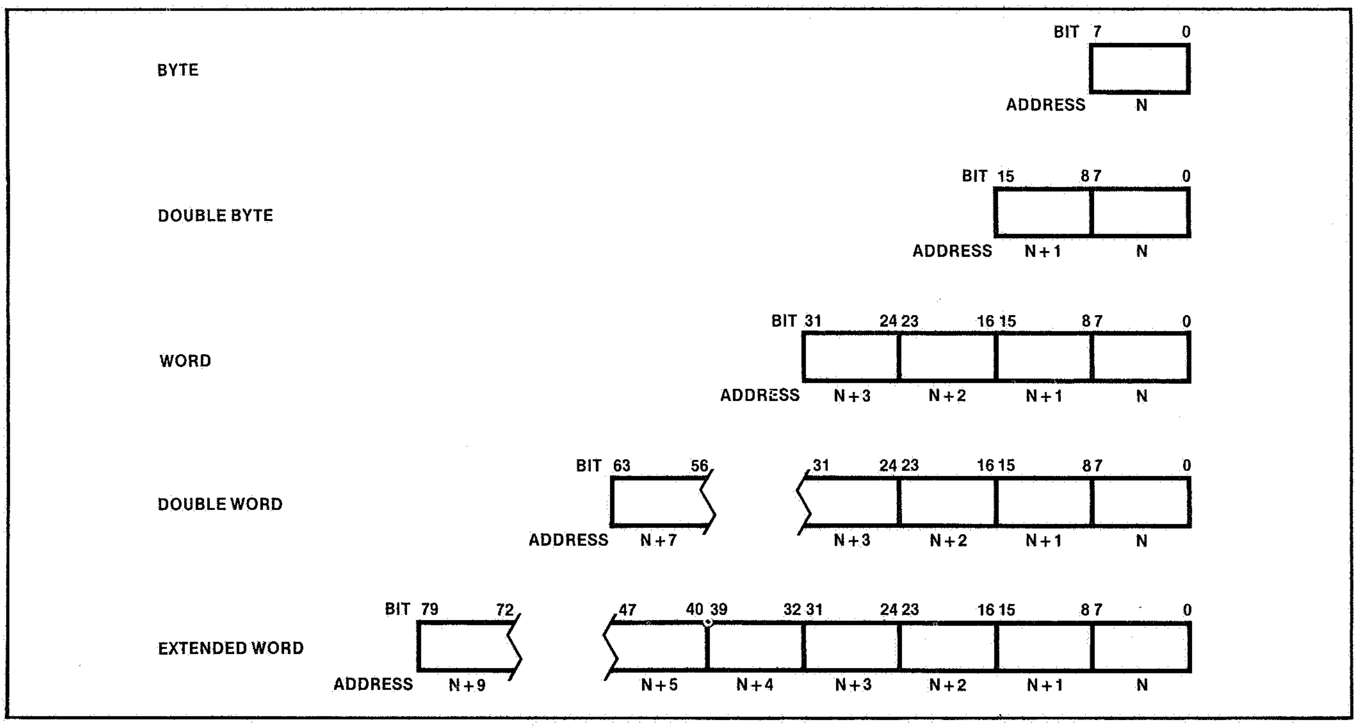

BYTE
BIT 7 0
ADDRESS N
DOUBLE BYTE
BIT 15 8 7 0
ADDRESS N+1 N
WORD
BIT 31 24 23 16 15 8 7 0
ADDRESS N+3 N+2 N+1 N
DOUBLE WORD
BIT 63 56 31 24 23 16 15 8 7 0
ADDRESS N+7 N+3 N+2 N+1 N
EXTENDED WORD
BIT 79 72 47 40 39 32 31 24 23 16 15 8 7 0
ADDRESS N+9 N+5 N+4 N+3 N+2 N+1 N

Figure 10. Basic iAPX 432 Data Lengths 171873-13

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 awaiting 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 occur, the memory system must ensure that the set of locations used to store the operand does not get changed to some interleaved combination of the two written values.

PROCESSOR PACKET BUS DEFINITION

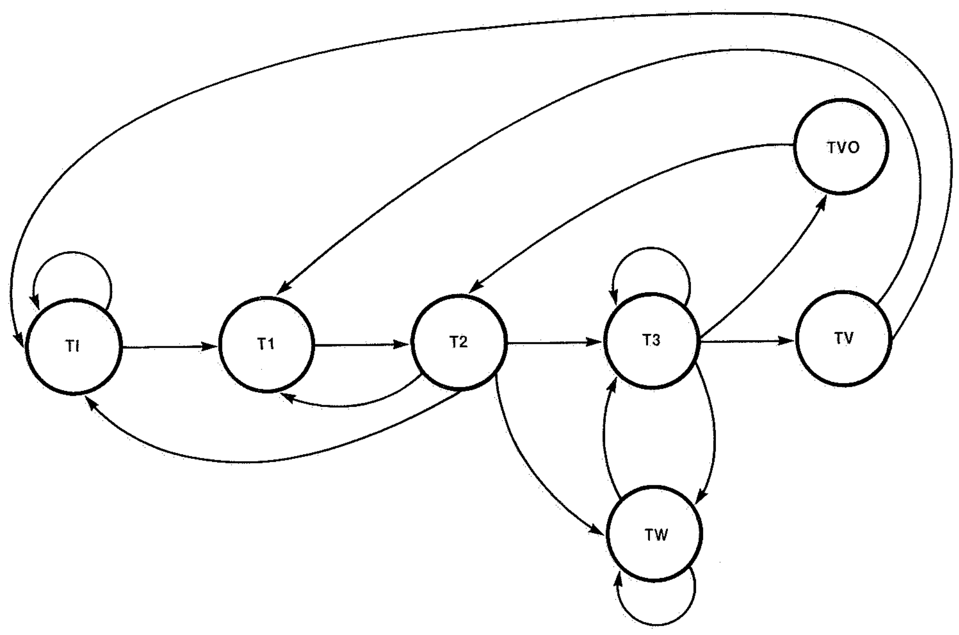
This section describes and defines the significance of the 19 signal lines that make up the Processor Packet bus, and the general scheme by which timing relationships on these lines are derived. Although this section defines all legal bus activities, the processors do not necessarily perform all allowed activities. Slaves to the Processor Packet bus must support all state transitions to ensure compatibility (refer to Figure 11 for Packet bus states).

The Processor Packet bus consists of 3 control lines:

- Processor Packet bus Request (PRQ),
- Enable Buffers for Output (BOUT),
- Interconnect Status (ICS).

This bus also includes sixteen 3-state Address-Control-Data lines (ACD15 through ACD0). PRQ has two functions whose use depends upon the application; i.e., PRQ either indicates the first cycle of a transaction on the Processor Packet 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 Packet bus transaction. The processor interprets the ICS input as an indication of one of the following:

- Whether or not an interprocessor communication (IPC) is waiting,
- Whether or not the slave requires more time to service the processor's request,
- Whether or not a bus ERROR has occurred.



Initial State	Next State	Trigger
Ti	Tl	Bus cycle desired
	Ti	No bus cycle desired
Tl	T2	Unconditional
T2	T3	ICS high
	Tw	ICS low
	Tl	Cancelled, Access Pending
	Ti	Cancelled, No Access Pending
T3	T3	Additional transfer required, ICS high
	Tw	Additional transfer required, ICS low
	Tv	All transfers completed, no overlapped access
	Tvo	Current write with overlapped access
Tv	Ti	No access pending
	Tl	Access pending
Tvo	T2	Unconditional
Tw	Tw	ICS low
	T3	ICS high

Figure 11. Processor Packet Bus State Diagram

171873-14

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-ACD0 (Address/Control/Data)

During the first cycle, (T1 or Tvo) of a Processor Packet 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 (T2), 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 T2, the access is cancelled and the ACD lines are not defined.

During the third cycle (T3 or Tw) of a Processor Packet bus transaction the processor presents a high impedance to the ACD lines for read transactions and asserts write data for write transactions.

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 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. The processor may simply enter T3 or it may first enter Tw for any number of cycles (as dictated by ICS).

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 access). 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 access, the bus state moves to Tv (the ACD lines will be high impedance). 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 transaction (see Table 2). 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

Table 1. ACD Specification Encoding

ACD 15	ACD 14	ACD 13	ACD 12	ACD 11	ACD 10	ACD 9	ACD 8
Access	Op	RMW	Length			Modifiers	
0 - Memory 1 - Other	0 - Read 1 - Write	0 - Nominal 1 - RMW	000 - 1 Byte 001 - 2 Bytes 010 - 4 Bytes 011 - 6 Bytes 100 - 8 Bytes 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	

IPC or reconfiguration request has been received. An iAPX 432 processor is required to record and service only one IPC or reconfiguration request at a time. Logic in the interconnect system must record and sequence multiple (possibly simultaneous) IPC occurrences and reconfiguration requests to the processor. Thus the logic that forms ICS must accomodate global and local IPC arrivals and requests for reconfiguration as individual events:

- Assert IPC significance on ICS for the arrival of an IPC or reconfiguration request.
- When the iAPX 432 processor reads interconnect address register 2, it will respond to one of the status bits for the IPC or reconfiguration request 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)
- 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. The interconnect system must signal the second IPC by raising ICS high for at least one cycle and then setting ICS low for at least one cycle during IPC significance time.

Table 2. ICS Interpretation

	Level		State
	High	Low	
IPC	None	Waiting	Ti, T1, T2*
Stretch	Don't	Stretch	T3, Tw
Err	Bus Error	No Error	Tv, Tvo

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

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 indicate that the current cycle is to be cancelled. (See Table 3.)

Table 3. PRQ Interpretation

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

BOUT (Enable Buffers for Output)

BOUT is provided to control external buffers when they are present. Table 4 and Figures 12 through 16 show its state under various conditions.

PROCESSOR PACKET BUS TIMING RELATIONSHIPS

All timing relationships on the Processor Packet bus are derived from a simple scheme and related to Table 5. Each timing diagram shown in the following pages (Figures 12 through 17) 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. The solid lines in Figure 18 show the state transitions initiated by the GDP.

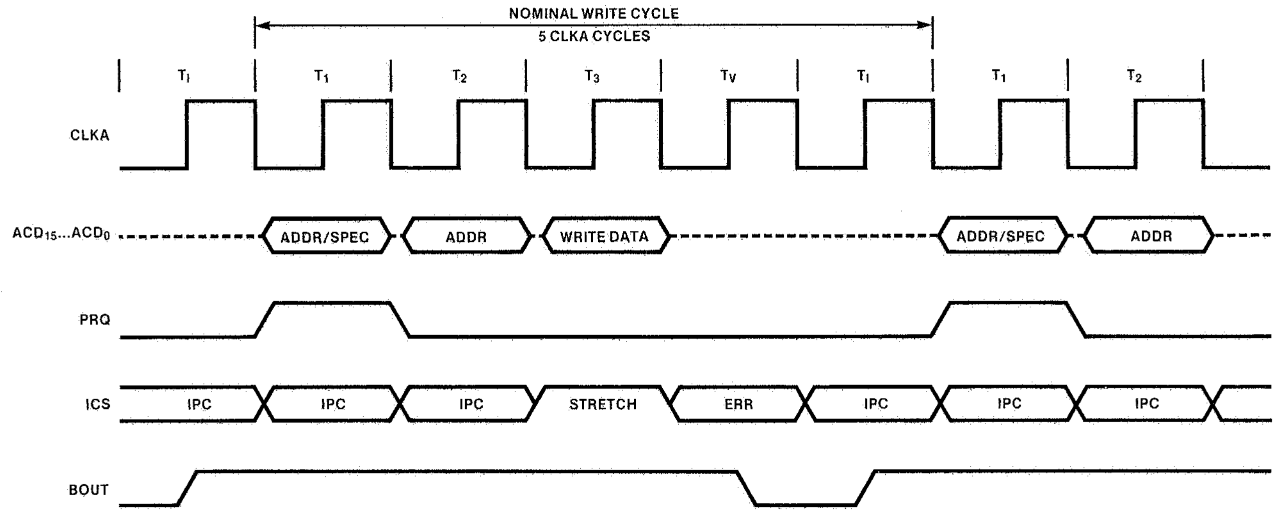
Any agent connected to the Processor Packet bus is recognized as either a processor or a slave. Examples of processors are the GDP and the IP. A memory system provides an example of a slave.

In all tranfers between a processor and a slave, the data to be driven are clocked three-quarters of a cycle before they are to be sampled. This allows adequate time for the transfer and ensures sufficient hold time after sampling. 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 AC characteristics section.

Table 4. BOUT Interpretation

BOUT	Always High	Low-to-High Transition or Low	High-to-Low Transition or Low	High-to-Low Transition or High
Write	T1, T2, T3, Tw, Tvo	Ti	None	Tv
Read	T1,T2	Ti,Tv	T3,Tw	None



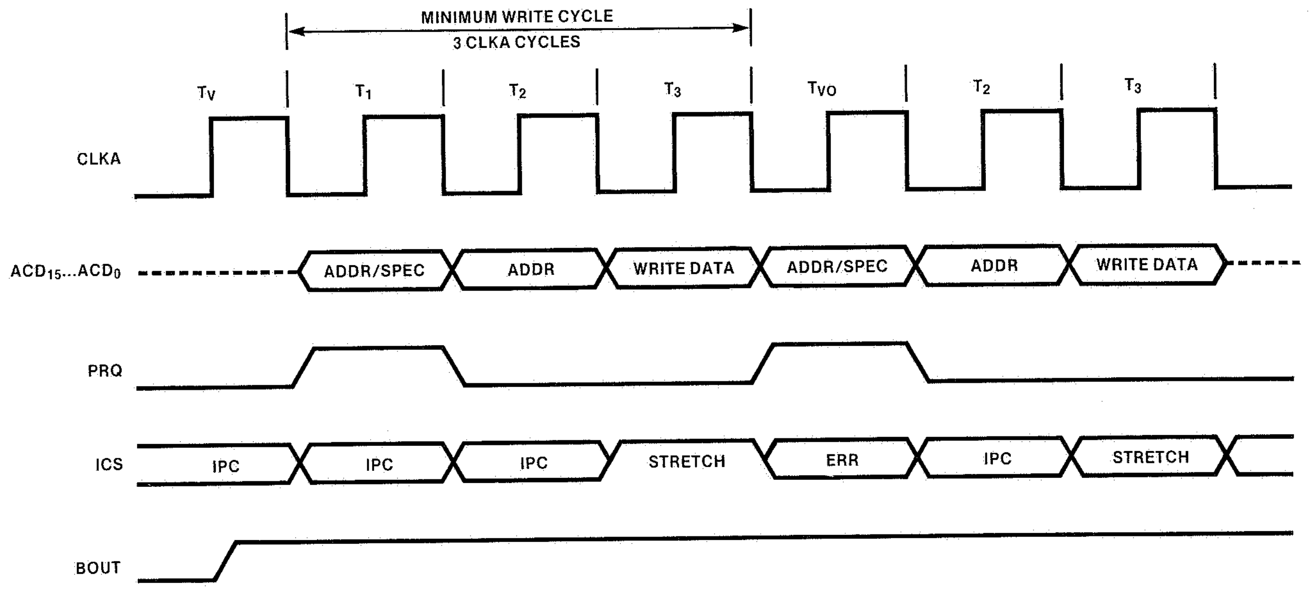
ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data1*	Lo-data1	T3
Hi-z	Hi-z	Tv
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2

*Undefined if single byte write

Figure 12. Nominal Write Cycle Timing 171873-15

Table 5. iAPX 432 Component Signaling Scheme

	Processor		Slave	
Inputs Sampled	ACD:	↓CLKA	All:	↑CLKB
	Others:	↑CLKA		
Outputs Driven	All (except BOUT):	↓CLKA	ACD:	↓CLKB
			Others:	↑CLKB
	BOUT:	↑CLKA		

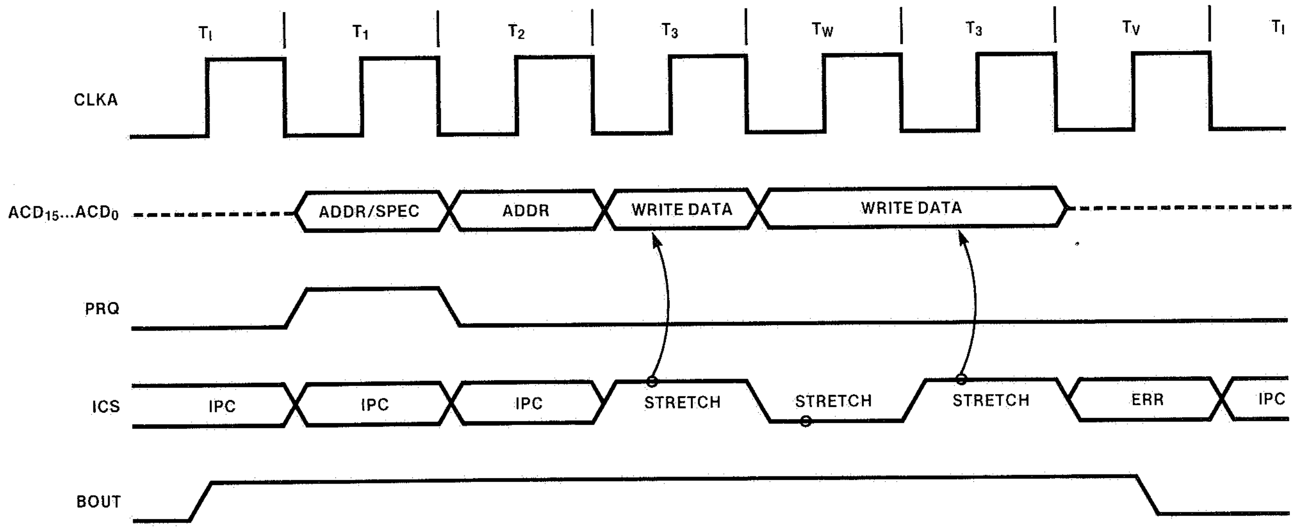


ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Tv**
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data1*	Lo-data1	T3
Spec	Lo-adr	Tvo
Hi-adr	Mid-adr	T2
Hi-data1	Lo-data1	T3

*Undefined if single byte write
**(Preceded by read cycle)

Figure 13. Minimum Write Cycle Timing

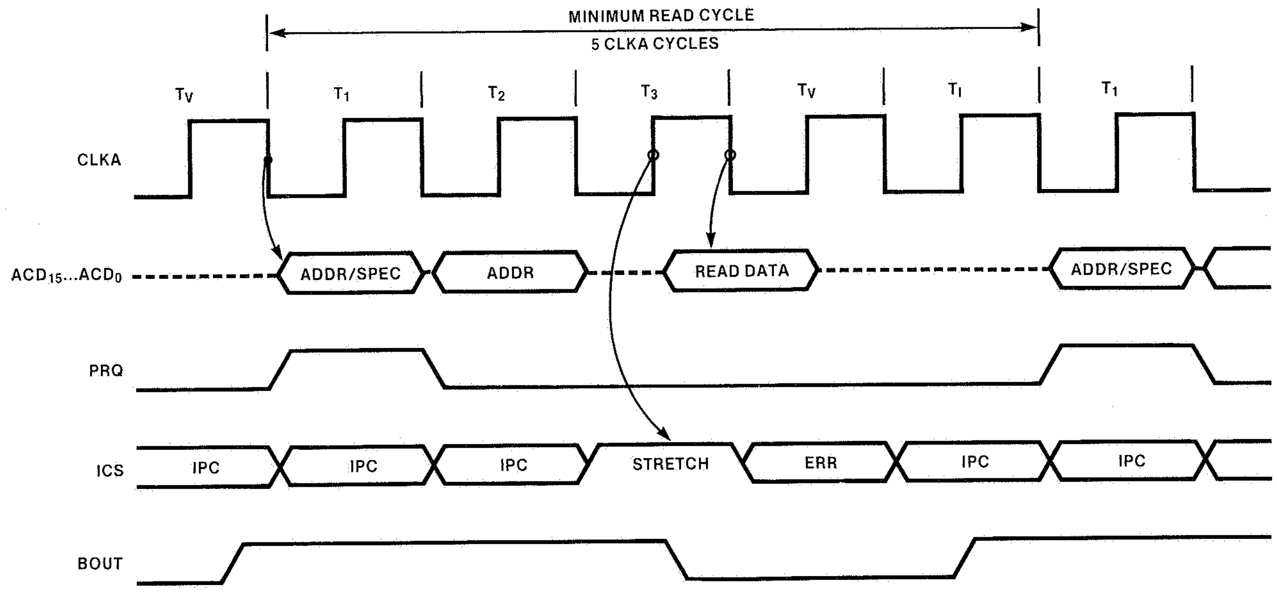
171873-16



ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data1	Lo-data1	T3
Hi-data2	Lo-data2	Tw
Hi-data2	Lo-data2	T3
Hi-z	Hi-z	Tv
Hi-z	Hi-z	Ti

Figure 14. Stretched Write Cycle Timing

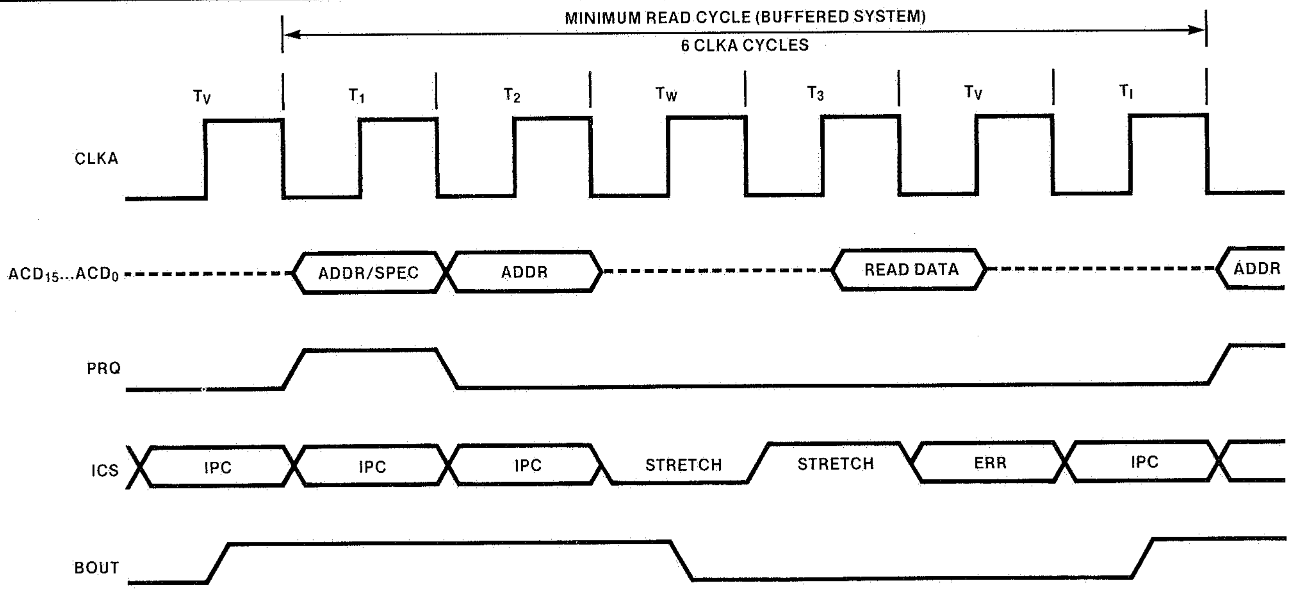
171873-17



ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Tv**
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data*	Lo-data1	T3
Hi-z	Hi-z	Tv
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1

*Undefined if single byte read
**(Preceded by a read): AD TIMING

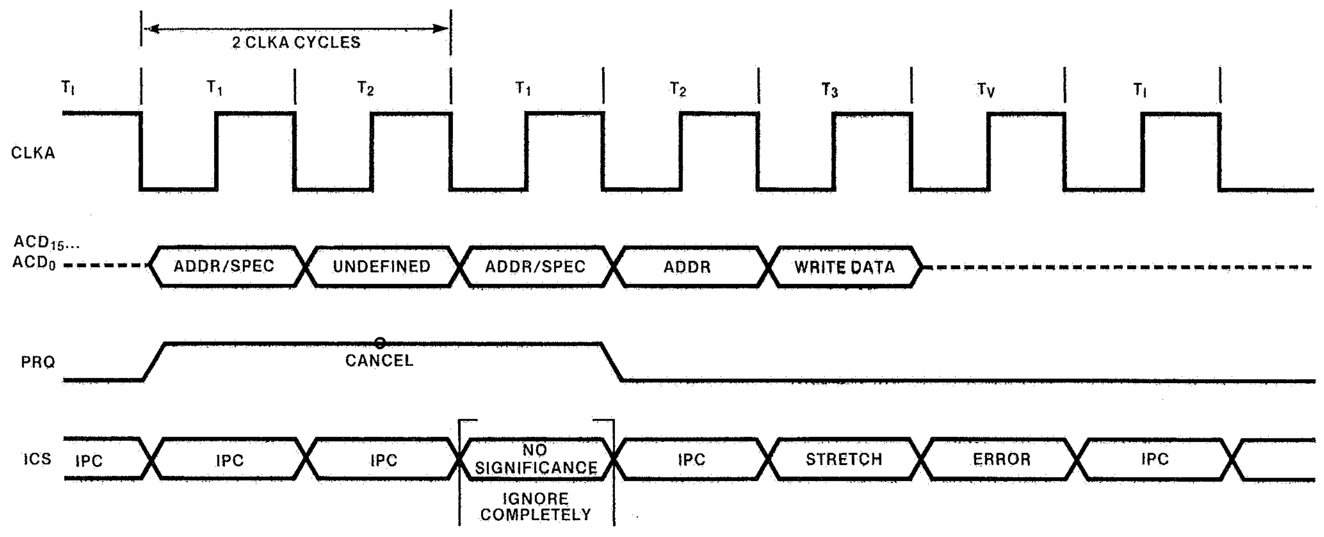
Figure 15. Minimum Read Cycle (Not Buffered) 171873-18



ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Tv**
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-z	Hi-z	Tw
Hi-data1*	Lo-data1	T3
Hi-z	Hi-z	Tv
Hi-z	Hi-z	Ti

*Undefined if single byte read
**(Preceded by a read)

Figure 16. Minimum Read Cycle (Buffered System) 171873-19



ACD15 ACD8	ACD7 ACD0	State
Hi-z	Hi-z	Ti
Spec	Lo-adr	T1
Undefined	Undefined	T2**
Spec	Lo-adr	T1***
Hi-adr	Mid-adr	T2
Hi-data*	Lo-data	T3
Hi-z	Hi-z	Tv
Hi-z	Hi-z	Ti

*Undefined if single byte write
**Access Cancelled
***New Access Started (Slave must support this subsequent access even though all processors may not implement it.)

Figure 17. Minimum Faulted Access Cycle

171873-20

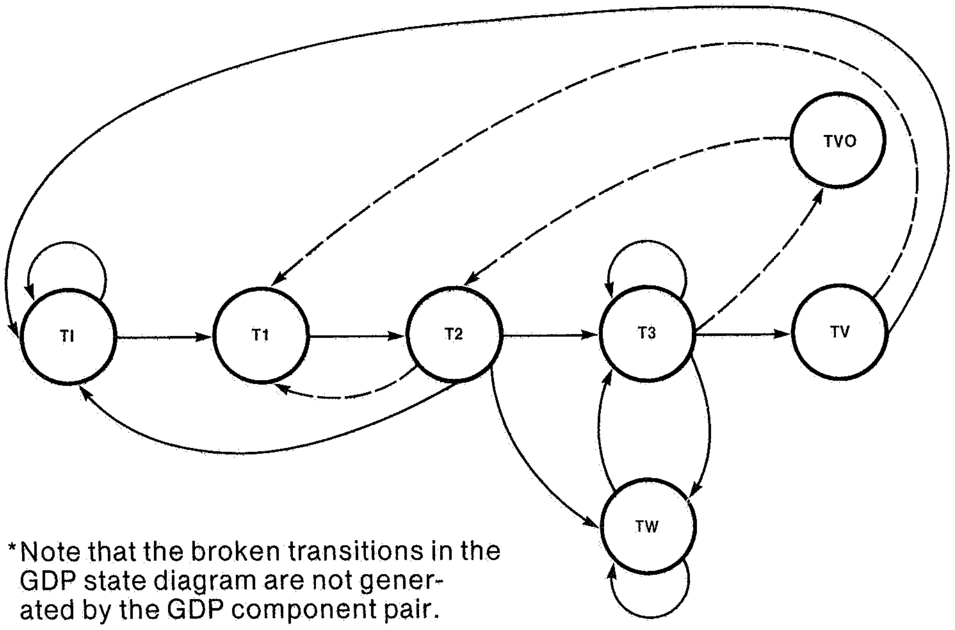

TVO
TI
T1
T2
T3
TV
TW
*Note that the broken transitions in the GDP state diagram are not generated by the GDP component pair.

Figure 18. GDP State Diagram 171873-21

43201 PIN DESCRIPTION

Processor Packet Bus Group

ACD15—ACD0 (Address/Control/Data lines, Inputs, high asserted)

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 11). 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 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 CLKA.

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.

Intra-GDP Bus Group

$UI_{15}...UI_0$ (Microinstruction Bus lines, Outputs, high asserted)

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.

$IS_6...IS_0$ (Interchip Status lines, Inputs, high asserted)

The 43201 receives information pertaining to interchip microprogram status from the 43202 over these lines.

System 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.

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/ results in a microprogram trap which causes the GDP to immediately terminate any bus transactions or internal operations which may be in progress at the time, reset to a known state, assert FATAL/, and await an IPC (which resets the GDP to the same state as INIT/ assertion does). The IPC will not be serviced for at least five clock cycles following CLR/ assertion.

If CLR/ is continuously asserted low for more than one clock cycle, it is ignored during alternate clock cycles (beginning with the second clock cycle) of continuous CLR/ low assertion.

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 checker does not drive the microinstruction bus; rather, it monitors the bus and compares the data on the bus to its internally generated result, signalling disagreement on its HERR/ line. This hardware error detection capability on the 43201 is provided mainly for test purposes. MASTER should be tied to V_{CC} for normal operation and tied low to enable hardware error detection 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 micro-instruction 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. HERR/ can drive one low power Schottky load.

Clock Group

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

Clock A (CLKA) provides the basic timing reference for the 43201. CLKB overlaps CLKA by nominally 1/4 cycle (90 degrees phase shift). All external signals are referenced to CLKA. Refer to the AC 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. If RDROM/ is low when INIT/ goes high, the 43201 goes into a special diagnostic mode. In this mode, with RDROM/ held low, the 43201 microinstruction sequencer steps through the 43201 microprogram ROM, sequentially displaying (but not executing) the 43201 microprogram on the uI_{15}—uI_0 lines. The RDROM/ feature is useful for testing. RDROM/ should be tied to V_{CC} for normal operation and tied low for testing .

Power and Ground Connections

V_{CC} (4 pins)

These pins supply +5 V±10 % referenced to GND pins.

GND (5 pins)

These pins supply ground reference for the 43201.

V_{BB} (Internally Generated)

This pin is connected to the substrate bias voltage of the 43201. An external low leakage 1 microfarad capacitor rated at 5 volts or greater should be used to bypass V_{BB}. V_{BB} is a negative voltage.

N.C. (No Connection, 4 pins)

43202 PIN DESCRIPTION

Processor Packet Bus Group

ACD_{15}—ACD_0 (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:

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

When the 43202 is in checker mode, the ACD pins are monitored by the hardware error checking logic and are in the high impedance mode.

PRQ (Processor Packet bus Request, Three-state Output, high 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, high 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:

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

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

BOUT is used to control external bus transceivers to buffer the 43201, 43202 from the Processor Packet 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. BOUT is not checked by the hardware error detection logic.

Intra-GDP Bus Group

uI_{15}—uI_0 (Microinstruction Bus lines, Inputs, high asserted)

The uI_{15}—uI_0 input lines provide the 43202 with microinstruction information sent from the 43201.

IS_6—IS_0 (Interchip Status lines, Outputs, high asserted)

The IS_6—IS_0 lines drive interchip microprogram status information from the 43202 to the 43201. IS_6—IS_0 are not checked by the hardware error detection logic.

System Group

PCLK/ (Processor Clock, Input, low asserted)

PCLK/ is asserted 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 four clock cycles before being asserted. PCLK/ is synchronous with respect to CLKA, but is generally unrelated to other interface timings.

CLR/ (Clear, Input, low asserted)

Assertion of CLR/ results in a microprogram trap which causes the GDP to immediately terminate any bus transactions or internal operations which may be in progress at the time, reset to a known state, assert FATAL/, and await an IPC (which resets the GDP to the same state as INIT/ assertion does). The IPC will not be serviced for at least five clock cycles following CLR/ assertion.

If CLR/ is continuously asserted low for more than one clock cycle, it is ignored during alternate clock cycles (beginning with the second clock cycle) of continuous CLR/ low assertion.

Hardware Error Detection Group

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

The MASTER input determines whether the 43202 is to function as a master or a checker. In master mode, the 43202 functions normally and drives all of its outputs. In checker mode, ACD_{15}—ACD_0 and PRQ enter the high impedance state and BOUT is unconditionally low. A 43202, whether master or checker, monitors the ACD_{15}—ACD_0 and PRQ lines and compares the data on them to its internally generated result, signalling disagreement on its HERR/ line. For normal operation, MASTER may be either left alone or tied high. MASTER must be tied low to disable the ACD_{15}—ACD_0 and PRQ outputs.

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

HERR/ is asserted low by the 43202 to indicate disagreement between the data appearing on the ACD_{15}—ACD_0 and PRQ pins and the internally generated result of the 43202. HERR/ is valid during CLKA and can normally be asserted by a 43202 every clock cycle. HERR/ is prevented from being asserted low during any clock cycle following a clock cycle in which a CLR/ low assertion is recognized by the 43202. HERR/ requires an external 2.2k ohm nominal pullup resistor.

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 ac electrical characteristics for exact statement of timing relationships. All external signals are referenced to CLKA.

Power and Ground Connections

V_{CC} (Power Supply, 4 pins)

These pins supply +5 V ±10%, referenced to GND pins.

GND (Ground, 5 pins)

These pins supply ground reference for the 43202.

N.C. (No Connection, 7 pins)

INSTRUCTION SET SUMMARY

Refer to Table 14 for the iAPX 432 General Data Processor operator set summary.

43201/43202 ELECTRICAL SPECIFICATIONS

Tables 6 through 13 and Figures 19 through 27 provide appropriate timing diagrams and tables to represent the complete electrical specifications for both the 43201 and 43202 components.

Table 6. iAPX 43201 Electrical Specification

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*	−1V to +7V
Power Dissipation	2.5 Watts

*43201 V_{BB} Pin with respect to GND −5V to 0V

Table 7. iAPX 43201 Electrical Specification

DC Characteristics

VSS = 0 Volts, VCC = 5 Volts ± 10% Ta = 0° C to 70° C

Symbol	Description	Min	Max	Units
Vili	Input Low Voltage IS6...IS0	−0.3	+0.7	V
Vihi	Input High Voltage IS6...IS0	3.0	VCC+0.5	V
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
Vol	Output Low Voltage (Microinstruction Lines) (Iol = −0.1 mA)	0	0.35 V	
Voh	Output High Voltage (Microinstruction Lines) (Ioh = 0.1 mA)	3.25	VCC	V
Vol	Output Low Voltage (Iol = 2.0 mA)	—	0.45	V
Voh	Output High Voltage (Ioh = −400 uA)	2.4	VCC	V
Icc	Power Supply Current (Sum of all VCC Pins)	—	400	mA
Iil	Input Leakage Current	—	±10	uA

Table 7. iAPX 43201 Electrical Specification (Cont'd.)

DC Characteristics

VSS = 0 Volts, VCC = 5 Volts ± 10% Ta = 0° C to 70° C

Symbol	Description	Min	Max	Units
Io	Output Leakage Current	—	±10	uA
Iol	@0.45 Vol			
	HERR/	—	.4	mA
	FATAL/	—	4	mA
	OTHER	—	2	mA
Ioh	@2.4 Voh	—	−0.1	mA

* For operation at 5 MHz or slower, the 43201 may be operated with Vihc minimum of 2.7 Volts.

Table 8. iAPX 43201 AC Characteristics

VCC = 5 ± 10% Ta = 0° C to 70° C

Symbol	Description	8 MHz Min	8 MHz Max	5 MHz Min	5 MHz Max	Unit
tcy	Clock Cycle Time	125	1000	200	1000	nsec.
tr, tf	Clock Rise and Fall Time	0	10	0	10	nsec.
t1, t2, t3, t4	Clock Pulse Widths	26	250	45	250	nsec.
tdc	Signal to Clock Set-up Time	5	—	5	—	nsec.
tcd	Clock to Signal Delay Time	—	55	—	85	nsec.
tis	Init to Signal Hold Time	15	—	20	—	nsec.
tie	Init enable Time	10	—	10	—	tcy
tdh	Clock to Signal Hold Time	25	—	35	—	nsec.
t_{OH}	Clock to Signal Output Hold Time	15	—	20	—	nsec.
tsi	Signal to INIT/ Set-up Time	10	—	10	—	nsec.
tuif	Microinstruction Bus Float Time	0	—	0	—	nsec.

The above specifications are subject to the following definitions and test conditions:

1. Note that tcy=t1 + t2 + t3 + t4 + 2*tr + 2*tf.
2. Pins under consideration were subjected to the following purely capacitive loading:
   C1 = 25 pF on HERR/
   C1 = 50 pF on uI15...uI0, IS6...IS0
   C1 = 70 pF on all remaining pins.
3. All timings are measured with respect to the switching level of 1.5 Volts. The switching point of CLKA and CLKB is referenced to the 1.8 Volt level.
4. CLKA and CLKB must be continuously applied for the 43201 to retain its state.

Table 9. iAPX 43201 Capacitance

Symbol	Parameter	Typical	Unit
Cin	Input Capacitance	6	pF
Cout	Output Capacitance	12	pF
Conditions:	fc=1 MHz, Vin=0V, VCC=5V, Ta=25° C Outputs in High Impedance state		

Table 10. iAPX 43202 Electrical Specification

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	−1V to +7V
Power Dissipation	2.5 Watts

Table 11. iAPX 43202 Electrical Specification

DC Characteristics

VSS = 0 Volts, VCC = 5 Volts ± 10% Ta = 0° C to 70° C

Symbol	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
Vili	Input Low Voltage ul15...ul0	−0.3	+0.7	V
Vihi	Input High Voltage ul15...u10	3.0	VCC+0.5	V
Vol	Output Low Voltage (Iol = 4.0 mA) ACD15...ACD0, PRQ (Iol = 8.0 mA) BOUT, HERR/	—	0.45	V
Voh	Output High Voltage (Ioh = −800 uA)	2.4	VCC	V
Voli	Output Low Voltage IS6...IS0 Ioli = 0.1 mA	—	0.35	V
Vohi	Output High Voltage IS6...IS0 Iohi = 0.1 mA	3.25	—	V
Icc	Power Supply Current (Sum of all VCC Pins)	—	455	mA
Ili	Input Leakage Current except MASTER	—	±10	uA
Ilim	Input Leakage on MASTER	—	−400	uA
Ilo	Output Leakage Current Vo = 0.45V .. VCC	—	±10	uA

* For operation at 5 MHz or slower, the 43202 may be operated with V_{ihc} minimum of 2.7 Volts.

Table 12. iAPX 43202 AC Characteristics

VCC = 5 ± 10% | Ta = 0° C to 70° C

Symbol	Description	8 MHz Min	8 MHz Max	5 MHz Min	5 MHz Max	Unit
tr, tf	Clock Rise and Fall Time	0	10	0	10	nsec.
t1, t2, t3, t4	Clock Pulse Widths	26	250	45	250	nsec.
tcy	Clock Cycle Time (tcy = t1 + t2 + t3 + t4 + 2*tr + 2*tf)	125	1000	200	1000	nsec.
tdc	Signal to Clock Set-up Time	5	—	5	—	nsec.
tcd	Clock to Signal Delay Time	—	55	—	85	nsec.
tdh	Clock to Signal Hold Time	25	—	35	—	nsec.
toh	Clock to Signal Output Hold Time	15	—	20	—	nsec.
ten	Clock to Signal Output Enable Time	15	—	20	—	nsec.
tdf	Clock to Signal Data Float Time	—	55	—	75	nsec.

The timing characteristics given below assume the following loading on output pins. Loading is given in terms of a fixed capacitance plus a DC current load.

Pins	Loading
HERR/	90 pF Iol=8 mA., Open Drain
BOUT	70 pF Iol=8 mA., Ioh=−800 uA
PRQ	70 pF Iol=4 mA., Ioh=−800 uA
IS6...IS0	50 pF MOS only
ACD15...ACD0	70 pF Iol=4 mA., Ioh=−800 uA

All output delays are measured with respect to the falling edge of CLKA except for BOUT. BOUT output delays are measured with respect to the rising edge of CLKA.

All timings are measured with respect to the switching level of 1.5 Volts. The switching point of CLKA and CLKB is referenced to the 1.8V level.

The 43202 is not capable of DC operation. For continuous data and logic state retention the CLKA and CLKB signals must be present.

Table 13. iAPX 43202 Capacitance

Symbol	Parameter	Typical	Unit
Cin	Input Capacitance	6	pF
Cout	Output Capacitance	12	pF
Conditions:	fc=1 MHz, Vin=0, Vout=0, VCC=5.0 V, Ta=25° C. Outputs in High Impedance state		

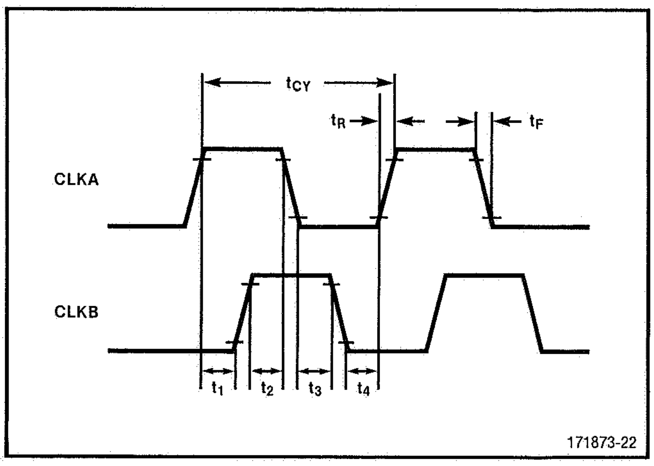


Figure 19. 43201 Clock Input Specification

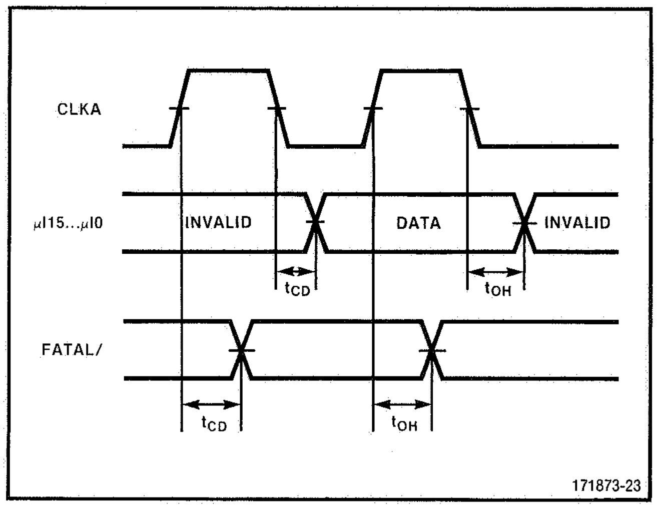


Figure 20. 43201 Output Timing Specification

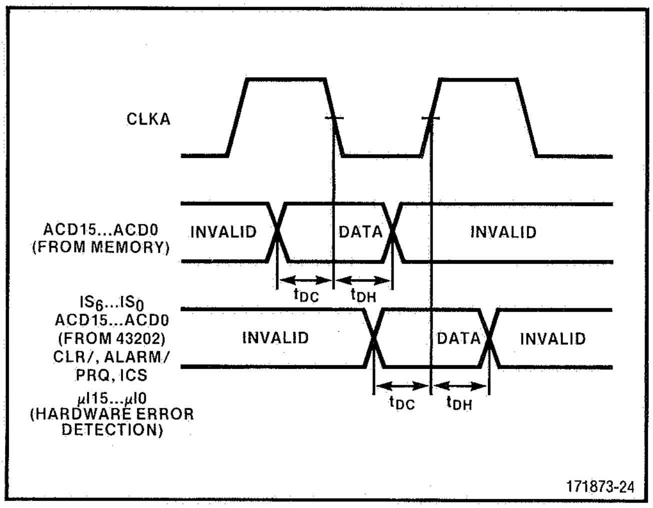

CLKA
ACD15...ACD0 (FROM MEMORY)
INVALID
DATA
INVALID
tDC
tDH
IS6...IS0 ACD15...ACD0 (FROM 43202) CLR/, ALARM/ PRQ, ICS μI15...μI0 (HARDWARE ERROR DETECTION)
INVALID
DATA
INVALID
tDC
tDH
171873-24

Figure 21. 43201 Input Timing Specification

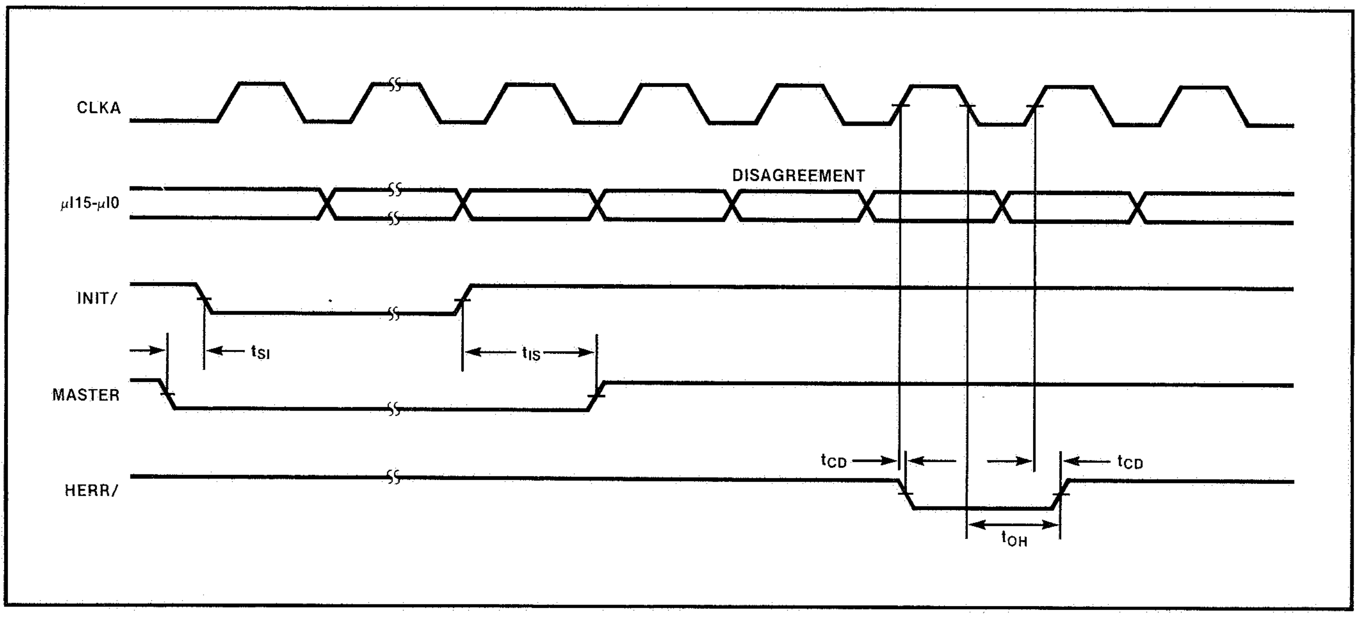


Figure 22. 43201 Hardware Error Detection Timing

171873-25

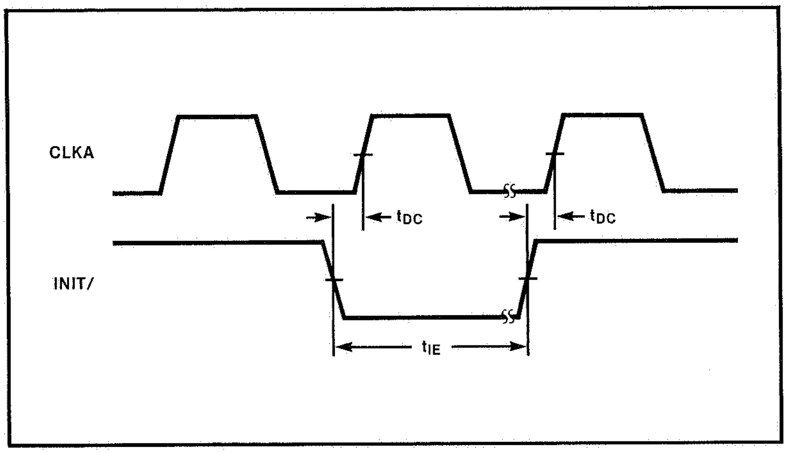


Figure 23. 43201 Initialization Timing 171873-26

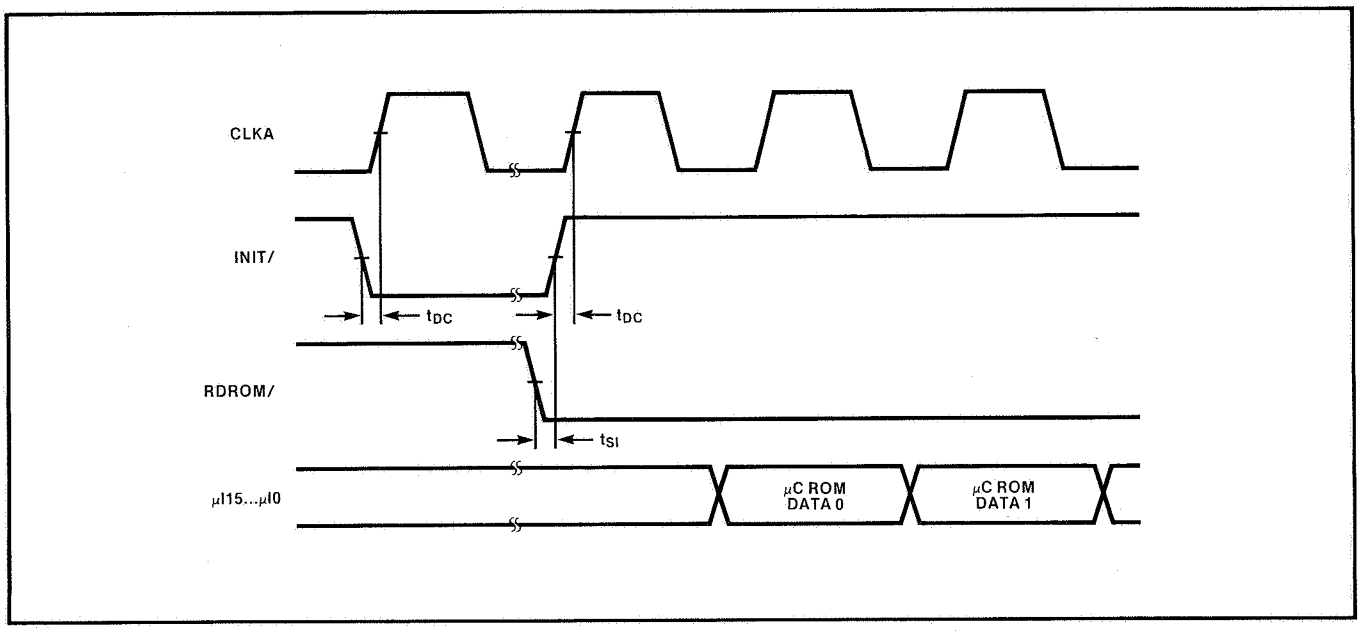


Figure 24. 43201 Microcode Interrogate Timing 171873-27

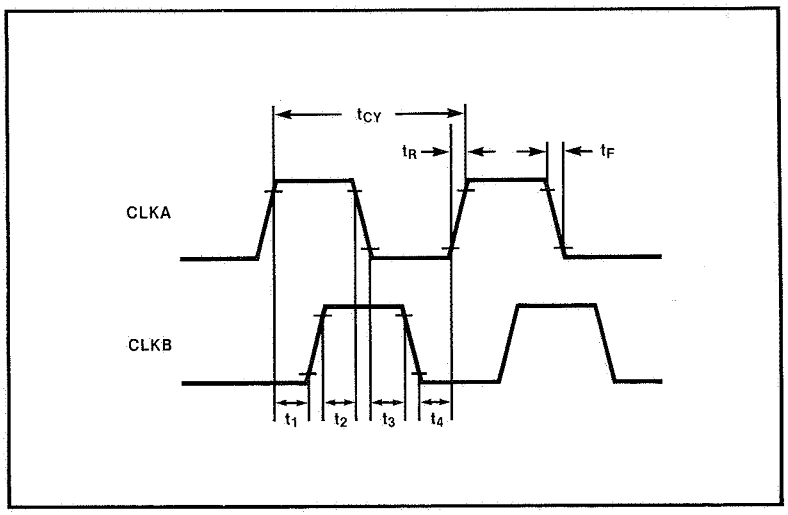


Figure 25. 43202 Clock Input Specification 171873-22

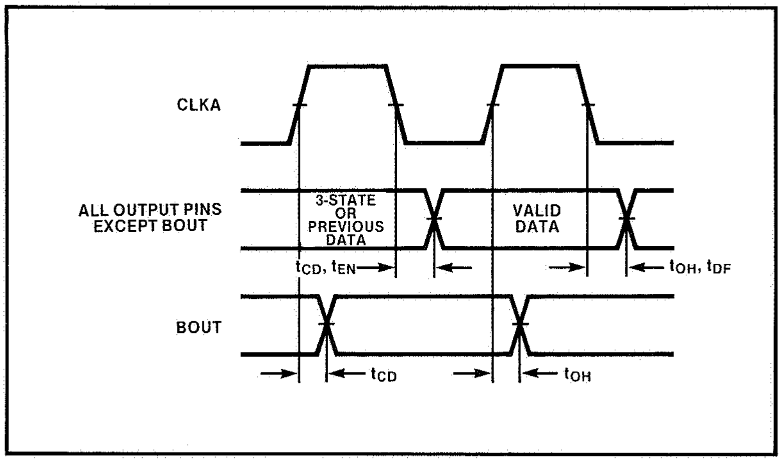


Figure 26. 43202 Output Timing Specification 171873-28

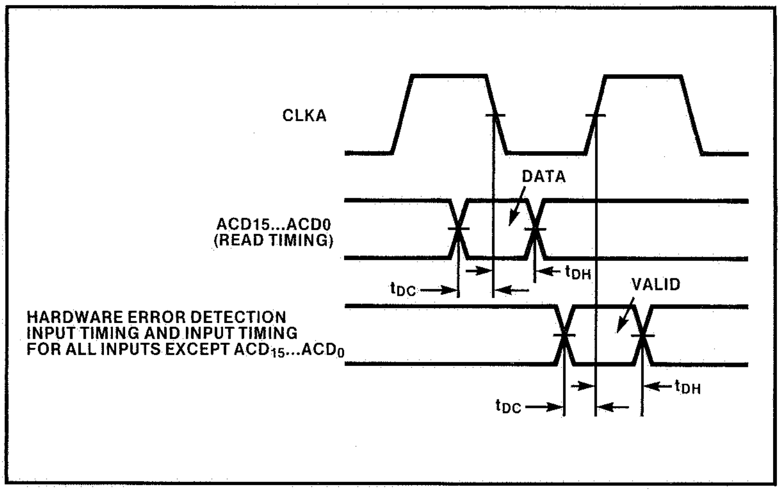

CLKA
DATA
ACD15...ACD0 (READ TIMING)
tDC
tDH
VALID
HARDWARE ERROR DETECTION INPUT TIMING AND INPUT TIMING FOR ALL INPUTS EXCEPT ACD15...ACD0
tDC
tDH

Figure 27. 43202 Input Timing Specification 171873-29

Table 14. General Data Processor Operator Set Summary

Character Operators	Short-Integer Operators	Integer Operators
Move Character Zero Character One Character Save Character AND Character OR Character XOR Character XNOR Character Complement Character Add Character Substract 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	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	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-Ordinal Operators	**Ordinal Operators**	**Short-Real 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	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	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

Table 14. General Data Processor Operator Set Summary (Cont'd.)

Short-Real Operators	Real Operators	Temporary-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	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	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	**Rights Manipulation Operators**	**Type Definition Manipulation Operators**
Copy Access Descriptor Null Access Descriptor	Amplify Rights Restrict Rights	Create Public Type Create Private Type Retrieve Public Type Representation Retrieve Type Representation Retrieve Type Definition
Refinement Operators	**Segment Creation Operators**	**Access Path Inspection Operators**
Create Generic Refinement Create Typed Refinement Retrieve Refined Object	Create Data Segment Create Access Segment Create Typed Segment Create Access Descriptor	Inspect Access Descriptor Inspect Access
Object Interlock Operators	**Branch Operators**	**Interconnect Operators**
Lock Object Unlock Object Indivisibly Add Short Ordinal Indivisibly Add Ordinal Indivisibly Insert Short Ordinal Indivisibly Insert Ordinal	Branch Branch True Branch False Branch Indirect Branch Intersegment Branch Intersegment without Trace Branch Intersegment and Link	Move to Interconnect Move from Interconnect
Process Communication Operators	**Processor Communication Operators**	**Context Communication Operators**
Send Receive Conditional Send Conditional Receive Surrogate Send Surrogate Receive Delay Read Process Clock	Send to Processor Broadcast to Processors Read Processor Status and Clock	Enter Access Segment Enter Global Access Segment Set Context Mode Call Context Call Context with Message Return from Context

intel®

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara 95051 (408) 734-8102 x598
Printed in U.S.A./Y-32/0281/50K/PS/GFH