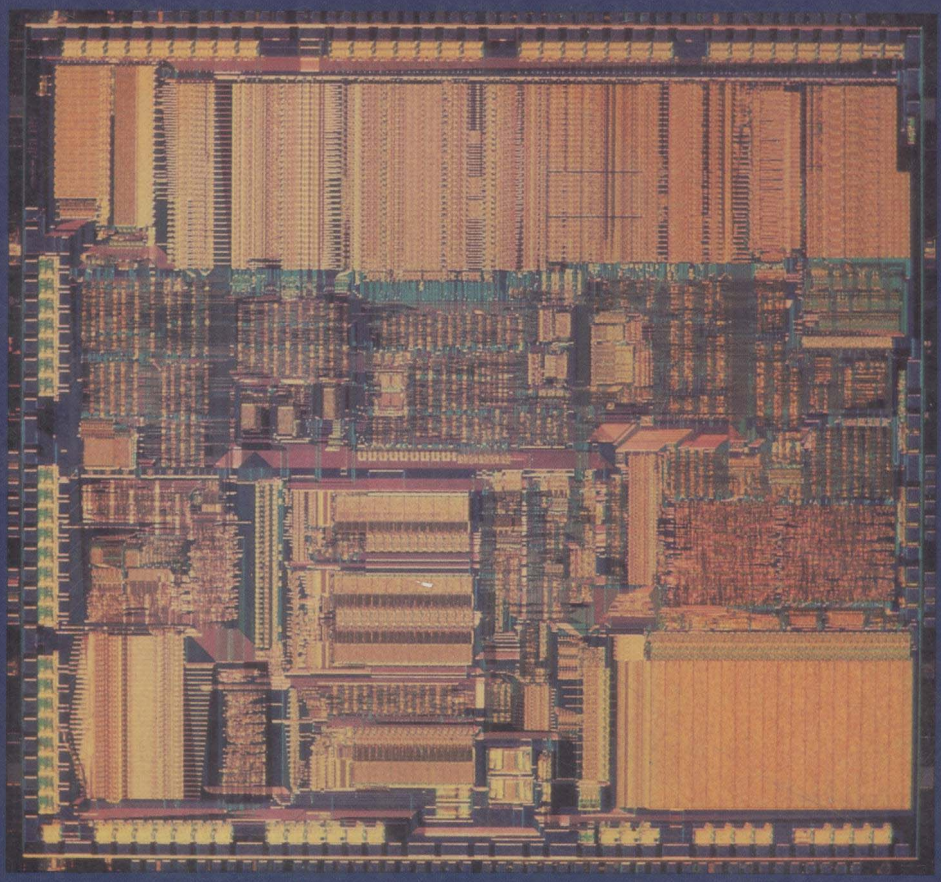
THE 80386/387 ARCHITECTURE



Stephen P. Morse
Eric J. Isaacson
Douglas J. Albert

THE 80386/387 ARCHITECTURE

Stephen P. Morse
Eric J. Isaacson
Douglas J. Albert

John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

Publisher: Stephen Kippur
Editor: Therese A. Zak
Managing Editor: Andrew B. Hoffer
Electronic Production Services: Publishers Network

Library of Congress Cataloging-in-Publication Data

Morse, Stephen P.
The 80386/387 architecture.

Includes bibliographies.
1. Intel 80386 (Microprocessor) 2. Intel 80387 (Microprocessor) I. Albert, Douglas J. II. Isaacson, Eric, III. Title.
QA76.8.I2684M67 1987 004.165 87-13366
ISBN 0-471-85352-6

Printed in the United States of America

87 88 10 9 8 7 6 5 4 3 2 1

THE 80386/387 ARCHITECTURE

To Erica, Megan, Melanie, and Anita

Acknowledgements

We would like to thank Richard Schell and John Oxaal for providing us with up-to-date reference information for the Intel and Weitek processors. We would also like to thank Joseph C. Krauskopf from Intel for reviewing the entire manuscript, and Chris Tice for reading the draft of the Weitek chapter, and correcting errors that we made.

CONTENTS

CHAPTER

1

INTRODUCTION

The 80386 is the latest member in an ever-evolving family of microprocessors. To best appreciate the current generation of microprocessors, we must look back and see where it all started and how we got to where we are. This chapter presents such a historical perspective, along with a review of some fundamentals of computers, the representation of numbers within computers, and the use of stacks in computers.

The remaining chapters in this book are organized in the following manner. The basic architecture of the 386 is presented in Chapters 2 and 3. The 386's 32-bit architecture is a significant change from the 8086 and the 286; you'll want to read much of Chapters 2 and 3 even if you're familiar with those predecessors. Chapter 4 presents the 387 architecture. This, for the most part, is similar to both the 8087 and 287 architectures and can be skipped if you're familiar with the floating-point instruction set. Chapter 5 deals with 386 features that are retained for compatibility with the 8086 and 286. Chapter 6 describes the support that the 386 gives to operating systems; the 386 is similar to the 286 in these respects. You may skip Chapter 6 if you are familiar with the 286, or if you aren't interested in operating-system software. Finally, Chapter 7 describes an interface designed by Weitek Corporation to provide floating-point processing even faster than the 387.

The History of Microcomputers

The evolution that led to the modern-day microcomputer can be broken down into two periods: a period of shrinkage followed by a period of enhancement. During the shrinkage period (circa 1940 to 1970) computers became smaller as the technology for fabricating their components evolved. This period culminated with a computer (albeit primitive even by 1970 standards) no larger than a postage stamp. The enhancement period (circa 1970 to the present) saw the postage-stamp computers become as powerful as their larger counterparts.

The Period of Shrinkage In the 1960s all electronic devices from radios and televisions to computers were built of bulky vacuum tubes. Computers of that vintage are sometimes referred to as first-generation computers, for example, IBM's 650 and 704. These computers were housed in large rooms containing several racks of electronic equipment. By the end of the decade, transistors and other solid-state devices began to replace vacuum tubes. Computers using this technology are called second-generation computers (the IBM 7090 and the Burroughs B5500, for example).

In the 1960s many discrete electronic components (transistors, resistors, etc.) were combined to form more complex electronic components called *integrated circuits*. An integrated circuit is fabricated on a wafer of silicon smaller than a postage stamp. It is mounted on a centipedelike structure that can be plugged into a system. This pluggable integrated circuit became known as a *chip*. Computers built from integrated circuits are the third generation computers (the IBM 360, the GE 635, and the Burroughs B6700). But integrated-circuit technology continued to advance, and by the early 1970s many of the components of a computer could be put together onto a single chip (Intel's 4004 and 8008). This led to the coining of the term *computer-on-a-chip*.

Computers-on-a-chip are called microcomputers or microprocessors. Although the terms are sometimes used interchangeably, there is a difference. A *microprocessor* is a single chip. It usually contains the control logic and arithmetic units of a computer but not the computer's memory or input/output devices. A *microcomputer* is an entire computer system consisting of a microprocessor chip, memory chips, and input/output devices. Sometimes the entire computer system is contained on one chip (Intel's 8048). This is called a *single-chip microcomputer*.

The Period of Enhancement The microprocessor era started with the introduction of Intel's 4004 and 8008 processors in 1971. This was the first generation of microprocessors. Both of these chips were designed for specialized applications—the 4004 in a calculator and the 8008 in a computer terminal. These microprocessors were intended as replacements for complex, custom-designed circuits. Now there was an alternative to designing a complex electronic circuit; one could design instead a simple circuit involving a microprocessor chip and write a program for it that simulated the functions of the intended complex circuit. Some typical applications in this period were electronic cash registers, intelligent typewriters, traffic light controllers, and microwave ovens. These are called *embedded* applications because the microprocessor is embedded into something other than a computer.

In 1974, when the 8008 matured into the 8080 (the second-generation microprocessor), a revolution occurred in the microprocessor field. Instead of being used as electronic logic replacements, 8080s were now appearing in applications which would have been infeasible if designed without microprocessors. Applications such as word processing machines, aircraft inertial navigation systems, and cruise missiles required either the ability to be easily modified (reprogrammed) or to fit into a small space, or both. The microprocessor provided these abilities, and the 8080 became the "standard" microprocessor to use.

It was now only a small step to go from a special-purpose, reprogrammable circuit such as a word processing machine to a general-purpose computing system based on a

microprocessor. One of the first such machines was the Intel Microprocessor Development System (MDS) introduced in 1974. Even though this was probably the first personal computer, Intel never wanted to call it that for several reasons. For one, they didn't want to attract the attention of the industry giants (IBM and DEC) who, coincidentally, were Intel customers; and for another, Intel knew the purchasing policies of most large corporations: an engineer could easily order a piece of laboratory equipment (such as a development system) but a request for a computer would have to go through the data processing department.

The microprocessor was now able to perform the computational tasks of the older and bulkier equipment and was inexpensive enough to find its way into the hands of the hobbyist. Many companies other than Intel began building 8080 chips, and some companies (notably Zilog) built enhanced versions of the 8080 (the Z80). Intel itself introduced an enhanced version in 1976 called the 8085. But the basic character of the 8080 wasn't to be significantly changed until 1978 when Intel came out with the 8086, the first member of the 86 family of microprocessors.

The 86 Family of Microprocessors Up to this point, the development of the microprocessor had followed the historical development of the mainframe computers that came before. But now Intel wanted to make a giant step forward that would put microprocessors in the lead. To this end they embarked on an advanced processor development. This processor, initially known as the 8816, incorporated the most advanced operating system and programming language concepts then known. Unfortunately, the 8816 took much longer to develop than was initially forecast. Intel was under continuing pressure to come out quickly with a processor that was better than the Zilog Z80. This provided the impetus for the development of an 8080 successor, namely the 8086. The 8086 was intended to tide Intel over until the introduction of the 8816, which by then was being called the 8800.

In 1978 Intel produced the 8086. This was the first microprocessor capable of working with sixteen bits of data at a time (the 8080 worked with eight bits). Two other companies quickly announced plans for 16-bit processors, namely Zilog with its Z8000 and Motorola with its 68000 (big numbers were in fashion at that time). This was the start of the third generation of microprocessors.

The 8088, an 8-bit version of the 8086, appeared in 1979. Two years later IBM announced its entry into the personal computer field and revealed that the 8088 would be the processor in its first personal computer—the IBM PC. With that endorsement behind them, the 8086 and 8088 were on their way to becoming the most popular microprocessors ever.

With the advent of the IBM PC, the nature of the microprocessor industry began to change. Instead of being used in a wide variety of embedded applications (traffic lights, microwave ovens, etc.), the bulk of the 16-bit microprocessors produced were now used in personal computers.

In 1982 Intel brought out enhanced versions of the 8086 and 8088, namely the 186 and 188. These chips embodied the 8086/8088 along with some of the support chips needed in a microprocessor system. This did not add any new capabilities but rather reduced the number of chips (and hence the cost) needed for a complete system.

That same year, 1982, Intel came out with the 286. This represented Intel's first big leap beyond the 8086. Plans for the 286 had begun four years earlier, long before the widespread use of personal computers. The primary reason for developing the 286 was the 8086's lack of memory management, a feature which Intel's competitors (mainly Motorola) provided for their 16-bit line. Another reason for Intel's commencing on the 286 was the continued slippage of the 8800 back in the late 70s. In fact, the 8800 slipped all the way to 1982 when it finally arrived (again renamed) as the iAPX 432!

The personal computer industry was flourishing by this time. IBM had set the standard with its PC and PC/XT machines, and others were producing close copies of the IBM machines. In 1983 Apple came out with a competitor to the IBM PC called the MacIntosh, which uses a microprocessor from Motorola's 68000 family. The second generation IBM personal computer was the IBM PC/AT (AT standing for advanced technology) which came out in 1984 and incorporates the 286 processor. The AT too became a standard which was widely copied.

Parallel to the development of the 86 family at Intel, Motorola was developing its 68000 family of microprocessors. The first two members of this family, namely the 68000 and the 68010 had 32-bit registers but could accommodate only 24-bit addresses. Furthermore, contrary to Intel's fears, early members of the 68000 family had great difficulty in supporting memory management. The 68020, introduced in 1984, corrected both these deficiencies.

Finally, in 1985, Intel formally unveiled the 386, the most significant advance yet within the 86 family. Not only was this Intel's offering for the next generation of IBM-compatible personal computers, it was also Intel's answer to Motorola's 68020 32-bit microprocessor. The 386 introduces a 32-bit general register set, an enormously expanded memory space, a more complete set of memory indexing modes, memory paging, and an enhanced facility for 8086 emulation. In 1987 IBM introduced its Personal System/2 line of computers, choosing the 386 for its high-end models and the earlier 86 family microprocessors for its low-end models. Some speculate that the 386 is the ultimate microcomputer architecture, since there is no pressing reason to have register sizes and memory addresses greater than 32 bits. We feel that it is dangerous to call anything ultimate in the world of computers: you may be assured that even as this book is being written, Intel is working on successors to the 386.

During the evolution in the microprocessor area, a parallel evolution was occurring in the area of microcoprocessors. A *coprocessor* is a subordinate processor that performs a specialized function for a general-purpose processor. The first popular coprocessor was the 8087, which performed floating-point computations for the 8086 and 8088. Concurrent with the development of the 8087, the Institute of Electrical and Electronic Engineers (IEEE) was developing a standard for microprocessor floating-point arithmetic. The 8087 was the first math processor to implement the standard being proposed at that time.

Besides being a coprocessor for the 8086, the 8087 served as a coprocessor for the 186 and 188 when those processors came along. But due to a different coprocessor interface incorporated in the 286, a modified version of the 8087 was needed to function with that processor: the 287. Finally, the 32-bit bus supported by the 386

dictated yet another coprocessor interface. By this time the proposed IEEE standard had been modified and formally adopted as a standard. Since a new coprocessor would be needed for the 386 anyway, the coprocessor was designed to implement the newly adopted standard. This new coprocessor, the 387, is therefore slightly incompatible with the 287 and 8087.

A word about nomenclature is in order here. Once upon a time the nomenclature was clear: everybody called the 8080 "the 8080," for example. When the 8086 was introduced, everybody called it "the 8086." Then Intel, in an attempt to make their products sound more impressive, replaced the prefix "80" with "iAPX" (please don't ask what APX means). They retroactively declared the 8086 to be the iAPX86, and referred to the successors as the iAPX186, iAPX188, and iAPX286. But outside of Intel the iAPX prefix never caught on. People were too entrenched in their habits of referring to the part numbers: 80186, 80188, 80286, 80287. Intel has finally surrendered; no mention of the iAPX386 appears in Intel literature, and 80386 and 80387 are the official names. We refer to the 8086 and 8087 as such, using the full part number, but for simplicity, we follow the common practice of omitting the "80" prefix on the names of successors: 186, 188, 286, 287, 386, 387. We refer to the entire family of processors as the "86 family."

Fundamentals of Computers

It is assumed that you are already familiar with the basic concepts of computers so we will just review them quickly here. A computer obtains data from an *input device*, processes the data, and delivers the final results to an *output device*. The particular processing to be done is specified by a list of instructions called a *program*. The program is stored in the computer's *memory*.

The operations of the computer are controlled by the *central processing unit*, or *processor* for short. The processor fetches instructions from the memory, decodes the instructions to determine what operations are to be performed, and executes the instructions by performing the operations. In order to perform the operations, the processor must sometimes send control signals to other devices within the computer. The operations that are performed during instruction execution consist of moving data and performing computations on data. The computer memory is used to supply inputs for the computations and to hold the results of the computations.

To see how all this ties together, let's analyze the execution of an *add* instruction. The processor sends a signal to the memory requesting the next instruction. The memory responds by sending an instruction to the processor. The processor then decodes the instruction and discovers that it's an *add* instruction. It then (1) sends out signals to the memory telling it to move two values to the processor, (2) adds the two values it received, and (3) sends out signals to the memory telling it to receive the result of the addition.

A *memory* is a collection of sequential *locations*, each having a unique *address*. Each location contains a sequence of *bits* (short for *b*inary dig*its*). These bits are the *contents* of the location. Each bit is either a 0 or a 1.

Registers, like memory, are also used to hold intermediate results. The registers are inside the processor, and therefore it's easier and faster to access values in registers than in memory. *Flags* inside the processor are used to keep track of what's going on. There are two kinds of flags—those that record information about the effects of previously executed instructions (status flags) and those that control the operations of the computer (control flags). An example of a status flag is a flag that indicates if a result is too big for the computer to handle. An example of a control flag is a flag that tells the computer to execute instructions at a slower rate, such as one per hour. It's also possible to have a flag that is both a status flag and a control flag. An example is the 386's NT flag (described later).

Representation of Numbers

We are accustomed to representing integers as a sequence of decimal digits, such as 365. This is interpreted as 3 hundreds, 6 tens, and 5 ones. It is sometimes called a base-ten representation. Integers in computers are usually represented as a sequence of binary digits (bits) such as 11010. This is the base-two representation of twenty-six: 1 sixteen, 1 eight, 0 fours, 1 two, and 0 ones. Binary numbers can be added, subtracted, multiplied, and divided directly (no need to convert them to decimal numbers first) as long as we remember that 1 plus 1 is 10 (1 two and 0 ones) and not 2. For example:

```
   1001     binary representation of nine
+  0101     binary representation of five
-------
   1110     binary representation of fourteen
```

We tend to get confused with long sequences of binary digits, although computers aren't perturbed the least bit. For example, 10110101 is the binary representation for one hundred eighty-one. To make things simpler, we have devised a scheme of compressing long sequences of binary digits by grouping the bits four at a time. Each group of four bits (sometimes called a *nibble*) is represented by a single character as shown in Table 1.1. Thus 10110101 is abbreviated to B5. This is called a *hexadecimal* number and is exactly the number system we would have used if we had been born with sixteen fingers.

The binary notation is perfect for describing positive numbers and zero. But when we want to allow for negative numbers, we need an additional mechanism to indicate the sign of the number. The simplest way to do this is to use the most significant (leftmost) bit of the number to indicate the sign. For example:

```
0000  0100    would be +4
1000  0100    would be −4
0111  1111    would be +127
1111  1111    would be −127
```

Table 1.1 Hexadecimal Representation

Group of Four Bits	Hexadecimal Digit	Value
0000	0	zero
0001	1	one
0010	2	two
0011	3	three
0100	4	four
0101	5	five
0110	6	six
0111	7	seven
1000	8	eight
1001	9	nine
1010	A	ten
1011	B	eleven
1100	C	twelve
1101	D	thirteen
1110	E	fourteen
1111	F	fifteen

Such a representation is called *sign-magnitude* representation and has one serious drawback: it requires a new set of arithmetic rules. This becomes obvious when we try to use binary arithmetic to subtract +1 from 0 and expect to get −1.

```
   0000  0000     0 in sign-magnitude
 - 0000  0001   + 1 in sign-magnitude
 ---------------
   1111  1111   − 127 in sign-magnitude
```

If we want to use the same binary arithmetic on signed numbers that we used on unsigned numbers, we need a signed-number representation in which 1111 1111 represents −1, not −127. Furthermore, subtracting +1 from −1 should give −2. Let's perform this subtraction to see what −2 should look like.

```
   1111  1111   here's − 1
 - 0000  0001   subtract + 1
 ---------------
   1111  1110   and call this − 2
```

This is called a *two's complement* representation; it has the property that binary additions and subtractions will give the correct two's complement result. For example:

```
   0000  0011   + 3 in two's complement
 + 1111  1110   − 2 in two's complement
 ---------------
   0000  0001   + 1 in two's complement
```

It also has the property that the most significant bit of every nonnegative (positive or zero) number is 0 and of every negative number is 1. Thus, just as in sign-magnitude representation, this bit serves as a sign bit.

The sign of a two's complement number can be changed by changing the value of each bit and adding +1. For example, we can obtain the two's complement representation of −5 from the two's complement representation of +5 as follows:

```
  0000 0101    +5 in two's complement
  1111 1010    +5 with each bit changed
+ 0000 0001    +1 in two's complement
-----------
  1111 1011    −5 in two's complement
```

We must be careful when lengthening two's complement numbers. If an 8-bit two's complement number is to be extended to 16 bits (so that it can be added to a 16-bit two's complement number, for example), some thought must be given as to what goes into the additional eight bits.

Suppose we want to add 0000 0001 (+1 in two's complement) to 0000 0000 0000 0011 (+3 in two's complement). In this case, there's no doubt that we would simply append eight 0's on the left side of the +1 and then add:

```
  0000 0000 0000 0011    +3 in two's complement
+ 0000 0000 0000 0001    +1 in two's complement
-------------------
  0000 0000 0000 0100    +4 in two's complement
```

However, if we want to add 1111 1111 (−1 in two's complement) to 0000 0000 0000 0011 (+3 in two's complement), we must append eight 1's to the left side of −1 (appending 0's would make it a positive number). The addition is then:

```
  0000 0000 0000 0011    +3 in two's complement
+ 1111 1111 1111 1111    −1 in two's complement
-------------------
  0000 0000 0000 0010    +2 in two's complement
```

Thus the extension of an 8-bit number to a 16-bit number looks like this:

Value	8-bit Representation	16-bit Representation
+1	0000 0001	0000 0000 0000 0001
−1	1111 1111	1111 1111 1111 1111

The rule for extending a two's complement number is to append additional bits on the left side of the number with each appended bit having the same value as the original sign bit. This process is called *sign extending*.