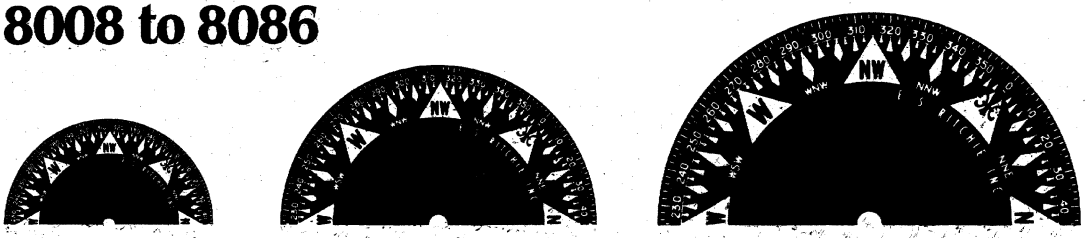
A mere six years of microprocessor evolution have yielded a three-orders-of-magnitude performance improvement.

Intel Microprocessors — 8008 to 8086



Stephen P. Morse,* Bruce W. Ravenel,* Stanley Mazor, and William B. Pohlman
Intel Corporation

Intel introduced its first microprocessor in November 1971 with the advertisement, "Announcing a New Era in Integrated Electronics." The fulfillment of this prophecy has already occurred with the delivery of the 8008 in 1972, the 8080 in 1974, the 8085 in 1976, and the 8086 in 1978. During this time, throughput has improved one-hundred-fold, the price of a CPU chip has declined from $300 to $3, and microcomputers have revolutionized design concepts in countless applications. They are now entering our homes and cars.

Each successive product implementation depended on fabrication innovations, improved architecture, better circuit design, and more sophisticated software, and throughout this development, upward compatibility not envisioned by the first designers was maintained. Here, we will try to provide an insight into the evolutionary process that transformed the 8008 into the 8086 and give descriptions of the various processors, emphasizing the 8086.

Historical setting. In the late 1960's it became clear that the practical use of LSI circuits depended on defining chips having

- a high gate-to-pin ratio,
- a regular cell structure, and
- a large standard part market.

In 1968, Intel Corporation was founded to exploit the semiconductor memory market, which uniquely fulfilled these criteria. Early semiconductor RAMs, ROMs, and shift registers were welcomed wherever small memories were needed, especially in calculators and CRT terminals.

*Currently with Language Resources, Sunnyvale, California.

In 1969, Intel engineers began to study ways of integrating and partitioning the control logic functions of these systems into LSI chips.

At this time, other companies (notably Texas Instruments) were exploring ways to reduce the time needed to develop custom integrated circuits. Computer-aided design of custom ICs was a hot issue then. Custom ICs are making a comeback today, this time, in the high-volume applications that typify the low end of the microprocessor market.

An alternate approach was to think of a customer's application as a computer system requiring a control program, I/O monitoring, and arithmetic routines, rather than as a collection of special-purpose logic chips. Drawing on its strength in memory, Intel partitioned systems into RAM, ROM, and single controller chips, i.e., CPUs.

Intel embarked on the design of two customer-sponsored microprocessors—the 4004 for a calculator and the 8008 for a CRT terminal. The 4004 replaced what otherwise would have been six customized chips usable by only one customer. Because the first microcomputer applications were known and easy to understand, instruction sets and architectures were defined in a matter of weeks. As programmable computers, their uses could be extended indefinitely.

Both microprocessors were complete CPUs on a chip and had similar characteristics. But because the 4004 was designed for serial BCD arithmetic and the 8008 for 8-bit character handling, their instruction sets differed.

The succeeding years saw the evolution that eventually led to the 8086. Table 1 summarizes the progression of features that took place during these years.

0018-9162/80/1000-0042$00.75 © 1980 IEEE

The 8008

Late in 1969, Computer Terminal Corporation (today called Datapoint) contracted Intel to do a pushdown stack chip for a processor to be used in a CRT terminal. Datapoint had intended to build a bit-serial processor in TTL logic, using shift register memory. Intel counterproposed that the entire processor be implemented on one chip. This processor was to become the 8008 and, along with the 4004, was to be fabricated using PMOS, the then-current memory fabrication technology. Due to the long lead time required by Intel, Datapoint proceeded to market the serial processor, and thus compatibility constraints were imposed on the 8008.

Most of the instruction set and register organization were specified by Datapoint. Intel modified the instruction set so the processor would fit on one chip and added instructions to make it more general-purpose. For although Intel was developing the 8008 for a specific customer, they wanted to have the option of selling it to others. And since Intel was using only 16- and 18-pin packages in those days, they chose to use 18 pins for the 8008 rather than design a new package for what was believed to be a low-volume chip.

8008 instruction set processor. The 8008 processor architecture is quite simple compared to that of today's microprocessors. The data handling facilities provide for byte data only. The memory space is limited to 16K bytes, and the stack is on the chip and limited to a depth of eight. The instruction set is small but symmetrical, with only a few operand addressing modes available. An interrupt mechanism is provided, but there is no way to disable interrupts.

Memory and I/O structure. The 8008 addressable memory space consists of 16K bytes. That seemed like a lot back in 1970 when memories were expensive and LSI devices were slow. It was inconceivable in those days that anybody would want to put more than 16K of this precious resource on anything as slow as a microprocessor.

The memory size limitation was imposed by the lack of available pins. Addresses are sent out in two consecutive clock cycles over an 8-bit address bus. Two control signals, which would have been on dedicated pins if such were available, are sent out with every address instead, thereby limiting addresses to 14 bits.

The 8008 supports eight 8-bit input ports and 24 8-bit output ports. Each port is directly addressable by the in-

Table 1.
Feature comparison—Intel microprocessors, 1972-1978.

	8008	8080	8085	8086
INTRODUCTION DATE	1972	1974	1976	1978
NUMBER OF INSTRUCTIONS	66	111	113	133
NUMBER OF FLAGS	4	5	5	9
MAXIMUM MEMORY SIZE	16K BYTES	64K BYTES	64K BYTES	1M BYTES
I/O PORTS	8 INPUT 24 OUTPUT	256 INPUT 256 OUTPUT	256 INPUT 256 OUTPUT	64K INPUT 64K OUTPUT
NUMBER OF PINS	16	40	40	40
ADDRESS BUS WIDTH	8*	16	16	20*
DATA BUS WIDTH	8*	8	8	16*
DATA TYPES	8-BIT UNSIGNED	8-BIT UNSIGNED 16-BIT UNSIGNED (LIMITED) PACKED BCD (LIMITED)	8-BIT UNSIGNED 16-BIT UNSIGNED (LIMITED) PACKED BCD (LIMITED)	8-BIT UNSIGNED 8-BIT SIGNED 16-BIT UNSIGNED 16-BIT SIGNED PACKED BCD UNPACKED BCD
ADDRESSING MODES	REGISTER IMMEDIATE**	MEMORY DIRECT (LIMITED) MEMORY INDIRECT (LIMITED) REGISTER IMMEDIATE**	MEMORY DIRECT (LIMITED) MEMORY INDIRECT (LIMITED) REGISTER IMMEDIATE**	MEMORY DIRECT MEMORY INDIRECT REGISTER IMMEDIATE INDEXING

*ADDRESS AND DATA BUS MULTIPLEXED.
**MEMORY CAN BE ADDRESSED AS A SPECIAL CASE BY USING REGISTER M.

struction set. The chip's designers felt that output ports were more important than input ports because input ports can always be multiplexed by external hardware under control of additional output ports.

One of the interesting things about that era was that, for the first time, users were given access to the memory bus and could define their own memory structure; they were not confined to what the vendors offered, as they had been with minicomputers. The user had the option, for example, of putting I/O ports inside the memory address space instead of in a separate I/O space.

Register structure. The 8008 processor contains two register files and four 1-bit flags. The register files are the scratchpad and the address stack.

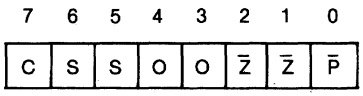
Scratchpad. The scratchpad file contains an 8-bit accumulator called A and six additional 8-bit registers called B,C,D,E,H, and L. All arithmetic operations use the accumulator as one of the operands and store the result back in the accumulator. All seven registers can be used interchangeably for on-chip temporary storage.

There is one pseudoregister, M, which can be used interchangeably with the scratchpad registers. M is, in ef-

Saving and restoring flags in the 8008

Interrupt routines must leave all processor flags and registers unaltered so as not to contaminate the processing that was interrupted. This is most simply done by having the interrupt routine save all flags and registers on entry and restore them prior to exiting. The 8008, unlike its successors, has no instruction for directly saving or restoring flags. Thus, 8008 interrupt routines that alter flags (practically every routine does) must conditionally test each flag to obtain its value and must then save that value. Since there are no instructions for directly setting or clearing flags, the flag values must be restored by executing code that will put the flags in the saved state.

The 8008 flags can be restored very efficiently if they are saved in a byte in memory in the following format:



most significant bit = bit 7 = original value of CARRY
bit 6 = original value of SIGN
bit 5 = original value of SIGN
bit 4 = 0
bit 3 = 0
bit 2 = complement of original value of ZERO
bit 1 = complement of original value of ZERO
bit 0 = complement of original value of PARITY

With the formatted information saved in a byte called FLAGS, the following two instructions will restore all the saved flag values:

```
LDA FLAGS     ; load saved flags into accumulator
ADD A         ; add the accumulator to itself
```

This instruction sequence loads the saved flags into the accumulator and then doubles the value, thereby moving each bit one position to the left. This causes each flag to be set to its original value for the following reasons:

- The original value of the CARRY flag, in the leftmost bit, will be moved out of the accumulator and wind up in the CARRY flag.
- The original value of the SIGN flag, in bit 6, will wind up in bit 7 and will become the sign of the result. The new value of the SIGN flag will reflect this sign.
- The complement of the original value of the PARITY flag will wind up in bit 1, and it alone will determine the parity of the result (all other bits in the result are paired up and have no net effect on parity). The new setting of the PARITY flag will be the complement of this bit (flag denotes even parity) and therefore will take on the original value of the PARITY flag.
- Whenever the ZERO flag is 1, the SIGN flag must be 0 (zero is a positive 2's-complement number) and the PARITY flag must be 1 (zero has even parity). Thus, an original ZERO flag value of 1 will cause all bits of FLAGS, with the possible exception of bit 7, to be 0. After the execution of the ADD instruction, all bits of the result will be 0 and the new value of the ZERO flag will therefore be 1.
- An original ZERO flag value of 0 will cause two bits in FLAGS to be 1 and will wind up in the result as well. The new value of the ZERO flag will therefore be 0.

The above algorithm relies on consistent flag values; i.e., the SIGN flag cannot be a 1 when the ZERO flag is a 1. This is always true in the 8008, since the flags come up in a consistent state whenever the processor is reset and since flags can be modified only by instructions which always leave the flags in a consistent state. The 8080 and its derivatives allow the programmer to arbitrarily modify the flags by popping a value of his choice off the stack and into the flags. Thus, the above algorithm will not work on those processors.

A code sequence for saving the flags in the required format is as follows:

```
     MVI  A,0      ; move zero in accumulator
     JNC  L1       ; jump if CARRY not set
     ORA  80H      ; OR accumulator with 80H hex
                   (set bit 7)
L1:  JZ   L3       ; jump if ZERO set (and SIGN
                   not set and PARITY set)
     ORA  06H      ; OR accumulator with 03 hex
                   (set bits 1 and 2)
     JM   L2       ; jump if negative (SIGN set)
     ORA  60H      ; OR accumulator with 60 hex
                   (set bits 5 and 6)
L2:  JPE  L3       ; jump if parity even (PARITY set)
     ORA  01H      ; OR accumulator with 01 hex
                   (set bit 0)
L3:  STA  FLAGS    ; store accumulator in FLAGS
```

fect, that particular byte in memory whose address is currently contained in H and L (L contains the eight low-order bits of the address and H contains the six high-order bits). Thus, M is a byte in memory and not a register; although instructions address M as if it were a register, accesses to M actually involve memory references. The M register is the only mechanism by which data in memory can be accessed.

Address stack. The address stack contains a 3-bit stack pointer and eight 14-bit address registers providing storage for eight addresses. The programmer cannot directly access these registers; instead, he manipulates them with control-transfer instructions.

Any one of the eight address registers in the address stack can serve as the program counter; the current program counter is specified by the stack pointer. The other seven address registers permit storage for nesting of subroutines up to seven levels deep. The execution of a call instruction causes the next address register to become the current program counter, and the return instruction causes the address register that last served as the program counter to again become the program counter. The stack will wrap around if subroutines are nested more than seven levels deep.

Flags. The four flags in the 8008 are CARRY, ZERO, SIGN, and PARITY. They reflect the status of the latest arithmetic or logical operation. Any flag can be used to alter program flow through the use of the conditional jump, call, or return instructions. There is no direct mechanism for saving or restoring flags, which places a severe burden on interrupt processing (see box at left for details).

The CARRY flag indicates if a carry-out or borrow-in was generated, thereby providing a multiple-precision binary arithmetic capability.

The ZERO flag indicates whether or not the result is zero. This provides the ability to compare two values for equality.

The SIGN flag reflects the setting of the leftmost bit of the result. The presence of this flag creates the illusion that the 8008 is able to handle signed numbers. However, there is no facility for detecting signed overflow on additions and subtractions. Furthermore, comparing signed numbers by subtracting them and then testing the SIGN flag will not give the correct result if the subtraction resulted in signed overflow. This oversight was not corrected until the 8086.

The PARITY flag indicates whether the result is of even or odd parity. This permits testing for transmission errors, an obviously useful function for a CRT terminal.

Instruction set. The 8008 instructions are designed for moving or modifying 8-bit operands. Operands are contained in the instruction itself (immediate operand), in a scratchpad register (register operand), or in the M register (memory operand). Since the M registers can be used interchangeably with the scratchpad registers, there are only two distinct operand addressing modes—immediate and register. Typical instruction formats for these modes are shown in Figure 1.

The instruction set consists of scratchpad-register instructions, accumulator-specific instructions, transfer-of-control instructions, input/output instructions, and processor-control instructions.

The scratchpad-register instructions modify the contents of the M register or any scratchpad register. This can

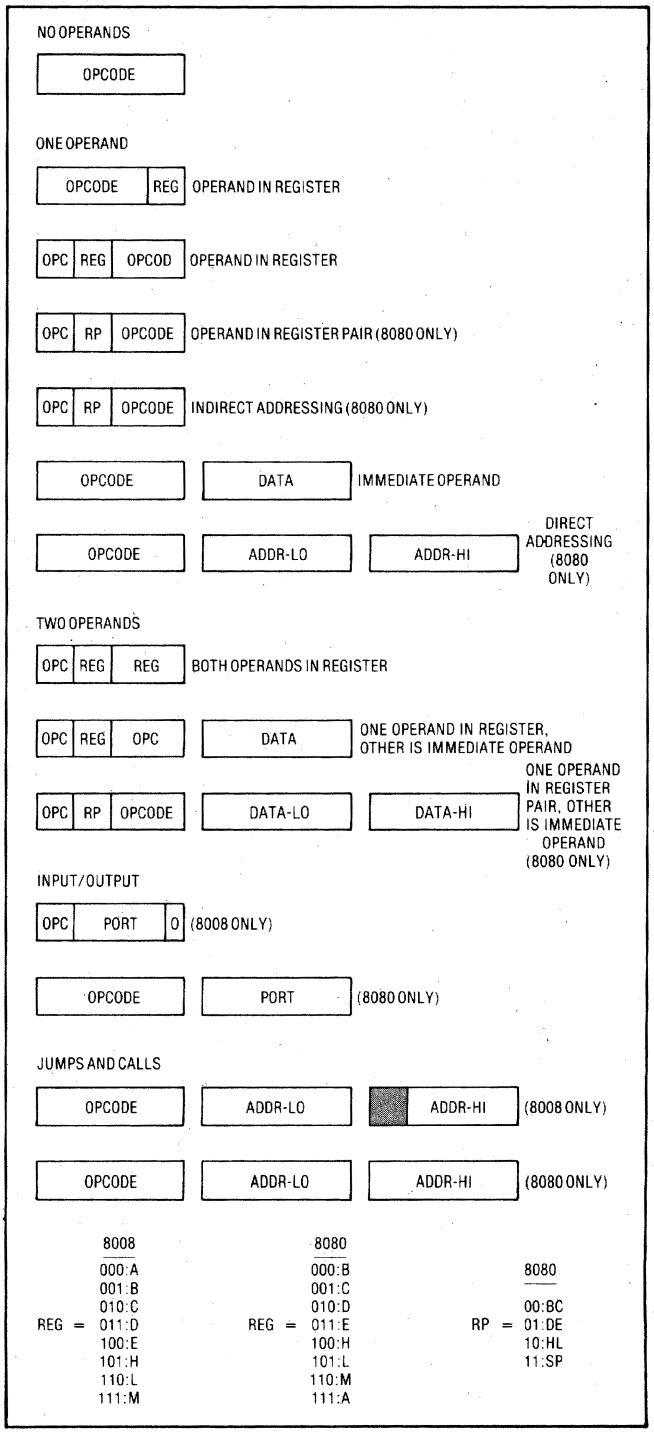


Figure 1. Typical 8008 and 8080 instruction formats.

involve moving data between any two registers, moving immediate data into a register, or incrementing or decrementing the contents of a register. The incrementing and decrementing instructions were not in Datapoint's specified instruction set; they were added by Intel to provide for loop control, thereby making the processor more general-purpose.

Most of the accumulator-specific instructions perform operations involving the accumulator and a specified operand. The operand can be any one of the scratchpad registers, including M, or it can be immediate data. The operations are add, add-with-carry, subtract, subtract-with-borrow, logical AND, logical OR, logical exclusive-OR, and compare. Furthermore, there are four unit-rotate instructions that operate on the accumulator. These instructions perform either an 8- or 9-bit rotate (the CARRY flag acts as a ninth bit) in either the left or right direction.

Transfer-of-control instructions consist of jumps, calls, and returns. Any transfer can be unconditional, or conditional based on the setting of any one of the four flags. Making calls and returns conditional was done only to preserve the symmetry with jumps. A short one-byte form of call—which will be discussed with interrupts—is also provided.

Each jump and call instruction (with the exception of the one-byte call) specifies an absolute code address in the second and third byte of the instruction. The second byte contains the six high-order bits of the address and the third byte the eight low-order bits. This inverted storage, which was to haunt all processors evolved from the 8008, was a result of a need for compatibility with the Datapoint bit-serial processor, which processes addresses from low bit to high bit. This inverted storage did have a virtue in those early days when 256 x 8 memory chips were popular: It allowed all memory chips to select a byte and latch it for output while waiting for the six high-order bits which selected the chip. This speeded up memory access.

There are eight input and 24 output instructions using up 32 opcodes. Each I/O instruction transfers a byte of data between the accumulator and a designated I/O port.

The processor-control instructions are halt and no-op. Halt puts the processor into a waiting state. The processor remains in that state until an interrupt occurs. No-op is actually one of the move instructions; specifically, it moves the contents of the accumulator into the accumulator, thereby having no net effect (move instructions do not alter flag settings).

Interrupts. Interrupt processing was not a requirement for the 8008. Hence, only the most primitive mechanism conceivable—not incrementing the program counter—was provided. Such a mechanism permits an interrupting device to jam an instruction into the processor's instruction stream. This is accomplished by having the interrupting device, instead of memory, respond to the instruction fetch; since the program counter isn't incremented, the instruction in memory that didn't get fetched won't be skipped. The instruction typically supplied by the interrupting device is a call, so that an interrupt service routine can be entered and then the main program can be resumed after interrupt processing is complete (a jump instruction would result in the loss of the main program return address). To simplify the interrupting device's task of generating an instruction, the 8008 instruction set provides eight one-byte subroutine calls, each to a fixed location in memory.

There are no instructions for disabling the interrupt mechanism; thus, this function must be realized with external hardware. More important, there are no instructions for conveniently saving the registers and flags when an interrupt occurs.

The 8080

By 1973, memory fabrication technology had advanced from PMOS to NMOS. As an engineering exercise, Intel decided to use the 8008 layout masks with the NMOS process to obtain a faster 8008. After a short study, the company determined that a new layout was required. It therefore decided to enhance the processor at the same time and to utilize the new 40-pin package made practical by high-volume calculator chips. The result was the 8080 processor.

In The Beginning...

In the beginning Intel created the 4004 and the 8008. And these processors were without enough memory and throughput. And Intel said, "Let there be an 8080," and there was an 8080 and Intel saw that it was good. And Intel separated the 8008 market from the 8080 market.

And Intel said, "Let there be an 8085 with an oscillator on the same chip as the processor, and let an on-chip system controller separate the data from the control lines. And Intel made a firmament and divided the added instructions which were under the firmament from the added instructions which were above the firmament. And Intel called the first set of instructions RIM and SIM. And the other instructions Intel never announced.

And Intel said, "Let the market below the 8085 market be served with a processor and let on-chip ROM appear." And Intel called the new processor the 8048. And the market it served Intel called the low end. And Intel saw that it was good.

And Intel said, "Let a new-generation processor serve the mid-range market. And let there be true 16-bit facilities in the mid-range. And let there be one megabyte of memory and efficient interruptible byte-string instructions and full decimal arithmetic." And Intel saw the collection of all these things, that it was good, and Intel called it the 8086.

And Intel said, "Now let us make a processor in our image, after our likeness, and let it have dominion over the high-end market." So Intel created the APX 432 in his own image, in the image of Intel created he it, data processor and I/O processor created he them. And Intel blessed them and said unto them be fruitful and multiprocess and revolutionize the microprocessor market and have dominion over the Z8000 and the M68000 and over every competitor that enters the market.

And Intel saw everything that he had made and, behold, it was good.

—S.P. Morse

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.