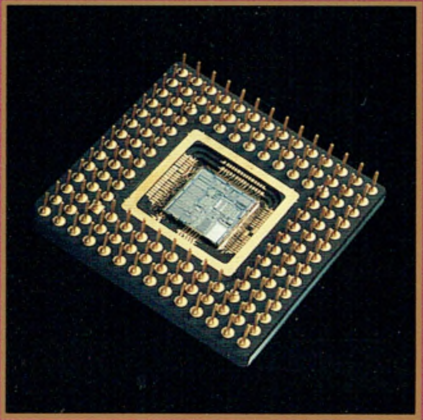
80386
A Collection of Article Reprints



80386

A Collection of Article Reprints

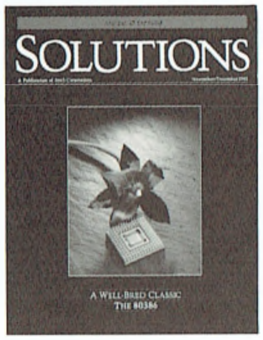
Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Distribution
Mail Stop SC6-59
3065 Bowers Avenue
Santa Clara, CA 95051

3/86

TABLE OF CONTENTS

ElectronicDesign
32-bit microprocessor plays Unix and MS-DOS programs at the same time


SOLUTIONS
A Well-Bred Classic
The 80386

INTRODUCTION

This collection of articles highlights the features, benefits, and market implications of Intel's new 32-bit microprocessor, the 80386. With its high level of performance and uniquely flexible architecture, the 80386 is an excellent match to the needs of a wide range of application areas—from factory machine control to engineering workstations and multiuser departmental computers.

The first three articles provide an overview of the 386, with concentration on application and system software programming models. The flexibility of the 386, as well as the performance characteristics of its architecture are highlighted. Two additional articles discuss hardware design aspects of the 386, including the special features of the 386 that allow implementation of fast, efficient main memory and cache designs. Finally, the last article provides an overview of the implications that the 386 is expected to have on microprocessor and application markets.

I hope you find this set of articles useful in understanding the 386 and the substantial capabilities it offers.

Sincerely,

Gene Hill

Gene Hill
386 Product Line Director

A HAYDEN PUBLICATION

ElectronicDesign®

FOR ENGINEERS AND ENGINEERING MANAGERS — WORLDWIDE

OCTOBER 17, 1985

COMPUTER-AIDED ENGINEERING
• Simulation software • Probabilistic fault grading
• Design system for programmable logic
• Fast general-purpose accelerator

Product Report:
Hybrid ADCs push up speed, add sample-and-hold amps

32-bit microprocessor plays Unix and MS-DOS programs at the same time

DESIGN ENTRY

32-bit microprocessor can run Unix and MS-DOS programs concurrently

On-chip paged and segmented MMUs and 3- to 4-MIPS performance are just two of the noteworthy features that distinguish the latest 32-bit μP.

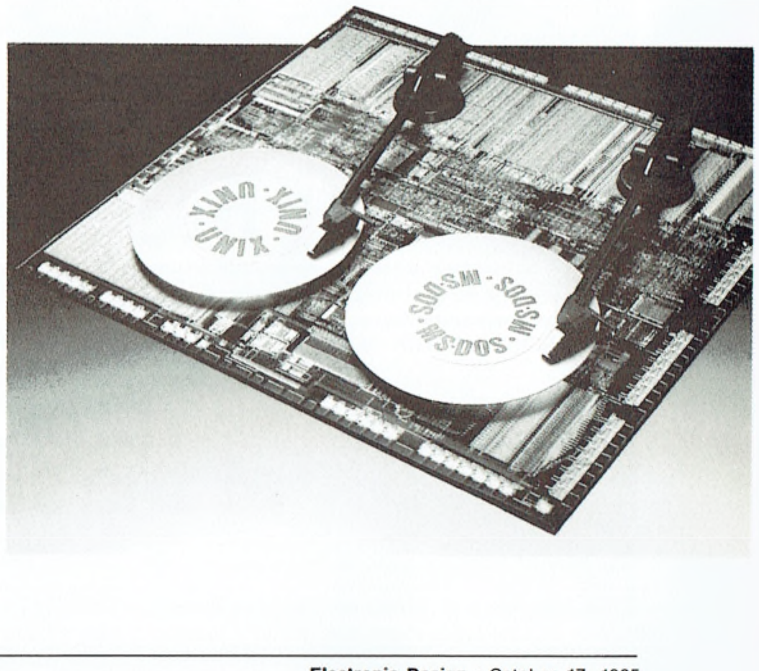
This is the first article in a three-part series on the 80386, the newest 32-bit microprocessor. It concentrates on the chip's hardware. The other articles will focus on the chip's features from the point of view of software, both operating systems and applications programs, and on the system design considerations.

With the appearance of the first 32-bit parts, microprocessor-based systems began to encroach on the performance of minicomputers in applications like engineering workstations, high-resolution graphics, robotics, and office automation. Only large systems, however, could furnish most of the features needed by such advanced applications—primarily throughput and memory support.

System size need no longer be equated with performance. One 32-bit microprocessor now delivers an unprecedented 3 million to 4 million instructions every second. The 80386 also is the first to put paged and segmented memory management units on chip. Thanks in large part to the internal MMUs; to its binary compatibility with its 16-bit predecessors, the 8086 and 80286; and to hardware for fast context switching, it has a virtual machine capability, enabling it to switch between programs written for different operating systems, such as Unix and MS-DOS. Hence it opens the wide world of MS-DOS programs to Unix systems.

Other features that add to the processor's punch are a pipelined architecture, a set of eight 32-bit general-purpose registers, a complete set of instructions for manipulating an extremely wide range of data types, and 11 addressing modes. (Included in the hardware is a barrel shifter that can shift up to 64 bits in a single clock cycle.)

But the processor's accomplishments do not end there. A fast local bus and separate data and address buses, coupled with a two-clock access cycle, result in the highest bus bandwidth of any microprocessor—32 Mbytes/s with a 16-MHz clock. The dual-clock bus timing accommodates high-speed local memories



Timothy J. Keating; Jan Willem L. Prak, PhD; and **Ken Shoemaker**
Intel Corp., 3065 Bowers Ave., Santa Clara, CA 95051; (408) 987-7471.

Cover: 32-bit microprocessor

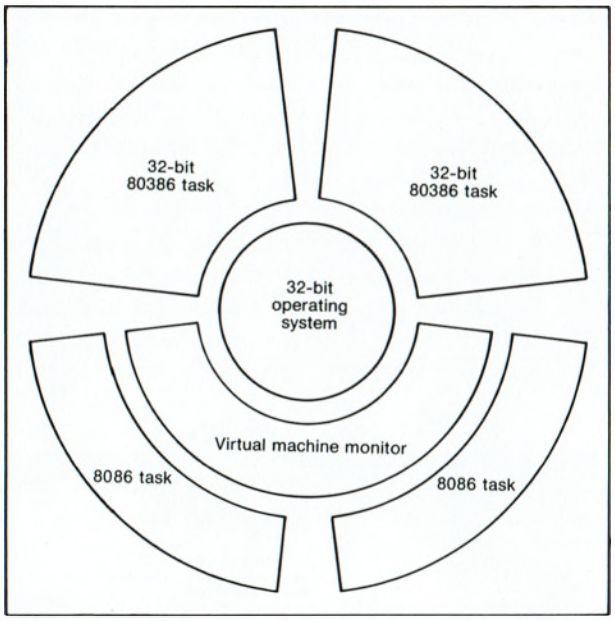
and caches (static RAM); to work comfortably with slower memories (dynamic RAM), addresses can be pipelined.

Finally, the chip carries hardware for system debugging and self-testing to an extent not seen before in microprocessors. Specifically, four debugging registers furnish the ability to set data breakpoints, in addition to the more familiar instruction breakpoints.

Chip holds 275,000 devices

Making possible all that capability on a single chip are over 275,000 transistors. Fabricated using Intel's 1.5-μm CHMOS III process, the 386 operates at a clock frequency of 12 or 16 MHz and consumes no more than 2 W. To allow nonmultiplexed buses and a large number of control lines, the device is housed in a 132-lead ceramic pin-grid array.

For memory management, the segmented and paged architecture, each backed up by a 32-entry cache, allows designers to select the scheme they require. For simple applications, segment sizes can reach 4 Gbytes (the entire physical address space of the chip), thus supplying a linear address space for those programs that require it. The logical address space itself is awesome for a microprocessor—64 terabytes (2^{46} bytes).



1. Suitable operating systems for the 32-bit 80386 microprocessor can run 16-bit applications as tasks by using a virtual machine mode. Both 32- and 16-bit programs—say, those written for Unix and MS-DOS—can run concurrently.

In more sophisticated applications, segmentation ensures protection and modularity for any size code or data structure, while paging affords physical memory management in virtual memory systems. The designer can use segmentation alone, paging alone, or join the two to make the most of their power: The choice of memory management fits the requirements of the wide spectrum of 32-bit systems.

As indicated, designers that have developed programs that run on the processors' forerunners—the 8086, 80186, and 80286 families—will not be left out in the cold. The 80386 serves as a virtual environment that allows 8086 programs to run as guests under new, appropriately written 32-bit operating systems (Fig. 1). Programs created for the 8086 would be encapsulated within an 80386 task, and since 80386 tasks can be protected and paged in virtual memory, those two features would then be available to 8086 programs.

Furthermore, because the segmented memory management and protection mechanism of the 80286 and 80386 are identical, no changes are necessary to the operating system routines in charge of managing the memory. If paging is called for, routines to manage the pages can be added with the help of the segmentation software. Both 16- and 32-bit code can coexist in the same system, and each can even call the other, allowing new 32-bit code to make use of current 16-bit programs.

What's more, multiple 8086 programs can concurrently run on the same 32-bit processor, with each program appearing to have its own processor and address space. Paging handles mapping the processor address spaces to the disk for swapping, and the protection mechanisms come through with all the hardware hooks needed to implement a virtual machine monitor. (A virtual machine monitor, in this case, is a program that maps resources from the virtual machine—the 8086—to the host operating system.) The hooks include I/O and address space protection traps and interrupt traps that allow the monitor to emulate the virtual environment. Operating systems, peripheral devices, and even bit-mapped graphics programs can thus be emulated.

Since memory management is done on chip, the processor avoids the serious performance degradations caused by wait states, which are inherent in systems that use off-chip memory management. Also, since the management functions are always available, memory management software standards are possible—a key point that allows freedom in porting operating systems between different 386-based systems.

Performed by an octet

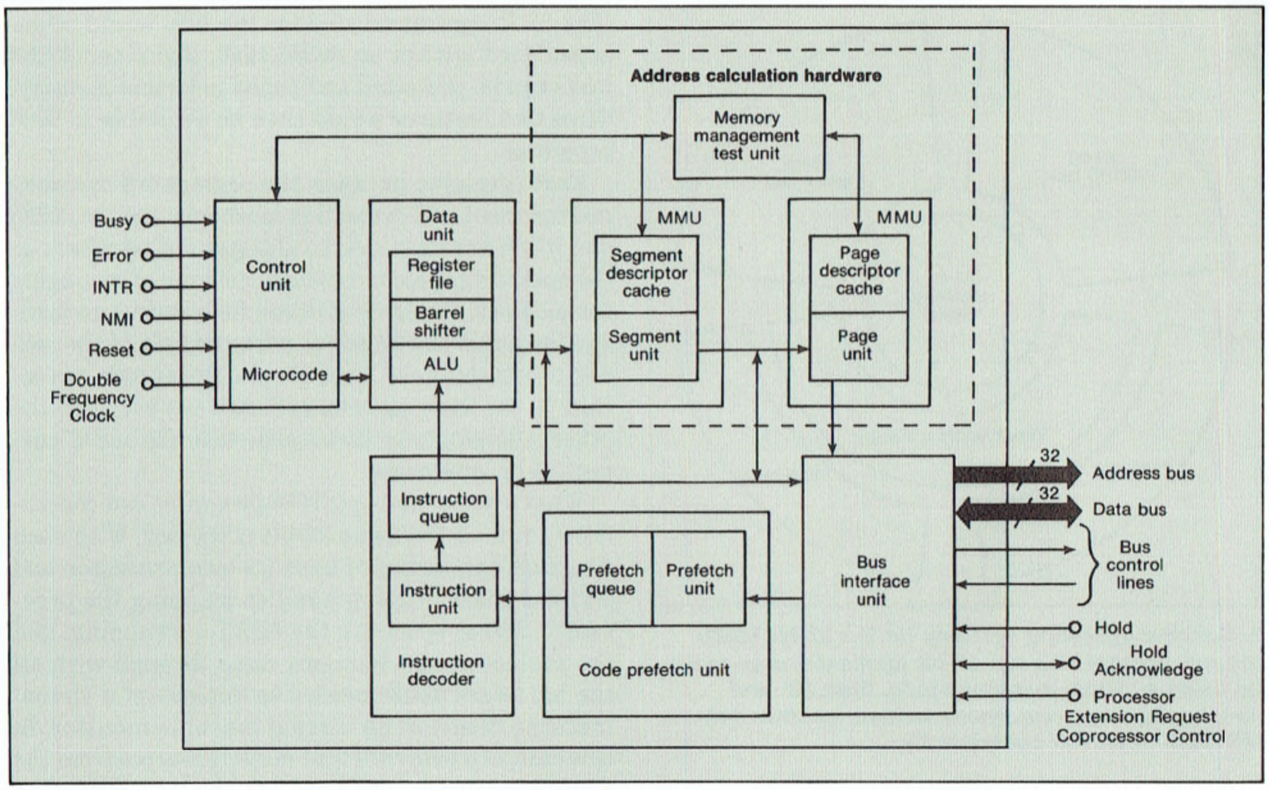
Within the chip the instruction processing is broken up into eight steps, with each performed by a separate optimized processing unit (Fig. 2). That scheme allows every step of any instruction to be completed quickly. It also permits several instructions to be in progress simultaneously, each in a different stage of execution.

The bus interface unit performs external bus accesses; while the bus is idle, the prefetch unit fills an operation-code queue. The instruction decoder deciphers the op codes to fill an instruction queue. In the data unit are the ALU, registers, and the 64-bit barrel shifter, all of which are manipulated by the microcode in the control unit. The latter unit, which also contains interrupt, processor, and coprocessor control circuitry, executes commands that come from the instruction queue.

The segmentation unit converts logical addresses into linear addresses using the chip's segment descriptor cache. The paging unit incorporates a 32-entry descriptor cache to convert linear addresses into physical addresses. Finally, the memory management test unit performs checks for protection violations and for the presence of pages in external memory, as well as other memory management functions.

Fast—and flexible

Although the processor can run at 16 MHz, it does not require premium-speed memories. To take advantage of slower chips for main memory, the chip adds a novel control input, called Next Address. The input allows external hardware to switch from the bus's fast two-clock access to a slower pipelined mode in order to increase the address access time without decreasing the data bus bandwidth. Address pipe-



2. Divided into eight subprocessing units, the 80386 microprocessor takes full advantage of pipelining to permit many of the subunits to operate in parallel. The architecture helps produce a speed of 3 to 4 MIPS.

Cover: 32-bit microprocessor

lining works by overlapping the address of the next bus cycle with the data already on the bus during the last clock of the previous bus cycle. Thus the pipelined cycle has a three-clock-cycle latency from address to data but consumes only two clocks on each of the address and data buses to maintain the 32-Mbyte/s bandwidth.

To remain compatible with 16-bit hardware, the processor can change the bus size during any bus cycle. That allows systems to mix 16- and 32-bit peripherals, buses, and memory boards, thereby enabling users to take advantage of the broad range of readily available peripherals.

An input signal called Bus Size 16 (BS_{16}) can be activated by, say, an external memory system. The chip will automatically convert transfers that are larger than 16 bits or are misaligned into two transfers. The entire function is invisible to the programmer, who can continue to write code independent of the bus width. As a result, the external hardware required to support both main 32-bit memory and 16-bit I/O, EPROM, or system bus is easy to design.

Full sets

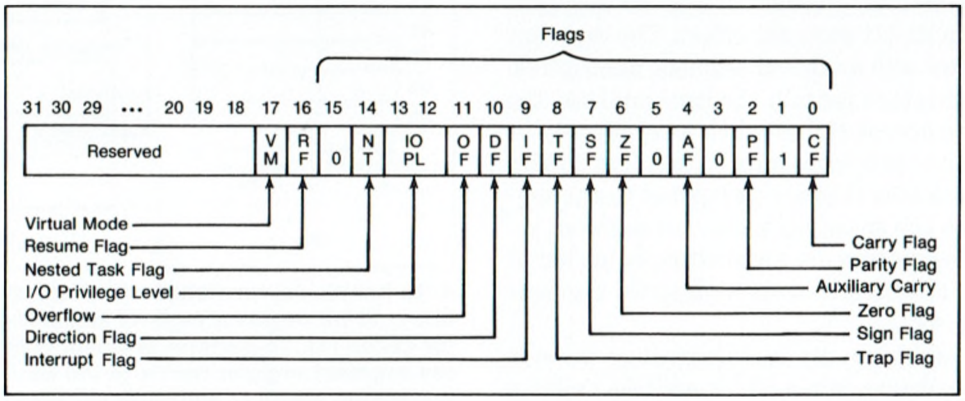
Within the data unit are eight interchangeable 32-bit general-purpose registers. Any of these registers can be used as accumulators for arithmetic or logical operations, as address registers, or as data registers. Additionally, they can handle 8- or 16-bit operations. For the latter, the low word (16 bits) of each register is directly accessible. For 8-bit operations four of the 16-bit registers can be split into pairs of separate 8-bit registers.

Full 32-bit instruction pointer and flags registers record and control the processor state. The instruction pointer contains the full 32-bit offset of the next sequential instruction to be executed. The flags register contains three kinds of flags (Fig. 3). The status flags reflect the outcome of computations such as parity, sign, and overflow. The control flags modify the operation of some instructions—for example, direction flag is set to control self-incrementing/decrementing indexes for block move instructions. System flags control system-wide resources and functions like I/O privilege, external interrupt masking, and virtual machine control.

Many options

The processor's instruction set is a superset of the 8086's that provides for full 32-bit data and address manipulation; among its enhancements are a full complement of bit manipulation commands. Most instructions operate with any general-purpose register and allow for 8-, 16-, or 32-bit operands. A wide spectrum of data types can be handled—integers, bytes, strings and bit strings up to 4 Gbits long.

On the average, the typical instruction length has been kept to just 3.1 bytes, and the average instruction time has been reduced to 4.4 clock cycles. Short instructions and low instruction clock counts form the base for the 386's 3- to 4-MIPS performance. Many factors combined to give these impressive results, some of the most notable being the fast 64-bit barrel shifter and multiplication and division hard-



3. The 13 flags in the 32-bit flags register can be divided into three classes: status, control, and system. The first class indicates the result of computations, the second modifies how some instructions are executed, and the third governs system-wide resources.

ware that yields 32-bit results in 0.5 to 2.4 μs (when the chip runs at 16 MHz). The barrel shifter operates in parallel with the ALU to speed bit manipulations, multiplication and division, and shifting and rotation.

Virtual memory management makes it possible for programs larger than physical memory to be transparently mapped from secondary storage to physical memory. Consequently, very large programs can be written without regard to the main memory size of different target systems. Address translation, swapping, and protection data are kept in memory-based lookup tables. Operating systems and memory management hardware use this information to translate from logical into linear into physical addresses and make program protection checks.

Within the processor, the first step in the translation is through the segmentation unit. Segmentation manages each user's logical memory—different-sized segments (up to 4 Gbytes) can be set up for code modules and different types of data (file buffers, program variables, work areas, and so forth). The programmer can also break up logical memory according to the organization of program routines, thus permitting different routines to be modularly compiled at separate times. By nature, both code and data vary in size; segmentation ensures the right virtual memory fit, assigning only as much memory as needed for the individual routines and data. It is a way of directly relating memory organization and usage to program code and data structures.

Logical addresses for segmentation

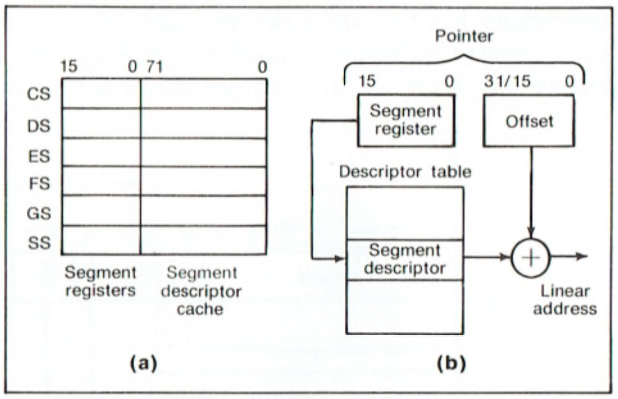
Under segmentation, the addresses that the programmer sees are called logical addresses. The logical address (48 bits) consists of a 16-bit segment selector and a 32-bit segment offset. The segment selector indexes into a table of segment descriptors. Segment descriptors contain the base address, the limit, and the access rights (such as read-only or read/write) and " privilege" information that specify the type of accesses that are permitted to the segment. To keep the speed high, current segment selectors and descriptors are automatically loaded at program initialization into six high-speed segment registers (Fig. 4a).

The 48-bit addresses are transformed by the segmentation hardware into a 32-bit address called a linear address. That transformation is performed by adding the segment offset to the 32-bit segment base value held in a segment register (Fig. 4b). In doing the transformation, the segmentation hardware checks the access against the limit and access rights for the segment used to ensure the integrity of the memory reference. If the chip's paging mechanism is disabled, the calculated linear address becomes the physical address used by the processor.

Further, programmers need only manipulate offsets as effective addresses. Instructions need not explicitly specify which segment register is used, because the correct segment register is automatically chosen for each instruction. Code references use the code segment (CS) register, data references use the data segment (DS) register, and stack references use the stack segment (SS) register. Three extra, uncommitted segment registers add flexibility—for example, when manipulating multiple data structures. The extra registers (ES, FS and GS), reflect the way programs are written, that is, as independent modules that require areas for code and data, a stack, and access to external data areas.

Since there is the potential for segments as large as 4 Gbytes, an efficient mechanism to manage physical memory becomes necessary. Although such management is not an issue for static systems, transferring large segments of hundreds of kilobytes between disk and physical memory can be slow and cumbersome in dynamic systems. Consequently, the processor uses its paged memory manager to handle the physical memory.

When operating under segmentation, the chip breaks up large segments into small fixed-size



4. To handle segmentation, 48-bit words are divided into a 16-bit segment register and a 72-bit descriptor cache (a). To perform segment computations, the segment register points to the desired segment descriptor, which is automatically loaded into the cache, and the descriptor is added to the offset value to form a 32-bit linear address (b).

Cover: 32-bit microprocessor

blocks. Instead of loading the entire segment, the operating system loads the individual pages as needed. That simplifies the management of physical memory. It also allows for small amounts of physical memory to be allocated for each task; rather than having many megabytes per task present in physical memory, the operating system only needs to keep a current working set of pages. The approach takes advantage of the locality of reference displayed by most programs and supports efficient management of physical memory, both main memory and swapping storage.

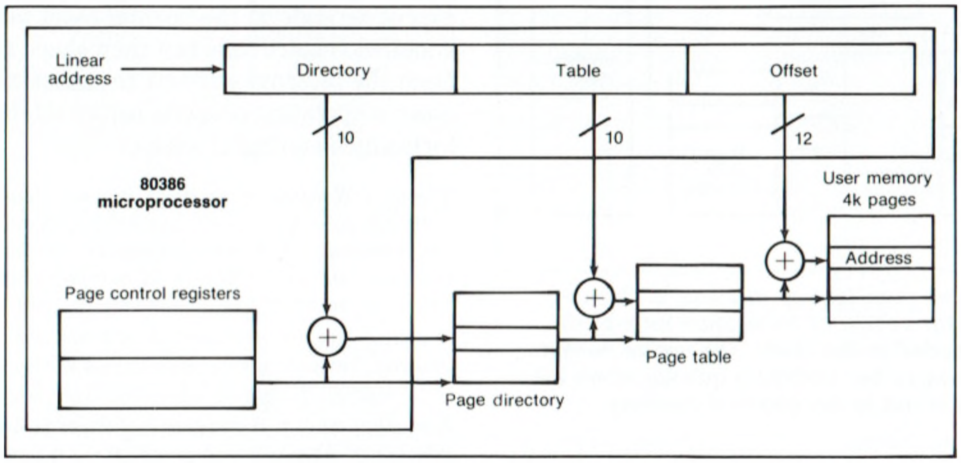
The page-based protection mechanism allows easy implementation of a demand-paged virtual memory system. To support page-based virtual memory, each 4-kbyte page can be designated as present or not present, dirty, accessed, user/supervisor, or read/write. If a page is not present, it must be restored to physical memory before it can be accessed. Whenever a page that is not present is accessed, the processor signals the operating system by raising a page fault indicator. The operating system page fault handler would typically pull the page in from disk, and then the faulting instruction is restarted. Every instruction may be restarted after a page fault.

The programmer controls paging through two levels of memory-based tables and two internal registers (Fig. 5). At the lower level, the page tables map the pages; at the higher level a page directory maps the page tables. An entry in a page directory or a page table requires four bytes, and each page directory and each page table is 4 kbytes long. Thus each contains 1024 entries. Hence only a single page directory, mapping 1024 tables, need be used to map the entire 4-Gbyte linear address space.

To ensure high performance, the memory-based page tables are not actually referenced for each physical address translation; instead the chip includes a 4-kbit associative cache for paging information. The cache, which is called a translation look-aside buffer, or TLB, contains the most recently used linear addresses and their translated physical addresses.

The cache is implemented with on-chip hardware consisting of control logic and a 32-entry page descriptor cache (Fig. 6). Since the address translation caches are part of the memory management unit, address translation can be done in parallel with other CPU activities.

The cache is organized as a four-way set-associative type. With such a cache and the 4k-page size, the processor can internally hold the mapping information for 128 kbytes of memory. Simulations show that this setup provides a hit ratio of greater than 98%, thus minimizing the performance impact of enabling the paging. The address generation time is the same when paging is enabled or disabled; however, there is a small performance penalty paid (2% to 4%) when page table entries must be fetched from exter-



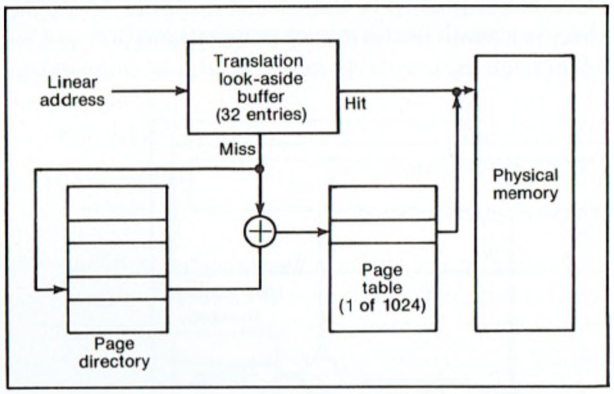
5. The two-level paging scheme starts with the page directory and then moves down to the page table, which maps the pages. Once the page is located, its address is added to an offset value to access the desired address in main memory.

nal tables on cache misses.

To allow multiple programs to execute concurrently, the operating system must be able to support multitasking. Because task switching occurs so frequently, special high-speed hardware to perform the operation was included on the chip, permitting a complete task switch with a single instruction or in response to an interrupt. A 16-MHz processor can save the state of one task (all registers), load the state of another task (all registers, including segment and paging registers if required), and resume execution in less than 17 μs.

Avoiding accidents

One of the most common problems in multitasking systems occurs when a task accidentally violates the address space of another, thereby using or modifying the other task's code or data. To counter that problem, the processor permits any task to have a separate address space, which is enforced by hardware. The protection mechanism also offers up to four privilege levels to protect sensitive code and data within a task. Additionally, read, write, read/write, and execution privilege can be granted on a segment or page basis. That allows a combination of operating system, system services, shared libraries, and application programs to reside in a common virtual address space, yet still to be protected.



6. To speed the page lookup process, a 32-entry page descriptor cache, or translation look-aside buffer, is included in the MMU. The cache permits page addresses to be looked up quickly when the desired page is not in the physical memory.

For software design, one of the most time-consuming phases is integration and debugging. External program debugging tools are used to test and monitor the operation of each module as it interacts with other modules, but hardware support for these software tools has been very limited, with only simple instruction breakpoint interrupts and single-step capabilities.

The processor contains a set of four debugging (breakpoint) registers into which linear addresses can be placed by a software debugger. If the processor attempts an access to the address in one of these registers, it will trap to the debugger. The processor can be set to trap when a memory read, a memory write, either a memory read or a write, or an instruction execution is attempted to the address in one of these registers. These registers thus simplify the design of sophisticated debuggers and allow the designer to implement features that were not possible without hardware support, such as breakpoints on data reads and writes and breakpoints in ROM.

To simplify the design of such diagnostics as power-up system confidence tests, the processor can be directed to test itself upon reset. When so requested, it enters a mode in which it tests over 75% of its 275,000 transistors. After the check is performed, a result signature is placed in two of the chip's general-purpose registers and the processor begins executing code at the normal reset location. User-supplied restart code can then check the signature from the internal self-test to ensure that the processor is operating properly before attempting to perform any meaningful work.□

Timothy Keating, a product manager, has been with Intel for four years and has held various positions in the company's U.S. and European operations. He received a bachelor's degree in computer and electrical engineering from the University of Santa Barbara.

A senior project manager, Jan Prak also has worked at Intel for four years. He earned doctoral and master's degrees at North Carolina State University.

Ken Shoemaker, a graduate of Purdue University's Electrical Engineering program, is a design engineer.

32-bit μP is a fine match for today's languages and operating systems

By supporting the data types, constructs, and other features of modern languages, a microcomputer can outperform minis while remaining compatible with its predecessors.

This is the second article in a series on the 80386 microprocessor. The first part, which appeared in the Oct. 17 issue (p. 115), focused on the chip's architecture. The final installment will cover system design considerations.

For many microprocessor-based systems, software has become the dominant development consideration. All too often, it also is the premier cause of delays. Systems like multitasking engineering workstations, multiple-cell process controllers, and multiuser transaction processors usually incorporate sophisticated software. To be successful, then, a high-end microprocessor must satisfy stringent software demands.

First and foremost, a 32-bit processor must accommodate efficient compilers. That way both operating systems and application programs will reflect the device's inherent abilities. Second, the chip must also deliver the high throughput required by advanced software. Third, the chip must support the large logical address spaces required for programs run by both engineering and business workstations. Fourth, ensuring that confidential data remains secure is a key concern in military and financial applications. Finally, compatibility with previous microprocessors is highly desirable, so that existing software remains usable.

The 32-bit 80386 makes the grade on all of these issues. Its extensive register set, as well as some of the types of data it handles, is geared to writing efficient compilers and guarantees fast context-switching in real-time systems. Further, its rich set of data types and operators make sure that it is fluent in nearly all of today's sophisticated languages. The chip's segmentation and paging units fill the bill when the large logical address spaces are the order of the day, and its levels of protected memories keep the most sensitive data safe. Lastly, to ensure that already written code remains current, the processor's protected mode enables it to execute unaltered 8086 and 80286 code automatically.

Rakesh Agarwal, **Greg Blanck,** and **Dana Krelle**
Intel Corp., 3065 Bowers Ave., Santa Clara, CA 95051; (408) 987-5778.

Unrestricted registers

Naturally, the chip's power is rooted in its main architectural features. Its set of 32-bit general-purpose registers, for one, may be used, without restriction, either to carry out calculations or to form memory addresses. The eight registers (EAX, EBX, ECX, EDX, ESP, EBP, ESI, and EDI) supply ample room to implement the most common addressing modes, or to accommodate a generous set of register variables without wasting silicon. Furthermore, the set incorporates the 286's 16-bit registers, some of which consist of 8-bit segments.

Three control registers and the flags register oversee machine behaviour and report the status of various operations. For their part, the six segment registers (CS, DS, ES, FS, GS and SS) can structure the 64-terabyte (2^{46}-byte) address space into separate logical spaces, or segments (Fig. 1). Each

Reprinted with permission from Electronic Design, Vol. 33, No. 25; copyright Hayden Publishing Co., Inc., 1985.

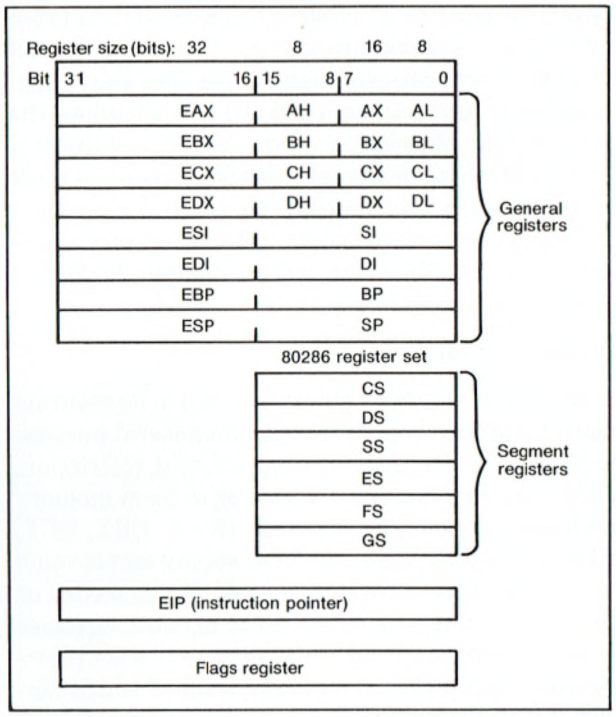
program can then have six logical address spaces mapped at one time, including four data spaces. Furthermore, six debugging registers permit up to four code or data breakpoints to be defined, making for previously unrealizable debugging ease.

One common use of registers is to store operands. For instance, to perform the simple computation Distance = Speed × Time, the following instruction sequence might be used:

```
MOV    EDI, ESI    ;   Save Speed in EDI
IMUL   EDI, EBX    ;   Multiply Speed with Time
```

Here, Speed is originally stored in ESI and Time in EBX; the result is stored in EDI. When it comes time to write the compiler, such a scheme is especially effective for languages like C that define register variables. The orthogonal qualities of the processor's addressing modes provide the user with several register variables.

If the variables do not reside in registers, operands must be fetched from memory. In its simplest form,



1. In the 80386 register set, the general registers can be used to store 8-, 16-, and 32-bit data (highlighted). Together with part of the segment registers they can substitute for the 80286 register set to execute 16-bit programs (overlay). The instruction pointer covers the full address space, and the large set of flags supports complex systems.

the effective address (or byte index) of an operand within a given logical address space can be directly specified in an instruction. A particular memory address, however, is often not known until the program is actually executing—say, in accessing a dynamic array and executing relocatable code. In those cases, the microprocessor forms an effective address by summing the contents of one, or two, general-purpose registers and an optional constant value, or displacement.

An effective address, EA, is generated by:

$$\text{Base} + \text{Index} \times \text{Scale} + \text{Displacement}$$

where the base value can be stored in any general register, and the index in any general register other than ESP, whose contents always point to the current stack and can be easily used for stack-relative addressing. The scale is a constant value—1, 2, 4, or 8 (depending on word length)—by which the index may be multiplied. The displacement field is also a constant; its value may range anywhere from -2^{31} to $+2^{31}-1$. Because the base, index, and displacement are optional, as is the scale when an index is present, the processor supplies 11 distinct 32-bit addressing modes. That set contains all of the most commonly used modes employed by compilers for advanced high-level languages (Table 1). In addition, direct and register operands are also furnished.

Base displacement

Assume that parameters are passed to a subroutine by pushing them onto a stack. The last parameter could be obtained from register EAX in two clock cycles by using the base-displacement mode:

```
MOV        DWORD PTR [ESP+4] , EAX
```

Similarly, suppose that the pointer nextRecord is stored in register EAX. In four clock cycles, it can be linked into a data structure—without additional overhead—using the instruction:

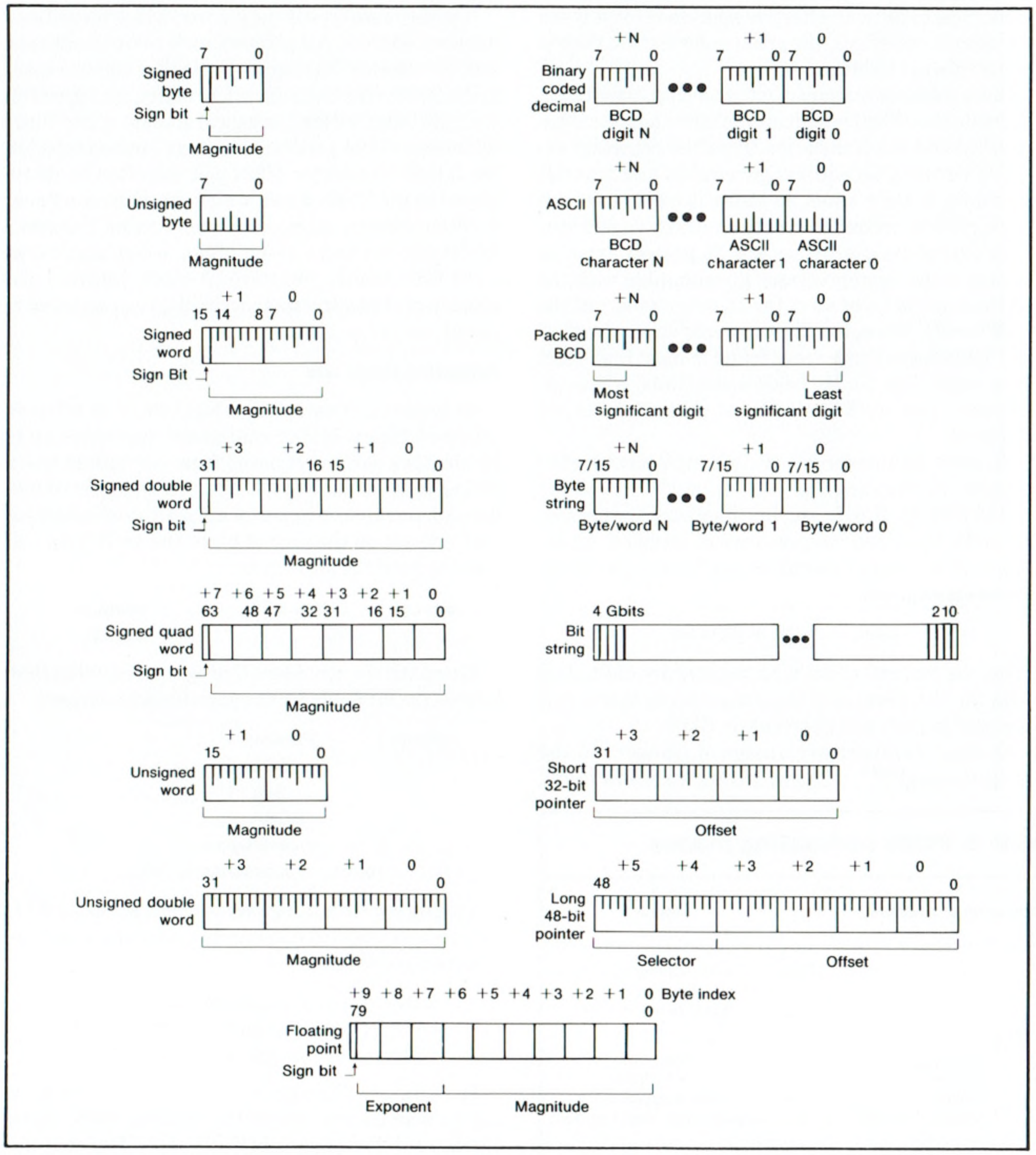
```
MOV     EAX, DWORD PTR [EAX+LinkOffset]
```

Finally, imagine that V points to a dynamically allocated integer vector and that the term V[i+2] should be initialized to 5. If V is loaded into the ESI register and is a register variable stored in EDI, the following statement performs the desired operation in three clock cycles, working with the based scaled-index displacement mode:

```
MOV     DWORD PTR     [ESI+EDI*4 + 8] , 5
```

The use of scaled indexing to generate a byte index into a 4-byte integer array eliminates the need for another instruction to generate the byte index.

The processor's high performance is manifested in the low clock counts required for each of the instructions. A lid is kept on clock counts because, when executing in-line code, the effective address formation for a given instruction starts during the last clock cy-



2. Binary coded decimals, bit and byte strings, and floating-point integers are among the numerous data types that the processor accommodates. Such dexterity lets the device meet the needs of virtually all high-level languages. (Unless otherwise noted the left-most digit is the most significant one.)

cle of the previous instruction. Since it takes only one clock cycle to form an effective address (except when an index is specified), the address formation time is almost always hidden.

Since memory management is on chip, time taken to create the effective address includes the logical-to-physical address translation. Thus the processor exhibits virtually no address generation and translation delays. Only when an index is used, a second clock cycle is needed to create an effective address.

To extend its usefulness and to protect existing software, the microprocessor is compatible with the addressing modes of all of the 16-bit members of the 8086 family. Moreover, the processor supports all of the fundamental data types found in most high-level languages (Fig. 2). Its basic operations, when applicable, can work with any of these data types (Table 2).

To start, an integer is a two's complement signed number in the range of -2^{31} to $+2^{31}-1$ (about ± 2.15 billion). If only smaller numbers are needed, 16- and 8-bit signed integers are also supplied. An example of an integer operation can be seen in the instruction sequence:

```
IMUL    EBX, V [EDX*4]
```

Here, the contents of the EBX register are multiplied with the Nth element of the integer array V, where N is stored in EBX and the result in EDX.

Ordinal numbers are unsigned integers in the range 0 though $2^{32}-1$. As with integers, 32-bit 16-bit, and 8-bit variants of ordinals are available.

Another data type is the pointer, which identifies a memory address. All pointers have two components: a 16-bit selector that names the logical address space and a 32-bit effective address, or offset, that specifies the byte index within the logical address space. Since addresses can be generated with an implied selector, the full 48-bit selector-offset pair can often be abbreviated as the 32-bit offset, which can address a linear 4-Gbyte address space. A compact version features a 16-bit selector and a 16-bit offset, which also works with 8086 family software (A more detailed discussion of the addressing mechanism can be found in part 1, Oct. 17, p. 115).

Accessing single bits

In system software, individual bits must often be accessed. Single bits or contiguous sequences up to 31 bits long can be retrieved from—or stored in—a string that extends to 4 Gbits. An example implements a semaphore by performing an atomic (locked) test-and-set on the second bit of the sema byte and waiting until the bit is clear:

```
waitLp:    LOCK BTS        sema, 2
           JC              waitLp
```

Alternatively, consider a C program fragment that defines the bit fields for the variable accessRights:

```
struct {        Present: 1;
                    DPL : 2;
                    Seg : 1;
                   ECRA : 4;
          }      accessRights;
          regVar = accessRights.DPL;
```

Assuming regVar is a register variable stored in ESI, this code translates directly into the processor instruction sequence:

```
MOV     CL, 2; setup field length
MOV     EAX, 5; setup field offset (1+4)
XBTS    ESI, accessRights, EAX, CL
```

To facilitate floating-point operations, the chip can be teamed with either the 80287 or 80387 math coprocessor. Both augment the types of data that the processor manipulates to encompass 64-bit long integers, 18-digit BCD integers, 32-bit single-real values, 64-bit double-real formats, and 80-bit extended-real digits. The coprocessors also furnish a comprehensive set of numeric instructions suited to these data types.

Table 1. 80386 addressing modes

Components of address			Mode
		Disp	Displacement only
Base			Base only
Base		+ Disp	Base displacement
	Index		Index only
	Index × Scale		Scaled index
	Index	+ Disp	Index displacement
	Index × Scale	+ Disp	Scaled index displacement
Base	+ Index		Based index
Base	+ Index × Scale		Based scaled index
Base	+ Index	+ Disp	Based index displacement
Base	+ Index × Scale	+ Disp	Based scaled index displacement
Register			Register operand
Immediate			Immediate operand

To accommodate BCD numbers, primitives are supplied that help set up both packed and unpacked decimal arithmetic. In addition, arithmetic operations on 80-bit packed decimal numbers are fielded by the math coprocessors.

The processor also is targeted for 64-bit operations such as multiplying integers and ordinals. Divisions that yield 64-bit products and dividends are also within its grasp, and multiple precision addition and substraction are implemented easily with the help of add-with-carry and subtract-with-carry operations.

More than math

Beyond arithmetic, the processor also carries out 64-bit shifts with 32-bit results. Those calculations are useful for creating high-performance loops to process unaligned, memory-based operands. A routine to move a bit string from one location to another —one double word at a time—might be desired in a bit-mapped graphics system (see the program, p. 167). First, however, the counter variable must be set and the source pointer, SP, incremented. Assume here that register ESI holds a double-word-aligned source address. Register CL stores the bit offset BO that points to the start of the bit string, and EDI contains the destination address.

The routine assumes that the destination is double-word aligned. If it is not (i.e., if an arbitrary bit field is selected), it is only necessary to move sufficient bits from the source to the destination to align the latter. Then the program can be used.

So far the processor has been looked at only from the application program's point of view. Equally important is the view from the side of the operating system that manages the application environment. By including many of the features required by an advanced operating system on chip, the processor establishes the framework to design fast, simple, and standard operating systems.

Such operating-system support is totally invisible to the application program. Functions featured on chip include a versatile memory management unit that encompasses segmentation and paging. Also, fast task switching in hardware facilitates efficient multitasking, while the ability to dynamically execute and address 16- and 32-bit code further enhances the chip's versatility.

An operating system, in addition to allocating memory, must isolate and protect each application from all others. Both segmenting and paging can be used to enforce this isolation. For example, the access control field of the segment descriptor handles many different types of information. A read-only segment, say, could be defined to hold sensitive system configuration data. Alternatively, an execute-only code segment might be created to ensure that a program will not be corrupted. Other segment types, such as the task state, are on hand to improve execution speed. And an operating system is free to dynamically create segments as required by the executing application. In fact, the chip can assign up to 16,384 segments to a task. Combined with the 32-bit offsets,

Table 2. Basic operations on expression types

	Type of expression					
Operation	**Integer**	**Ordinal**	**String**	**Bit string**	**Floating-point**	**BCD**
Move To and from memory, convert precision	Yes	Yes	Yes	Yes	Yes[1]	Yes[2]
Arithmetic Add, subtract, multiply, divide, negate	Yes	Yes	No	No	Yes[1]	Yes[2]
Logical And, Or, exclusive OR, shift	Yes	Yes	No	No	No	No
Compare	Yes	Yes	Yes	Yes	Yes[1]	Yes[2]
Transcendental functions	No	No	No	No	Yes[1]	No

[1]Available directly with 80287 or 80387 math processors.
[2]ASCII and decimal adjustment instructions are supplied to implement efficient loops. BCD is supported directly by the 80287 and 80387 math coprocessors.

that makes for the aforementioned logical address space of 64 terabytes that is available for each task. Besides relocation and protection, descriptors also deliver semantic controls for each segment, used to specify the word length of its contents.

Because the processor's segmentation mechanism is capable of incorporating various degrees of protection in a system, the integrity of a program address is always guaranteed. Specifically, an application-generated offset is added to the segment's base value to create the linear address. If the offset is greater than that allowed by the segment descriptor's limit field, a protection violation occurs. A semantic control bit, called the Granularity bit (G bit), interprets the limit field within the descriptor: A value of 0 indicates an absolute limit (byte-granular limit); 1 signifies that the restriction specifies a page (page-granular limit).

Levels of privilege

The limit fields are supplemented by program level protection, which is based on privilege levels. The access right byte of each segment descriptor contains a descriptor privilege level (DPL) that identifies the access permitted for a particular segment. Privilege levels range from 0 to 3, with the first being the most privileged (Fig. 3). At any point, the processor's current privilege level (CPL) is determined by the privilege level of the executing code segment.

As a rule, a program has the right to access segments that are no more privileged than itself and to call other programs that are at least as privileged. Interlevel transfers are fielded by a special descriptor, the call gate, which furnishes a controlled entry point to more privileged routines.

Intertask protection is furthered by defining an individual local descriptor table (LDT) for every task. Using a separate LDT logically isolates each applications address space.

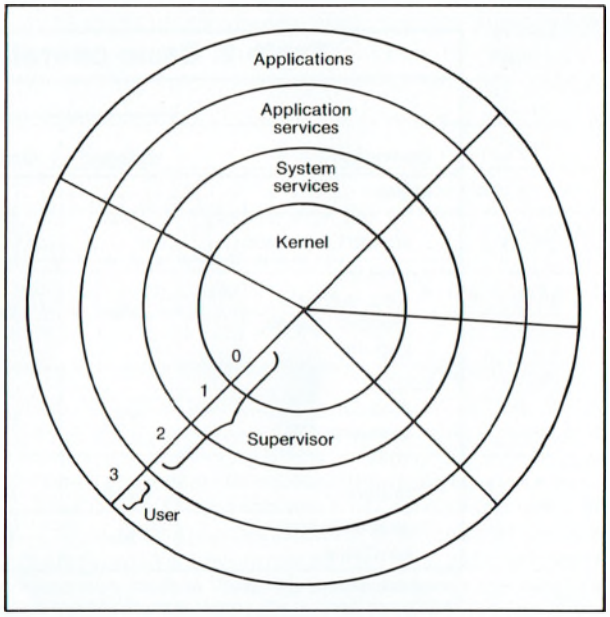
Several other semantic control bits reveal the processor's power. One, the Default Size bit (D bit) determines whether code segments will assume 32-bit (D is set to 1) or 16-bit (D is set to 0) operands and operations. The Expand Down bit (E bit) defines whether a given data segment needs an expand-down or expand-up stack, and interprets the limit field accordingly. Importantly, all of these semantic controls permit dynamic switching between 16- and 32-bit code, segment by segment. That power makes it easy to vary between unchanged 8086 or 80286 software and new 32-bit code.

Double-level page map

The processor incorporates a two-level page translation mechanism that allows any 4-kbyte linear address page to be mapped into any arbitrary 4-kbyte section of physical memory. The double-level page map for the current task is located in memory by a pointer in control register 3 (CR3), which is also called the page directory base register. All page tables are located on page boundaries, to ensure that they are efficiently manipulated. For each page in the logical memory space, a page descriptor is defined (Fig. 4). It contains the physical address corresponding to each logical page, as well as other page

A double-word shift

```
       MOV EBX, Length      ; Set loop counter to
                                string length
       LODS EAX, [ESI]      ; Grab first DWORD,
                                increment SP
       MOV EDX, EAX         ; EDX = first DWORD
loop:  LODS EAX, ESI        ; Grab next DWORD,
                                increment SP
       SHRD EAX, EDX, CL    ; Align DWORD shifted by
                                CL bits
       XCHG EDX, EAX        ; Swap EAX with EDX
       STOS [EDI], EAX      ; Store aligned data,
                                increment EDI
       DEC EBX              ; Decrement loop count
       JA loop              ; Repeat loop if count = 0
                            ; R indicates contents of R
```



3. Of four protection levels, the top three (0–2) are assigned to the supervisor. The lowest is allotted to user code. Such a system is more suitable for multitasking than one with a single supervisory level.

status information.

To speed translating from linear to physical addresses, the chip carries a page descriptor cache, also known as a translation look-aside buffer (TLB). Its 32 entries can obviate full address translation more than 98% of the time. Hence, address translation degrades performance in only 2% of all cases.

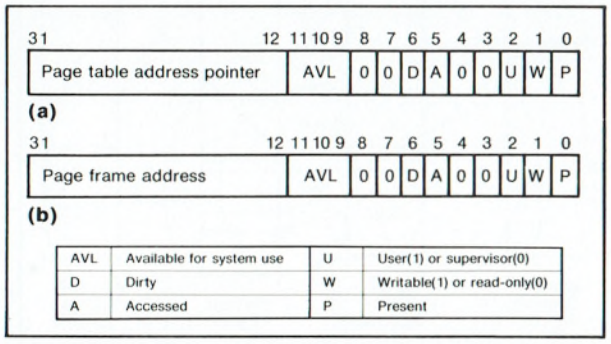
The User bit and the Write bit assign varying degrees of protection to each page (Table 3). In addition, two status bits are present for the system designer to track page usage. The Accessed bit is true if a page or page directory has been accessed by the MMU; the Dirty bit is true if the page or directory has been written to.

The processor's paging mechanisms thus form a powerful base for implementing the key operating system services necessary to paged virtual memory management. Specifically, these are algorithms to allocate and replace memory page frames and to store a virtual page in secondary memory.

Four address space formats

The memory management units furnish a flexible means to fit address space architecture to different applications. Four major address architectures can be directly implemented. For example, the segmented and paged model (Fig. 5a), which unleashes the chip's full power, is the one used for the Unix System V operating system. There, each task has a code segment, data segment, and stack segment; the processor's segmentation unit furnishes the per-task segments, and the paging unit allows the working set kept in main memory to be effectively managed.

The segmented mode (Fig. 5b) is very useful when segments are not large and when compatibility with



AVL	Available for system use	U	User(1) or supervisor(0)
D	Dirty	W	Writable(1) or read-only(0)
A	Accessed	P	Present

4. For every page, the directory descriptor (a) and the table descriptor (b) each contain 12 flag bits. In addition to governing memory management and protection, flag bits can be employed by the system designer for proprietary functions.

previous processors in the 8086 family is a must. When system security and reliability are important, the protection scheme established by segmentation mechanism is particularly helpful.

Some designers prefer a completely flat logical architecture in which paging is used to manage physical memory. The processor makes possible this memory arrangement in the paged linear model (Fig. 5c). Since the processor's paging mechanism is on chip, the device performs better than systems that use this model with off-chip MMUs.

Finally, the completely linear model is the design of choice in many real-time applications (Fig. 5d). It suits very compact, user-designed operating systems because niether segmentation nor paging need be employed. In this model, the program address is also the physical address.

Multitasking a must

For systems that require multitasking or even multiple environments, the processor's task state segment, or TSS, is useful. This is a repository for all task state information that allows tasks to be switched quickly with a hardware assist by storing and reloading information from old and new TSSs in a single instruction. The chip remains compatible with the 80286's task-switching mechanism. Since the TSS includes the state of the page directory base register, the user can assign a different page map for each task with a single instruction.

The chip packs the most power when the semantic control, flexible address space architecture, and multitasking assistance mechanisms are all combined. Application programs written for each of the following environments, for instance, could exist concurrently—in fact, there could be multiple instances of each environment executing concurrently. And because all semantic control bits of the processor are switched when a task is switched, a new environment can be entered with one instruction.

One environment could be the segmented and paged model, executing 32-bit code under Unix System V. Another might be the segmented and paged model, executing 16-bit 80286 object code. A third environment could apply the paged linear model to execute 32-bit code transported from linear Unix environments like Berkeley 4.2. A fourth environment might directly execute unchanged 8086 and 8088 object code using the segmented and paged model—say, programs written for the IBM PC.

To emulate the operation of the 8086 and 8088 pro-

32-bit microprocessor

cessors within its protected virtual address mode environment, the processor is controlled by the Virtual Mode bit (VM bit) in the flags register. In a virtual 8086 code segment, the segment register semantics are the same as in the 8086 itself. Thus unchanged 8086 code can be executed in the processor's protected mode—even code that follows the frowned-upon practice of employing segment registers for temporary storage.

Just like an 8086

In the virtual 8086 mode, the processor looks just like an 8086 to the application program; the value in the segment selector register is shifted left by four bits to compute the actual segment base. As in the 8086, and in real modes of the 80286 and 80386, each segment is 64 kbytes long. And even though the processor can generate larger offsets, that memory is not accessible.

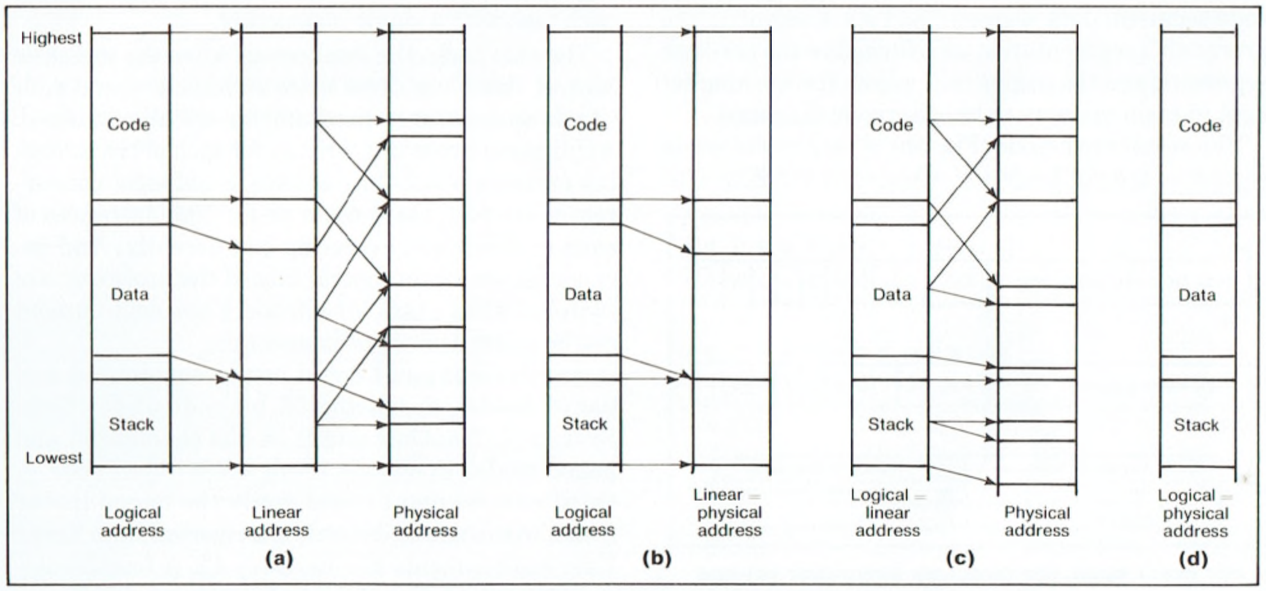
Interrupts cause an automatic switch out of the virtual 8086 mode, so that the operating system's interrupt handler can process the interrupt or hand it back to the 8086 program, if appropriate.

Each virtual 8086 code section generates linear addresses up to 1 Mbyte long. To supply separate address spaces for each of these, the operating system must create a separate page directory for each task. The paging mechanism also can be used to simulate the address wraparound at 1 Mbyte employed by the 8086. Nevertheless, the processor's protection scheme remains intact in the virtual 8086 mode because such programs always execute at the least privileged level.□

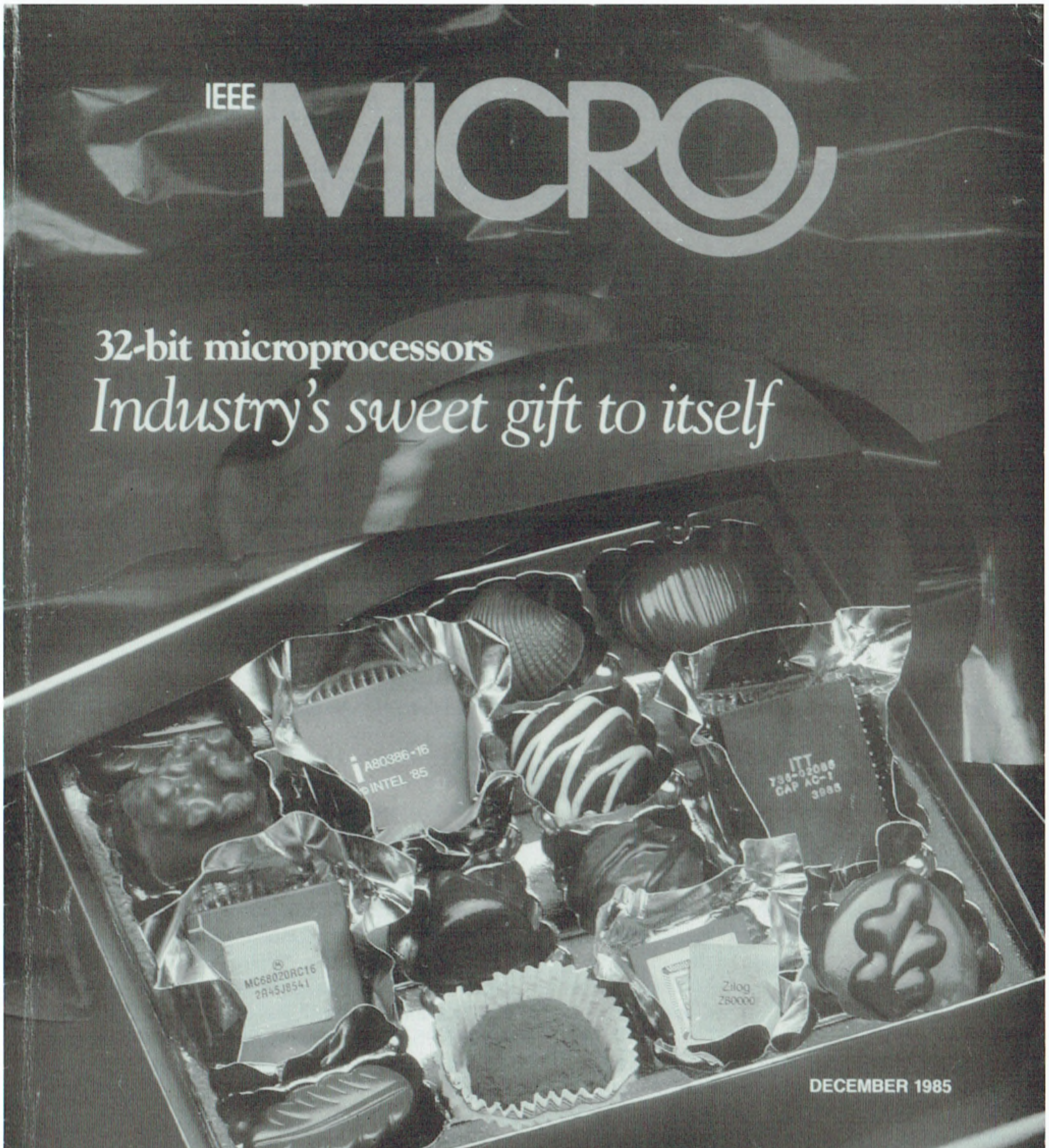
Senior engineer Rakash Agarwal is a five-year veteran of Intel. His bachelor's degree is from the University of British Columbia and his master's was awarded by the University of Toronto.

Greg Blanck is a design engineer who has worked at Intel for a year. He holds a BSEE in electrical engineering from Case Western Reserve University.

Dana Krelle was formerly assigned to the 80286 microprocessor and is a product marketing engineer for the 80386. He holds a BSEE from the University of Michigan and an MBA from the University of California at Berkeley.



5. The processor directly implements four address architectures. The segmented and paged model (a) unleashes the device's full power. The segmented model (b) is appropriate when the fields are fairly short. A linear model is possible in which paging oversees physical memory (c), and a completely flat model is the choice for real-time applications (d).


IEEE MICRO
32-bit microprocessors
Industry's sweet gift to itself
A80386-16
INTEL '85
ITT
MC68020RC16
2R45J8541
Zilog
Z80000
DECEMBER 1985
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

The Intel 80386—Architecture and Implementation

Khaled A. El-Ayat and Rakesh K. Agarwal
Intel Corporation

The Intel 80386 represents the state of the art in high-performance, 32-bit microprocessors. It features absolute object code compatibility with previous members of the iAPX 86 family of microprocessors, including the 80286, 80186, 80188, 8086, and 8088. This protects major investments in application and operating systems software developed for the iAPX 86 family, while offering a significant enhancement in performance. The 80386's architecture and performance should allow it to be used in a wide range of demanding applications—e.g., in engineering workstations, office systems, robotic and control systems, and expert systems.

The 80386 implements a full 32-bit architecture with a 32-bit-wide internal data path including registers, ALU, and internal buses; it provides 32-bit instructions, addressing capability, and data types, and a 32-bit external bus interface. It extends the iAPX 86 family architecture with additional instructions, addressing modes, and data types. It incorporates a complete memory management/unit. The 80386 extends the 80286 segmentation model to support four-gigabyte segments and to provide a standard two-level paging mechanism for physical memory management. System designers can use segmentation or paging or both, without performance penalties, to meet their memory management requirements.

The 80386 architecture is complemented by a bus interface that uses only two clocks per bus cycle; this allows efficient interfacing to high-speed as well as low-speed memory systems. At 16 MHz, the bus can sustain a 32-megabyte-per-second transfer rate. Other bus features include dynamic bus sizing to support mixed 16/32-bit port interfacing and a dynamically selectable pipelined mode to facilitate high-speed memory interleaving and allow longer access times.

The 80386 is implemented in Intel's CHMOS-III 1.5-micrometer process. Typical instruction mixes indicate an average processing rate of 4.4 clocks per instruction and an overall execution rate of three to four MIPS. To facilitate system debugging, the chip incorporates hardware debug features and self-testing.

80386 base architecture

Different microprocessor applications require different types of architectural support. Some applications—such as those running under Berkeley UNIX—may prefer a linear address space. Others that manage a multitude of dynamic data structures may require hardware-enforced rules to protect the visibility of the dynamically created objects. The 80386 architecture supports these diverse require-

0272-1732/85/1200-0004$01.00 © 1985 IEEE

ments by providing the user with several memory management and addressing models. Further, its repertoire of addressing modes, data types, instructions, and special constructs make it well suited to modern high-level languages.

The base architecture of the 80386 encompasses the register model, data types, addressing modes, and instruction set. It forms the basis for high-level-language compiler code generation and for assembly-language-level application programming. Other features of the machine useful for implementing operating systems are discussed in the section on OS architecture.

Registers. The 80386 possesses several on-chip register sets to support various machine features. Figure 1 shows the eight general-purpose registers available for calculations and memory addressing, the flags register, and the instruction pointer. Other registers include control registers, six segment registers used to structure the four-gigabyte address space and to facilitate system debug, and six debug registers used to control the setting of up to four code or data breakpoints.

The 32-bit general registers are named EAX, EBX, ECX, EDX, ESP, EBP, ESI, and EDI. To allow 16-bit operations and to provide compatibility with the 16-bit members of the iAPX 86 family, eight 16-bit registers are superimposed onto the low-order parts of the 32-bit registers. Similarly, there are eight 8-bit registers that are aliases for the lower and upper halves of each of the 16-bit registers. Operations on 8-bit or 16-bit registers affect only the corresponding superimposed registers. For example, the carry out of bit 7 during an 8-bit add is not propagated into bit 9 of the destination; instead, the carry flag (CF) of the flags register is set appropriately. This is true for all condition code settings in the flags.

Operand addressing. 80386 operands may reside on the chip (in registers), in main memory, or in the I/O address space. Furthermore, an operand may be implied in the instruction or specified explicitly as a part of the instruction.

Storing operands in registers generally provides the fastest method of processing data. The contents of any 80386 general register can be operated on by any arithmetic or logical operator. Alternatively, 8-, 16-, or 32-bit constants (immediates) can be embedded directly in an instruction. Sixteen- and 32-bit operations may specify 8-bit sign-extended or zero-extended immediates. Table 1 includes sample instructions employing registers and immediates as

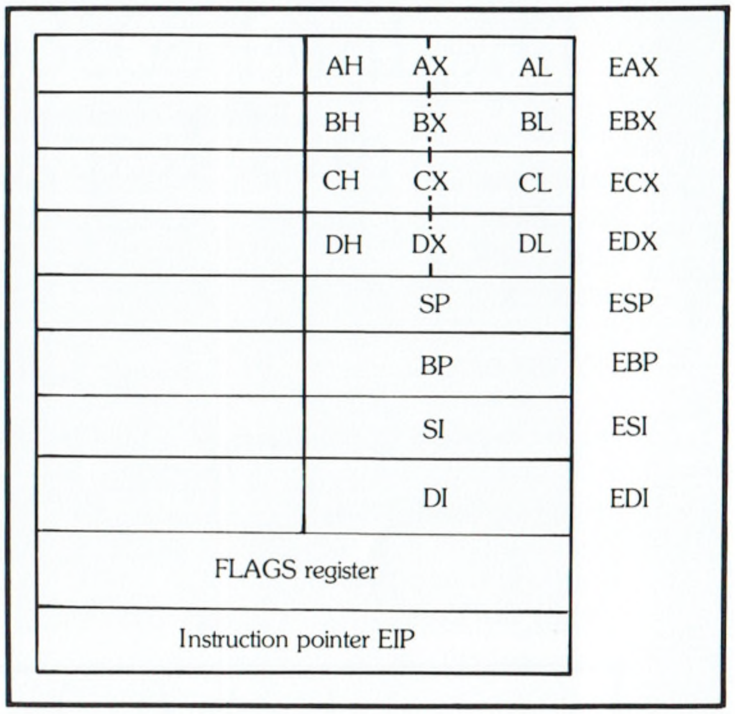


Figure 1. The 80386 general register set, FLAGS, and instruction pointer.

operands. In general, register-to-register operations execute in two clocks on the 80386. At a clock rate of 16 MHz, this translates to 125 nanoseconds per operation.

Most operands are stored in main memory. The 80386 has a full complement of address generation mechanisms for specifying the effective address of such operands. These mechanisms were developed in response to the storage paradigms present in high-level languages.

In its simplest form, the effective address of a memory operand can be encoded directly in an instruction. Usually, however, a particular memory address is not known until the program is actually executing. In this case, the effective address can be obtained by summing the contents of one or two general-purpose registers and an optional immediate value or *displacement*. This register-based effective address scheme can be summarized as

$$[\text{base register}] + [\text{index register}] * (\text{scale}) + [\text{displacement}].$$

Here the base register is any general-purpose register and the index register is any general-purpose register

Table 1.
Examples of operand addressing in the 80386.

Instruction	Clocks	Semantics
INC EAX	2	Increment contents of EAX by 1.
IMUL EBX, -3	9	Multiply the integer in EBX by -3.
CMP CX, 0	2	Compare contents of CX with 0 and set condition codes.
MOVSX EAX, SI	3	Sign extend the contents of the 16-bit register SI and move into EAX.
MOV DWORD PTR [56], -12445654	2	Assign -12445654 to the 32-bit integer at address 56.
JMP jumpTable[EBX*4]	10	Jump to the address stored at entry EBX of jump table.
SUB DX, WORD PTR [EBP+EDI*2-10]	7	Subtract from DX the 16-bit quantity at address [EBP+EDI*2-10].

$$\left\{\begin{array}{c}\text{EAX}\\\text{EBX}\\\text{ECX}\\\text{EDX}\\\text{ESP}\\\text{EBP}\\\text{ESI}\\\text{EDI}\end{array}\right\} + \left\{\begin{array}{c}\text{EAX}\\\text{EBX}\\\text{ECX}\\\text{EDX}\\\text{—}\\\text{EBP}\\\text{ESI}\\\text{EDI}\end{array}\right\} \cdot \left\{\begin{array}{c}2\\4\\8\end{array}\right\} + \left\{\begin{array}{c}0\\\text{8-bit displacement}\\\text{32-bit displacement}\end{array}\right\}$$

Figure 2. 32-bit memory addressing modes.

Table 2.
80386 support of high-level-language memory addressing.

Storage class	Type specifier	Addressing mode
Static	Scalar	[disp]
	Structure	[disp]
	Array of scalars	[disp+index]
	Array of structures	[disp+index]
Automatic	Scalar	[base+disp]
	Structure	[base+disp]
	Array of scalars	[base+disp+index]
	Array of structures	[base+disp+index]
Heap	Scalar	[base]
	Structure	[base+disp]
	Array of scalars	[base+index]
	Array of structures	[base+disp+scale]

other than ESP. The scale specification is a constant value, either 2, 4, or 8. If specified, it scales the index register by the required amount, thus simplifying indexing into arrays of multibyte elements. The displacement field is also a constant, its value ranging from -2^{31} to $+2^{31}-1$. Figure 2 shows all of the 80386's 32-bit memory addressing modes, and Table 2 correlates high-level-language addressing forms with those modes. (Example of memory addressing appear in Table 1.)

Data types. As shown in Figure 3, the 80386 directly supports the fundamental data types found in most high-level languages. The basic operations provided by the 80386 for each of these data types are shown in Table 3. Most of these operations execute in two clocks when register or immediate operands are used. Furthermore, because of pipelining and the two-clock memory bus, stores to memory also execute in two clocks.

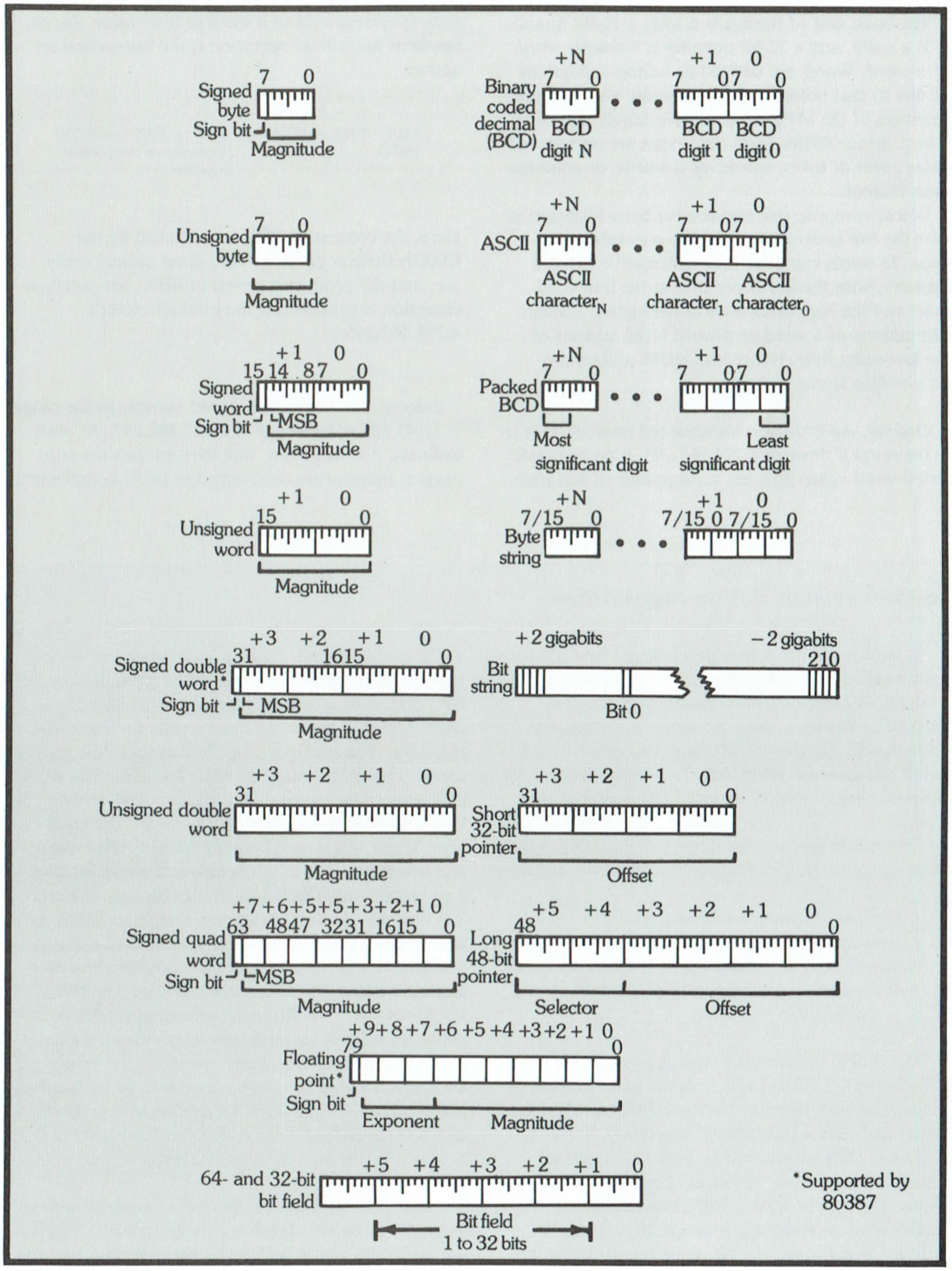


Figure 3. 80386 data types.

The basic unit of storage is a *byte*; a 16-bit quantity is a *word*, and a 32-bit quantity is a *double word*, or *d-word*. Words are defined as having a length of 16 bits so that notational compatibility with the other members of the iAPX 86 processor family will be retained. In the 80386, most data types are represented in the form of bytes, words, or d-words, or combinations thereof.

Words comprise two consecutive bytes in memory, with the low-order byte at the lower-numbered address. D-words comprise four consecutive bytes in memory, with the low-order byte at the lowest address and the high-order byte at the highest address. The address of a word or d-word is the address of the low-order byte. Hence, the 80386 utilizes the little-endian storage scheme.

Ordinal. An ordinal is an unsigned number. If it is in the range 0 through 4,294,967,295, it corresponds to a d-word value. If it has a magnitude of less than zero, it corresponds to a word or byte value. An example of an ordinal operation is the instruction sequence

```
MUL   EBX,vec[EDX*4]  ; EBX : = EBX * vec[EDX]
INTO                  ; Generate an exception if
                      ;overflow
```

Here, the content of EBX is multiplied by the EDXth element of the d-word-sized ordinal array vec, and the product is stored in EBX. An overflow exception is generated if the product exceeds 4,294,967,295.

Integer. An integer is a signed number in the range −2,147,483,648 through +2,147,483,647. As with ordinals, d-word, word, and byte integers are supported. Integers are represented in two's-complement

80386 system debug capabilities

A large portion of system development time is usually devoted to system debugging and verification. The magnitude of the problem is strongly influenced by system complexity at both the software and the hardware levels. In highly complex systems, external hardware and software debug aids alone cannot provide the level of support needed; internal CPU assistance is required.

To facilitate system development and real-time system debugging, the 80386 provides the following capabilities:

- detection of instruction breakpoints,
- detection of data reference breakpoints,
- specification of four separate breakpoint addresses,
- instruction single-stepping, and
- a one-byte trap instruction.

The 80386 has six system debug registers (see figure). The first four, DR0 to DR3, store the required breakpoint addresses. Registers DR6 and DR7 contain debug status and control information, respectively. Registers DR4 and DR5 are reserved by Intel. Breakpoint addresses must be linear addresses of instructions or data items. The control register, DR7, specifies the conditions under which a breakpoint is recognized and includes enable/disable masking fields, the breakpoint type, and the breakpoint length fields.

The enable/disable masking fields determine whether a detected breakpoint condition will be recognized by the CPU and whether an exception will be generated or simply stored in the debug status register for future examination. The breakpoint type field indicates the type of memory reference—e.g., an instruction execution, a data write reference, or a data read/write reference—that is intended to cause the system break. The breakpoint length field is used primarily for data references and selects byte, word, or double-word ranges for data item breakpoints. This field is needed because of a problem that arises in data referencing. Simply specifying the starting address of a data item is too restrictive and is insufficient for matching a breakpoint condition. The problem exists because there are three different data item lengths (8, 16, and 32); under erroneous conditions the generated address and data type length may not exactly match the specified breakpoint condition. The length field adds flexibility by selecting a range in which breakpoints can occur. Instruction breakpoints always specify a one-byte length field, since system breakpoints should uniquely specify the byte-granular starting address of intended instructions.

Let us illustrate the use of the debug capability with an example. To cause a break at a particular instruction, the user loads the starting address of that instruction into one of the breakpoint address registers, DR0 to DR3. The

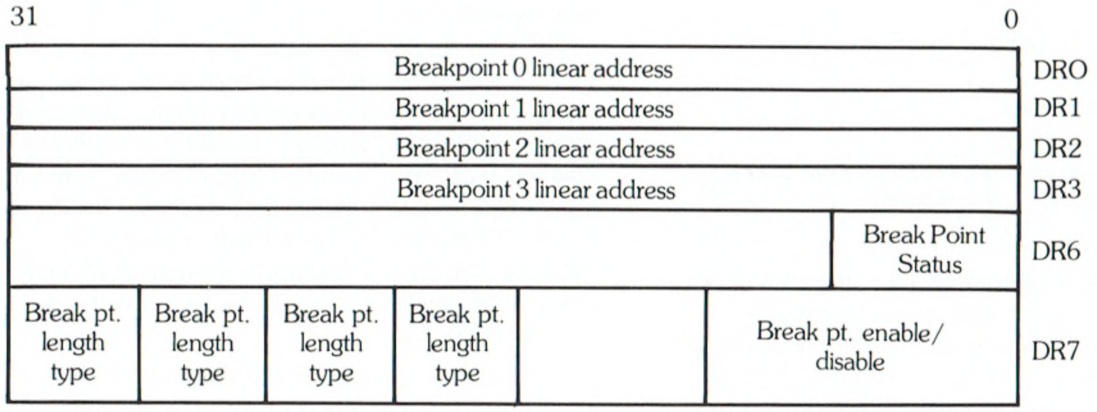
Table 3.
Data types supported by the 80386 instruction set.
(Floating point is available when numeric coprocessor is added.)

	Data type					
Operation	Ordinal	Integer	BCD	Floating point	String	Bit string
Move to/from memory, convert precision	X	X	X	X	X	X
Arithmetics add, subtract, multiply, divide, negate	X	X	X	X		
Logicals AND, OR, XOR, shift	X	X				
Compare	X	X	X	X	X	X
Transcendentals				X		

corresponding enable bit for the selected register must be set, and the type and length field must be set to instruction break (length = one byte). When the CPU is certain it is about to execute that instruction, it completes the execution of the current instruction, and a debug exception is generated. Note that if a successful branch or transfer of control precedes the intended breakpoint instruction, the break does not occur. An instruction break occurs before the instruction causing it is executed, whereas a data reference break occurs after the instruction causing it is executed.

The single-stepping-by-instruction feature forces an exception after each instruction execution. It can be used for system monitoring on an instruction-by-instruction basis. The one-byte trap instruction causes a software trap when executed and is useful for debugging exception-handling code.

80386 system debug registers.


31
0
Breakpoint 0 linear address
DRO
Breakpoint 1 linear address
DR1
Breakpoint 2 linear address
DR2
Breakpoint 3 linear address
DR3
Break Point Status
DR6
Break pt. length type
Break pt. length type
Break pt. length type
Break pt. length type
Break pt. enable/ disable
DR7

notation. This allows a common set of instructions for addition and subtraction. For example,

```
SUB   ESP, 5
```

subtracts five from ESP whether ESP stores an integer or an ordinal. The settings of the overflow, sign, zero, and carry flags allow a program to determine whether a signed or an unsigned overflow has occurred. However, special instructions are provided for determining overflow in multiply and divide operations involving integers, since an integer multiply has its own rules for overflow and an integer divide produces its own unique bit patterns.

Pointers. A pointer is a memory address. There are two types of pointers in the 80386: near pointers and far pointers. A near pointer is another term for an effective address. A far pointer has two components: a word-sized selector and a d-word-sized effective address. The selector names the logical address space in which the effective address resides. This ability to define logical address spaces gives a user greater flexibility in structuring memory. (This is discussed in greater detail below.) To retain compatibility with 16-bit members of the iAPX 86 processor family, the 80386 also supports pointers having word-sized selectors and word-sized effective addresses.

Bit fields. The 80386 can do fetches from, or perform stores into, contiguous bit sequences of up to 31 bits each, where such bit fields themselves reside in a bit string of up to four gigabits. Single-bit values can also be tested and modified. Furthermore, bit fields can be scanned for the first set bit in either a forward or a reverse direction. This feature can be used to implement the *set* type of Pascal. For example, if *col* is an object of the type *set of color*, then the Pascal fragment

```
while c in col do
```

can be translated into

```
BSF   EAX, col    ; Find first set element. Store in EAX
JZ    loopExit    ; Exit if none left
```

Floating-point operations. By adding the 80287 floating-point coprocessor, or the higher-performance 80387, the user can extend the 80386 instruction set to support 32-bit, 64-bit, and 80-bit IEEE-standard floating-point arithmetic directly. These coprocessors provide the accuracy and performance demanded by numerically intensive applications such as robotics and graphics.

Multiprecision operations. The 80386 provides limited support for 64-bit operands. Integer and ordinal multiply and divide operations may have 64-bit products and dividends; the multiplier, multiplicand, divisor, quotient, and remainder are limited to d-word quantities. Multiprecision add and subtract operations can be easily synthesized with the add-with-carry (ADC) and subtract-with-borrow (SBB) operations.

Besides multiprecision arithmetic operations, the 80386 provides double-width shift instructions that accept a 64-bit input and generate a 32-bit output. These instructions can be viewed as generalizations of the normal logical shifts, except that the value shifted in is not zeroes but is specified by the contents of another 32-bit operand. Double-width shifts are especially useful for buffering intermediate data when performing operations on unaligned bit strings. A barrel shifter within the 80386 makes the execution times of these instructions independent of the size of the shift—any register-based, double-shift operation can be done in three clock periods.

Logical addresses. Thus far we have discussed memory addresses only in the context of effective addresses. We shall now investigate the logical address spaces provided by the 80386 to allow convenient memory partitioning. Each logical address space is named by means of a word-sized selector. All memory addresses have two components: the selector that names the logical address space (or segment) and an effective address (or offset) that indexes into the named logical address space. The full selector:offset form of address is the far pointer mentioned previously. The selector is not usually directly specified in an instruction's operand field; it is instead stored in a segment register.

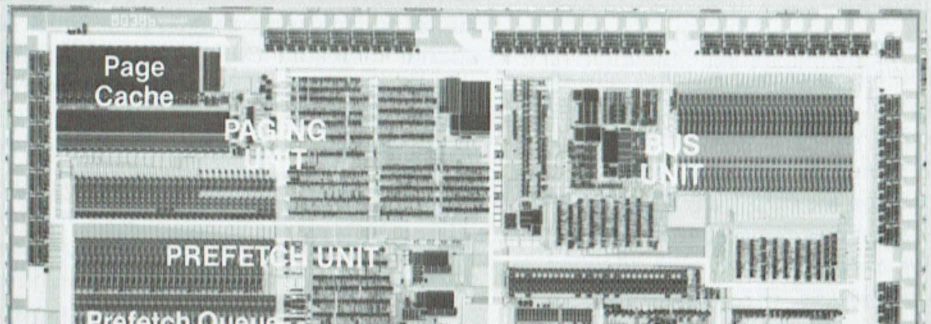
There are six segment registers named CS, DS, ES, FS, GS, and SS, as shown in Figure 4. Segment registers are not usually encoded in instructions; the segment register to be used is instead implied in the operand type. For example, code is fetched from the logical address space named by the selector in CS, at the offset specified by the instruction pointer EIP. Similarly, the stack is located in the logical address space named by the selector in SS, with the top-of-stack at offset ESP.

For memory operands not residing in the stack, the implied segment is usually DS. If a memory operand's default segment register is not desired, it

The 80386 physical implementation

The 80386 measures 390 mils on a side and is implemented in CMOS with Intel's CHMOS-III process, which provides 1.5-micrometer geometries and two layers of metallization. This allows 16-MHz operation with low power consumption. The chip contains over 275,000 transistors and is housed in a 132-pin pin grid array to simplify external interfacing and improve reliability.

To meet the high performance objectives of the 80386 architecture, the chip's designers organized the CPU into eight pipelined logical units and provided for a high degree of execution overlap among them. The units, which are shown in the die photo are the bus interface unit, the instruction (code) prefetch unit, the instruction decode unit, the segmentation unit, the paging unit, the protection test unit, the control unit, and the data unit. The last three comprise the execution section of the CPU, which consists of a microengine, a register file, an ALU, a barrel shifter, and miscellaneous control logic. On-chip memory management is implemented by the protection test, segmentation, and paging units.



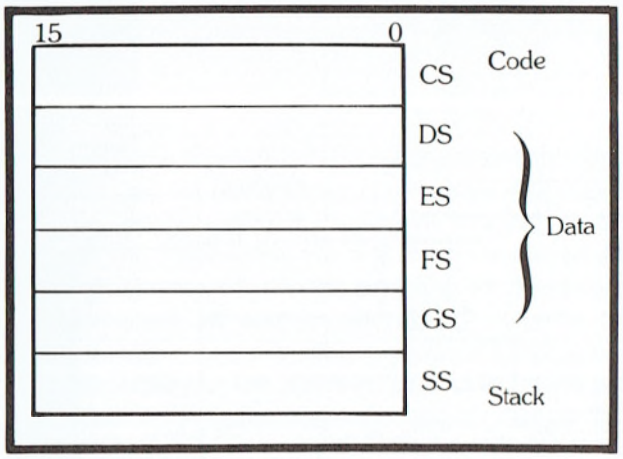


Figure 4. Segment registers.

can be overridden by placing one of six unique prefix bytes just before the instruction at which the override is to take effect. This allows rapid switching between different address spaces without requiring that a segment register be loaded with the correct selector value every time. It should be noted that segment descriptors are cached on the chip from their respective descriptor tables to allow rapid address translation and protection checking. Along with allowing the creation of address spaces, selectors form the basis for the 80386 memory management and protection scheme.

OS architecture—the memory management and protection model

Many computational environments require memory to be protected from unauthorized access. Furthermore, the allocation and deallocation of memory according to a process's needs must be managed. The 80386 provides a comprehensive set of mechanisms for supporting these requirements. The overall memory address generation structure is shown in Figure 5.

The logical address's selector and offset components specified by the instruction are mapped into a *linear address* via segment tables. Figure 6 shows how the selector specifies a segment descriptor. A segment descriptor is an eight-byte record. The main information it contains is the linear address and size of the base of the segment, and the type of reference that is allowed to be made into the segment. (Segment access rights are discussed below.) The linear address is constructed from a logical address by adding the offset to the linear base address. If this address exceeds the bounds of the segment (as specified by its size), a segmentation exception is signaled. An instruction encountering such an exception is fully restartable. Note that there are two segment tables of 8192 entries each (see Figure 6 again). This permits a total of 16,383 logical address spaces,* or roughly 14 bits of addressability. Since the offset furnishes 32 bits of addressability, the total logical address space of the 80386 provides 2^{46} bits of addressability.

The linear address is passed through a two-level page map table to generate a physical address. The physical address is the address delivered to the microprocessor's external bus for a memory access.

If a particular environment requires only one logical address space, the selector mechanism can be easily bypassed. This is done by defining one large logical address space spanning the entire linear address space and loading the corresponding selector into all segment registers. The difference between a logical address and a linear address then disappears, since the effective address component of the logical address matches the linear address. Similarly, if a linear address to a physical address mechanism is not required, paging can be disabled by resetting a bit in a control register. When paging is disabled, the linear address bypasses the page table look-up and appears directly as the physical address.

Since both translation steps are optional, the 80386 can allow the user to choose from one of four distinct views of memory:

- Unsegmented unpaged memory. Here both translation steps are bypassed, thereby making the effective address the same as the physical address. This is useful, for example, in a low-complexity, high-performance controller application, which requires a simple view of memory.
- Unsegmented paged memory. Here memory is viewed as a paged linear address space. Protection and management of memory is done via paging. This view is favored by some operating systems—e.g., Berkeley UNIX.
- Segmented unpaged memory. Here memory is viewed as a collection of logical address spaces. The advantage of this view over a paged approach is that

*Entry zero of the global descriptor table is special-cased to indicate a *null selector*. Hence, only a total of 16,383 descriptors is supported.

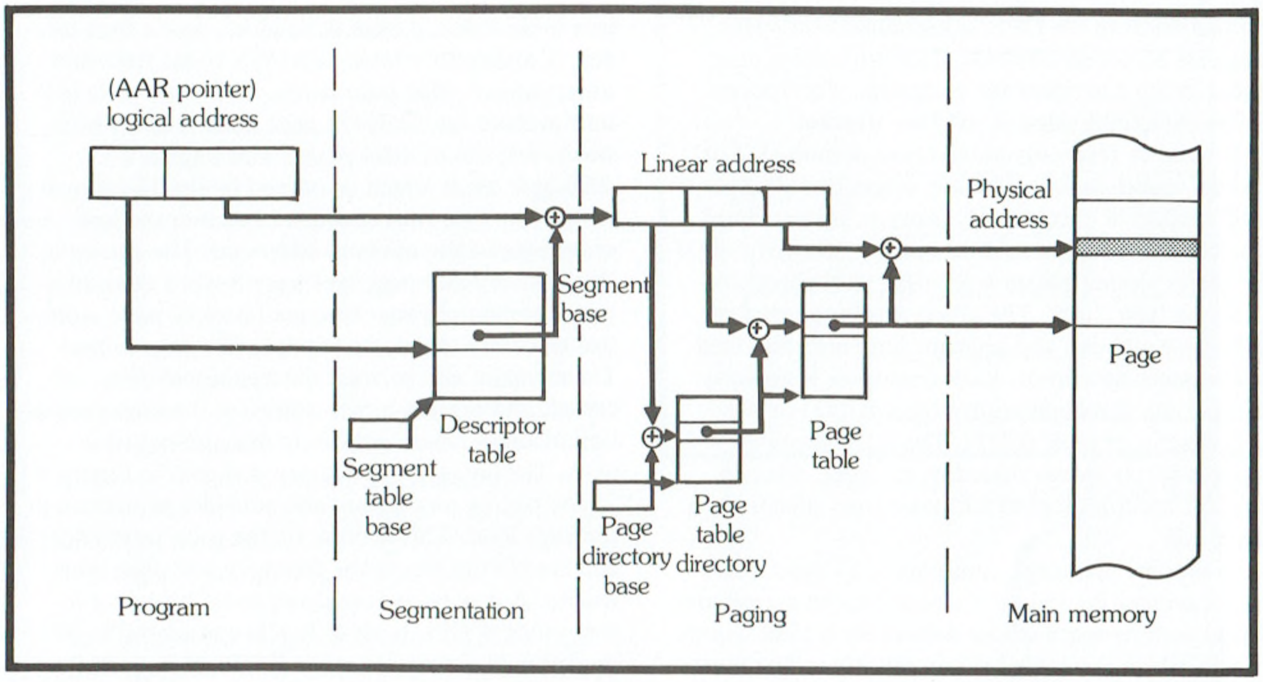


Figure 5. Memory address translation mechanism.

it affords protection down to the level of a single byte. Furthermore, unlike paging, it guarantees that when a linear address is generated the translation information is on-chip. Hence, segmented unpaged memory results in predictable access times.

• Segmented paged memory. This is the most comprehensive view of memory supported by the 80386. It uses segmentation to define logical memory partitions and paging to manage the allocation of memory within the partitions. Operating systems such as UNIX System V favor this view.

Segmentation mechanism. Memory protection is enforced in the 80386 through a concept called a *privilege level*. At any instant the processor is in one of four privilege levels. The current privilege level, or CPL, is stored as a number in the range 0 through 3, with level 0 being the most privileged level and level 3 the least. Furthermore, every logical address space (segment) has a privilege-level attribute associated with it called the descriptor privilege level, or DPL. The DPL is encoded in a field in the descriptor for a segment. Data access to a logical address space by a process operating at a given CPL is disallowed if the CPL is less privileged than the DPL of the logical address space. The CPL itself can change whenever CS is loaded with a new segment. Thus, the CPL is usually the DPL of the current code segment.

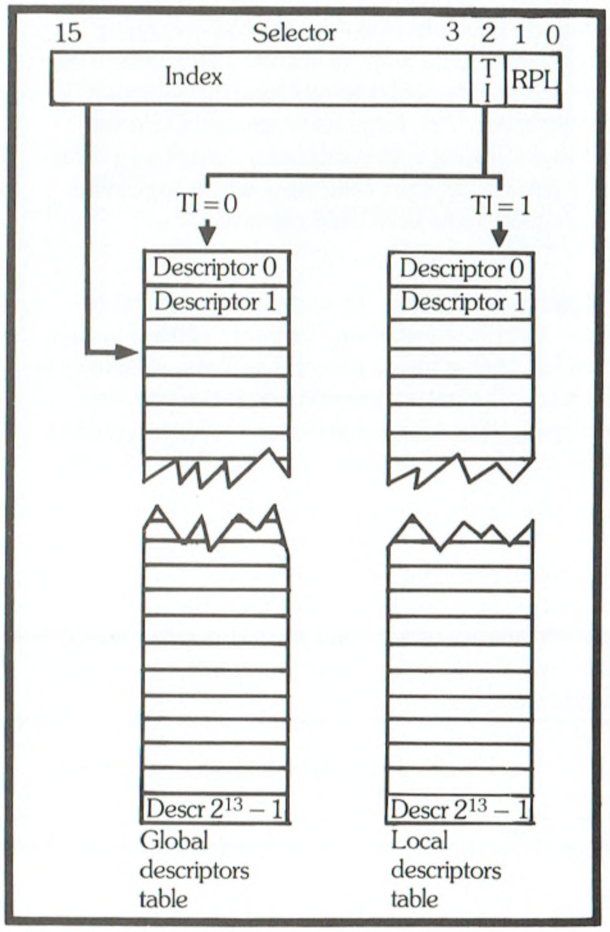


Figure 6. How a selector names a descriptor.

In addition to the DPL, a logical address space possesses an access attribute. This attribute is also stored in the descriptor for a segment. For address spaces containing data, it specifies whether read/write or read-only accesses are permitted. For address spaces containing code, it specifies whether read/execute or execute-only accesses are permitted. An access to a logical address space is allowed only if the access request passes a privilege level check and an access type check. The access attribute, privilege level, base address, and segment limit are contained in a segment descriptor. Each descriptor is an entry in either the global descriptor table (GDT) or the local descriptor table (LDT). The GDT specifies logical address spaces shared by all tasks, whereas the LDT specifies logical address spaces specific to a single task.

To support memory management, all descriptors have a *present bit* and an *accessed bit*. An access to a logical address space whose present bit is clear causes a fault, which is signaled via an interrupt. The fault handler can bring the logical address space's contents into the linear address space, set the linear address space's base and limit in the descriptor, and set the present bit. The faulting instruction can then be restarted. This implements a demand-swapping scheme for logical address spaces. The accessed bit is provided to help with the replacement policy implementation. This bit is set whenever a selector defining a logical address space is loaded into a segment register, thereby indicating which logical address spaces have been used recently.

Paging mechanism. To complement logical-to-linear address translation, the linear address is translated via paging into a physical address, as shown in Figure 5. The paging scheme involves a standard, two-level, table look-up process. The linear address to be translated is divided into three fields: a directory table index, a page table index, and a byte index. The directory table index is a 10-bit field that selects one of 1024 page tables. This page table is in turn indexed by the 10-bit page table index, which selects one out of 1024 pages. This page is a 4096-byte block which is indexed by the 12-bit byte index. The byte thus addressed specifies the low-order byte of the operand addressed. The entries in the directory and page tables are d-word quantities that store the physical base addresses of page tables (for directory tables) or of pages (for page tables). These entries also provide the traditional dirty, accessed, and present bits to allow for the implementation of replacement policies in demand-paged systems. The layout of each entry is shown in Figure 7.

The paging mechanism also provides protection at the page level. This is done via the user/supervisor and read/write bits in the directory and page table entries. A process is considered to be executing in user mode if CPL is set to 3; it is considered to be executing in supervisor mode if CPL is 2, 1, or 0. Therefore a user process is allowed to read a page only if both the directory entry and page table entry for it have the user/supervisor bit set. Similarly, a user write is allowed only if both entries have both the user/supervisor bit and the read/write bit set. A supervisor process is allowed to read or write all pages without restraint. Any violation of the page protection rules causes a paging exception. An instruction encountering such an exception is fully restartable. This permits the construction of demand-paged systems and, it should be noted, allows the use of the copy-on-write trick in implementing UNIX's *fork* primitive.

Task switching support. In addition to the memory management and protection facilities described above, the 80386 assists the operating system by providing hardware to implement task switching. This

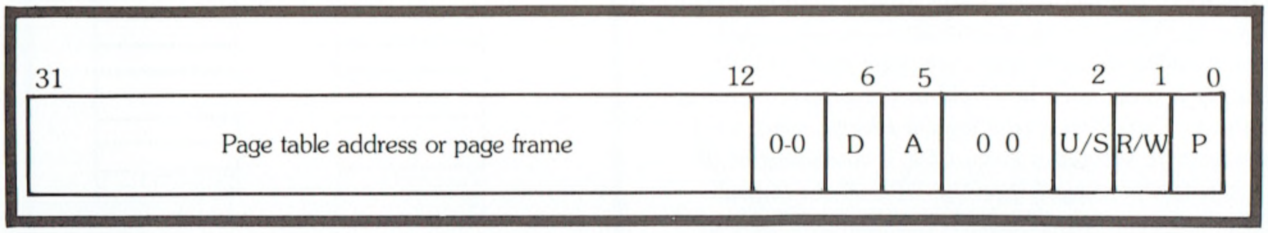


Figure 7. Directory table and page table entries.

hardware allows fast and efficient task switching in multitasking applications.

In task switching, the state of a task is stored in a data structure called a task state segment, or TSS. The fields of the TSS store images of the general registers, flags, instruction pointer, and segment registers, and an image of a pointer to the page directory table base and the local descriptor table. The TSS is itself described by a specially tagged descriptor residing in the global descriptor table. For a task switch to be performed, a call or jump is made via a selector that names such a descriptor. The processor recognizes the TSS and does a task switch by saving the current machine state in the currently active TSS and loading the state in the target TSS.

Compatibility with the 80286. Full compatibility with the 80286—to protect investments in operating systems and applications software—was clearly one of the key objectives of the 80386 architecture. Such compatibility was achieved by making the 80386 instruction set object code compatible with the 80286 instruction set; all architectural extensions to the 80286 instruction set, data types, and addressing modes adhere to strict compatibility rules. Furthermore, as the discussion earlier in this article showed, the basic 80286 memory management model was maintained and extended by the 80386.

Compatibility with the 8086. The 80386 is compatible with the 8086 in the same way that the 80286 is compatible with the 8086. The 80386 is similar to the 80286 in that it too powers up in 8086-compatible mode (also called real mode). Real mode is useful for initialization and for configuration of the processor data structures needed to run in native 80386 protected mode. The recommended method for running 8086 code is called virtual 86 mode.

To allow 8086 code to run harmoniously with its native code, the 80386 employs the notion of a virtual 8086, or VM86, task. Except for I/O and interrupt-related instructions, a VM86 task executes code using 8086 semantics. Memory addresses generated are treated as linear addresses and are subject to translation via paging. Mapping each VM86 task's linear address space to a different physical address virtualizes 8086 memory. However, operations that use global resources, such as I/O and interrupt operations, also need to be virtualized. This is done by trapping the instructions for such operations and allowing the virtual machine monitor (executing as a native 80386 program) to emulate them. VM86 mode is a key feature of the 80386—it allows the large body of 8086 software to run concurrently with native code in high-performance environments, and thereby allows 86, 286, and 386 tasks to run simultaneously.

80386 implementation

To meet the 80386's performance objectives, the chip's designers gave careful consideration to the internal implementation of the architecture—i.e., the microarchitecture. Performance enhancements in advanced microprocessor CPU designs can be obtained through

- advances in technology and circuit design,
- higher clock rates,
- wider data paths,
- advances in system and bus architecture, and
- advances in microarchitecture.

The first four of these were discussed earlier along with their effects on performance. The following sections describe the 80386 microarchitecture, focusing on performance-enhancing features such as pipelining and parallelism.

The 80386 is organized as eight logical units, with each unit assigned a task or step in the fetching and execution of each instruction. This arrangement allows as much parallel execution of the instruction stream as possible. The units are pipelined and, for the most part, operate autonomously. The units and their interconnections are shown in Figure 8 in the functional block diagram of the 80386.

The eight units are the bus interface unit, the prefetch unit, the instruction decode unit, the control unit, the data unit, the protection test unit, the segmentation unit, and the paging unit. The control, data, and protection test units comprise the execution section of the CPU.

The bus interface unit interfaces the CPU to the external system bus and controls all address, data, and control signals to and from the CPU. The prefetch unit is responsible for fetching instructions from memory. It uses an advance-instruction-fetch pointer to prefetch code from memory and store it in a temporary code queue. This queue also acts as a buffer between the prefetch unit and the instruction decode unit. Since addresses generated by the prefetch unit are linear, they must be translated to physical addresses by the paging unit before the prefetch bus cycle request can be sent to the bus interface unit.

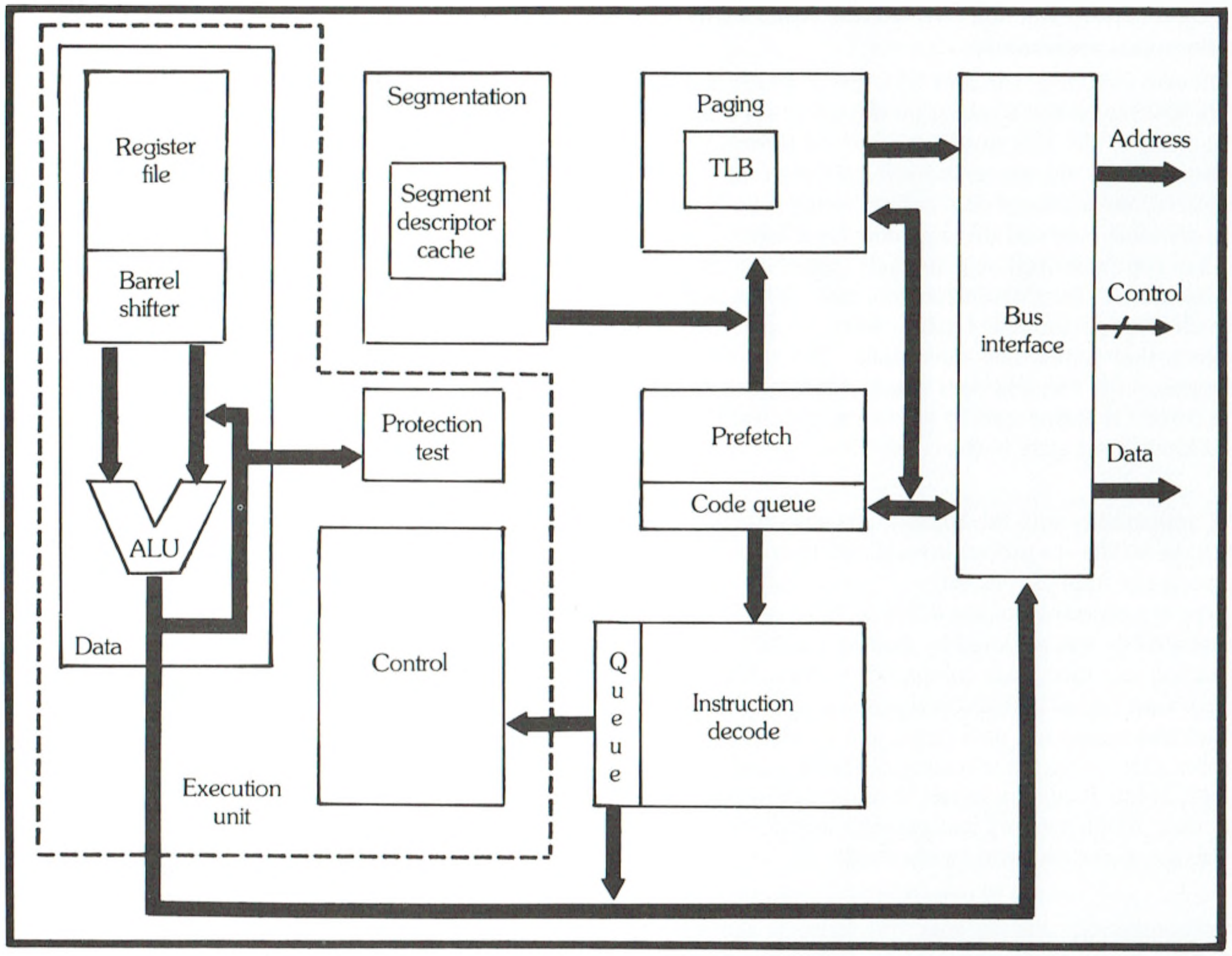


Figure 8. Block diagram of 80386.

The instruction decode unit prepares and decodes instructions for immediate execution by the execution unit. It does this by fetching bytes of code from the prefetcher's code queue, transforming them into a fully decoded instruction, and then storing that instruction in a three-level decoded instruction queue. The execution unit then operates on the decoded instruction, performing the steps needed to execute it.

Instructions requiring memory references send their requests to the segmentation unit for logical address computation and translation and segment protection violation checking. The segmentation unit produces a translated linear address which the paging unit then translates into a physical address. The paging unit also checks for paging violations before it sends a bus request and the address to the bus interface unit and external bus.

Pipelining and parallelism. Advanced microprocessors are normally pipelined by overlapping the fetching, decoding, and execution of instructions. In the 80386 microarchitecture, however, the operations of all eight of the logical units are overlapped. This allows the parallel and autonomous operation of the units. They can simultaneously operate on different instructions, thereby significantly boosting the overall instruction processing rate of the CPU. For example, while the bus interface unit is completing a data write cycle for one instruction, the instruction unit can be decoding another, and the execution unit processing a third.

The sections below describe each of the logical units of the 80386 and discuss how each was designed to maximize the benefits of pipelining.

Bus interface unit. The bus interface unit provides a high-speed interface between the CPU and the system. Its function is to efficiently meet the CPU's requirements for external bus transfers during code fetches, data fetches, paging unit requests, and segmentation unit requests. To accomplish this, it has been designed to accept and prioritize multiple internal bus requests so that it can make maximum use of the available bus bandwidth in servicing those requests. This activity is overlapped with any current bus transaction. As mentioned earlier, the 80386 bus uses only two clocks per cycle; if the pipelined mode is used, the 80386 bus is capable of starting the next address of a new bus cycle before the completion of a current bus transaction.

Prefetch unit. This unit is responsible for prefetching instructions from memory. It stores the aligned code in its code queue for efficient decoding by the instruction unit. It maintains a linear address pointer and a segment prefetch limit that are initially obtained from the segmentation unit to be used as a prefetch instruction pointer and for checking segment limit violations, respectively. The prefetcher attempts to keep its code queue filled with valid bytes of code by sending prefetch bus cycle requests to the bus interface unit through the paging unit. Prefetch bus cycle requests are made whenever the prefetch code queue is partially empty or after the occurrence of a control transfer. The prefetcher's bus cycle requests are assigned a lower priority than execution-related, operand fetch/store bus cycle requests and page-miss processing and segmentation-specific bus cycle requests. At zero wait states, there is no interference between prefetch and data bus cycles. Idle cycles are used to prefetch code from memory and keep the code queue filled.

Instruction decode unit. This unit decodes and prepares instructions for processing by the execution unit. Whenever the instruction unit's own queue or pipe is partially empty, it fetches bytes of code from the prefetcher code queue, decodes and prepares them, and stores the result in its own three-word-deep queue. The decoded instruction queue words are very wide; they contain all the instruction fields the execution unit needs to immediately execute the instruction without further decoding. The combined prefetch unit/instruction unit pipe operates on a two-clock cycle. The instruction unit, however, can decode at only one clock per opcode byte.

Execution unit. The next logical unit in the pipe is the execution unit. As mentioned previously, it is composed of the control, data processing, and protection test units. Its responsibility is to execute the instruction given to it. It does so by using its own resources as well as by communicating control and sequencing information to other logical units needed to complete the execution of the instruction. The fully decoded instruction is popped out of the instruction queue, and the execution unit uses its various fields, such as microcode starting addresses, operand references, data types, and ALU operators, to execute it.

The control section consists of a microcode-driven engine that has special-purpose hardware for decoding, assisting, and speeding up microcycle execution. The data processing section—or data path—contains all data registers, an ALU, a barrel shifter, multiply/divide hardware, and special control logic; it performs the data operations selected by the control section. The protection test section performs all static segmentation-related violation checks under microcode control.

The microengine has a two-clock execution latency, but by overlapping microinstruction fetching and execution and by using the delayed microjump technique, it provides an execution rate of only one clock per microcycle (62.5 ns at 16 MHz). To enhance the effective instruction processing rate of the execution unit further, parallel or overlapped execution of instructions is employed. Since memory reference instructions, including stack push and pop instructions (heavily utilized in procedure calls), constitute a large portion of the instruction mix in a typical program, a special technique is used to reduce the number of clocks needed to execute such instructions. The method used partially overlaps the execution of every memory reference instruction, including stack push and pop instructions, with the execution of the preceding instruction. This parallel execution of two instructions enhances the instruction processing rate of the CPU—with a typical mix of instructions, it yields a nine-percent improvement in performance.

The implementation of the execution unit required the addition of a 32-bit internal bus and special control logic to ensure the correct completion of the current instruction, prevent the use of stale register values, and provide the control needed to handle the simultaneous execution of two instructions.

Segmentation unit. This unit performs effective address computation upon request of the execution unit. It does this logical-to-linear address translation at the same time it does bus cycle segmentation violation checks. (Static violation checks—e.g., of segment descriptors—are performed by the protection test unit and are not part of the bus cycle activity.) The translated linear address is then sent to the paging unit along with bus cycle transaction information. The paging unit then becomes responsible for requesting bus service from the bus interface unit.

Paging unit. Linear addresses generated by the segmentation or code prefetch units are passed on to the paging unit, where they are translated into physical addresses. As explained in the section on architecture above, paging translation is implemented through a two-level page relocation mechanism. To improve performance, the paging unit uses a 32-entry translation look-aside buffer (TLB) to perform the translation. Page table and page frame entries are cached into the TLB. This results in a translation time of one half of a clock period—much faster than if memory-based page tables were referenced instead of the TLB. The combined logical-to-physical address translation pipe, which includes effective address formation, segmentation, and paging relocation, requires only two clocks of processing. No additional clocks are needed for paging since TLB look-up and translation are performed in the same clock (second phase) as the linear address calculation.

Integrated segmentation and paging. The 80386 implements a segmentation-plus-demand-paging memory management function for virtual memory translation and protection violation checking. To enhance system performance, memory management functions are integrated into the chip. All virtual-to-physical address translation, segmentation, and paging violation checking are performed by the on-chip memory management unit (MMU), which is imple-

80386 bus operation

The 80386 external interface (Figure B) connects the chip to memory and peripherals over a high-performance bus. This interface provides separate address and data pins to support efficient high-speed bus transfers and bus cycle pipelining. It uses its other pins to maintain simple and efficient interfaces—pins are dedicated to cy-

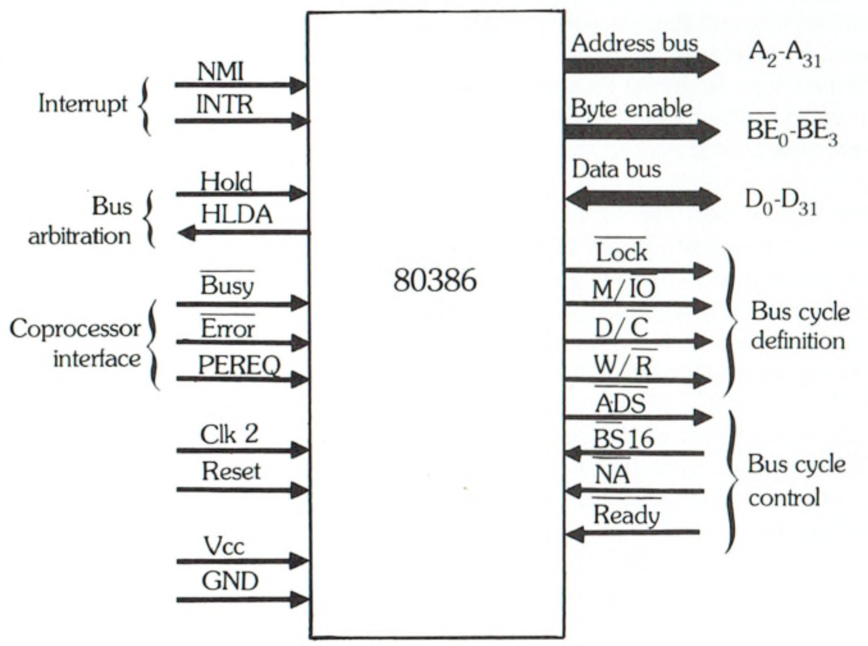


Figure B. iAPX 80386 external interface.

mented by the segmentation, protection test, and paging units. All memory management protection and translation descriptors are cached into the on-chip MMU. Segment descriptors are cached into segment descriptor caches in the segmentation unit, while page descriptors are cached into the TLB. Virtually all protection and translation activities utilize the on-chip descriptors.

There are several advantages to integrating memory management and the CPU on the same chip:

• The implementation can take advantage of pipelining and parallel execution by overlapping the steps from effective address formation to linear address and physical address generation. Under such an arrangement the next memory address will already be known while the current transaction is still on the bus. The logical-to-physical address translation in many cases will overlap other bus cycles and will be completed by the time the current bus cycle is finished. Pipelining to an off-chip MMU would clearly be more difficult.

• No additional clocks need to be added to the bus cycle to implement the MMU translation, since the memory management functions are internal and not part of the external bus cycle.

• System complexity and cost are reduced, and board design is simplified, since no off-chip memory management components are needed.

Special-purpose hardware. To enhance performance, the 80386 implements several important pro-

cle definition, cycle control, interrupts, bus arbitration, and coprocessor support.

Some typical 80386 bus cycles are shown in Figure C. With zero wait states, each bus cycle consists of only two clocks; at 16-MHz operation, the 32-bit bus can sustain a 32-megabyte-per-second transfer rate. When the bus operates in pipelined mode, the address of the next bus cycle is sent on pins A31-A2 before the current cycle is completed. This mode, which is dynamically selectable from the interface, can be used to implement a fast interleaved memory subsystem with inexpensive dynamic RAMs. In such a system, the current bus transaction uses one bank of memory while the new address is used to access another. The pipelined mode can also be used to overlap the current bus cycle with address decode delays of the next cycle, making available longer system access times. Slower memory or I/O systems use the $\overline{\text{READY}}$ pin to extend the current bus cycle as needed. Dynamic bus sizing allows software-transparent interfaces to 16-and 32-bit ports.

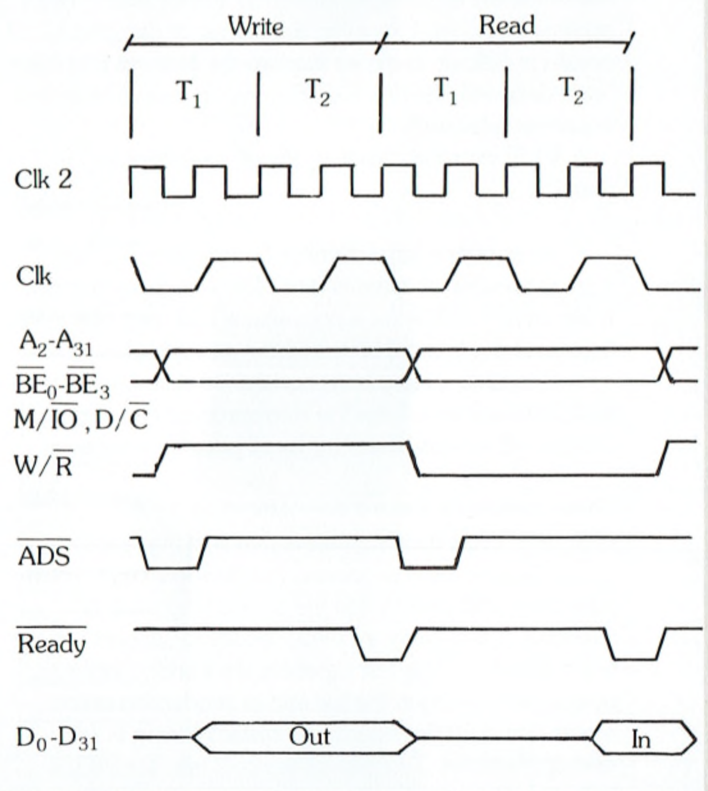


Figure C. 80386 bus operation.

cessing functions with special-purpose hardware. These special blocks include the TLB for paging translation, a 64-bit barrel shifter, multiply/divide hardware, multiple ALUs and adders, and miscellaneous random-logic control function implementations that take the place of PLA or microcode approaches that are too slow.

Translation look-aside buffer. As explained above, the time required to perform the two-step table look-up paging translation function is significantly reduced by the use of a TLB. With 32 entries in its cache, the TLB enjoys a relatively high hit ratio of 98 percent. A TLB miss occurs when an entry is not in the cache and references to memory-based page

Design for testability

To guarantee high quality in a complex VLSI microprocessor, the chip development team must

- use good, conservative design techniques,
- employ rigorous design verification at all levels,
- carefully choose the criteria against which manufacturing, testing, and quality assurance will be measured, and
- ensure the testability of the device.

Intel's goal is to manufacture the highest-quality silicon as inexpensively as possible. Adequate device testing helps achieve that goal. However, the testing of complex VLSI circuits is difficult. Here we explore the features that have been designed into the 80386 to make testing easier and ensure its adequacy.

A VLSI microprocessor is difficult to test for two reasons:

- It contains a large number of transistors but has a small number of external pins. The external pin count is too small to allow for a convenient and thorough interface to all the internal blocks of the microprocessor.
- It exhibits a fairly large number of states. It is virtually impossible to force the microprocessor to sequence through all possible states under all possible conditions.

These limitations make it very difficult to control and observe a VLSI microprocessor that is being tested.

To alleviate these problems, the 80386's designers incorporated test circuits into the device to ensure its testability. They designed these circuits to support proven test techniques. The test capability thus added facilitates component testing in the lab and in production environments and simplifies board and system testing in end user applications. Test capabilities built into the 80386 include

- self-test of all large PLAs,
- self-test of the control ROM and microengine, and
- signature analysis.

Special microcode and instructions are also provided to facilitate testing, and there is special test circuitry for the translation look-aside buffer (TLB).

PLA testing. In the 80386, self-test of large PLAs is needed because of the large number of possible input combinations. A maximal polynomial counter is used as a pseudorandom vector generator to drive the inputs to the PLA and to ensure sequencing through all possible combinations of the inputs. This is illustrated in the accompanying figure. As the inputs change in a pseudorandom fashion, the outputs from the PLA are fed into a linear feedback shift register with appropriate polynomial coefficients to accumulate a unique *signature* of the block. This signature is then read and analyzed by special software to determine correctness. This technique is referred to as *signature analysis*.

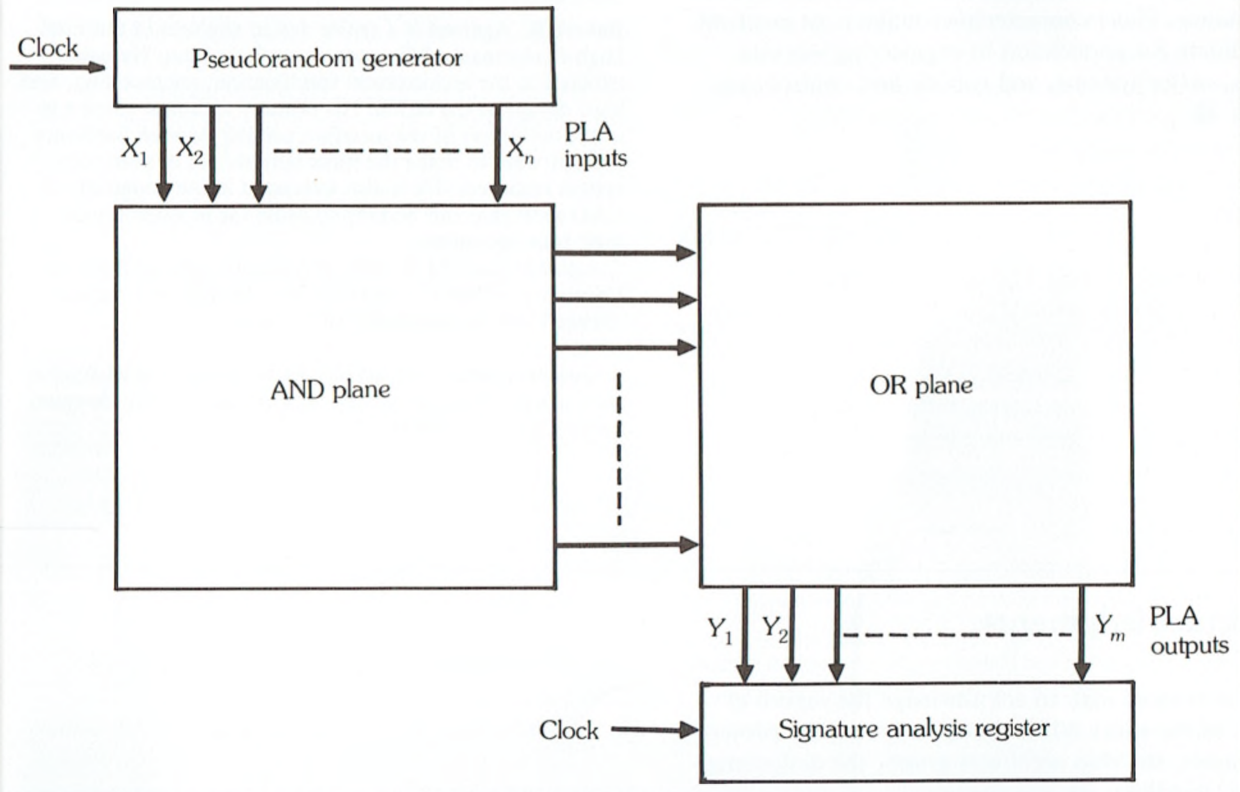
Signature analysis has been used for many years—primarily at the board and system level—to facilitate testing and fault diagnosing. It has the advantage of being able to reduce a large amount of data, uniquely encode it, and store it in a register (the linear feedback shift, or signature, register). It should be noted that the polynomials used to configure this register must be carefully chosen to provide maximum test coverage and error detection.

Control ROM testing. Self-test of the microcode ROM consists of sequencing through all addresses and word locations and, as in PLA testing, accumulating the result in a signature register. The accumulated microcode signature is then read by software. The microcode address counter logic is used to do the sequencing in the tables are needed to complete the address translation. Even with a high hit ratio, the memory-referencing table look-up function is clearly time-consuming and should be designed to minimize the degradation of system performance. The 80386 paging unit incorporates special-purpose hardware to implement the table look-up functions instead of using the more economical but slower microcode approach. TLB miss processing by this special hardware consumes only nine clocks, whereas a microcode-driven look-up would consume three times as many clocks.

Barrel shifter. A 64-bit barrel shifter is provided to implement shift, rotate, and bit manipulation in-

ROM test sequence, to share hardware, and to provide address sequencing logic for the microengine self-test.

TLB test circuitry. Testing the TLB RAM and CAM (content-addressable memory) for all entries and with all data sensitivity patterns would be very cumbersome and time-consuming without special assistance. To simplify such testing and improve its coverage, the 80386's designers added special-purpose hardware—two registers, plus special instructions to control them—to allow for the direct writing of any data pattern into the RAM or CAM blocks of the TLB. This hardware can also force the comparison of any data pattern against existing entries in the TLB to test the TLB's matching capability.



PLA testability.

structions efficiently as well as to assist in multiply and other miscellaneous operations. It can shift any data type by any shift count in a single clock, and it is significantly faster than an ALU/microcode implementation.

Multiply/divide hardware. A one-bit-per-clock multiply/divide mechanism permits a 32-bit multiply or divide in 40 clocks, maximum. To speed multiply execution, additional special hardware is included to detect early completion of a multiply and correctly terminate the operation. Early completion of a multiply is detected when all significant multiplier bits have been exhausted and the final product can be obtained by an appropriate shift operation. Since many multiply operations do not require the full 32-bit multiply, the average number of clocks needed for a multiply is 20.

The 80386 represents the state of the art in 32-bit microprocessor architecture and performance. It maintains object code compatibility with existing members of the 86 family of microprocessors, thus protecting end users' investments in software. Its implementation exploits pipelining and parallel execution to provide high performance. These characteristics make it an excellent candidate for application in engineering workstations, office systems, and robotic and control systems.

Acknowledgments

The authors wish to acknowledge the superb efforts of the entire 80386 team, including the project managers, the chip architects group, the design engineers, and the mask designers.

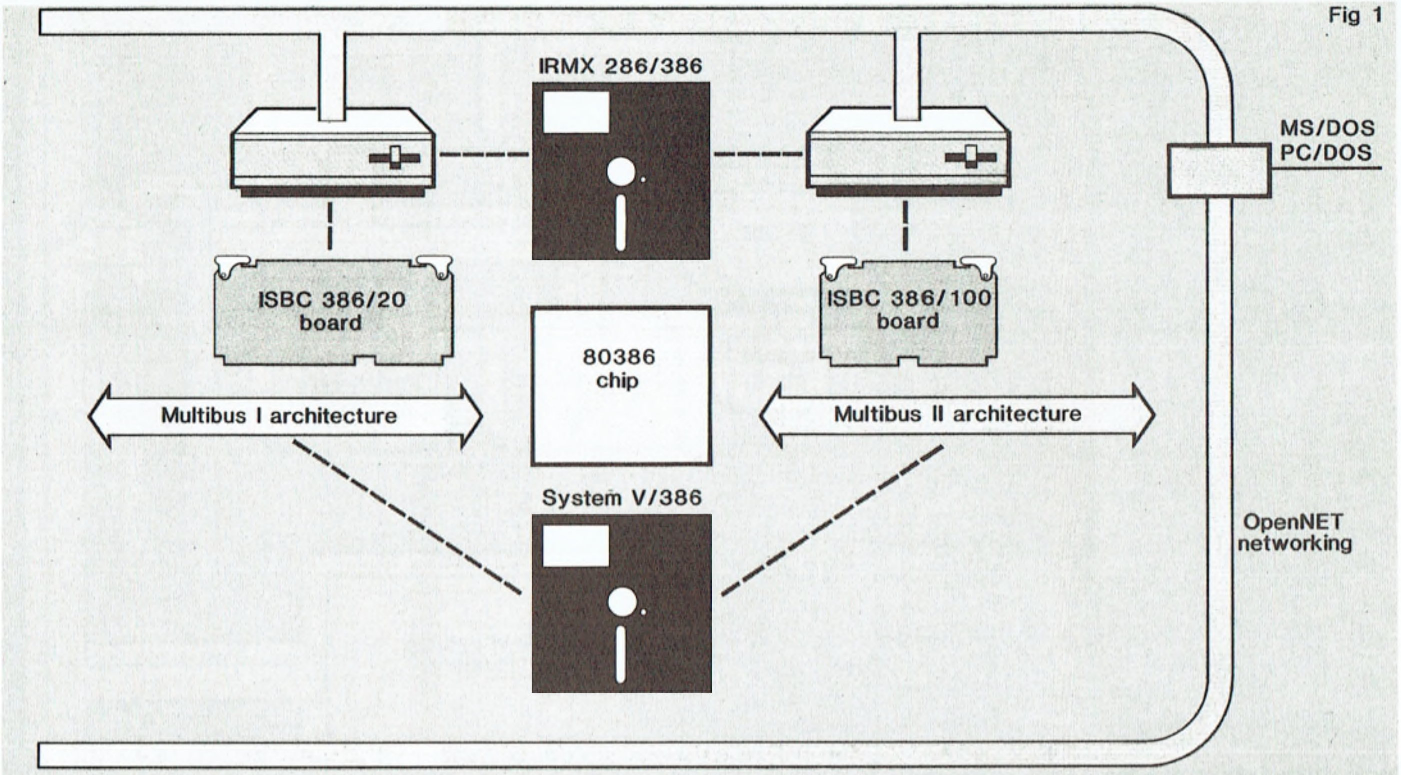
Khaled A. El-Ayat is a project manager in the Intel High-Performance Microprocessor Operation. With Intel eight years, he has contributed to the definition, design, and development of the company's microprocessors. He designed the control structures of the 80386. His technical interests include microprocessor architectures and all aspects of VLSI design.

A member of the IEEE, El-Ayat holds a BSc from Cairo University, an MSc in electrical engineering and computer science from the University of Toronto, and a PhD in electrical engineering and computer science from the University of California, Santa Barbara. He is a part-time lecturer in microprocessors at the University of Santa Clara.

Rakesh K. Agarwal is a senior design engineer in the Intel High-Performance Microprocessor Operation. He contributed to the architectural specification, microcoding, and logic design of the 80386. His primary technical interest is in the definition of the interface needed between hardware and software to make the most optimal use of computer system resources. He is also interested in the design of CAD tools that can be used to assist the architecture-to-logic transformation.

Agarwal received the BSc in computer science from the University of British Columbia and the MSc in computer science from the University of Toronto.

Questions about this article can be directed to El-Ayat at Intel Corporation, Mail Stop SC4-59, 2625 Walsh Avenue, Santa Clara, CA 95051.



Building with the 386

Glen Shires takes a look at Intel's new processor, the 386, and at what it will provide to systems builders

When Intel introduced its high performance 80386 microprocessor at the end of last year, the company stressed that it had been designed to be a most powerful building block for computer systems.

It can easily be connected to high performance memory and I/O subsystems, so system designers need to design their hardware to take full advantage of the computational power of the 386.

To do this, Intel has optimised the external bus for access to the external memory and I/O. Only two clock cycles are required for the 386 to perform a 32 bit bus access, allowing a sustained bus throughput of up to 32Mbytes/sec at 16MHz. Used with cache memory, this high-speed bus can obtain maximum bus bandwidth with cost-effective components.

The 386 also accommodates slower memory and peripheral subsystems. A READY# input to the 386 is used to indicate to the 386 when the bus access has been completed, while the external subsystem can hold READY# inactive to make the 386 stretch the bus cycle with any number of wait-state clock cycles until the subsystem has completed the access.

The 386 has a 32 bit data bus and a 32 bit physical address bus. The 386 instruction set can access data a byte, word, or double-word (dword) at a time (8, 16, or 32 bit access). If a word or dword access is not aligned and crosses two dwords on the 32 bit external bus, the 386 will automatically generate two bus cycles to perform the access.

Nearly every type of 32 bit memory system, for any processor, requires only the upper address bits and uses individual enables for the four lower bytes rather than the two least significant address bits, A1 and A0. The 386 directly generates the upper address bits (A31:2) and the individual byte enables instead of the two least significant address bits.

Although not often necessary, if the two least significant address bits must be generated they can easily be decoded from the byte-enables with four gates. Generally, the only time the least significant address bits are required is when a standard system bus is used.

The 386 bus has been designed so that a variety of memory types and peripherals can be connected. To simplify bus interface logic, only one signal is used to indicate the beginning of a bus cycle (Address Status—ADS), and individual signals are used to indicate write/read (W/R), memory I/O (M/IO) data/code (D/C).

Not only are the 386 bus signals carefully chosen to make interfacing to memory and I/O as simple as possible, but the entire bus is designed to allow very high performance with cost-effective memory. Like some other microprocessors, the 386 uses separate address and data buses (each use a separate set of pins rather than being multiplexed onto the same pins), which allows the bus to run faster. However, the 386 also externally pipelines the address and data bus. Not only can the address for the next bus cycle be pipelined, but the next address can be output by the 386 before the current bus cycle has completed. This allows all address output and buffer delays to overlap completely the current bus cycle memory and I/O systems without any effect on performance. In this way high bus performance can be obtained from slower and less expensive memory and I/O components.

For maximum flexibility external hardware has complete control of when and if the next 386 bus cycle address is pipelined. If the address is non-pipelined it stays valid through the end of a bus cycle. When the external hardware activates the 386 Next-Address pin, the address bus is pipelined and the address for the next bus cycle becomes available before the current bus cycle ends.

A pipelined address provides more time for memory to respond to an access with data without slowing the overall bus cycle time. Therefore, slower and more cost-effective memories can be used without sacrificing bus bandwidth.

A DRAM requires a precharge time between back-to-back accesses in which it must remain idle to guarantee that it does not lose data. In order to get the maximum throughput from DRAMS, two or more interleaved banks can be used so that accesses can be made immediately back-to-back, so long as each access is to a different bank. While one bank is pre-charging, another bank is performing the current access.

With more elaborate DRAM control logic further advantage of the interleaved banks can be gained when using the pipelined address capability of the 386. While one bank of the DRAMs is completing a bus access cycle, the pipelined address can be set up on the address inputs of a second bank of DRAM so the next read access could actually begin before the current access is completed.

For a 16MHz 386 with 150ns DRAMs, so long as the 386 is able to pipeline the address by two cycles, a read access can begin one cycle before the read bus cycle starts and then require only two bus cycle clocks (zero wait-states). With continuous accesses to different banks, the first access will take one wait-state, allowing the second access to be pipelined by two cycles for zero wait-states, and then the next access is pipelined by one clock, again requiring one wait-state. This means sequential accesses can maintain an average of 0.5 wait-states with inexpensive 150ns DRAMs in a 16MHz 386 system.

Fig 2

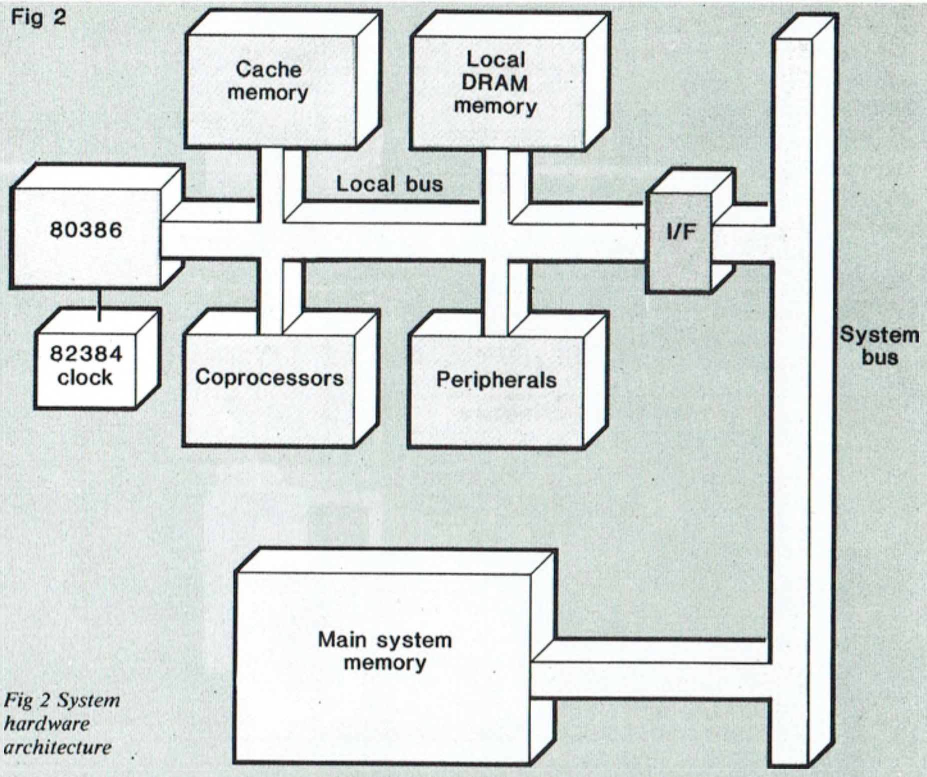
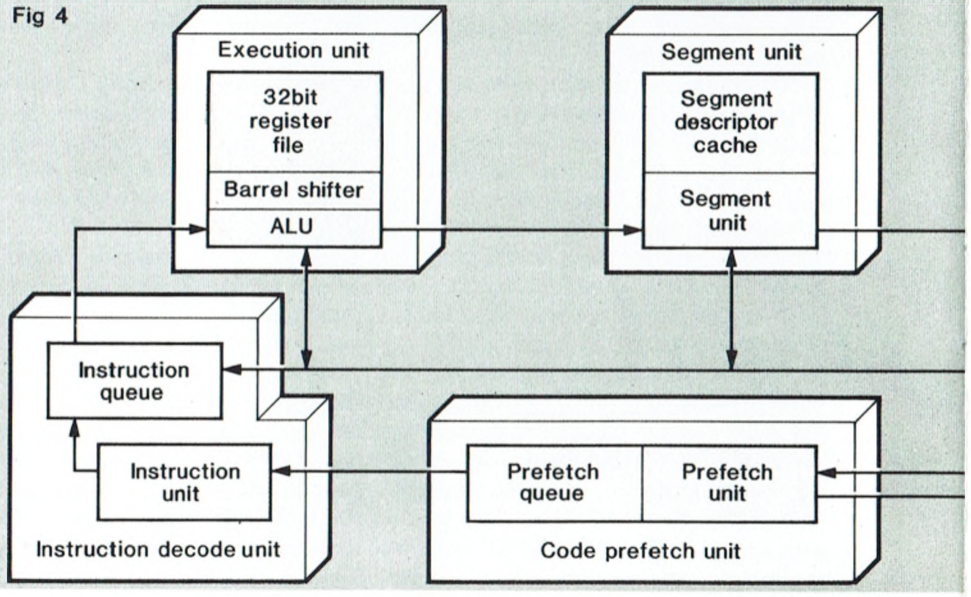


Fig 2 System hardware architecture

Because the 386 allows its external address to be pipelined, cost-effective DRAMs can be used while still maintaining high memory throughput —up to 32Mbytes/sec with 100ns and up to 25.6Mbytes/sec with 150ns DRAMs. Although DRAM memory can be used for very high performance, even higher bus bandwidth can be obtained using static RAMs (SRAMs). At 16MHz 55ns SRAMs are fast enough to complete an access within two clocks without interleaving and without address pipelining. This means that all bus accesses can be completed with zero wait-states. The 386 optimally accommodates these very fast memories by not forcing the address to be pipelined.

In the case when the address of the next bus cycle is not known until after the current cycle has finished (such as indirect jumps or data-dependent accesses) or following an idle bus cycle, access would be actually delayed by a clock. Instead, the 386 can generate bus cycles either with or without address pipelining. Unlike other microprocessors, the 386 is flexible enough to use a simple

Fig 4



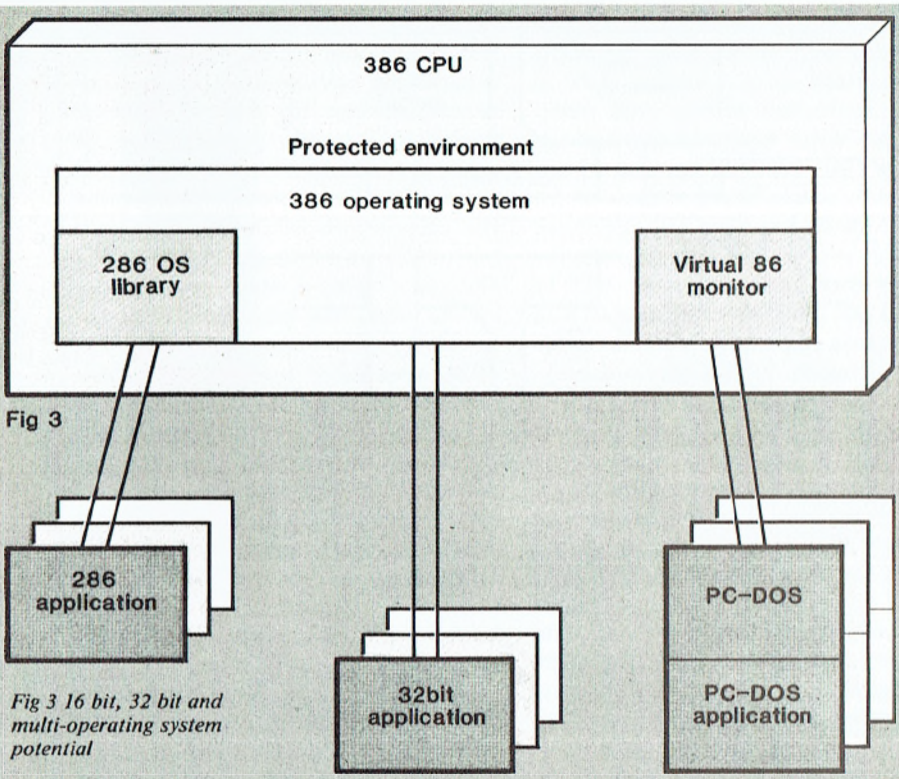


Fig 3

Fig 3 16 bit, 32 bit and multi-operating system potential

memory interface that provides all the benefits of both pipelined and non-pipelined address buses. Allowing the use of pipelining is most optimal for slower memories such as DRAMs; not forcing pipelining to be used is most optimal for fast memories, such as SRAMs.

The 386 does not pipeline the address when it is not available soon enough, and , if the memory subsystem can complete the access in two clocks, it simply activates READY # , so no penalty is paid for the lack of a pipelined address.

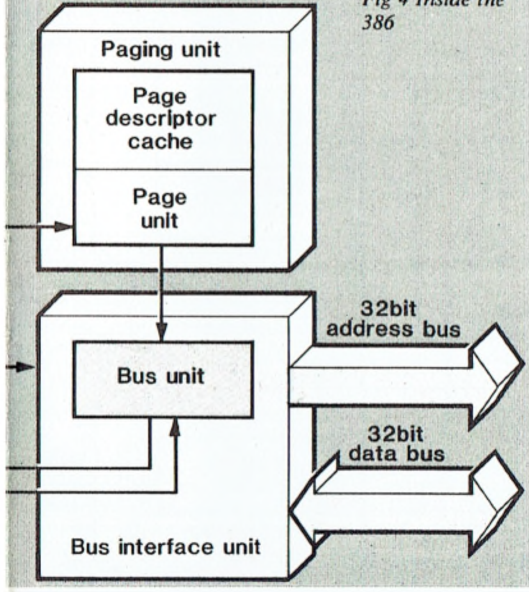


Fig 4 Inside the 386

Although SRAMs provide the highest bus performance, DRAMs can provide much more memory space for much less cost and board space.

A cache system can be used to take advantage of the best features of both and can be built to allow the most often accessed code and data to be stored in the fast SRAMs and less often used code and data to be stored in the slower DRAMs. This lowers overall memory cost while still providing near zero wait-state performance.With 32kbytes of cache, it will have an approximate hit rate of 86%, which means that 86% of the time the data or code in the cache and bus accesses require zero wait-states; with 64k of cache the hit rate increases to near 92%.

While current silicon technology prevents useful general-purpose caches being placed on the processor chip, relatively small on-chip MMU caches can be very effective. The 386 caches both segment and page descriptors. By placing a 32-entry TLB (translation lookaside buffer) page descriptor cache on the 386, over a 98% hit rate has been achieved.

The simplicity of the 386 bus also minimises the logic required to connect to common peripherals and ROM/EPROMs. Discrete logic, PALs, or gate arrays can be used for efficient, customised interfacing to the specific set of peripherals required.

Because the 386 has instructions to perform byte, word, and dword accesses, 8, 16 and 32 bit peripherals can be connected directly to the local bus.

The 386 also has a 'bus size 16 bit' input, which makes it easy to use the 386 on either a 32 bit or a 16 bit bus. This input controls the width of the 386 external bus for the current bus cycle. When inactive the 386 uses its full 32 bit data bus to perform byte, word or dword accesses. When the 16 bit signal is activated, only the lower sixteen data pins are used and unaligned, or 32 bit accesses are broken up into multiple 16 bit word accesses.

The 16 bit bus capability is useful with 8 and 16 bit peripherals because the local bus has fewer interconnections and is less costly. Also, peripherals can be placed at 16 bit word boundaries to duplicate the I/O address used in other 16 bit systems, perhaps for software compatibility with 8086 and 80286 systems.

In addition, if access time to ROM/EPROMs is not critical—for when they are only used for system initialisation—then a 16 bit bus can be used so that only 2bytes of ROM/EPROM width are required rather than 4bytes.

The 386 permits up to 256 different types of interrupts. Some of these are traps generated internally by the 386 such as divide-by-zero or protection violations. The majority, however, are free for use as external hardware interrupts and may be used to indicate to the 386 when a certain I/O device requires servicing.

A 8259A programmable interrupt controller can be connected directly to the 386 to allow eight vectored interrupts to be used, which along with the 386 Non-maskable Interrupt pin provide for nine hardware interrupts. Multiple 8259A chips can also be cascaded if more hardware interrupts are required. To move data between the RAM and peripherals, it is possible to interrupt the processor, have it save its state, perform the transfer, and then restore the processor state. However, it is more efficient to move large amounts of sequential data with a DMA controller, so as not to disturb the processor's execution. The 386 provides hold (HOLD) and hold acknowledge (HLDA) pins which permit a DMA controller or another processor to tri-state the 386 bus signals and take over the bus to perform such transfers. A very common use of DMA is to perform transfers between memory and mass storage, such as in the swapping of virtual memory.

The 386 permits segments of various sizes, from one byte up to 32 (4Gbytes) which is sufficient for nearly any code of data structure. With a possibility of such large segments, it is easy for the programmer to place his data and code into segments, but it would often be very difficult, if not impossible, for the operating system implementing virtual memory to find space in physical RAM to swap in the entire segment. Therefore the 386 also implements paging underneath segmentation. If used, a segment can be split up into several regularly sized pages, possibly with each page corresponding to the size of a block of mass storage. This not only allows portions of segments to reside in RAM, but also permits segments to be split up into several parts to fit any free, but not necessarily contiguous, RAM, and eliminates memory fragmentation problems.

While segmentation allows the software programmer to place his code and data into natural and efficient structures, and provides the operating system with valuable information of which portions of memory are good candidates for swapping, paging underneath segmentation allows efficient hardware implementation of virtual memory.

In implementing a demand-paged system, a standard DMA controller can be used to move one or more sequential blocks (pages) of data between the RAM and mass storage. After each series of sequential blocks is moved, the DMA controller normally interrupts the processor and waits for further commands from the processor.

If several blocks are being moved into several non-sequential pages, the DMA must stop after each page and interrupt the processor. The processor must then send new commands to the DMA controller for the next page. This will decrease the processor performance because the processor must stop and handle an interrupt after each page and the overall data transfer from mass storage transfer may be slower. Mass storage is usually implemented as a spinning disk in which sequential blocks pass under the read/write head. If there is not enough time between the end of one page and the beginning of the next for the DMA to interrupt the processor and the processor to reprogram the DMA, then the next page will be missed and the DMA will have to wait for the disk to go around again before transferring the next page.

The Intel 82258 Advanced DMA controller chip can be used to solve both these problems: fewer interrupts to the processor and quicker multi-page transfers. The processor can generate a list of source or destination blocks for the DMA, for example, to swap a series of non-contiguous pages. Then the 82258 can use its data chaining feature to swap all the pages automatically, without interrupting the processor and without reprogramming the 82258 between each non-contiguous page. Therefore the 82258 provides good performance in a demand-paged environment.

The 386 has fast general purpose computation. In addition, for applications requiring high-precision, floating point, or trigonometric or logarithmic calculations, numeric coprocessors can be used to increase further the system performance. A numerics-intensive system might have the numerics coprocessor performing the calculations while the processor performs memory organisation and I/O housekeeping chores.

The 386 has an optimised coprocessor interface, allowing the 80287 and 80387 numeric coprocessors to be directly connected to the 386.

Intel's 80287 provides high-performance numeric support for cost-sensitive 80386 designs, and its 16 bit bus connects directly to either processor and allows numerics instructions to be placed in the processor instruction stream, intermixed with the standard processor instructions.

The forthcoming 80387 has a 32 bit external bus and can also be connected directly to the 386. The 80387 is a fully software compatible superset of the 80287, executes at a faster internal speed, and uses a faster external bus to achieve 8 times the performance of the 80287. The 386 automatically recog-

Fig 5

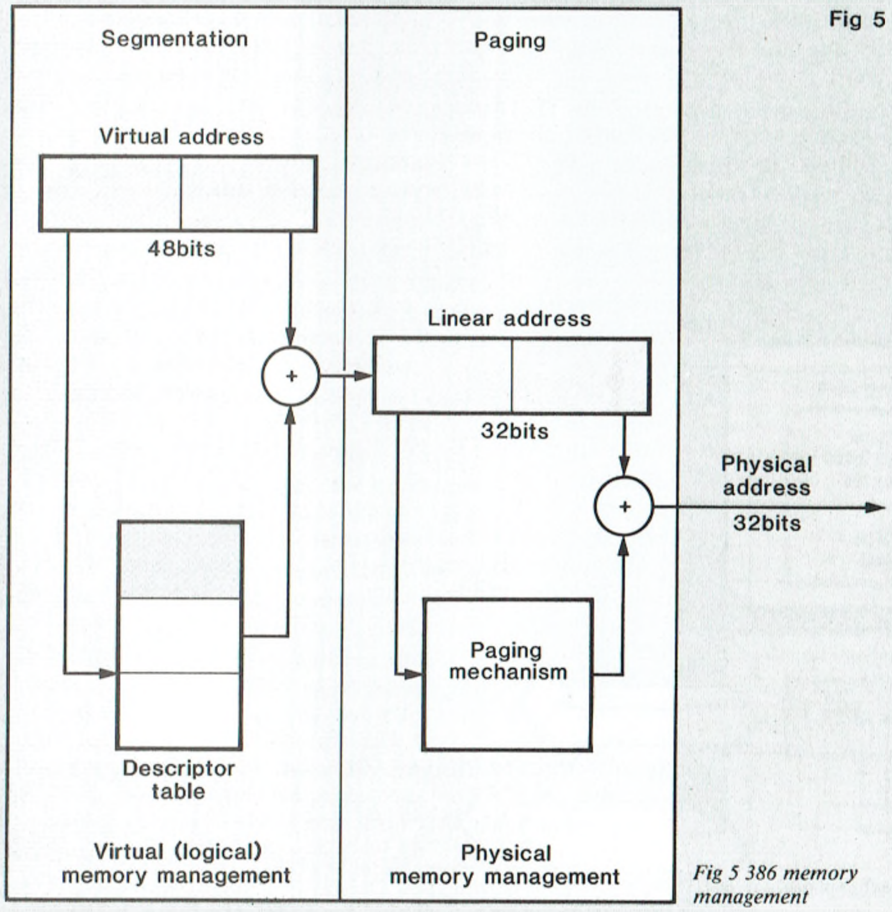


Fig 5 386 memory management

Fig 6

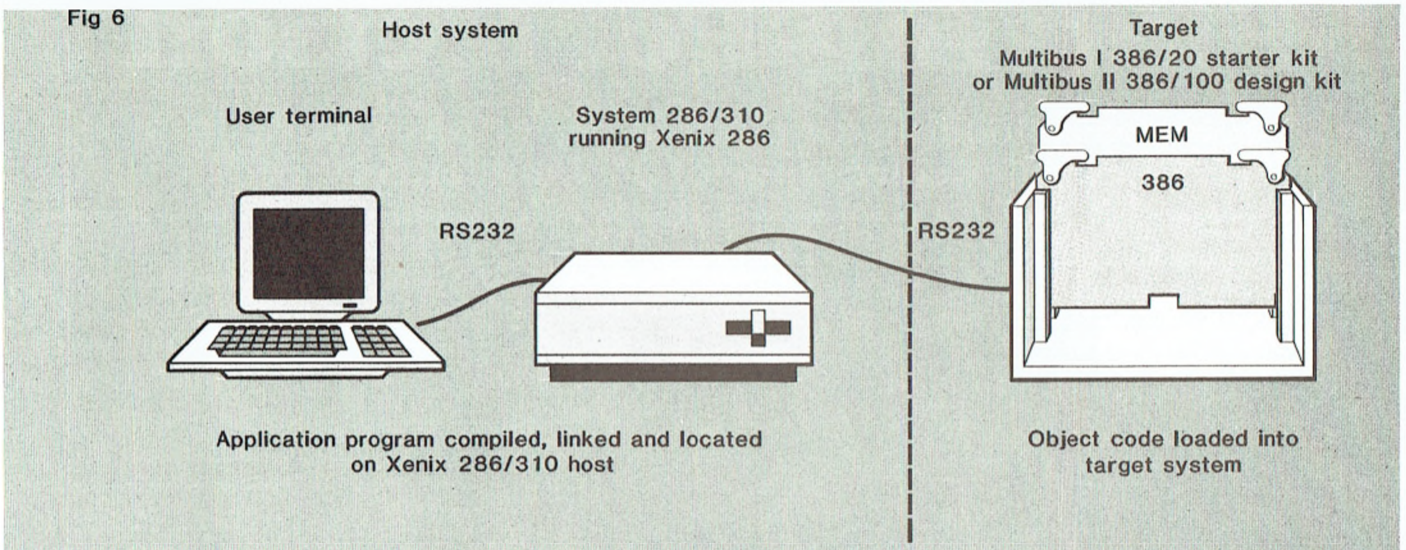

Host system
User terminal
System 286/310 running Xenix 286
RS232
Target
Multibus I 386/20 starter kit or Multibus II 386/100 design kit
MEM
386
RS232
Application program compiled, linked and located on Xenix 286/310 host
Object code loaded into target system
Host user sets breakpoints, starting location etc., via monitor package

Fig 6 The basis for a 386/Multibus/Xenix system

nises whether the coprocessor is an 80287 or 80387 and performs the appropriate 16 bit or 32 bit transfers.

Numerics-intensive applications can run 10 to 100 times faster when a numeric coprocessor is used.

The flexibility of the 386 bus makes it easy to connect it to 'industry-standard' system buses such as Multibus I and Multibus II. The 386 bus signals can be easily translated into 80286-style signals to use the Multibus I bus controller (82288) and arbiter (82289).

The 386 bus size 16 bit input is useful in connecting to a 16 bit bus such as the Multibus I bus.

For higher performance on a system bus, the 386 can also be used with the high-speed, advanced 32 bit Multibus II, which uses message passing protocols to achieve a bandwidth up to 40Mbytes/sec.

Design example

An engineering workstation typically requires high computational power and a very large memory space to perform such tasks as design simulation and verification. In addition, the ability to run multiple programs at the same time allows the designer to maximise this effectiveness, because he can work on one part of the design while simulating another part in the background. In many cases, the ability of the workstation quickly to create and manipulate detailed graphical pictures of the design is also essential to allow the designer to enter and view his work efficiently. Finally, because most designers work in teams rather than individually, it is important that their workstations can easily communicate with each other.

A complete 386 system design meets all these needs. Its fast processing speed and large physical and virtual address space allow large simulations to be performed efficiently. For high performance with large amounts of cost-effective DRAM, a cache is used to allow near zero wait-state operation. The 386 segmentation and paging capabilities, together with 82258 advanced DMA and the intelligent fixed disk controller (82062), combine to form a powerful and efficient implementation of virtual memory. The 386 on-chip memory management not only provides efficient translations for virtual memory, but also offers protection and multitasking capabilities.

While the computational speed and flexibility of the 386 allows it to perform all the calculation, graphics and communication functions for some workstations, even greater performance can be obtained when specialised coprocessors are used. The 80278 or 80387 Numeric Processor can be used to provide fast floating-point arithmetic, allowing complex and accurate simulations to complete quickly. Intel's upcoming 82786 Graphics/Display coprocessor can quickly generate and manipulate complex text and graphics images as well as create an efficient windowing environment. The 386 simply supplies high-level graphics commands and the 82876 performs the tedious task of pixel updates and display functions. The 82586 LAN coprocessor can handle high-speed communication between workstations: sending and receiving multiple data frames, generate protocols, and perform error checking without microprocessor intervention.

To round out the system the 386 also connects to slave peripherals, including two serial ports for the keyboard and a mouse or modem, and a floppy disk controller. This 386 subsystem could be constructed on a board with a Multibus interface to allow it to be installed in an 'industry standard' system.

This would then provide access to the processors and peripheral devices on other Multibus boards within the same system.

Real-time example

A complex industrial robot requires precision, real-time calculations for fine positioning and axis control. A large physical address space is often required to allow complex sequences and vast amounts of data to be immediately available.

A 386 system design meets the needs of an industrial robot. The 32 bit 386 integers and 80 bit 80287/80387 floating-point numbers can easily provide the speed and accuracy of computation required for the fine positioning resolution of industrial robots. □

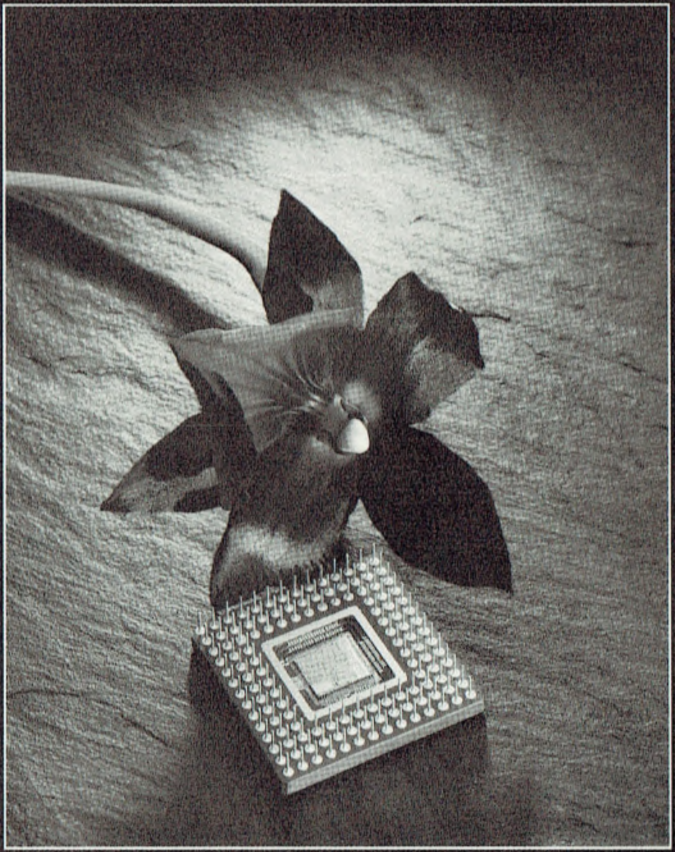
Glen Shires is a member of Intel's US 386 development team. Intel UK is based in Swindon, Wilts. Tel: (0793) 696000.

SPECIAL 32-BIT ISSUE

SOLUTIONS

A Publication of Intel Corporation

November/December 1985



A WELL-BRED CLASSIC
THE 80386

80386 Cache Design

by Glen Shires

Intel's 80386 microprocessor can perform at a sustained rate of 3-4 Million 32-bit Instructions Per Second (MIPS). In order to take advantage of the full potential of the 386, a high performance memory system is required so that the 386 can quickly access code and data. Not only must the memory system be fast, but many of the larger applications that the 386 is capable of executing require large amounts of memory for the program and data. The system designer has the challenge of providing a vast amount of high-speed memory at a reasonable cost.

Traditional microprocessor systems have used DRAMs (Dynamic Random-Access Memories), which can provide cost-effective memory and operate at the speeds required by earlier generations of microprocessors.

In comparison, the 386 is capable of very fast memory accesses. At 16MHz, the 386 has a 50% faster bus than any other microprocessor and can perform each bus cycle in only 125 nanoseconds for a maximum bandwidth of 32 Megabytes/second. The 386 obtains this speed not only from its high clock rate, but because it requires only two clocks for a bus cycle (competitive processors require three or more).

The 386 can make the most of the speed of DRAMs because of its ability to pipeline the next bus cycle address while the current bus cycle is in progress, and its ability to use interleaved DRAM arrays. For the memory requirements of traditional applications, fast DRAMs can be used for a high performance, cost-effective memory system. But the power of the 386 permits much larger applications to be run which require very large amounts of memory. Such a vast amount of memory can be implemented cost-effectively with slow, inexpensive memories. For the highest performance, more expensive, fast memories are required.

A cache memory system provides the answer to this dilemma. It combines the speed of high speed Static RAMs (SRAMs) with the cost-effectiveness of slower, less expensive DRAMs.

Cache Configuration			Cache Performance	
Size	Associativity	Line Size	Hit Rate	Performance Ratio over non-cached DRAM
1K	direct	4 bytes	41%	0.91
8K	direct	4 bytes	73%	1.25
16K	direct	4 bytes	81%	1.35
32K	direct	4 bytes	86%	1.38
32K	2-way	4 bytes	87%	1.39
32K	direct	8 bytes	91%	1.41
64K	direct	4 bytes	88%	1.39
64K	2-way	4 bytes	89%	1.40
64K	4-way	4 bytes	89%	1.40
64K	direct	8 bytes	92%	1.42
64K	2-way	8 bytes	93%	1.42
128K	direct	4 bytes	89%	1.39
128K	2-way	4 bytes	89%	1.40
128K	direct	8 bytes	93%	1.42
no cache – 2 CLK SRAM access			(100%)	1.47
no cache – 4 CLK pipelined DRAM			—	1.00

Table 1: Performance of Various Cache Configurations

WHAT IS A CACHE?

A cache memory is simply a fast and relatively small memory (typically SRAM) that is located between the processor and larger main memory (typically DRAM). The intent of a cache system is to make all of the large main memory appear to respond as fast as the high-speed SRAMs respond. To do this, the memory system keeps all of the RAM code and data in DRAM and also keeps a copy of only the most often accessed code and data in the fast SRAMs. This provides a large amount of RAM memory (implemented in inexpensive DRAM) and still allows most code and data accesses to be quickly completed by using the faster, smaller SRAMs.

What Information is Stored in a Cache?

The fast SRAM cache memory is much smaller than the main system DRAM and therefore can only contain a copy of a small fraction of the information in the DRAM. In order to make the cache useful, it must contain information that the processor will likely require in the near future.

Because most computer programs have a habit of re-accessing the same memory locations soon after the first reference to these locations, it is possible to assume that memory that recently has been accessed will probably be required by the processor in the near future. A cache memory can be designed to store the most recently used information and have a good probability that the processor will soon require much of the same information again.

For the first reference to a location, the processor will generally have to wait for the access from the lower-speed main memory. While the processor is performing this access, the cache will automatically place the data in its memory. When the processor needs to access this same location again, it can go directly to the high-speed cache memory.

When the cache becomes full and it needs to store the contents of a new memory location, it must replace the contents of an old location with the contents of the new location. In this way the cache maintains the more recently used memory locations.

Hit-Rate Indicates Effectiveness

A cache is only effective if a large percentage of the processor accesses require information that is already in the cache rather than the slower main memory. The percentage of the processor accesses that are already located in the cache is called the cache hit-rate. If, on average, 90% of the processor accesses are found in the cache and only 10% require DRAM accesses, then the cache is said to have a hit-rate of 90%.

Because the DRAM read-cycle is not generally begun until after it is determined that the required memory location is not in the cache, cache miss read-cycles actually take longer than a DRAM read-cycle would if no cache was used. While cache hit accesses are much faster than straight DRAM, cache miss cycles are slightly slower. If the hit rate is too low, then using the cache can actually decrease overall performance. An effective cache requires a hit rate of well over 50-60%, which corresponds to a minimum size of approximately 2K-bytes.

The 386 cache designer has the freedom to choose the cache size and configuration to determine the hit rate, thus controlling the overall cache effectiveness.

Reprinted with permission from Solutions Magazine, 1986.

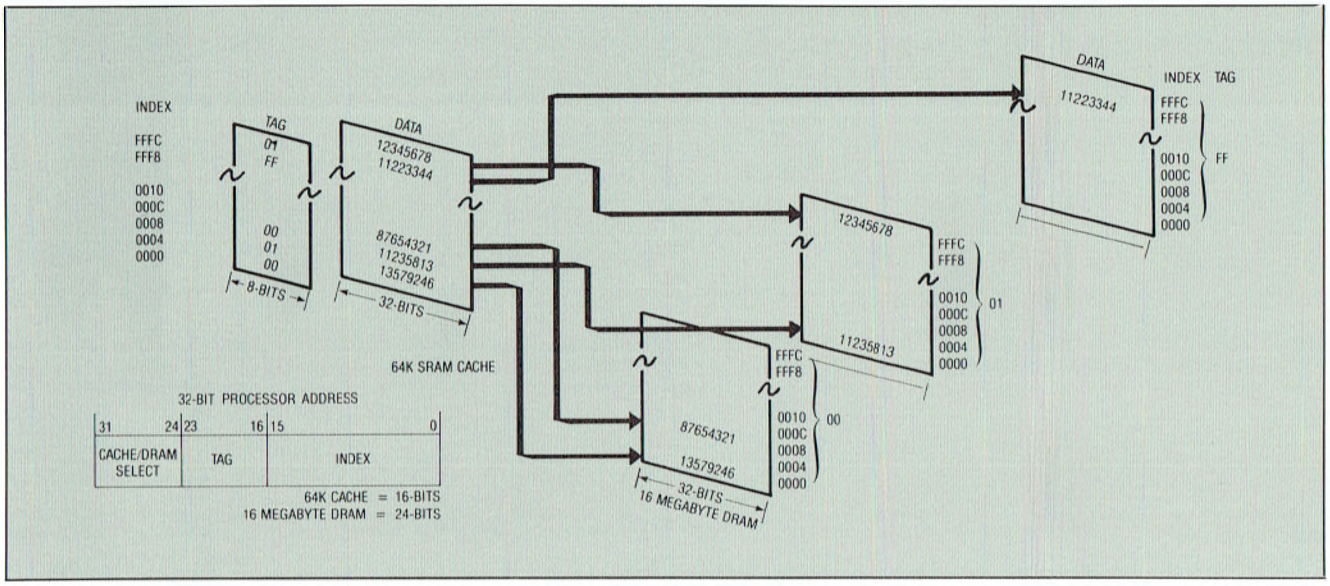


Figure 1: 64K Direct-Mapped Cache

CACHE DESIGN
Several factors which influence the effectiveness of the cache have to be considered during design:

Cache Size
The larger the cache, the more information it can store. A larger cache has a greater likelihood of containing the information that the processor needs, and therefore has a higher hit-rate.

Cache SRAMs contain two types of information. The data SRAMs contain the contents of the memory locations stored in the cache. The tag SRAMs are used to store information to "remember" the full DRAM address of each of these entries. The term "cache size" refers only to the size of the data SRAMs and neglects the size of the tag SRAMs.

A typical cache size could be 64K, because that is the largest size possible when using the minimum number of currently available SRAM chips (eight 16Kx4 SRAM chips for 64K of 32-bit data).

The performance for various configurations of caches based on Intel simulations is shown in Table 1. As reference points, the bottom of the table compares these cache configurations with two non-cached configurations: a full zero-wait-state SRAM memory and a non-cached two-wait-state address-pipelined/three-wait-state unpipelined DRAM memory. Listed for each configuration is the hit rate and the performance ratio compared to the non-cached DRAM. All of these cache configurations combine code, data, and stack information for the best cost/performance ratio.

Cache Associativity
When the cache becomes full and needs to store the contents of a new location, it must overwrite an old location. The cache entry in which the new information is to be replaced is determined, in part, by the cache associativity. The cache associativity determines how many different places in the cache any particular DRAM location can be mapped into.

At one extreme is the fully associative cache, which is the most flexible and most complex organization. It allows the contents of the new location to be placed into any entry of the cache. When new information needs to be stored in the cache, an algorithm is used to determine which cache entry is least likely to be required again in the near future (possibly the least-recently-used), and that entry is overwritten by the contents of the new location. Then, whenever the processor makes a memory access, every entry of the cache must be checked to determine if the information is stored in the cache.

At the other extreme is the direct-mapped cache, which is the simplest organization. Each location in DRAM is directly mapped into only one cache entry. When the cache needs to store the contents of a new location, there is only one cache entry that can possibly be used. Also, when the processor makes a memory access, only one entry of the cache must be checked to determine if the information is stored in the cache.

Figure 1 illustrates a 64K direct-mapped cache with a tag field large enough to cache up to 16M-bytes of DRAM. The following discussion will be based on a cache and DRAM of this size. For a different size DRAM and cache SRAM, the number of bits for each field differs, but the concepts apply directly.

The 386 32-bit physical address is divided into three fields: Select, Tag, and Index. The 8 most-significant bits (Select) are used by the chip-select logic to determine if the memory access is to the cache/DRAM subsystem. The 24 least-significant bits (Tag and Index) determine the 16M-byte DRAM address. The 24 least-significant bits are also broken up into two fields for the cache SRAM. The 16 least-significant bits (Index) are used as an index to address the SRAMs. The middle 8-bit (Tag) field is used as tag information.

The cache SRAM is partitioned into two parts: data and tag. The 64K bytes (16K entries x 32-bits) of data SRAM contain the contents of the memory locations stored in the cache and are addressed by the Index field. The two least-significant bits of the Index field are used as byte-enable select information, and only the upper 14-bits of the Index field actually address each of the data SRAMs.

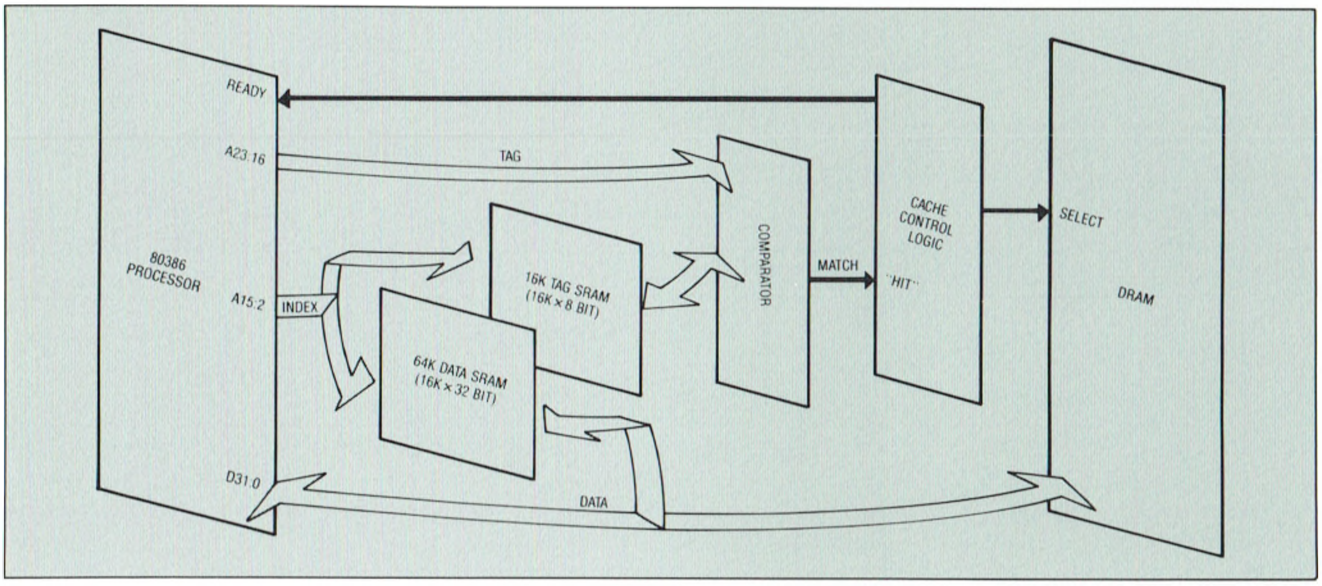


Figure 2: 64K Direct-Mapped Cache

The tag SRAM is also addressed by the upper 14-bits of the Index field. The tag SRAMs are used to store the Tag field, which corresponds to the upper DRAM address bits. Since the Tag field is only 8-bits wide, the tag SRAMs are 16K by 8-bits.

Whenever the processor makes an access to this direct-mapped cache, the Index field is used to address the tag SRAMs and the tag SRAM contents are compared with the processor Tag field. If they match, then the cache contains the data of the accessed location in the data SRAMs. If a match is found on read accesses, a "cache hit" is said to occur, and the read-data is simply read out of the cache data SRAMs. If a match is not found on read accesses, a "cache miss" is said to occur, and a DRAM access must be performed to obtain the data. Normally, on a read cache miss, the DRAM data are not only returned to the processor, but also placed into the cache.

Because the cache is usually used for every memory access, the tag matching and cache or DRAM accesses must be performed in hardware for the cache to be effective. Figure 2 shows a block diagram of the hardware required for this direct-mapped cache.

Although the direct-mapped cache has the advantage of using very simple hardware, its disadvantage is that each DRAM location corresponds to only one cache entry. It is very possible to have a program that constantly accesses two DRAM locations that happen to correspond to the same cache entry. Every time one location is accessed, it is placed into the cache replacing the data from the other location. When the other location again needs to be accessed, it too must read from the slower DRAM and replace the data of the first location in the cache. This constant thrashing makes the cache ineffective for these particular memory locations.

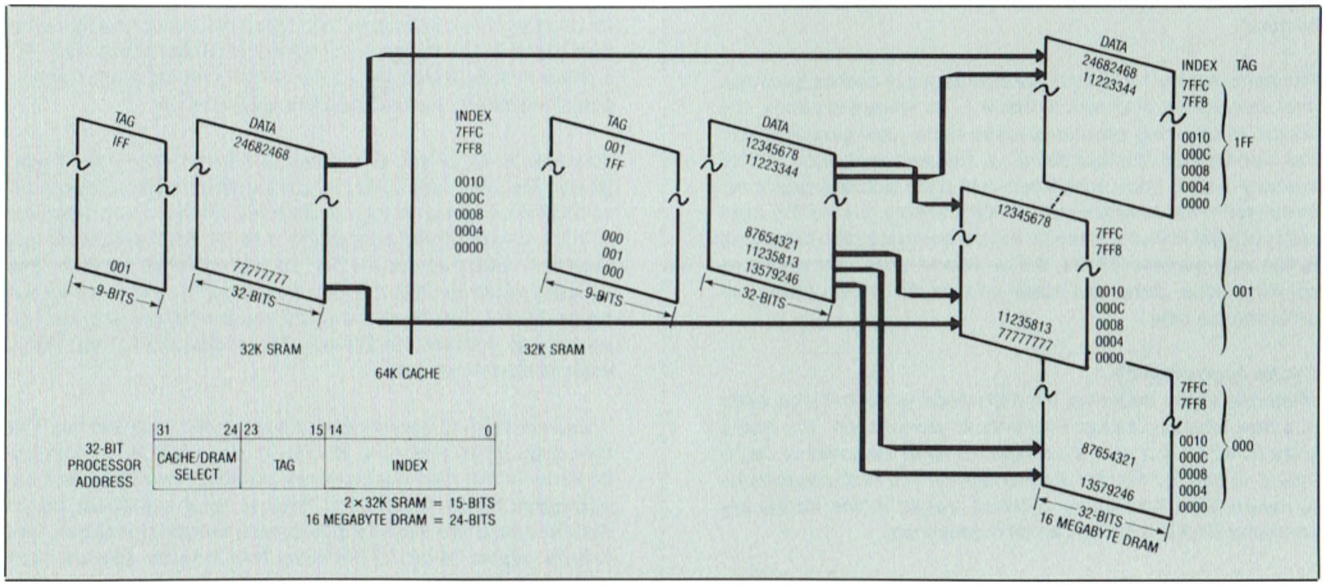


Figure 3: 64K Two-Way Associative Cache

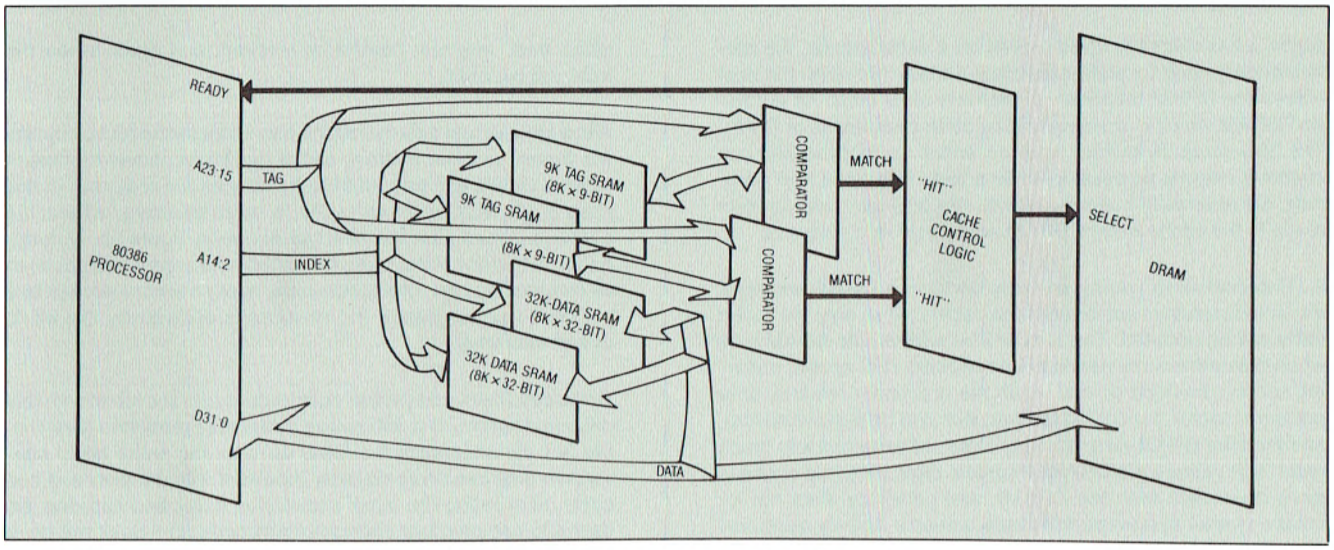


Figure 4: 64K Two-Way Associative Cache

This problem is eliminated with a two-way associative cache, as illustrated in Figure 3. Each DRAM location can potentially be located in either of two cache entries. This has the advantage in that no two locations can constantly thrash between a single cache entry (but there is still the chance that three or more locations will thrash). Therefore, the hit-rate for a two-way associative cache is somewhat higher than a direct-mapped cache when both caches have the same amount of data SRAM. Figure 4 shows a block diagram of a two-way cache. The implementation requires approximately twice as many chips. Because each cache Index has two entries, two tag SRAMs (both bigger than the direct-mapped implementation) and two comparators to match the tags are required.

Similarly, a four-way associative cache can be constructed for an even higher hit-rate, but it requires additional tag SRAMs and control hardware. Finally, a fully associative cache requires very complex hardware because it allows each DRAM location to map to any cache entry and therefore requires a comparator on each tag location (a structure often called "Content-Addressable Memory").

In previous generations of computers, it often made sense to use a two-way or four-way associative cache because fast SRAMs were small and very expensive relative to comparators and random control logic. For a reasonable size cache, many SRAM chips were required, and the additional logic to allow two-way or four-way was minimal relative to the SRAM cost and board space.

Today, however, fast SRAMs have become much larger, less expensive, and require less board space. Because the number and cost of SRAMs for a reasonably-sized cache has dropped much more than the cost and board space for the comparators and control logic, it now makes sense to minimize the comparators and control logic and increase cache effectiveness by simply making the cache larger. Therefore, large, direct-mapped caches are generally preferable over smaller, associative caches when implementing the cache out of chips commonly available today.

As fast 16Kx4 SRAM prices decrease, a 32-bit direct-mapped 64K cache for 16M-bytes of DRAM can be implemented with only 10 SRAMs (eight data and two tag). It would make little sense to double the cost and board space to implement a two-way associative 64K cache, which requires 21 SRAMs (sixteen data and five tag 8Kx4 SRAMs) for a relatively small improvement in performance. If more performance is required, it is generally more cost-effective to increase the size of the cache rather than the level of associativity. Therefore, the typical cache implementation is direct-mapped.

Write Policies

Read accesses with a cache are relatively straightforward. When the data are found in the cache, they are used. When the data are not found in the cache, a DRAM cycle is performed.

The simplest write policy is called "Write-through". For every processor write-access, a DRAM write-cycle is performed to update the active bytes in DRAM so that the DRAM data are kept up-to-date. In addition, if the write location is found in the cache, the active bytes are also updated in the cache.

The processor write-accesses that update all 32-bits could always update the cache, whether or not the location was already contained in the cache. However, because all four cache bytes are always assumed to be valid, it is impossible to update the cache with a new location if the processor does a write-access with only some but not all bytes active. Therefore, most write-through schemes will only update the cache if the cache already contained the location of the write-access.

A simple variation on write-through is "Buffered-write". Like write-through, the DRAM is updated after every processor write-access. Unlike write-through, the DRAM write-access is buffered so that the processor and cache can immediately begin the next bus cycle. Usually, only one level of write-accesses can be buffered; if two consecutive writes are performed, the second write must wait until the first finishes. If, however, a write-access is followed by read-accesses and the data is found in the cache, then the processor read-accesses can occur while the DRAM is busy writing.

Buffered-write requires slightly more control logic and fully latched DRAM address and data buses. Because the majority of processor accesses are not writes, and because it can take the cache a few clocks to update itself on a write-access, the performance gained by buffered-write is usually not worth the cost unless the DRAM response is relatively slow (such as a dual-port DRAM which is in heavy use by other processors or DMA). The 386 internally buffers write-accesses so that the 386 can continue internal processing while a write bus cycle is in progress. By externally buffering writes, the 386 can also perform reads to the cache while a DRAM write cycle is in progress.

A "Deferred-write" policy will only perform a DRAM access if the write location is not found in the cache. If it is, only the cache entry will be updated. The data will be written into DRAM only when the cache entry needs to be replaced. (For speed, buffering can be provided so that when the processor requires data not in the cache, the DRAM read-access can be performed first, and then the DRAM write-access.) Deferred-write requires much more logic, allows the DRAM to contain stale data (the cache is more up-to-date than the DRAM), and generally does not increase overall system performance unless a heavily accessed dual-port DRAM is used.

Most microprocessor caches will probably use write-through or buffered-write caches for cost/performance reasons. A typical microprocessor cache uses the write-through policy. With the addition of extra DRAM address latches, and by substituting registered transceivers for the DRAM data transceivers, buffered-write can be implemented with a small change in the control logic.

Bytes Per Line

The most straightforward cache for a 32-bit processor is 32-bits wide (4 bytes per line). However, there is no reason why the cache and DRAM cannot be made wider, possibly 64-bits for 8 bytes per line. Because most programs generally access locations near those accessed recently, filling each cache entry with 64-bits rather than 32 increases probability that future accesses will be found in the cache. With 64-bit wide cache and DRAM, each 32-bit access can be made at full speed, with no penalty for accessing both 32-bit words simultaneously.

Unfortunately, more bytes per line require more cache SRAMs, more DRAM chips, and wider buses. In many cases, the incremental increase in performance cannot justify the extra cost. A typical 386 cache implementation uses 4 bytes per line.

Physical vs. Virtual Address Caches

Most larger systems that require a cache will also use virtual memory. This brings up the question of whether caches should use the virtual or the physical addresses; that is, whether the cache should come before or after the Memory Management Unit (MMU).

Figure 5a shows a virtual address cache. Because the cache is located before the MMU, the processor can make very fast accesses to the cache. Cache hits do not require the MMU conversion delay, so the cache bus cycles can be performed several clocks faster than if the conversion was required. Unfortunately, because the same virtual addresses are often used by different tasks, the entire cache must be flushed every time the processor switches tasks (all entries become invalid). Therefore, any time a new task is invoked, the cache is effectively empty and all the first accesses are cache misses, requiring DRAM cycles. Also, if interrupts are handled as a separate task with their own virtual addresses, the cache must be emptied when each interrupt handler is invoked, and again when the interrupt is exited.

Although a virtual address cache can make cache hit bus cycles run faster, a virtual address cache quickly becomes inefficient when a significant amount of task changes are required. In this case a physical address cache is more effective, where the cache is placed after the MMU as shown in Figure 5b. Unfortunately, when the MMU is implemented as a separate chip, such as required by non-Intel processors, every memory access to a physical address cache (hit or miss) is delayed by the MMU conversion time.

While designers using other microprocessors are faced with this difficult dilemma, the 386 solves both these problems (see Figure 5c). By integrating the MMU on-chip, the entire MMU conversion time can be completely pipelined with the previous bus cycle, eliminating the usual conversion delay and allowing the cache to use physical addresses efficiently so it does not have to be flushed for task switches or interrupt handlers. Therefore, the 386 provides the speed of virtual address caches without the constant cache flushes that make virtual address caches inefficient. Every 386 cache design uses physical addresses.

TYPICAL CACHE IMPLEMENTATION

Figure 6 shows an actual implementation of a 386 cache. This is a 64K, direct-mapped, write-through, physical address cache. It consists of only 10 SRAM chips, and a few TTL and PAL (Programmable Array Logic) chips. Some of the PAL control logic and the TTL latches can be shared with the DRAM and I/O subsystems.

On a read-cycle, the 386 Index address bits are latched and then sent to the Tag and Data SRAMs. The comparator checks the Tag SRAM data against the 386 Tag Address. If they match, the 386 READY line is activated and the data in the Data SRAMs

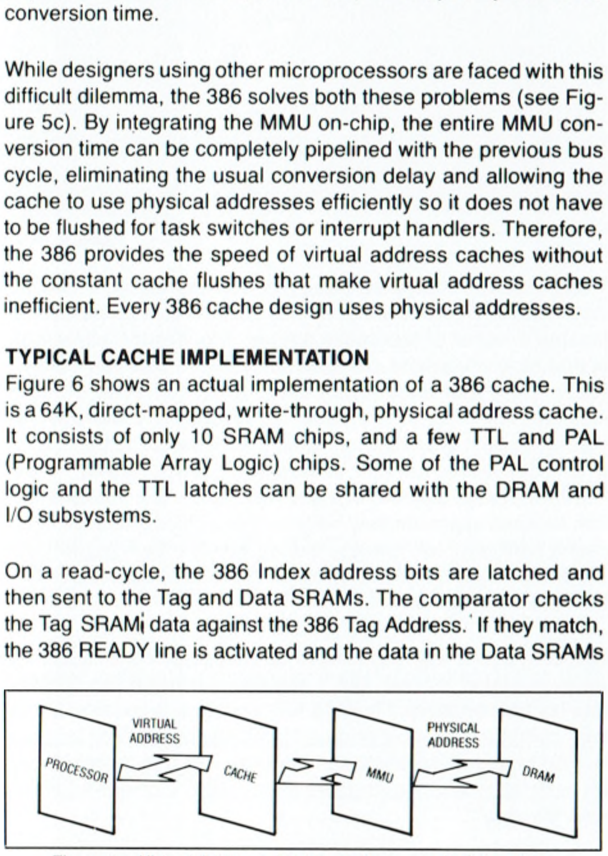


Figure 5a: Virtual Address Cache requires constant flushing

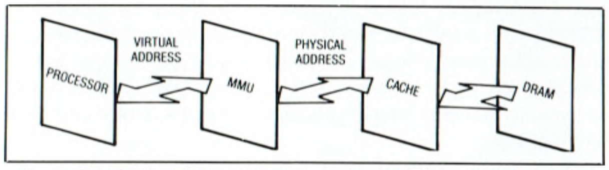


Figure 5b: Physical Address Cache slowed by MMU conversion delay

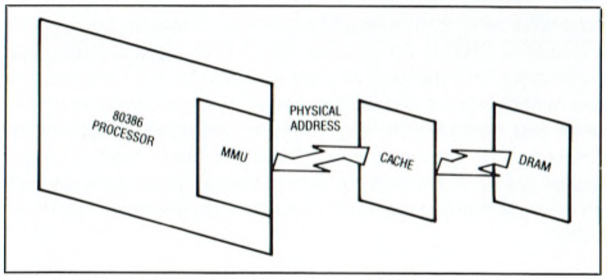


Figure 5c: 80386 Physical Address Cache eliminates constant flushing and conversion delay

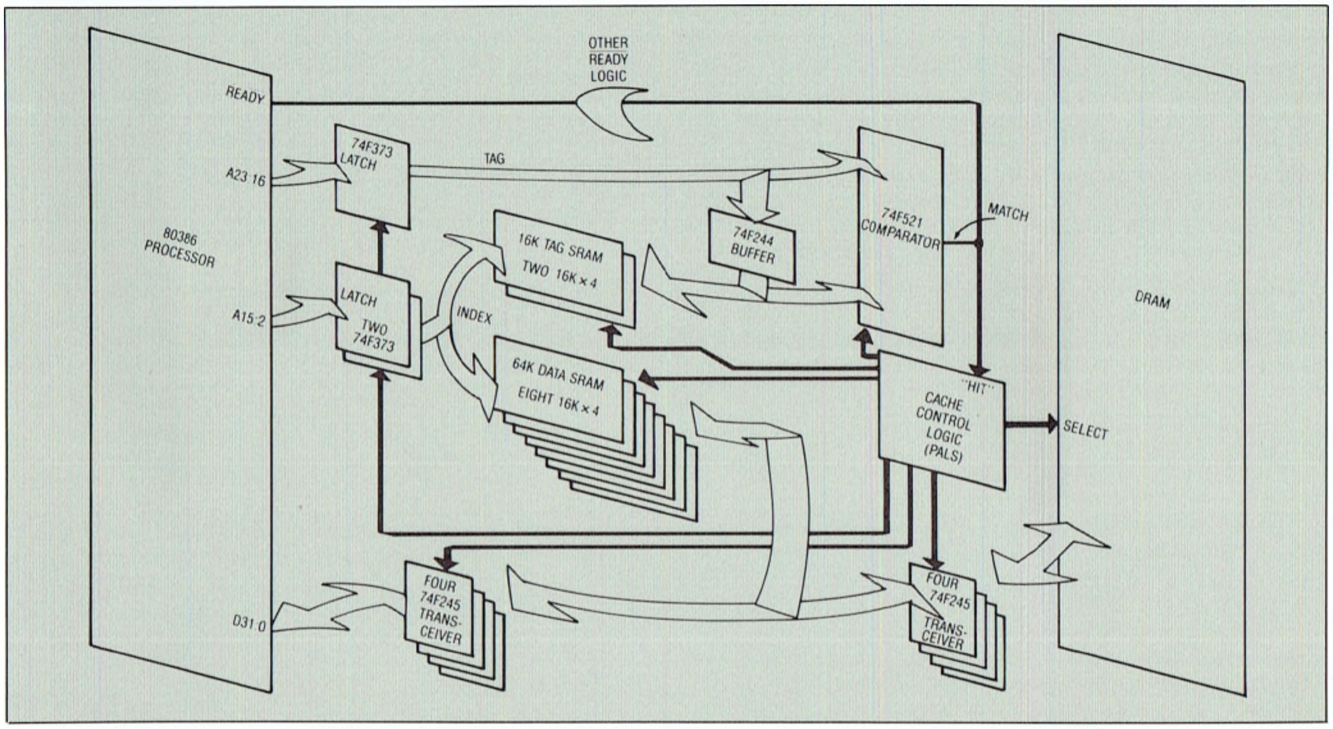


Figure 6: 64K Direct-Mapped, Write-Through, Physical Address Cache

are sent to the 386, providing a zero-wait-state cache hit read cycle.

A state machine, implemented in PALs, follows each 386 bus access and generates the required control signals. These PALs sample the 386 status signals and the comparator to generate the appropriate read/write and cache hit/miss cycles.

The critical paths that determine the required access times for the SRAMs are shown in Table 2.

Performance

The performance for this cache is shown in Table 3. Also shown is the overall performance when the cache is used with a DRAM subsystem that always requires four clocks per access.

Any read access found in the cache runs at zero-wait-states, independent of the DRAM subsystem speed. Because the cache can run with and without address pipelining, these performance times are independent of address pipelining unless the DRAM-writes take advantage of the pipelining.

Tag SRAMs: tag access, compare and activate READY in 2 CLKs												
2× CLK period		386 address valid		74F373 latch		74F521 compare		74AS32 "OR"		386 READY set-up		Tag SRAM access time
2× 62.5	–	40	–	8	–	11	–	6	–	20	=	40nS @ 16MHz
2× 83.3	–	44	–	8	–	11	–	6	–	22	=	75nS @ 12MHz

Data SRAMs: data access and pass to 386 in 2 CLKs										
2× CLK period		386 address valid		74F373 latch		74F245 transceiver		386 Data set-up		Data SRAM access time
2× 62.5	–	40	–	8	–	7	–	10	=	60nS @ 16MHz
2× 83.3	–	44	–	8	–	7	–	11	=	96nS @ 12MHz

Table 2: Critical Paths Determine SRAM Access Speeds

		Using 4 CLK DRAMs	
Access type	CLKS per access	CLKS	wait-states
read-hit	2	2	0
read-miss	2 + DRAMread	6	4
write	DRAMwrite	4	2

Table 3: Performance of Cache Implementation

CONCLUSION

Because the 386 has a very fast two-clock bus, it can optimally interface with a cache. A cache implementation can be used to provide a large memory with very high performance at a fraction of the cost of zero-wait-state RAM. By permitting the 386 to access code and data at these speeds, the maximum potential of the 386 can be obtained. □

L.F. ROTHSCHILD, UNTERBERG, TOWBIN
55 WATER STREET, NEW YORK, N.Y. 10041
TEL: (212) 412-1000 CABLE: PROBATICA

TECH NOTES

December 1985

Tech Notes, a monthly publication, is intended to present an overview of our strategy, discuss sector and company fundamentals and review technology stock performance. In addition, we will feature topical selections from our technology research effort.

Stephen E. Yoken, Director of the Technology Research Group
(212) 412-1822

T-01,02,03,08,42,44,73

L.F. ROTHSCHILD, UNTERBERG, TOWBIN
55 WATER STREET, NEW YORK, N.Y. 10041
TEL: (212) 412-1000 CABLE: PROBATICA

Technology Strategy Review

Outlook—More Positive Environment Unfolding

In our November Technology Strategy Review we hypothesized a more positive environment for technology companies in 1986. Our only real reservation was in the area of demand recovery—i.e., whether the moderate rate of growth forecast for GNP would offer enough stimulus to generate higher levels of consumption and start revenue growth accelerating. Recently, the Federal Reserve Board has gone to great lengths to convince everyone of its accommodative intentions toward 1986. With the likelihood, therefore, of stable interest rates, and no signs of inflation, we feel that a serious decline in the consumer sector is unlikely. Meanwhile, a combination of other favorable forces is at work (as discussed in November), such as a dollar-related currency swing, continued technology company restructuring, inventory replenishment and a new computer industry product cycle.

It seems to us that most high technology companies will be able to record better earnings results next year even in an environment of moderate GNP growth because most moves toward downsizing and other expense reduction programs will have been in place for some time. In addition, computer companies are beginning to report improved order rates. If we superimpose the various new product cycles on this scenario, then the stage is set for some leveraged operating earnings gains. It is important to note that a very sensitive early indicator—lead times—is now giving significant postive signals, as discussed later in this report. The major short-term risk to our bullish scenario is the potential plateauing in the recent order growth momentum sometime in the first half of 1986. While this may prove to be temporary until the various end-markets pick up some steam, it could be the basis for some profit taking in technology stocks. We are not at that point yet, but we do feel a correction could occur early in January 1986, as the pressure for year-end portfolio performance eases.

Taking all these factors into account, we continue to recommend accumulation of selected high quality technology stocks, and advise being particularly aggressive on any pullback. We are also strongly persuaded that valuations will continue to rise and that upside surprises in reported earnings are very possible in 1986. We feel that more attractive values have existed and still do exist in the computer and related sectors, given their currently more visible fundamentals versus the semiconductor issues. This difference will probably continue until investors see more evidence of recovery. As we point out later, the larger semiconductor issues have merely moved back to the tops of their recent trading ranges; it is the smaller companies that the market has thus far favored with the greater gains. As the evidence mounts that this recovery in the semiconductor industry could be of longer-term duration, we expect investors to shift toward the larger companies in this industry. We would accumulate semiconductor issues for the long term, being particularly aggressive on any market-related corrections. Paul Johnson, our semiconductor analyst, comments on his favorite stocks in "The Year Ahead: Outlook by Industry" issue of the *Notes for Portfolio Managers* (December 19, 1985).

The Semiconductor Industry

Current Business

Clearly, this is the most difficult cycle we have ever seen from the point of view of analysis and forecasting. It has also become the most controversial on Wall Street, raising debate to feverish levels and the ire of company managements even higher.

First of all, different companies in the industry are prepared for different types of recovery. While some have permanently removed plant and people from the operating cost mix, believing that a downsized operating structure is more appropriate to the industry of the late 1980s, others have kept people and plant in place, creating substantial underabsorbed overhead, on the theory that the rebound will be just like every other cycle and being well positioned early will be crucial. Needless to say, there is merit to both positions simply because there are enough differences from past cycles to justify the former strategy and enough similarities to argue for the latter.

The most furious debate is over capacity and inventories. Let's try to sum up the situation:

- Consumption has been running above orders for some time, eating up inventory;
- This is a seasonally good time of the year for the industry, as is the first calendar quarter;
- Most companies have removed substantial capacity from the market, and much that remains is obsolete and very costly to keep;
- People have been removed from the rolls, and we feel that this is the key to the capacity issue;
- Six to nine months will be required to reopen plant, restart production lines, hire and retrain; and
- Examples of improvements in pricing exist and, although they are modest at the moment, we are persuaded that more will occur.

Now let's talk of lead times, which we mentioned earlier. On a recent trip to Silicon Valley, we were quoted a current history of lead times by one company: 90 days ago, lead times of two weeks; 30 days ago, lead times of five to six weeks; and now, lead times of 10-12 weeks. As observers of this industry, we feel what now prevails is much closer to normal. *This is what happens when the cycle is in the process of restarting itself.* Other companies agreed with this information. Current orders appear to be across the board—emanating from both distributors and OEMs—thus differentiating the present situation from the 1982 false start experience. It should be noted that orders are not, for the most part, explosive; but design activity levels have been very active for quite a while. Orders are still concentrated in the 30-day turns or 90-day turn and fill business. Clearly, more visibility on order rate growth must occur before the end-user commits for the longer term.

Current Investor Outlook

The stocks, as one might have expected, shot up from the lows of late September and early October back to prices seen earlier this year—that is, the prices for the mainstream companies. The smaller niche companies—where product focus and dynamics have carried the business for the past two years and where one can identify genuine proprietary technology, leadership and a competitive edge—have all outperformed in recent weeks. We are not recommending avoidance of the larger companies, just great selectivity. Whether the established companies are able to reorient product mix is the issue we would focus on. While this may be intellectualizing—since the stocks have always moved as a group—we suspect that investors will, over time, pick out the special companies as they usually do.

At last, slowly but surely, we are seeing the computer industry reorder, just as its own business has begun to improve. It is our feeling that the various new product families will provide the principal thrust to the improvement expected. As momentum accelerates, we expect to see various new products emerge from the data communications sector—in particular, products related to networking functions. In addition, substantial new workstations are likely as the 32-bit microprocessor is designed into product in 1987.

Finally, the military is going to continue to provide substantial end-market demand, particularly for custom and semi-custom product as electronics continue to penetrate, and where foreign competition has no entry. Our analysis has confirmed our intuitive beliefs about how and when to buy these stocks. If this is a false start, then a trade will have developed short term. In any case, we are convinced that the potential for a new industry upturn is very great. We cannot ignore all the signs of historical significance. At present, therefore, our worst case is for an order lull through the spring and summer and a firming up in late 1986. We feel that 1987 could be a very good year because of the 32-bit microprocessor and all the associated products that it will spawn.

Attractive Companies

Convergent Technologies—For the last two months, Jim Magid has been beating the drum on Convergent Technologies in the Spotlight Group. Of course, the company has accommodated us with a decent third quarter, a very thorough outline of management's long-term game plan at its November analyst meeting, a 40% equity interest in a vertical market value-added software company, a squeaky clean balance sheet and now, the coup de grace, a merger with 3Com—one of the best local area networking companies around. As Jim has pointed out, this combination of two creative and rapidly growing technology companies in the computer industry has come at an opportune moment given the current dynamics of networking and the opportunity for Convergent to develop a full value-added systems offering. Another obvious plus is the combination of two very good management groups. With new product coming for AT&T, a renewed and enlarged relationship with Burroughs and more OEM arrangements yet to be unveiled, the long-term outlook for Convergent has improved dramatically. Our 1986 earnings estimate is $0.75-$1.00 per share; 1987 prospects are tentatively being pegged at $1.50-$2.00. We like this stock.

Emulex Corp.—Emulex is beginning to look like a solid stock with new management members and a strong new product introduction program planned. As Fred Cohen has indicated, with the outlook brightening for Digital Equipment Corp., it is easy to get enthusiastic about Emulex—a supplier of DEC-compatible peripherals. Fiscal 1986 (June) earnings are expected to advance to $0.75 per share from $0.59, on sequentially better quarters. For fiscal 1987, Fred's preliminary projection is $1.25-$1.50.

The table on page 7 lists those stocks recommended in our November Technology Strategy Review that we continue to find attractive, plus Convergent Technologies and Emulex. Convergent has been in our Spotlight Group for some time.

32-Bit Microprocessors

Last month in *Tech Notes* we highlighted two technology subjects, local area networks and the 32-bit microprocessor, including reports written by our analysts on the local area network and its related personal computer impact. This month we are offering various articles written especially for *Tech Notes* on the 32-bit microprocessor.

Some clients have commented that the Intel 32-bit product, the 80386, is a non-event in the sense that it is right on the "trend line" of overall performance and functionality and offers no real advantage over the competition. We disagree with this characterization and are dedicating this month's *Tech Notes* to a discussion of the various issues.

L.F. ROTHSCHILD, UNTERBERG, TOWBIN
55 WATER STREET, NEW YORK, N.Y. 10041
TEL: (212) 412-1000 CABLE: PROBATICA

Needless to say, we are convinced of the importance of this generation of microprocessors and of Intel's participation. The emergence of the 32-bit micro has destroyed the distinction between computer companies. All levels of the industry are now directly competitive and can offer products of comparable functionality. This new set of dynamics pumps up the intensity of competition to a magnitude never before experienced.

For the computer manufacturers the question is not whether to develop a 32-bit based product line, but which 32-bit micro to choose. The discussions that follow should be helpful in analyzing the issues.

Stephen E. Yoken,
Director of the Technology Research Group
(212) 412-1822
December 16, 1985

Attractive Companies

Company	Price (12/16/85)	52-Week Range	Earnings Per Share 1984	1985E	1986P	P/E Ratios 1985E	1986P	Ind. Div.	Yield
AMP, Inc. (AMP)	37	38- 27	$1.87	$1.00	$1.50	37.0X	24.7X	$0.72	1.9%
Ashton-Tate (TATE) 1,3	19	19- 6	0.78	0.80A	1.65	23.8	11.5	None	None
Convergent Tech. (CVGT) 1,2,3	12	12- 5	Def.	0.25	0.75	48.0	16.0	None	None
Data General (DGN)	48	76- 31	2.60	0.91A	2.00	52.7	24.0	None	None
Digital Equipment (DEC)	133	127- 85	5.73	6.40A	8.00	20.8	16.6	None	None
Emulex (EMLX) 1,3	12	15- 5	0.88	0.59	0.75	20.3	16.0	None	None
IBM (IBM)	152	154-117	10.77	10.80	13.00	14.0	11.7	4.40	2.9
Intel (INTC) 1,3	31	32- 20	1.50	0.00	0.50	NM	62.0	None	None
Perkin-Elmer (PKN)	31	31- 20	1.30	1.77A	1.85	17.5	16.8	0.60	1.9
Siliconix (SILI) 1,3	23	24- 11	1.29	1.50	2.0	15.3	11.5	None	None

1 Within the past three years L.F. Rothschild, Unterberg, Towbin has acted as manager or co-manager for offerings of securities of this company.
2 A principal of L.F. Rothschild, Unterberg, Towbin is a director of this company.
3 L.F. Rothschild, Unterberg, Towbin maintains a trading market in the shares of this company.

32-Bit Microprocessor Market—The Race Just Got Tougher

Intel's Announcement

In mid-October, Intel formally introduced the iAPX 386—commonly known as the 80386 or simply the 386—at a gala affair in San Francisco. The 386 represents Intel's entry into the 32-bit microprocessor market, which at last count totaled 50 different offerings. Since its introduction, the 386 has received much attention from the electronics industry and the press. In fact, just about every major trade publication has featured the product in one format or another. Although many 32-bit alternatives have already been announced, we believe the 386 is the most significant one to date and will be the most successful. We think this product will have a major impact not only on the success of all other microprocessors, but on the future of the entire computer industry as well.

Why All The Attention

The market for 32-bit microprocessors have received a great amount of attention, and for good reason. Although still in its infancy, it is expected to grow in excess of 60% annually to total $200 million by 1990, up from $15-$20 million this year. If the market for peripheral circuits, software and development systems is added in, the total should approach $1-$2 billion by 1990.

Besides the large market potential, the 32-bit microprocessor area is important for another reason. The industry has witnessed the development of the 4-bit Intel 4004, the first microprocessor, the 8-bit microprocessors, and the 16-bit processors. The 32-bit processors represent the next and, to some degree, the final step in this evolution. Future processors will certainly increase in complexity and integration but, other than a few specialty products, there will not be a big demand for either 48- or 64-bit devices. Even general purposes mainframe and supermini computers use only 32 bits. 32 bits will become the standard for microprocessors—this technology represents the last round to enter the general purpose microprocessor market.

Not all applications currently need the power of a 32-bit microprocessor. Nevertheless, if a manufacturer offers a migration path from 8 to 16 to 32 bits, the high-end processor demonstrates the manufacturer's continued commitment to customers and enhances the value of its low-end offerings. Having a premier product enhances a manufacturer's full line of integrated circuits (ICs) as well as its microprocessors. Leading edge ICs are designed into the most advanced systems, giving the semiconductor manufacturer a view into the systems manufacturer's future product plans. Moreover, being at the forefront in microprocessors is equated with overall technological leadership because micros are the most complex integrated circuits made. This type of achievement strengthens the ties between the semiconductor supplier and its customers.

In addition to all the other benefits mentioned, in ten years the market for 32-bit microprocessors—including supporting software, peripheral circuits and development systems—is guaranteed to be very large, and estimated to approach $10 billion by the mid 1990s.

The 32-Bit Microprocessor Market

The market for 32-bit processors encompasses three broad areas—upward migration of current applications, the replacement of minicomputers in many applications and "new applications" that will grow out of the enormous price/performance increase the new processors offer. The market segments for these products are office automation and personal computers, scientific and engineering workstations, telecommunications and factory automation. New applications include expert systems, robotics, video graphics, transaction processing, signal processing, and speech recognition.

Although very high performance will be critical for most 32-bit applications, it is not the only important feature. For instance, in *office automation applications,* compatibility with existing software will be the critical factor for success. At the low end it is estimated that $5-$6 billion worth of software has been written for the IBM PC and its clones. The installed base of users will dictate the success in this market; therefore, microprocessor suppliers cannot ignore the importance of this large software library.

In the *scientific and engineering* world, different issues are important. This market will always pay a premium for increased performance; therefore, microprocessor speeds will be critical, but here, too, software compatibility is very important. The UNIX operating system has become the standard along with Ethernet networking; thus, to be successful, manufacturers will have to offer very high performance microprocessors that support these features.

Many people believe that, given the wide variety of applications for these processors, the market will be fragmented and that many different processors will be successful. Although this may prove to be true, with earlier generations the bulk of the market went to just a few manufacturers. In the 8-bit world, of the ten major manufacturers that introduced products, five control 85% of the market today. At 16 bits, four of the six processors now introduced control 85% of that market. We believe a similar consolidation will emerge at 32 bits with three or four key manufacturers controlling at least 80% of the market.

The market for 32-bit microprocessors has been slow to develop, and several explanations have been offered. Although many observers believe that the market will continue its slow expansion pace because there are so many alternatives, we disagree. Designing a 32-bit system is a very difficult task. In many ways, a system at this level of sophistication is as hard to design as a full 32-bit minicomputer. We believe the main reason the market has been slow to develop is because of the lack of tools available to support the design. Microprocessor suppliers must provide users with development systems and emulators, systems and support software, and peripheral and support circuits. With these tools, the new 32-bit systems will take a long time to develop; without the tools, the systems may never get completed.

It is important to remember that the relationship between a microprocessor supplier and a customer is very symbiotic: the success of one partner promotes the prosperity of the other. The level of sophistication of the supplier's support, development tools and the completeness of the supplier's product offerings are all critical to the success of the partnership.

The Success of the Intel 386

It was not until Intel introduced the 386 that all of the major suppliers presented their offerings. Intel has been supporting microprocessor designs for over ten years—longer than any other supplier—and with the 386 the company has demonstrated its continued commitment to being the leader in microprocessors. With each previous microprocessor generation, Intel has shown that it knows which support tools are necessary to be successful. We believe this experience will be reflected in its 32-bit products. At the 386 announcement, Intel introduced system software, high level language compliers, emulators, operating system support, development systems and board level products.

Most forecasts of 32-bit microprocessor usage estimate that office automation and personal computers will account for at least 80% of the unit volume by 1990. In this market, 32-bit microprocessors will be used to extend the power of microcomputers, replace low-end minicomputers and expand the functionality of both. At the low end the new computer systems will need to be able to tap the estimated $5 billion worth of software now written for the IBM PC, which operates on the Intel 8088, 8086 and the 286 microprocessors. Since the 386 is fully compatible with previous generations of Intel microprocessors, it is the only device that will allow easy access to the library of software. In the replacement of minicomputers, 32-bit microcomputers will need to offer very high performance and support very large amounts of memory. The 386 boasts a performance of 3-4 million instructions per second (MIPS) and is able to address 4 billion bytes of physical memory and 64 trillion bytes of virtual memory—a memory capacity as powerful as many of the very high performance superminicomputers. We have very little doubt that the 386 will become the standard 32-bit microprocessor used in office automation applica-

L.F. ROTHSCHILD, UNTERBERG, TOWBIN
55 WATER STREET, NEW YORK, N.Y. 10041
TEL: (212) 412-1000 CABLE: PROBATICA

tions, and given the estimated size of this market, success there will insure success in the whole 32-bit market.

Our forecasts for the engineering and scientific market indicate that these products will account for 5%-10% of the unit volume in 1990. Despite the fact that it will be dwarfed by the office automation market, the engineering market usually receives most of the attention primarily because engineers are usually the first to try out new products, and since the 32-bit market is still very young, engineering applications will be the first to adopt the new technology. The 386 will be particularly attractive to the end users in this market because of its unique ability to support two operating systems simultaneously, e.g. an engineer will be able to run a CAE application under UNIX while using a word processing or spreadsheet program, such as Lotus 1-2-3, under MS-DOS.

Intel has not historically been very visible in the engineering world. In the past, the company has not won any major engineering workstation designs, although its previous microprocessors have been extremely successful. We believe the 386 will be a winner in this market. The processor was built in high-performance CMOS and boasts excellent performance at 3-4 MIPS. The 386 also addresses a tremendous amount of memory—more than any other processor—which is important because engineering applications require vast amounts of memory support. The memory capacity of the 386 will support the future needs of this market for many years to come. Although many observers will try, it is very difficult to compare the performance of the 386 with other microprocessors. Nevertheless, when the smoke clears, we believe the evaluations will show that the 386 offers excellent performance and is as powerful as any other microprocessor available.

Although performance is considered the only truly important requirement in the engineering world, other factors are becoming increasingly more important. The IBM PC/AT has become the low-end entry-level computer of choice throughout the engineering world because of its power and aggressive pricing. This has allowed engineers access to the $5 billion worth of software written for the IBM PC. Following on this trend, the Intel 386 was designed to support both UNIX and MS-DOS applications at the same time. Therefore, computer systems built around this microprocessor will be the only ones to offer the performance of a superminicomputer while accessing the IBM PC's extensive software base.

Given the performance characteristics and the very advanced technical features of the 386, combined with Intel's vast experience in microprocessor development support and the pervasiveness of previous Intel microprocessors, we believe that Intel will be very successful in the 32-bit market and that the 386 and follow-on processors will become the 32-bit standards.

Paul Johnson (212) 412-1568

Impact of the Intel 80386 on the CAD/CAM/CAE Industry

Overview

The CAD/CAM/CAE industry is quickly moving from a minicomputer-based system environment to one based on 32-bit microprocessors. Many of the companies offer turnkey systems based on Digital Equipment's VAX 32-bit minicomputers, and other computer manufacturers are now in the process of choosing microprocessor-based systems either to be developed in-house or brought from third party vendors. Many believe that the 32-bit microprocessor race has already been won by those firms with products announced in the last two years. The problem is that shipments of many of those products have not yet begun in quantity. On the horizon there is a chip set that is beginning to catch hold of the engineering and scientific marketplace for computers—the Intel 32-bit microprocessor, the 80386.

Since previous articles in this issue have described the intricacies of the chip, it is not necessary to give another detailed description of its capabilities—in this piece we outline some of the possible impacts the product will have on the CAD/CAM/ CAE market. Suffice it to say that we anticipate a low-cost, readily available computer priced at $8,000-$12,000—$2,000-$4000 more than an IBM PC/AT fully loaded with acceptable graphics. The offering would most likely come from IBM, which, in our opinion, owns the low end of the engineering computer market. The IBM PC, which for the sake of simplicity we will call the PC 386, will be mass-produced and will quickly approximate the shipment volumes of the PC/AT.

One of the obvious impacts of the Intel 80386 is to bring many of the companies currently offering software on the Intel 16-bit 80286, 8086 and 8088 microprocessors to a more powerful microcomputer processor. The software that currently runs on the Intel family of microprocessors (including Flight Simulator) will also run on the 80386—at roughly twice the speed. The real boost in power is achieved by recompiling the software to take advantage of the larger bus on the chip—this will increase its speed more like four times. Today a company can obtain from Intel a complier for the C and PL/M high level languages; a Fortran complier will be available in 1986 and an Ada complier in 1987.

Software written in the C and PL/M languages (most of the software written for a UNIX operating system and for flexibility is written in C) can today begin being formatted for the 80386 on hardware ranging from the Digital VAX to the IBM PC. Intel has also provided software development tools to make it easier to write new software for the processor—Intel is the largest supplier of software development tools to the industry. The operating systems supported by the 80386 are MS-DOS, UNIX and iRMX (Intel's real-time operating system). Both UNIX and MS-DOS are the *de facto* operating systems for the CAE marketplace with UNIX soon to become the standard for all offerings from the CAE vendors. The 80386 runs UNIX and MS-DOS concurrently so a user could execute a processing-intensive program and simultaneously begin working on another project. Today a lot of time is wasted as users wait for programs to be executed, particularly in the low-end segment of the market.

Effect on Specific Companies

A low-cost, extremely powerful computer that runs today's software is exactly what the customer wants. We believe the PC 386 is the computer that engineering managers would like to see on every engineer's desk. The PC 386 could possibly be ready for volume shipment in the fourth quarter of 1986, but is more likely to become available in the first quarter of 1987. With this in mind, a PC 386 and microVAX (II and III) arrangement would be just what the the customer is looking for—a low-cost front end and a low-cost distributed node. Some companies have an advantage, particularly those that already have ported their software to the IBM PC. This advantage, however, does not come to those companies that have tailored their software to co-processor boards to enhance speed today instead of waiting for the power of the PC 386 tomorrow (more like a year from tomorrow). Tektronix/CAE Systems and Valid Logic are two of the largest CAE vendors to have done so. Daisy Systems has added National Semiconductor's 16-bit microprocessor, the 32016, to its high-end offerings, but the software can also run on the other processors such as the 80386. So *Daisy* can now have a high end that encompasses parallel processing architecture with two 80386 microprocessors—one way to maintain high margins. *Mentor Graphics* is preparing an IBM PC/AT offering for the first quarter of 1986, although it will not run the full gamut of software that Mentor's Apollo-based offerings do. The company is also transferring its software out of the Pascal language and into C to increase the flexibility to different hardware systems and, more than likely, to prepare for the PC 386.

An interesting added aspect of the PC 386 is that it brings the current low-cost offerings that run just on personal computers into the fold of full fledged CAE offerings. Companies such as Futurenet, Viewlogic, Case, P-CAD, etc., all are now running on the same hardware as the full line CAE vendors and could put pricing pressure on the CAE vendors that have only partial products offerings. And the low-cost CAE vendors could cause pricing problems for the companies with more overhead, such as the full line CAE vendors.

In the CAD/CAM sector of the market there could be a major dislocation of the existing minicomputer-based CAD offerings if the companies have difficulty porting their software out of the minicomputer environment down to the microprocessor level. The low-cost CAD software offerings of today will be available on hardware that is not far from the processing speeds of the minicomputers. *Intergraph, Computervision, IBM, GE/Calma, Schlumberger/Applicon,* et.al., are all based on minicomputer-based offerings and could well have problems offering integrated software at the low end that does not pirate the high end. IBM, Applicon and Computervision could have major problems porting their mainstream software to the microprocessor level. These companies could actually lose market share to the low-cost CAD offerings such as *Autodesk's* AutoCAD or *T&W Systems'* Versacad. Computervision currently has a software package running on the IBM PC, but it does not interface with the existing software in the company's mainstream product, the CDS 4000. Computervision is also making the mistake (in our opinion) of porting its software to the 68020 environment and will probably have trouble bringing that same software up on a PC 386. Intergraph could, in our opinion, replace the National Semiconductor 32032 in the Interpro 32 with an 80386 and not be set back too much by software changes, since Intergraph has chosen the AT&T System V version of UNIX which is also the operating system on the 80386. We do not expect Intergraph to announce this change for another six months. The incremental revenues will be nil in 1986 but could be $50 million, or 3,000 units, in 1987.

The bottom line is that **Autodesk, PDA Engineering, Daisy Systems, and Mentor Graphics will all benefit from the Intel 80386 by expanding their existing markets for the low-cost systems.** The pricing pressure mentioned earlier will probably be insignificant since customers at this point in the industry's development are becoming concerned with the security of future systems; hence, the data created will be compatible with future offerings from a vendor. Intergraph is currently working on porting its software down to the microprocessor level—the company's first product will be annonced in the second quarter of 1986 and shipments will begin in the third. Intergraph will adopt a rather cautious approach to porting its software since the company takes great care not to upset its customers. Computervision is also porting its software down to the microprocessor level, but we envision that the customer will be less impressed by the Sun Microsystems licensed workstations once the PC 386 is announced. On a final note, the forward-looking companies will be quick to adopt the PC 386 to some degree in their product offerings over the next 18 months.

Peter D. Schleider (212) 412-1572

L.F. ROTHSCHILD, UNTERBERG, TOWBIN
55 WATER STREET, NEW YORK, N.Y. 10041
TEL: (212) 412-1000 CABLE: PROBATICA

Convergent Technologies and the Intel 80386

Background

The Personal Computer

The personal computer, as it entered business and industry, was defined by the 1977 Apple II, consisting of a keyboard, monitor, floppy disc drive, and software for spreadsheet analysis, word processing, and data base management. The use of a modem tied machines to the outside world of data bases and electronic mail, but the computer functioned more as a dumb terminal in these configurations. Used by an individual, with files and data entered by hand, machines started showing up on office desks, laboratory benches, and the factory floor. The personal and individual nature of the interaction with the computer and its low price, compared to other business and industrial equipment, saw personal computers percolate into the business world before the computer hierarchy knew what was happening.

The IBM Did Not Change The Basic Personal Computer

The introduction of the IBM PC did nothing to change the basic configuration of the personal computer; it still consists of the same elements. The most powerful PC's, such as the IBM/AT, are essentially similiar to the early Apple, and most of the software packages being used are only but zippier versions of the early favorites. The power has grown, so that memory of 64K bytes on the first IBM PCs compares with 640K bytes common today, the limit available for use under the MS-DOS operating system. The limit of 640K bytes is beginning to prove inadequate for sophisticated users today—i.e., a tenfold increase in memory in less than a decade is not enough. Even so, the essential structure of PCs and their use today is more similiar to the 1977 Apple II than is commonly recognized.

From Personal to Organizational Computer

The success of the IBM PC changed the attitude of business toward the personal computer. Viewing them as a threat to traditional computer and information hierarchies at first, the MIS departments took over the purchase of personal computers and software, control of the data, and access to the machine in large corporations and IBM succeeded in Apple's footsteps. The personal computer became the organizational computer in business and industry.

Organizational Computers Were Islands

Although organizational PCs were supported by computer departments, they still function as isolated, independent workstations for the most part. These islands are just now being connected by bridges, and the new generation of microprocessors is essential for PCs to be used as more than dumb terminals in such applications. Networks enable these organizational computers to share costly peripherals, communicate with limited corporate databases, and function as terminals on corporations' larger data processing networks. Personal computers, connected horizontally to each other and vertically to mainframes for data and communications, are just now being recognized as replacements for the tens of millions of dumb terminals in the installed base.

The 80386 Bridges the Islands

As these workstations gain access to corporate data from mainframes and are used for analysis and communications, the next generation of computer usage is unfolding. The 1960s were characterized by batch processing mainframe computers. The 1970s were the era of the minicomputer and distributed computing on mainframes. The next era should witness the integration of these two into broad interconnections of networked intelligent workstations, bringing mainframe power to the desktop, and connecting horizontally and vertically into hierarchies. The segmented memory of the Intel 80386 and its ability to run UNIX multiprocessing and multitasking at the same time as MS-DOS applications open the door for this structural change from isolated personal or organizational computers to an interconnected network providing more structured information, current and fresh, than any single user could provide for himself.

Cost/Performance Advances of Thirty Times in a Decade

The evolution of software and functionality of computers has continued without pause. The rate of cost/performance improvement in the desktop computer industry has been between 25% and 40% annually for the past decade of the personal computer, and that rate should continue. That means the performance for the same cost increases ten to thirty times over a decade.

Memory, Logic, and Peripherals All Join in Progress

The desktop computer of 1987 should have ten to thirty times the capability of the original Apple II at the same cost, or perhaps as much as one hundred times the performance at a higher cost, when implemented in an engineering workstation. This is the result of progress in semiconductors and peripherals. Recording densities of floppy and hard discs, soon to be followed by the introduction of high capacity optical storage media, have paralled this trend. By 1987, for example, technology employed in today's digital compact audio disc, or CD, can result in a device capable of 500 megabytes of storage for $500, about the same cost as the Apple II's 140K bytes in floppy disc drives went for a decade earlier, or a 125-fold increase, overlooking the important fact that you can't record on the digital disc yet. With the high resolution monitor screens the next generation computers will have, the digital CD may rival printing for the distribution of some forms of information. The examples of semiconductor memory improvements are well known, but similar gains in logic have not been fully recognized.

Microprocessors Determine the Computer Capabilities

The real performance gains come from the changes in the capability of microprocessors and coprocessor chips that form the brains of these machines, the operating systems that turn the machines into responsive, intelligent, interactive creations, and the networks that can link these machines together.

Operating Systems Converge in the 80386

The evolution of the Intel 8086 architecture as implemented in MS-DOS on the IBM computers has captured the attention of most observers, but Intel's proprietary iRMX real time operating system, for laboratory, factory, and industrial applications, has paralleled the success of MS-DOS in a different arena. The 80386 runs these operating systems, as well as UNIX, all at the same time and with the ability to share data among operating systems and applications. Its compatibility with the past while opening new vistas for performance is the "revolutionary" aspect of the 80386. The "more than evolutionary" capability of the Intel 80386 is probably being underestimated because the full potential of its predecessor, the 80286, has still not been fully unleashed in desktop computers, so the question as to why we need more than we have today is uppermost in critics' minds. The answer of course is that with 40% annual cost performance gains, the entire character of the personal computer, as it is used in business, is changing in a radical way.

The 80386 Is Revolutionary and Evolutionary at the Same Time

The Intel 80386 is revolutionary in that it addresses an almost infinite amount of memory in comparison with today's processors, opening true multiprocessing, multitasking, high resolution display, with the ability to interface with much more information than any one person could enter in a lifetime. Software advances that utilize the total capability of this device are probably more than five years away, but because the 80386 is compatible with existing 8086 software, it will be implemented to enhance existing software and applications, and its true revolutionary nature will show up gradually in networking and in putting more complex software into simpler seeming applications.

Multitasking Adds Speed to Today's Application

An example of how the 80386 is compatible with but better than today's processors can be seen in the use of multitasking. Multitasking is being able to do more than one thing at a time. Obvious examples are communications and a spreadsheet simultaneously, receiving data and sending it. Current windowing software is inadequate, present color displays are of too low a resolution to hold enough information on the screen at one time, present operating systems do not allow true multitasking to work, and the 640K memory limit on MS-DOS means that not enough memory is available to be able to do anything very large. The 80386 can solve all of those problems. But the first feature will be speed. Running several times faster than the 80286, the 80386 will run all applications a little faster, but multiprocessing will allow it to run some applications at an almost infinitely faster rate to the user. Consider a spreadsheet that must wait for recalculation, which takes a long time with a big spreadsheet. With the 80386 and multitasking, the sheet can be run in the background as two or more simultaneous tasks. So, data entry in the foreground goes on without waiting for calculations, communications, or printing from a background of the identical spread sheet. The same application works with complex word processing documents, which would make a big difference for a law firm, for example.

The Intel 80386 and Its Capabilities

Memory—The 80386 addresses 4 gigabytes of memory, or 4 billion bytes of memory, compared with 1 megabyte possible on the 8088 and 6,700 times the 640K bytes of memory available under MS-DOS. With virtual memory, the 80386 addresses 64 tetrabytes, or 70 billion bytes of memory. The magnitude of this is impossible to visualize easily. It is 100 million times the memory available under MS-DOS. If purchased in 256K DRAMS at $2.00 each it would take 2 billion devices worth $4 billion. It can easily be seen that this amount of memory access, equal to more than 256,000 bytes for everyone in the United States, exceeds the capability of existing software and data bases. But what is not so apparent is that in structures with mainframe computers at the top and 386-based PCs as terminals, the information locked inside today's mainframes, properly configured, can become available to many more users.

Software—The 80386 is capable of running many operating systems simultaneously. It can run 386 software, being a mainframe in power itself, 286 applications, and conventional PC-DOS software at the same time. With the Virtual Monitor 8086, the 386 runs multiple 8086 applications as if each had its own 8086. This frees future software from the confines of PC-DOS while maintaining compatibility with existing machines. For example, it is possible for Convergent Technologies to implement its

L.F. ROTHSCHILD, UNTERBERG, TOWBIN
55 WATER STREET, NEW YORK, N.Y. 10041
TEL: (212) 412-1000 CABLE: PROBATICA

MegaFrame and other Frame computers in the 80386, and retain AT&T UNIX system V compatibility with the UNIX operations presently run on series 68000 microprocessors, while adding Convergent's 8086-based CTOS as well as PC and MS-DOS applications.

Convergent Among The First And Best With The 80386

Convergent Technologies should be among the first and best to implement the 80386 in workstations and in minicomputers. Convergent's existing workstation line, the NGen, is just benefitting from the introduction of the 80286 module, which Burroughs has introduced as the B 28. A review of the power of the 286 module shows that Convergent has much more performance on its NGen systems than is commonly available on IBM PC/ATs using the same 80286 microprocessor.

The modularity of the NGen system allows for upgrades without obsoleting existing equipment or software. Fully compatible with the 80186 CPUs, the 80286 adds power and features. On Convergent's 286 module, memory management allows accessing up to 4 megabytes of real memory, at any given time, out of an address space that is much larger under virtual memory. Virtual memory management allows up to 16 megabytes of virtual memory. Convergent will achieve this by using the disc as cache memory. CTOS software allows true concurrency up to one megabyte, and new software will allow concurrency up to 4 megyabytes soon, well in advance of other operating systems for the 80286.

Concurrency means running several tasks at the same time, a special advantage of CTOS. Convergent is implementing the 80286 to access much larger segments of memory than can be used on operating systems such as MS-DOS, which so far are limited to only 640K bytes of direct access memory, and Convergent will be able to access 4 megabytes on the 286 NGen module that Burroughs is buying. This means that Convergent's 80286 module, shipping now, can access as many applications (called contexts) as desired at the same time. Networked together, these workstations can share costly printers and high capacity disc files, and be integrated into mainframes controlled communications systems using the MegaFrame minicomputer. In an organizational context, the system is more powerful than isolated, non-networked, PC's.

A principal application of the 80286 is to support multiple programs serving multiple applications through a server, the master workstation. In addition to supporting a traditional workstation such as a word processor or spreadsheet user, the station can control communications or a network, printer and other applications at the same time. This is because the 80286 has such a large memory address space. Looking to the 80386 in the same application, multiple mainframe applications can be shared over the network, making the combination of 80386 workstations and mainframes much more powerful than the same number of machines would be in isolation. Thus, with a more powerful master, the sharing of high-priced high quality peripherals is spread over a larger base. Today's Convergent machines are high-priced relative to PC or AT competitors running PC tasks in stand alone applications, but when the values of the integrated systems are viewed, the costs are comparable. **However, the introduction of 80286 and 80386 NGen modules and Frames should lower the workstation cost of Convergent equipment, per user, giving the company a performance and cost advantage in the organizational market. Including 3Com networking makes this a very powerful system, and we expect to see it adopted soon by a major new OEM customer, itself an impressive computer company.**

With the 80386, Convergent's task of making the machine PC-compatible is much easier, so we expect to see Convergent implement the 80386 throughout its product line, from NGen modules through Mega Frame. We expect future NGen 80386 processors to be compatible with previous Ngens running CTOS, and PC compatible as well. That means that today's users of Convergent's NGen machines can take advantage of the PC-DOS software within a year, if they desire. If Intel's timetable is met, we would expect to see Convergent introduce a 386 module by the fall of 1986. Because Convergent has been a leader in UNIX applications, the company understands and has software which uses multitasking and multiprocessing applications for the business world. We expect these to be key features as new major OEMs choose Convergent systems in 1986, but in all probability, Convergent's ability to appeal to another major computer company may well be the integration of PC and other applications on the 386, in the context of full compatibility of strong present product line.

James I. Magid (212) 412-1574

DOMESTIC SALES OFFICES

ALABAMA

Intel Corp.
5015 Bradford Drive
Suite 2
Huntsville 35805
Tel: (205) 830-4010

ARIZONA

Intel Corp.
11225 N. 28th Drive
Suite 214D
Phoenix 85029
Tel: (602) 869-4980

Intel Corp.
1161 N. El Dorado Place
Suite 301
Tucson 85715
Tel: (602) 299-6815

CALIFORNIA

Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500

Intel Corp.
2250 E. Imperial Highway
Suite 218
El Segundo 90245
Tel: (213) 640-6040

Intel Corp.
1510 Arden Way, Suite 101
Sacramento 95815
Tel: (916) 920-8096

Intel Corp.
4350 Executive Drive
Suite 105
San Diego 92121
(619) 452-5880

Intel Corp.*
2000 East 4th Street
Suite 100
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114

Intel Corp.*
San Thomas 4
2700 San Thomas Expressway
Santa Clara, CA 95051
Tel: (408) 986-8086
910-338-0255

COLORADO

Intel Corp.
3300 Mitchell Lane, Suite 210
Boulder 80301
Tel: (303) 442-8088

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (303) 594-6622

Intel Corp.*
650 S. Cherry Street
Suite 915
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

Intel Corp.
26 Mill Plain Road
Danbury 06810
Tel: (203) 748-3130
TWX: 710-456-1199

EMC Corp.
222 Summer Street
Stamford 06901
Tel: (203) 327-2934

FLORIDA

Intel Corp.
242 N. Westmonte Drive
Suite 105
Altamonte Springs 32714
Tel: (305) 869-5588

Intel Corp.
6363 N.W. 6th Way, Suite 100
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

FLORIDA (Cont'd)

Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33702
Tel: (813) 577-2413

GEORGIA

Intel Corp.
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel: (404) 449-0541

ILLINOIS

Intel Corp.*
300 N. Martingale Road, Suite 400
Schaumburg 60172
Tel: (312) 310-8031

INDIANA

Intel Corp.
8777 Purdue Road
Suite 125
Indianapolis 46268
Tel: (317) 875-0623

IOWA

Intel Corp.
St. Andrews Building
1930 St. Andrews Drive N.E.
Cedar Rapids 52402
Tel: (319) 393-5510

KANSAS

Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 345-2727

MARYLAND

Intel Corp.*
7321 Parkway Drive South
Suite C
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

Intel Corp.
7833 Walker Drive
Greenbelt 20770
Tel: (301) 441-1020

MASSACHUSETTS

Intel Corp.*
Westford Corp. Center
3 Carlisle Road
Westford 01886
Tel: (617) 692-3222
TWX: 710-343-6333

MICHIGAN

Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel: (313) 851-8096

MINNESOTA

Intel Corp.
3500 W. 80th Street
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

Intel Corp.*
Parkway 109 Office Center
328 Newman Springs Road
Red Bank 07701
Tel: (201) 747-2233

NEW JERSEY (Cont'd)

Intel Corp.
75 Livingston Avenue
First Floor
Roseland 07068
Tel: (201) 740-0111

NEW MEXICO

Intel Corp.
8500 Menual Boulevard N.E.
Suite B 295
Albuquerque 87112
Tel: (505) 292-8086

NEW YORK

Intel Corp.*
300 Vanderbilt Motor Parkway
Hauppauge 11788
Tel: (516) 231-3300
TWX: 510-227-6236

Intel Corp.
Suite 2B Hollowbrook Park
15 Myers Corners Road
Wappinger Falls 12590
Tel: (914) 297-6161
TWX: 510-248-0060

Intel Corp.*
211 White Spruce Boulevard
Rochester 14623
Tel: (716) 424-1050
TWX: 510-253-7391

NORTH CAROLINA

Intel Corp.
5700 Executive Center Drive
Suite 213
Charlotte 28212
Tel: (704) 568-8966

Intel Corp.
2700 Wycliff Road
Suite 102
Raleigh 27607
Tel: (919) 781-8022

OHIO

Intel Corp.*
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528

Intel Corp.*
Chagrin-Brainard Bldg., No 300
28001 Chagrin Boulevard
Cleveland 44122
Tel: (216) 464-2736
TWX: 810-427-9298

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73116
Tel: (405) 848-8086

OREGON

Intel Corp.
10700 S.W. Beaverton
Hillsdale Highway
Suite 22
Beaverton 97005
Tel: (503) 641-8086
TWX: 910-467-8741

PENNSYLVANIA

Intel Corp.
1513 Cedar Cliff Drive
Camphill 17011
Tel: (717) 737-5035

Intel Corp.*
455 Pennsylvania Avenue
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077

Intel Corp.*
400 Penn Center Boulevard
Suite 610
Pittsburgh 15235
Tel: (412) 823-4970

PUERTO RICO

Intel Microprocessor Corp.
South Industrial Park
Las Piedras 00671
Tel: (809) 733-3030

TEXAS

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628

Intel Corp.*
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-8087
TWX: 910-860-5617

Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490

Industrial Digital Systems Corp.
5925 Sovereign
Suite 101
Houston 77036
Tel: (713)988-9421

UTAH

Intel Corp.
5201 Green Street
Suite 290
Murray 84123
Tel: (801) 263-8051

VIRGINIA

Intel Corp.
1603 Santa Rosa Road
Suite 109
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

Intel Corp.
110 110th Avenue N.E.
Suite 510
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086

WISCONSIN

Intel Corp.
450 N. Sunnyslope Road
Suite 130
Chancellory Park I
Brookfield 53005
Tel: (414) 784-8087

CANADA

BRITISH COLUMBIA

Intel Semiconductor of Canada, Ltd.
301-2245 W. Broadway
Vancouver V6K 2E4
Tel: (604) 738-6522

ONTARIO

Intel Semiconductor of Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
TELEX: 053-4115

Intel Semiconductor of Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
TELEX: 06983574

QUEBEC

Intel Semiconductor of Canada, Ltd.
620 St. Jean Blvd.
Pointe Claire H9R 3K3
Tel: (514) 694-9130
TWX: 514-694-9134

*Field Application Location

DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35805
Tel: (205) 837-6955

†Hamilton/Avnet Electronics
4812 Commercial Drive N.W.
Huntsville 35805
Tel: (205) 837-7210
TWX: 810-726-2162

Pioneer/Technologies Group Inc.
4825 University Square
Huntsville 35805
Tel: (205) 837-9300
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5100
TWX: 910-950-0077

Kierulff Electronics
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
TWX: 910-951-1550

Wyle Distribution Group
17855 N. Black Canyon Highway
Phoenix 85023
Tel: (602) 866-2888

CALIFORNIA

Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (818) 701-7500
TWX: 910-493-2086

Arrow Electronics, Inc.
1502 Crocker Avenue
Hayward 94544
Tel: (408) 487-4600

Arrow Electronics
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
TLX: 888064

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
TWX: 910-595-2860

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6051
TWX: 910-595-1928

Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel: (619) 571-7500
TWX: 910-595-2638

†Hamilton/Avnet Electronics
20501 Plummer Street
Chatsworth 91311
Tel: (818) 700-6271
TWX: 910-494-2207

†Hamilton/Avnet Electronics
4103 Northgate Boulevard
Sacramento 95834
Tel: (916) 920-3150

Hamilton/Avnet Electronics
3002 G Street
Ontario 91311
Tel: (714) 989-9411

Hamilton/Avnet Electronics
19515 So. Vermont Avenue
Torrance 90502
Tel: (213) 615-3909
TWX: 910-349-6263

Hamilton Electro Sales
9650 De Soto Avenue
Chatsworth 91311
Tel: (818) 700-6500

†Hamilton Electro Sales
10950 W. Washington Boulevard
Culver City 90230
Tel: (213) 558-2458
TWX: 910-340-6364

Hamilton Electro Sales
1361 B West 190th Street
Gardena 90248
Tel: (213) 558-2131

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
TWX: 910-595-2638

Kierulff Electronics
10824 Hope Street
Cypress 90430
Tel: (714) 220-6300

Kierulff Electronics, Inc.
1180 Murphy Avenue
San Jose 95131
Tel: (408) 971-2600
TWX: 910-379-6430

Kierulff Electronics, Inc.
14101 Franklin Avenue
Tustin 92680
Tel: (714) 731-5711
TWX: 910-595-2599

†Kierulff Electronics, Inc.
5650 Jillson Street
Commerce 90040
Tel: (213) 725-0325
TWX: 910-580-3666

Wyle Distribution Group
26560 Agoura Street
Calabasas 91302
Tel: (818) 880-9000
TWX: 818-372-0232

CALIFORNIA (Cont'd)

†Wyle Distribution Group
124 Maryland Street
El Segundo 90245
Tel: (213) 322-8100
TWX: 910-348-7140 or 7111

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 843-9953
TWX: 910-595-1572

†Wyle Distribution Group
11151 Sun Center Drive
Rancho Cordova 95670
Tel: (916) 638-5282

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 910-338-0296

Wyle Military
18910 Teller Avenue
Irvine 92750
Tel: (714) 851-9958
TWX: 310-371-9127

Wyle Systems
7382 Lampson Avenue
Garden Grove 92641
Tel: (714) 851-9953
TWX: 910-595-2642

COLORADO

Arrow Electronics, Inc.
1390 S. Potomac Street
Suite 136
Aurora 80012
Tel: (303) 696-1111

†Hamilton/Avnet Electronics
8765 E. Orchard Road
Suite 708
Englewood 80111
Tel: (303) 740-1017
TWX: 910-935-0787

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

†Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

†Pioneer Northeast Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373

FLORIDA

†Arrow Electronics, Inc.
350 Fairway Drive
Deerfield Beach 33441
Tel: (305) 429-8200
TWX: 510-955-9456

†Arrow Electronics, Inc.
1001 N.W. 62nd Street
Suite 108
Ft. Lauderdale 33309
Tel: (305) 776-7790
TWX: 510-955-9456

†Arrow Electronics, Inc.
50 Woodlake Drive W., Bldg. B
Palm Bay 32905
Tel: (305) 725-1480
TWX: 510-959-6337

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
TWX: 510-956-3097

†Hamilton/Avnet Electronics
3197 Tech. Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

Hamilton/Avnet Electronics
6947 University Boulevard
Winterpark 32792
Tel: (305) 628-3888
TWX: 810-853-0322

†Pioneer Electronics
221 N. Lake Boulevard
Suite 412
Alta Monte Springs 32701
Tel: (305) 834-9090
TWX: 810-853-0284

†Pioneer Electronics
674 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
TWX: 510-955-9653

GEORGIA

†Arrow Electronics, Inc.
3155 Northwoods Parkway, Suite A
Norcross 30071
Tel: (404) 449-8252
TWX: 810-766-0439

Hamilton/Avnet Electronics
5825 D. Peachtree Corners
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer Electronics
5835B Peachtree Corners E.
Norcross 30092
Tel: (404) 448-1711
TWX: 810-766-4515

ILLINOIS

†Arrow Electronics, Inc.
2000 E. Alonquin Street
Schaumberg 60195
Tel: (312) 397-3440
TWX: 910-291-3544

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (312) 860-7780
TWX: 910-227-0060

MTI Systems Sales
1100 West Thorndale
Itasca 60143
Tel: (312) 773-2300

†Pioneer Electronics
1551 Carmen Drive
Elk Grove Village 60007
Tel: (312) 437-9680
TWX: 910-222-1834

INDIANA

†Arrow Electronics, Inc.
2495 Directors Row, Suite H
Indianapolis 46241
Tel: (317) 243-9353
TWX: 810-341-3119

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
TWX: 810-260-3966

†Pioneer Electronics
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

KANSAS

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel: (913) 888-8900
TWX: 910-743-0005

KENTUCKY

Hamilton/Avnet Electronics
1051 D. Newton Park
Lexington 40511

MARYLAND

Arrow Electronics, Inc.
8300 Guilford Road #H
Rivers Center
Columbia 21046
Tel: (301) 995-0003
TWX: 710-236-9005

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
TWX: 710-862-1861

†Mesa Technology Corporation
16021 Industrial Drive
Gaithersburg 20877
Tel: (301) 948-4350
TWX: 710-828-9702

†Pioneer Electronics
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 948-0710
TWX: 710-828-0545

MASSACHUSETTS

†Arrow Electronics, Inc.
1 Arrow Drive
Woburn 01801
Tel: (617) 933-8130
TWX: 710-393-6770

†Hamilton/Avnet Electronics
100 Centennial Drive
Peabody 01960
Tel: (617) 532-3701
TWX: 710-393-0382

MTI Systems Sales
13 Fortune Drive
Billerica 01821

Pioneer Northeast Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 863-1200
TWX: 710-326-6617

MICHIGAN

Arrow Electronics, Inc.
755 Phoenix Drive
Ann Arbor 48104
Tel: (313) 971-8220
TWX: 810-223-6020

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-242-8775

Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
TWX: 810-273-6921

†Pioneer Electronics
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

Hamilton/Avnet Electronics
10300 Bren Road East
Minnetonka 55343
Tel: (612) 932-0600
TWX: (910) 576-2720

†Pioneer Electronics
10203 Bren Road East
Minnetonka 55343
Tel: (612) 935-5444
TWX: 910-576-2738

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
TWX: 910-764-0882

†Hamilton/Avnet Electronics
13743 Shoreline Court
Earth City 63045
Tel: (314) 344-1200
TWX: 910-762-0684

NEW HAMPSHIRE

†Arrow Electronics, Inc.
3 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
TWX: 710-220-1684

Hamilton/Avnet Electronics
444 E. Industrial Drive
Manchester 03104
Tel: (603) 624-9400

NEW JERSEY

†Arrow Electronics, Inc.
6000 Lincoln East
Marlton 08053
Tel: (609) 596-8000
TWX: 710-897-0829

†Arrow Electronics, Inc.
2 Industrial Road
Fairfield 07006
Tel: (201) 575-5300
TWX: 710-998-2206

†Hamilton/Avnet Electronics
1 Keystone Avenue
Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
TWX: 710-940-0262

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
TWX: 710-734-4388

†Pioneer Northeast Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
TWX: 710-734-4382

†MTI Systems Sales
383 Route 46 W
Fairfield 07006
Tel: (201) 227-5552

NEW MEXICO

Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
TWX: 910-989-1151

Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
TWX: 910-989-0614

NEW YORK

†Arrow Electronics, Inc.
25 Hub Drive
Melville 11747
Tel: (516) 694-6800
TWX: 510-224-6126

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
TWX: 510-224-6166

†MTI Systems Sales
38 Harbor Park Drive
P.O. Box 271
Port Washington 11050
Tel: (516) 621-6200
TWX: 510-223-0846

†Pioneer Northeast Electronics
1806 Vestal Parkway East
Vestal 13850
Tel: (607) 748-8211
TWX: 510-252-0893

†Pioneer Northeast Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
TWX: 510-221-2184

Pioneer Northeast Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
TWX: 510-253-7001

NORTH CAROLINA

Arrow Electronics, Inc.
5240 Greendairy Road
Raleigh 27604
Tel: (919) 876-3132
TWX: 510-928-1856

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819
TWX: 510-928-1836

Pioneer Electronics
9801 A-Southern Pine Boulevard
Charlotte 28210
Tel: (704) 524-8188
TWX: 810-621-0366

OHIO

Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
TWX: 810-459-1611

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
TWX: 810-427-9409

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 433-0610
TWX: 810-450-2531

OHIO (Cont'd)

†Hamilton/Avnet Electronics
4588 Emery Industrial Parkway
Warrensville Heights 44128
Tel: (216) 831-3500
TWX: 810-427-9452

†Pioneer Electronics
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
TWX: 810-459-1622

†Pioneer Electronics
4800 E. 131st Street
Cleveland 44105
Tel (216) 587-3600
TWX: 810-422-2211

OKLAHOMA

Arrow Electronics, Inc.
4719 S. Memorial Drive
Tulsa 74145
Tel: (918) 665-7700

OREGON

†Almac Electronics Corporation
1885 N.W. 169th Place
Beaverton 97006
Tel: (503) 629-8090
TWX: 910-467-8746

Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
TWX: 910-455-8179

Wyle Distribution Group
5250 N.E. Elam Young Parkway
Suite 600
Hillsboro 97124
Tel: (503) 640-6000
TWX: 910-460-2203

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000

Pioneer Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122

†Pioneer Electronics
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778

TEXAS

†Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
TWX: 910-860-5377

†Arrow Electronics, Inc.
10899 Kinghurst
Suite 100
Houston 77099
Tel: (713) 530-4700
TWX: 910-880-4439

Arrow Electronics, Inc.
10125 Metropolitan
Austin 78758
Tel: (512) 835-4180
TWX: 910-874-1348

†Hamilton/Avnet Electronics
1807 W. Braker Lane
Austin 78758
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75062
Tel: (214) 659-4100
TWX: 910-860-5929

†Hamilton/Avnet Electronics
4850 Wright Road #190
Houston 77477
Tel: (713) 780-1771
TWX: 910-881-5523

†Pioneer Electronics
9901 Burnet Road
Austin 78758
Tel: (512) 835-4000
TWX: 910-874-1323

†Arrow Electronics, Inc.
3375 Brighton-Henrietta Townline Road
Rochester 14623
Tel: (716) 427-0300
TWX: 510-253-4766

Arrow Electronics, Inc.
7705 Maltage Drive
Liverpool 13088
Tel: (315) 652-1000
TWX: 710-545-0230

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
TWX: 510-227-6623

Hamilton/Avnet Electronics
333 Metro Park
Rochester 14623
Tel: (716) 475-9130
TWX: 510-253-5470

Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-2641
TWX: 710-541-1560

Pioneer Electronics
13710 Omega Road
Dallas 75234
Tel: (214) 386-7300
TWX: 910-850-5563

Pioneer Electronics
5853 Point West Drive
Houston 77036
Tel: (713) 988-5555
TWX: 910-881-1606

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

Wyle Distribution Group
1959 South 4130 West, Unit B
Salt Lake City 84104
Tel: (801) 974-9953

WASHINGTON

†Almac Electronics Corporation
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
TWX: 910-444-2067

Arrow Electronics, Inc.
14320 N.E. 21st Street
Bellevue 98007
Tel: (206) 643-4800
TWX: 910-444-2017

Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 453-5874
TWX: 910-443-2469

WISCONSIN

†Arrow Electronics, Inc.
430 W. Rausson Avenue
Oakcreek 53154
Tel: (414) 764-6600
TWX: 910-262-1193

Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
TWX: 910-262-1182

CANADA

ALBERTA

Hamilton/Avnet Electronics
2816 21st Street N.E.
Calgary T2E 6Z2
Tel: (403) 230-3586
TWX: 03-827-642

Hamilton/Avnet Electronics
6845 Rexwood Road Unit 6
Mississauga, Ontario L4V1R2
Tel: (416) 677-0484

Zentronics
Bay No. 1
3300 14th Avenue N.E.
Calgary T2A 6J4
Tel: (403) 272-1021

BRITISH COLUMBIA

Hamilton/Avnet Electronics
105-2550 Boundry Road
Burmalay V5M 3Z3
Tel: (604) 272-4242

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
TWX: 04-5077-89

MANITOBA

Zentronics
590 Berry Street
Winnipeg R3H OS1
Tel: (204) 775-8661

ONTARIO

Arrow Electronics Inc.
24 Martin Ross Avenue
Downsview M3J 2K9
Tel: (416) 661-0220
TELEX: 06-218213

Arrow Electronics Inc.
148 Colonnade Road
Nepean K2E 7J5
Tel: (613) 226-6903

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units G & H
Mississauga L4V 1R2
Tel: (416) 677-7432
TWX: 610-492-8867

†Hamilton/Avnet Electronics
210 Colonnade Road South
Nepean K2E 7L5
Tel: (613) 226-1700
TWX: 05-349-71

†Zentronics
8 Tilbury Court
Brampton L6T 3T4
Tel: (416) 451-9600
TWX: 06-976-78

Zentronics
564/10 Weber Street North
Waterloo N2L 5C6
Tel: (519) 884-5700

Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 225-8840
TWX: 06-976-78

QUEBEC

Arrow Electronics Inc.
4050 Jean Talon Quest
Montreal H4P 1W1
Tel: (514) 735-5511
TELEX: 05-25596

Arrow Electronics Inc.
909 Charest Blvd.
Quebec 61N 269
Tel: (418) 687-4231
TLX: 05-13388

Hamilton/Avnet Electronics
2795 Rue Halpern
St. Laurent H4S 1P8
Tel: (514) 335-1000
TWX: 610-421-3731

Zentronics
505 Locke Street
St. Laurent H4T 1X7
Tel: (514) 735-5361
TWX: 05-827-535

†Microcomputer System Technical Demonstrator Centers

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Japan

FRANCE
Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin en Yvelines
France

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Piper's Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Seidlstrasse 27
D-8000 Munchen 2
West Germany

Order Number: 231737-001

Printed in U.S.A./HE-307.C/0486/20K/BL LD
Microprocessors