CHAPTER 6.4
MICROPROCESSORS

S. Tewksbury

INTRODUCTION

This chapter reviews microprocessors, ranging in complexity and performance from 8-bit microprocessors widely used in embedded systems applications to very high performance 32- and 64- bit microprocessors used for powerful desktop, server, and higher-level computers. The history of the evolution of microprocessor architectures is rich in the variety of hardware/software environments implemented and the optimization of architectures for general-purpose computation. A variety of simple 8-bit microprocessors introduced over 30 years ago continue to thrive, adding to the basic microprocessor architecture a variety of other analog and digital functions to provide a *system-on-a-chip* solution for basic, intelligent products. These include digital cameras, electronic toys, automotive controls, and many other applications. These applications represent cases in which the programmable capabilities of microprocessors are embedded in products to yield "intelligent modules" capable of performing a rich variety of functions in support of the application. Many of these applications are intended for portable products (such as cameras) where battery operation places a premium on the power dissipated by the electronics.

Starting from the early 8-bit microprocessors, most manufacturers (Intel, Motorola, and so forth) followed a path of evolutionary development driven by the exponentially increasing amount of digital logic and memory that could be placed on a single IC. The class of microprocessor generally called *microcontroller* made use of the additional circuitry available on an IC to integrate earlier external circuitry directly onto the microcontroller IC. For example, 8-bit microcontrollers based on the Motorola 6800 and Intel 8085 8-bit microprocessors have evolved to include substantial nonvolatile memory (able to hold programs without power), significant internal RAM (above to provide storage for data being manipulated), digital ports (e.g., serial ports, parallel ports, and so forth) to the external world, and various analog circuit functions. Among the analog circuit functions are analog-to-digital and digital-to-analog converters, allowing the microcontroller to capture information from sensors and to drive transducers. With increasing circuitry per IC, functions such as the analog-to-digital converters have evolved into more complex components—for example, allowing a multiplicity of different analog input signals to be converted into digital signals using an analog multiplexing circuit and allowing programmable gain to amplify the incoming analog signal to levels minimizing the background noise included in the digital signal.

As the density of VLSI ICs has increased, earlier microprocessors requiring the full IC area have become smaller in area required. The 8- and 16-bit controllers can now be embedded in VLSI ICs to extend many of today's applications to "intelligent" systems. This embedding of microprocessor cores into other digital circuit functions has extended to the area of programmable logic (general-purpose arrays to logic that can be user programmed to perform a custom function). For example, Altera programmable logic ICs includes IP (Intellectual Property) cores such as a 200 MHz 32-bit ARM9 core processor along with embedded RAM cores. Similar capabilities exist for other manufacturers of programmable logic. Today, the combination of programmable logic allows a user to efficiently implement custom logic functions of considerable complexity (i.e., the Xilinx Virtex family of programmable logic provides 8 million gates and can operate at a 300 MHz clock rate) while

including the familiar functionality of well-established microprocessors for overall control. As the VLSI technologies continue to advance, the more powerful microprocessors of today will eventually migrate into these embedded applications as even more powerful microprocessors emerge for the desktop/laptop computer market.

Another evolutionary path was to create increasingly powerful microprocessors for applications such as desktop computers. Drawing power directly from a plug, minimizing power dissipation is less a priority (aside from the inevitable side issue of heating due to power dissipation). Instead, raw computing performance is the objective. Having to provide connections among integrated circuits to handle different peripheral functions (floating point arithmetic units, modules managing memory access though cache and other techniques, image processing modules, and so forth) can create a substantial performance barrier because of data transfer rate limitations. Migrating such peripheral functions essential to the basic computer operation can greatly improve the performance of the overall computer (microprocessor and peripherals). As an example, the evolution of the Motorola 68000 family starts with the basic MC68000. It then progressed to the MC68010, to the MC68020, to the MC68030, to the MC68040, and beyond not by changing the core microprocessor architecture greatly but rather by moving "onto the chip" such functions as the floating point accelerator, the memory management unit, virtual memory control, cache memory to reuse data/instructions loaded from RAM, and so forth.

The greater computational power of these early 16/32-bit microprocessors (such as the Motorola MC68000) is attractive in many embedded applications. As a result, microcontrollers have evolved beyond the 8-bit microprocessor cores to include these classical 16/32-bit microprocessor cores. At the same time, the microcontrollers retain the integration of other peripherals that support the applications (such as automotive and entertainment) into which they are embedded.

Returning to the microprocessors used for desktop computers, workstations, servers, and so forth, more recent advances have challenged and overcome many limitations that were once considered fundamental. Microprocessors such as the Pentium introduced in 2001–2002 operate at clock rates well above 1 GHz, a clock rate that was deemed unrealizable not long ago. Figure 6.4.1 shows the evolution of clock rates for Intel microprocessors. A high-end microprocessor recently demonstrated by Intel operates with voltages near 1 V

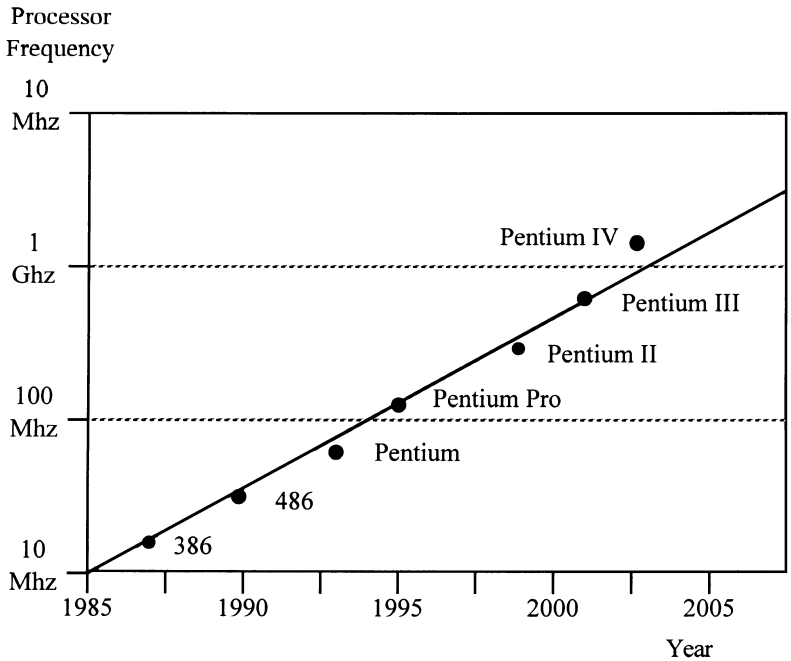


FIGURE 6.4.1 Clock rates of Intel Microprocessors. (From R. Ronen et al., "Coming Challenges in Microarchitectures and Architecture," 2001.)

and at power levels of over 100 W. That corresponds to currents of 100 A or greater. To respect how large this current is, not that most homes have 100 A or 200 A services to operate the entire house. Today's high-end microprocessors employ well over 100,000,000 transistors on a single integrated circuit not much larger than a few square centimeters. Internally, the architectures have become sufficiently sophisticated that the hardware/firmware providing operating system functions are a fundamental part of the microprocessor. The architectures themselves have become highly refined.

The IBM Power4 microprocessor's basic description illustrates this sophistication, With 170 million transistors, the Power4 microprocessor is a 64-bit microprocessor operating at frequencies greater than 1 GHz, using an eight-instruction-wide design with superscalar operation and out-of-order execution of instructions. More than 200 instructions can be pending completion in each of the two on-chip processor cores.

Programming these contemporary microprocessors requires sophisticated programming support tools and generally require that programs be written in higher-level programming language (such as C++). Programming basic programs in assembly language is virtually impossible.

In a very real sense, microprocessors have become ubiquitous throughout our technologically driven society. Dozens appear in an automobile, invisible to the owner but providing the intelligence and control that are at the heart of the advanced automotive systems. Portable video game players contain some of the most remarkable microprocessors available, supporting advanced visualization in the palm of your hand.

CMOS Technology

Digital circuits today are based on complementary MOS (CMOS), CMOS having emerged as a preferred technology for a variety of reasons, perhaps the most important being the low dc power dissipation provided. Despite the dominance of pure CMOS, a hybrid version of CMOS and bipolar devices (called BiCMOS) has been developed to provide the advantages of low power in CMOS with the capabilities of high speed in bipolar circuits. Such BiCMOS technologies are used, for example, in the Intel Pentium microprocessors.

Digital electronics used a standard +5 V supply voltage over several generations, driven in part by the earlier dominance of bipolar circuits. With the migration to a dominance of CMOS, lower voltage operation has been possible and supply voltages have been decreasing significantly over the past decade. Initially, voltages were decreased to 3.3 V, with subsequent advances leading to further reductions to voltages of about 1 V. These reductions in supply voltage provide substantial performance advantages. For example, the lower voltage leads to a lower power consumption per device, an important reduction for holding overall chip power dissipations within acceptable limits as the number of devices per chip has increased. In addition, the steady decrease in feature sizes leads to cases in which the electric fields within the IC increase if the voltages are not reduced. Reduction of supply voltage in combination with reductions in feature sizes has helped prevent excessive electric fields and their associated effects. Beyond the simple reduction of externally provided supply voltage, contemporary microprocessor chips have evolved to include sophisticated internal power management subsystems—generating internal voltages as needed as well as allowing selective turning off of various sections of the system to reduce power consumption. Although voltage levels have decreased over the past several years, today's advanced, high-performance microprocessors require remarkably high current supplies (several tens of amps up to 100 A or higher). These high currents present complications associated with voltage losses along power interconnections within the IC because of the nonzero resistance of those lines and voltage drops due to that resistance. Such complications have led to specialized metalization layers for distribution of power throughout the IC.

The trends toward very high performance microprocessors are quite visible through their impact on successive generations of personal computers. Less visible have been the dramatic advances in microprocessor based systems intended for embedded applications (e.g., microprocessors embedded in automobiles, in audio equipment, in toys, and so forth). Microprocessor architectures from earlier generations of microprocessors provide the core computing power for these embedded applications while the addition of a wide range of peripheral functions (including analog input and output) have provided essentially single-chip solutions for intelligent embedded systems. In contrast to high-performance desktop computers, which receive power from a wall plug, these embedded applications often receive their power from batteries. Minimization of power dissipation is essential to provide long battery life.

These two application areas have led to two distinct roadmaps for future generation technologies—one roadmap for the high-performance microprocessors and another roadmap for the battery-powered, lower-performance microprocessors. The roadmap generated at regular intervals by the Semiconductor Industry Associates (SIA) organization provides not only a perspective on the current state of the technologies but also on the planned progression of those technologies over the next 10 years. The roadmaps can be viewed at the SIA website.

General Overview

Contemporary microprocessors include many of the functions that in earlier generation computers were implemented on separate integrated circuit chips. In addition, their architectural designs have become highly sophisticated, supporting not only routine user programs but also the underlying operating system associated with the computers. To provide a basic overview of computer architectures, the basic architecture of earlier generation systems implementing the computer on one or more PC boards, rather than on a single integrated circuit, is used here. The reader is advised, however, that far more sophisticated systems have replaced these earlier generation systems.

Figure 6.4.2 illustrates a typical multichip computer based on the earlier generation Intel microprocessors and peripheral chips. The role of the various blocks is as follows.

Processor. The block labeled "processor" was one of the earlier generation microprocessors (e.g., 8088 and higher generations). The internal processor architecture is discussed in the next section. In general, the processor manages the operation of the entire system, communicating with the peripheral functions using a "system bus" shown as the horizontal line. This system bus includes parallel wires (address lines) to select the peripheral and an internal data storage item in the peripheral (or memory), parallel wires (data lines) to transfer data to and from the peripherals and memory, a read/write line to specify whether data are being transferred to or from the microprocessor, and interrupt line(s) allowing external peripherals to "interrupt" the currently

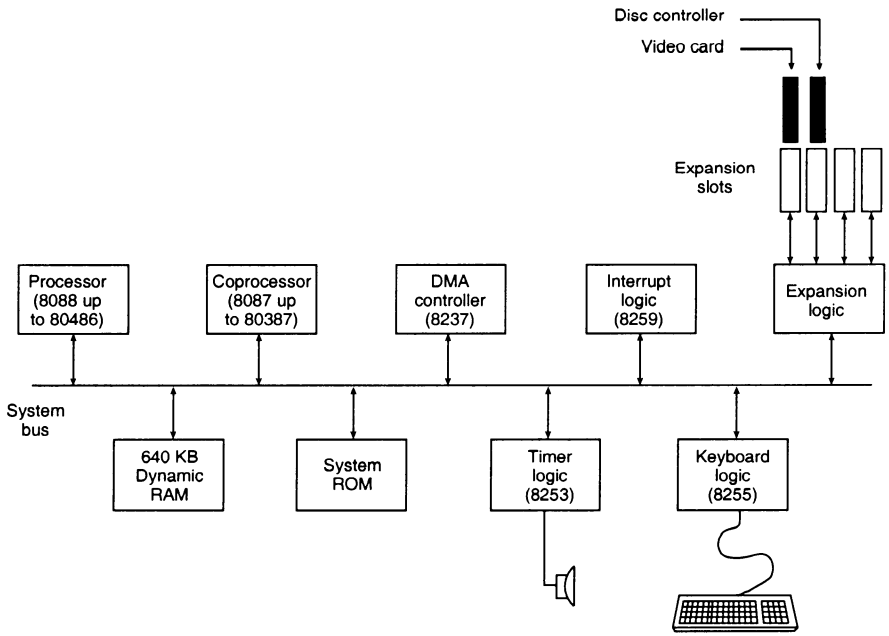


FIGURE 6.4.2 Block diagram of a typical PC motherboard.

executing program to allow the peripheral to communicate at its request with the microprocessor. The bus also includes other lines for functions such as providing resets and clocks.

Dynamic RAM. The dynamic RAM (640k shown, corresponding to the DOS system RAM capacity in those earlier computers) provides storage of both data and instructions. To access a particular item in the RAM, the address lines provide the location of the specific information involved in the data transfer. Today's RAM is far larger than the 640k shown, with standard personal computer memory capacities approaching 1 Gbyte.

System ROM. This nonvolatile memory provides the startup information needed to start the computer in a desired state. During startup, special logic in the processor accesses this startup information.

Disk Controller. The external disk hard drive stores user programs, operating systems, and other information, retained when the power is turned off. The disk controller manages the exchange of data between the microprocessor system (and its RAM) and the disk drive. There has been a substantial evolution in the philosophy and technologies of disk controllers over the past two decades, evolving from manufacturer specific controllers to standardized controller functions and data interfaces. Today's computer systems employ disk drives of substantial capacity (tens to hundreds of Gbyte capacity). The technologies used in contemporary disk drives have advanced dramatically in parallel with the advances in VLSI technologies, providing not only much larger capacities but also smaller and more rugged drives.

Cache Memory. Although the speeds of microprocessors have increased dramatically, memory components have seem a less dramatic increase. Memory access times (DRAMs) are typically several tens of nanoseconds. With microprocessor clock rates having increased to 1GHz and higher, a read or write of external memory can take the microprocessor about 50 to 100 clock cycles. Static RAMS have substantially faster response times, but have lower data storage capacities per IC. This has led to the introduction of "staged" memories, with instructions (and in some cases data) transferred from the slow DRAM into fast SRAM where the instructions (and data) can be retrieved more quickly. If an instruction is reused (and in the case of a loop instruction sequence) or data are reused (as in several computational algorithms), the second use of the instruction (data) is more quickly retrieved from the SRAM. Rather sophisticated algorithms are used to decide when information in a cache can be replaced by new information and to search the cache for instructions (content-addressable memories). Contemporary microprocessors have included one or more stages of cache within the IC. First-level caches are smaller than second-level caches, but significantly faster (two to three cycles for first-level cache versus six to 10 cycles for the second-level cache).

Video Card. The video card provides access to the display of the computer. As in all other components, video displays have seen dramatic improvements over the past 20 years, including not only the transition to high-resolution color displays but also to flat panel displays such as used with laptop computers. The expectations of the display drivers (hardware and software) have increased sufficiently that video cards continue to play a major role in the performance of the overall computer system.

Interrupt Logic. While the processor is in control, it determines to whom it will talk and is oblivious to any needs originating at a peripheral. For example, the serial port of the computer may receive data (e.g., through a modem) and the processor must be made aware that there are data to be read before the next data byte is received by the serial port. Interrupts provide the mechanism through which the peripherals can "interrupt" the processor's current activities and request that it handle requests originating in the peripherals. The "8259" shown for the interrupt logic is an classic integrated circuit included on computer board to interface interrupts from the peripherals for presentation to the processor. The interrupts ask the microprocessor to interrupt its operation and service the peripheral requesting service. The microprocessors allow external inputs to be turned on or off. When an interrupt occurs, the software program branches to a predetermined location where commands determine which peripheral generated the interrupt and then to a location where the program handles the needs of the interrupting peripheral.

Keyboard Logic. The "8255" component shown for interfacing the computer to a keyboard was also an integrated circuit designed specifically for this purpose.

DMA Controller. The direct memory access controller provides a high-speed means of streaming data to and from the computers RAM. When active, this replaces the normal single word at a time operations provided by the processor across the system bus.

Coprocessor. The capabilities of operations performed by the microprocessor is limited by the number of transistors available, leading to cases in which some desired operations on data that would normally be provided by the processor must be performed instead by an auxiliary integrated circuit. The classic example was floating point computations, since the standard microprocessor's arithmetic unit was designed to operate only on integers. Addition of the coprocessor allowed floating point instructions to be added to the instruction sets, the data being exchanged between the microprocessor and the floating point coprocessor (bypassing the internal arithmetic unit of the microprocessor) for fast hardware execution. The alternative is a slower sequence of instructions executed within the microprocessor to perform the floating point computations as a program. Like many of the other peripherals shown, such floating point coprocessors have migrated into the microprocessor IC. This has led to contemporary architectures with multiple arithmetic units and an ability to execute more than one arithmetic function at a time. This ability is actually employed, and contributes to the ability of microprocessors to execute (on average) one instruction for each clock period.

Timer. The timer function (shown as the earlier 8253 integrated circuit) allows a variety of timing functions to be performed. These include measuring time, generating timing pulses for various peripherals, and so forth.

Not Shown. Not shown in Fig. 6.4.2 are several other peripheral functions such as serial ports for modems and parallel ports for printers. Specialized integrated circuits were also developed to handle these peripheral ports and have evolved over time to include several types of peripheral ports (USB, Firewire, RGB TV, SCSI, and several others). These "standardized" interfaces allow components from different manufacturers to be used in combination with different computers and have played a substantial role in the availability of a rich set of reasonably priced peripherals.

Microprocessor Architecture

Next, the basic internal architecture of the processor shown in Fig. 6.4.2 is discussed. Again, it is necessary to review the general architecture from the perspective of earlier generation microprocessors because of the great complexity of today's high-end microprocessors. Figure 6.4.3 shows the general architecture of an elementary microprocessor. Such architectures are typical of the 8-bit microprocessors used in microcontrollers.

The microprocessor's internal bus is shown as an address bus and a data bus, extending to the external peripherals as shown in Fig. 6.4.2. The data bus "width" (number of parallel wires) defines the size of the data/instruction unit transferred into and out of the microprocessor. Typically, this data bus width also defines the type of microprocessor. An 8-bit microprocessor has an 8-bit data bus, a 16-bit microprocessor has a 16-bit data bus, and so forth. In the case of the microprocessor reading an instruction from the external memory, the data bus width also constrains the number of instructions available. The basic sequence of operations in performing an instruction includes first the reading of the instruction itself, then the reading of any constants associated with the instruction (e.g., numerical constants, address offsets, and so forth) and then, if the data for the instruction is stored in memory, the memory locations where these alterable data items are stored. With an 8-bit data bus, the first instruction item can represent up to 256 instructions, normally setting the limit on the number of instructions in the microprocessor's instruction set. With a 16-bit data bus, the first instruction can represent up to 64,000 instructions (allowing a rich instruction set or an organization of the instruction op code and operand information in a highly efficient manner, as in the Motorola 68000 processor family).

The width of the address bus defines the amount of memory data storage supported. In 8-bit and 16-bit microprocessors, the width of the address bus was twice that of the data bus. For example, an 8-bit microprocessor would have a 16-bit address bus, able to address $2^{16} = 65{,}536$ (equal to "64K" in computer jargon) memory locations. More memory can be addressed (as in the PC example in Fig. 6.4.2 using "memory banks," essentially an external selection of which set of a multiplicity of 64K memory units is accessible through the address bus. A 16-bit microprocessor with a 32-bit address bus is able to directly address $2^{16} = 4.3$ billion memory locations.

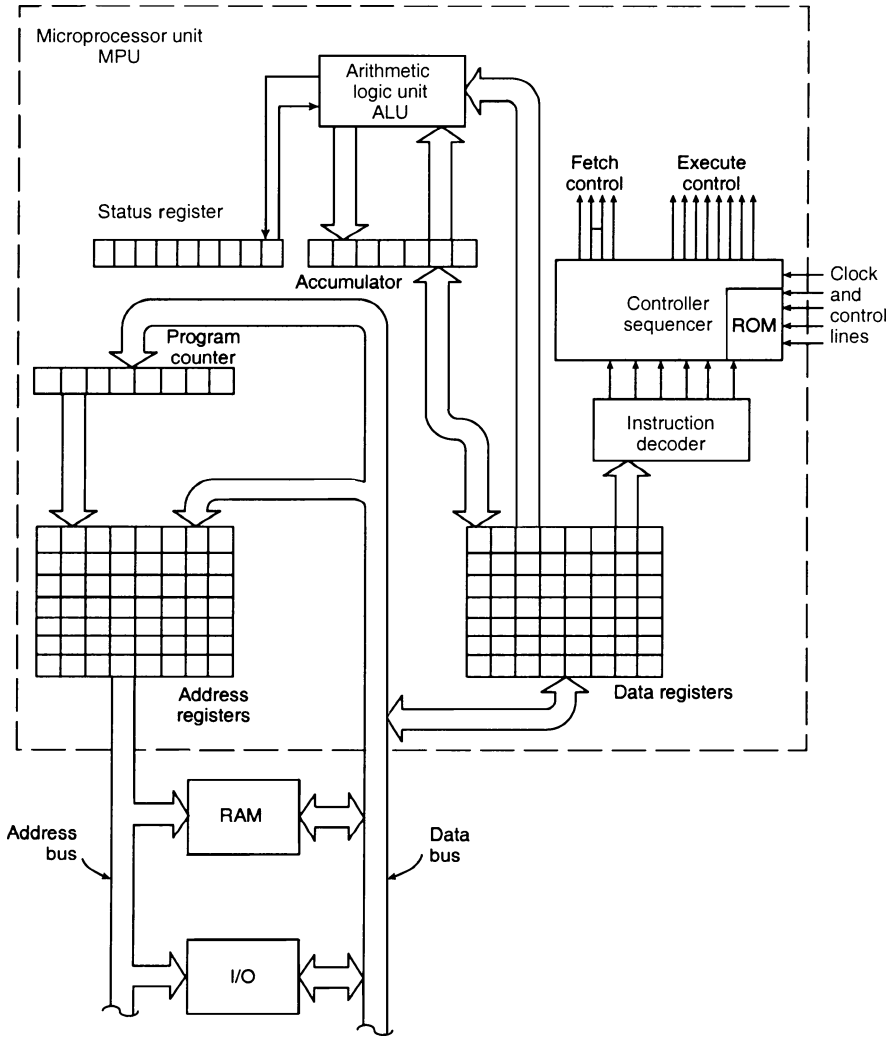


FIGURE 6.4.3 Architecture of an elementary microprocessor.

The internal logic units of the microprocessor shown in Fig. 6.4.3 provide the following functions.

Arithmetic Logic Unit. The arithmetic logic unit (ALU) can perform a variety of arithmetic and logic functions on data inputs (either on two input data items or on a single input data item). Standard arithmetic functions are add, subtract, multiply, and divide. Logic functions include the bit-wise AND, OR, XOR functions. These are also functions allowing the data word to be shifted in position. The specific operation to be performed is determined by input control signals generated by the controller. If a given ALU function requires more than one clock period (e.g., the multiply operation), then the controller/sequencer controls the multiple addition steps to complete the multiplication using a single multiply instruction.

Figure 6.4.3 shows a register receiving the output of the ALU and feeding that output back into the ALU during the next cycle. This provides, for example, an efficient means of adding a long string of numbers—this particular function leading to the name "accumulator" associated with the register.

Microprocessors initially used a single arithmetic unit. However, many programs would allow more than one arithmetic operation to be performed at the same time. By including multiple arithmetic units (perhaps different types of arithmetic units or other functions), more than one execution cycle can be launched at the same time. Such architectures are known as superscalar architectures and have become standard in 32-bit and higher microprocessors. The superscalar architectures, combined with high-performance cache memories and pipelined techniques can execute more than one instruction per clock cycle. For example, the Motorola PowerPC G4 microprocessor supports 1064 MB/s data rates and executes up to 16 simultaneous instructions per clock cycle.

Status Register. The status register is closely associated with the arithmetic logic unit and holds information regarding the data. This information is extracted by analyzing the data, deciding, for example, whether the number is zero, whether the number is negative, whether a carry or borrow state was generated from the high-end of the data in the operation, whether the result is an overflow (larger than the maximum number allowed), and so forth. This information is used most often in establishing the condition in conditional instructions (e.g., the assembly language equivalent of "if the number is negative then do this").

Instruction Decoder and Controller/Sequencer. The instructions are stored as binary data words (e.g., 8-bit data for an 8-bit microprocessor) in memory, using binary codes to represent the specific task to be performed in completing the reading of the instruction (if a multiword instruction) and carrying out the operation to be performed on the instruction. This binary code must be decoded and then provided to a sequential logic circuit that can generate the various electronic signals to control the various components within the microprocessor. For example, if the instruction says to add the number 10 to the number stored in the accumulator, then the first instruction defines the operation (addition) and the data to be used. The number 10 would be stored in memory along with the instruction so a second read of the memory would be required (for an 8-bit microprocessor). Following the reading of the number 10 into the microprocessor, the controller/sequencer would apply the two data items to the arithmetic unit and tell the ALU to perform an operation. Then, the controller/sequencer would transfer the result to the location specified in the instruction. Although sounding like a complex task, the controller/sequence can be efficiently implemented using efficient realizations for the basic set of stored information and feedback through registers to sequence the current state to the next state until the entire instruction has been executed.

Data Registers and Address Registers. Figure 6.4.3 shows two banks of registers, one the set of data registers and the other the set of address registers. The distinction between the data and address notations is largely based on the use of address registers to provide part of a memory address (e.g., the base address of a large table). Otherwise, the registers are similar and provide the function of "scratch pad" storage on the microprocessor chip. By taking data inputs to the ALU from these registers (and storing results from the ALU into these registers for later reuse), the delays associated with transferring data to and from external memory are avoided. As an example of the use of this scratch pad data storage, consider the following three instructions: (a) X = B + C, (b) Y = D + E, and (c) Z = X * Y. Upon completion of the first instruction, the value of X could be stored in data register D0. On completion of the second instruction, the value of Y could be stored in data register D1. Then at some later instruction, instruction (c) could be performed by retrieving the values of X and Y stored in the data registers D0 and D1. An example of using the address registers to generate addresses is illustrated by the example of addressing external memory to obtain successive components of a vector V(j). The address (base address) of the first component V(0) could be stored in address register A0. Then, the ALU/accumulator can be used to generate the sequence 0, 1, 2, ... By using the so-called indexed addressing mode, the address in A0 and the number from the ALU are added without having to perform an explicit addition instruction in the ALU.

Program Counter. The program counter is a counter that is automatically incremented after each instruction has been retrieved from the external RAM so that the number in the counter is the address for the start of the next instruction. This automatic incrementing reflects the typical case of instructions being executed in sequence. In the case when the program jumps to some other place, the offset between the current value of the program counter and the desired value of the program counter is added to the count in the program counter. Since the program counter always points to the next address, this automatically causes program execution to jump to the desired instruction in memory.

Included among the address registers discussed previously (or defined separately) is a specialized register called the "stack." This register performs a critical function when external interrupts occur or when the program calls a subroutine stored in some area of memory. Once the program jumps to the subroutine (or the "subroutine" used to respond to the interrupt), the microprocessor has lost the value of the next instruction that would have been executed (memory address following the current instruction). When such jumps occur, the location of the *next* instruction in sequence, in particular the address given by the program counter, is stored automatically in the memory location whose address is in the stack register and the contents of the stack pointer is either incremented (or decremented depending on the microprocessor family used). The address in the stack pointer at any time is the memory location in which the last item was stored, allowing that address to be used to retrieve that item. On retrieving an item, the address in the stack pointer is decremented (or incremented) so that the address points to the previous memory location. In this manner, the stack pointer operation implements in memory a "last in, first out" data structure. The memory structure is generally called a *stack*, suggesting a stack of addresses from which you retrieve the address at the top of the stack, exposing the address underneath. Return from the subroutine to the point in the program from which the jump was called is achieved simply by placing the address stored in the stack in the program counter.

Instruction Execution Sequence

The overall execution of an instruction involves a basic sequence of three steps, starting with the reading of the instruction from memory. The three steps are (1) the reading of the instruction, (2) the decoding of the instruction within the microprocessor, and (3) execution of the instruction (including the storage of the result of an instruction). Figure 6.4.4 illustrates this sequence of steps in combination with a clock signal driving the progression. The time from the start of the reading of the instruction (called "fetch" in Fig. 6.4.4) to the completion of the instruction including writing of any results into storage is called the instruction cycle time. The example illustrated in Fig. 6.4.4 is merely representative of several different approaches that can be used to arrange the orderly completion of these three basic steps. For example, in some 8-bit microprocessors, a single clock cycle is used to complete a single basic instruction. When the clock is in one state (e.g., high), the instruction information is fetched and when the clock signal is in the other state (e.g., low) the instruction is decoded and executed. There are also important cases in which a multiplicity of clock cycles are needed for some instructions. For example, a multiply operation in a microprocessor with only a addition-based ALU requires that a sequence of multiplies and adds be performed to complete the overall instruction. In this case, each addition operation will consume one or more clock cycles.

Figure 6.4.4 suggests that only one of the three operations can be performed at any given time. However, the fetch operation involves interfacing to the external memory (setting addresses and reading external data into the microprocessor), whereas the other operations are performed within the microprocessor. It is therefore possible to arrange for the fetching of the next instruction to be started while the decoding/execution stages of the current instruction are being completed. It is also possible to separate the decode and execution operations and perform them in parallel. For example, while executing the current instruction the decoder could be decoding the next instruction while at the same time the microprocessor is fetching still one more instruction. The approach is called "pipelining" (reflecting the output from one step being fed into the next step while a new action is being input to that first step). Pipelining provides a faster throughput of instructions (more instructions per second can be executed) but does not intrinsically reduce the time to execute a given instruction (i.e., pass a single instruction through the pipe). Figure 6.4.5 illustrates the basic principle, here taking the basic instruction cycle as having four phases—fetching an instruction (F), decoding the

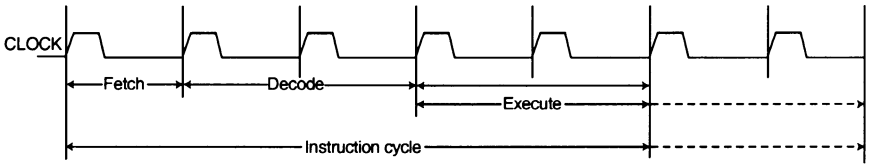
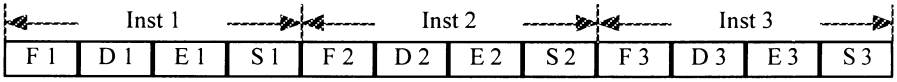
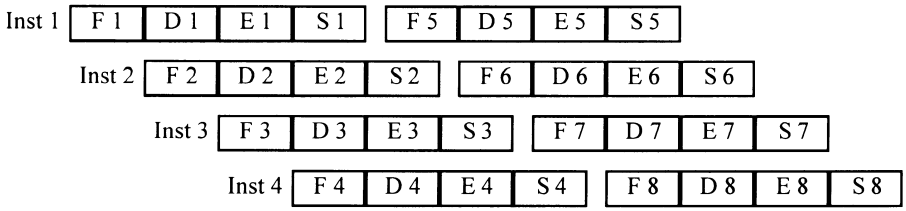


FIGURE 6.4.4 Instruction cycle timing.



(a) Sequential completion of each instruction in order.



(b) Pipelined execution (fetching next instruction while executing current instruction).

FIGURE 6.4.5 Instruction step sequencing: (a) nonpipelined; (b) pipelined.

instruction (D), executing the instruction (E), and storing the result (S). Figure 6.4.5*a* shows the operation without pipelining, each instruction starting after the previous has completed. In Figure 6.4.5*b*, each of the four steps can be performed in parallel, allowing a higher throughput rate for instructions. Optimizing the performance of a pipeline is complicated by the common use of branch instructions in programs. To handle such cases, sophisticated schemes to manage the pipeline have been developed. Modern high-performance microprocessors are characterized by an increasing number of pipeline stages (e.g., the Pentium Pro has more than 10 pipeline stages).

Serial Data Control

Microprocessors interface with a variety of devices that transmit and receive data in serial streams. The interface logic converts serial data to parallel data and vice versa. A programmable communication interface is used to handle this conversion and samples for data transmission, both synchronous and asynchronous.

Serial protocol involves three possible serial communication techniques. Two of the most widely used are *synchronous* and *asynchronous* serial communication. The third, *isosynchronous*, is a hybrid.

Asynchronous communication is best suited for data transmission between two devices where data are sent at low speed in intermittent, small groupings. A typical application is data entry through a CRT keyboard. Since the receiving device has no way of knowing when data will be sent, the asynchronous format requires *framing information* for each character transmitted. This enables the receiver, such as a universal synchronous-asynchronous receiver-transmitter (USART), to detect a valid data signal properly.

While the receiver waits on a dead line for possible transmission, it constantly samples for the leading edge of the start bit to occur. Sampling is performed by the USART much faster than the transmission baud rate (bit rate). For example, a receiver may sample the signal edge at eight times the baud rate. If the baud rate is 100 (100 bits/s), the USART would sample for the leading edge every $^1/_{800}$ s. Once the edge is detected, however, the receiver must ensure that it is receiving a valid signal and not a transmission caused by line noise. Upon detecting a possible start it is receiving a valid signal and not a transmission caused by line noise. Upon detecting a possible start bit, the receiver steps off a half-bit time to see that the bit is a logical 0. Once the start bit is detected, the receiver times a 1-bit time sampling for the remaining data and stop bits. Through this routine, the receiver synchronizes on each character transmitted.

In a synchronous protocol there is clocking between the two systems. The transmitter generates the clock and transmits data on the leading edge of the clock pulse. The receiver uses this clock to read in the serial data stream.

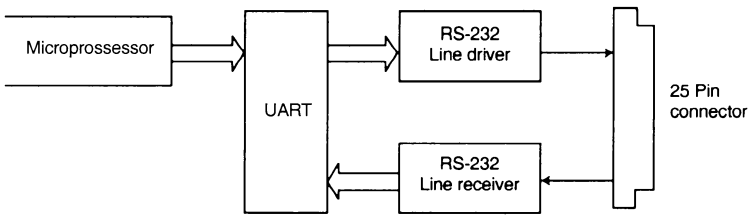
With synchronous protocol, bit synchronization is not necessary, as data are expected continuously. Still, the receiver must establish a reference indicating where data begin and end. With a synchronous protocol, characters are grouped into records, and *framing characters* are added to the record. These framing characters are SYN (or synch) characters. The USART uses the SYN characters to determine the boundaries of the message. SYN characters consist of a synchronization pattern, a pattern not likely to occur during a normal message. They are generated by the transmitter. The receiver samples these SYN characters bit by bit to establish its reference.

The isosynchronous format of serial communication retains the clock interconnect of the synchronous protocol, but it does not generate SYN characters. As with the asynchronous format, a start bit is generated. As the protocol uses clock synchronization, the repetitive samplings of the asynchronous format are eliminated. The hybrid protocol reduces the amount of MPU time devoted to message recognition, eliminates software overhead required by framing characters, and implements a greatly simplified protocol.

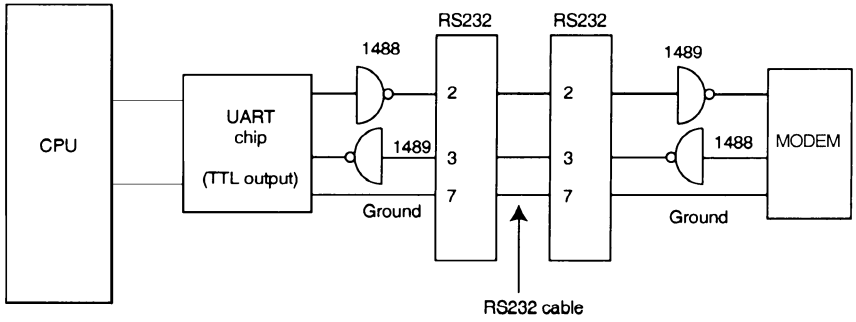
In receiving and transmitting the serial protocol in any form, the USART strips (or inserts) the framing characters and bits from the serial data stream and converts the data into a parallel format to be placed on the data bus. Figure 6.4.6 shows the elementary function of a serial I/O interface.

As far as the MPU is concerned, the USART consists of the data bus buffer, a control, and status registers. The receive data and transmit data buffers lie passively in the path of received and transmitted data and do not need direct access. The USART receives the serial-data-in signal and transmits the serial-data-out control signal. The MPU can service the transmissions on an interrupt-driven basis.

The rate of data transfer in serial data communications is stated in bps (bits per second). Another widely used terminology is baud rate. Digital systems normally are based on signals that can take one of two possible values (one corresponding to logic "1" and the other to logic "0"). However, it is also possible to allow a signal to take on a larger number of discrete values. In such cases, each signal value represents more than a single binary bit. For example, if a signal amplitude can assume one of four (eight) different values, each signal amplitude received represents two (three) bits. The baud rate is the number of signal values transmitted per second. If the baud rate is constant, then the number of bits per second transmitted increases as the number of



(a)



(b)

FIGURE 6.4.6 Generalized serial port: (*a*) using RS-232 interface; (*b*) using voltage level converters such as MC1488 and MC1489 chips.

TABLE 6.4.1 RS232 Comparison with RS422 and RS423

	RS232	RS422	RS423
Maximum cable length (ft)	50	4000	4000
Maximum speed (baud)	20K	10M/40 ft	100K/30 ft
		1M/400 ft	10K/300 ft
		100K/4000 ft	1K/4000 ft
Logic 1 voltage level	–3 to –25	A > B	–4 to –6
Logic 0 voltage level	+3 to +25	B > A	+4 to +6

signal levels increases. Serial data communications usually is designed for transmission over as long a distance as possible, with the result that there is significant attenuation (and noise) associated with the received signal level, causing transmission of binary signals to be more robust. This dominance of binary signal transmission has led to the terms *bps* and *baud rate* often being used (though incorrectly) as equivalent.

The serial port of a personal computer is a familiar serial data transmission port. The RS232 standard is common, with characteristics shown in Table 6.4.1. Notice that the voltage levels representing logic 1 and 0 levels are not conventional digital logic levels but rather positive and negative voltages, respectively. These voltage levels are one of the reasons why power supplies for personal computers must provide +/– 12–15 V in addition to the standard digital logic supply voltages. The RS232 standard allows serial data transmission over a modest distance at a modest rate. Table 6.4.1 also shows the characteristics of two extensions of the RS232 standard, both providing substantially higher data rates (or modest data rates over very long distances).

The serial port evolved from peripherals designed to connect a computer to the telephone network through a modem (modulator/demodulator). Table 6.4.2 shows the pins of an RS232 cable using a 25 pin D-type connector. Normally, only a subset of the control signals shown are used, allowing a smaller number of cable pins. Table 6.4.3 shows the pins of a 9-pin serial connector in which the common signals are retained. These tables show that the RS232 connection provides two serial data streams, one transmitted data and the other data being received. In addition, there are a number of control signals that manage the exchange of data between the computer and the peripheral. Because of the origins in connecting computers to telephone networks, terminology related to that connection are still used today. The connectors on opposite ends of an RS232 cable are not equivalent. For example, if one connector is configured according to the connector pins listed in Table 6.4.3, then the other end will require that its transmit data be placed on pin 2 (for reception by the far end) and that its receive data be taken from pin 3 (from transmission by the far end). Control signals also show different pin connections on opposite ends of the cable. The two ends are distinguished by the terminology DTE (Data Terminal Equipment, e.g., the computer) and DCE (Data Communications Equipment, e.g., the connection to the telephone wire). RS232 ports have evolved to provide connections to a wide range of peripherals, well beyond the single example of a modem. Often, the full set of control signals is not required. In cases not requiring end-to-end control signals, only the R × D, T × D, and GND signals shown in Tables 6.4.2 and 6.4.3 are needed.

TABLE 6.4.2 RS232 Pins

Pin	Description
1	Protective ground
2	Transmitted data (T × D)
3	Received data (R × D)
4	Request to send (RTS)
5	Clear to send (CTS)
6	Data set ready (DSR)
7	Signal ground (GND)
8	Data carrier detect (DCD)
9/10	Reserved for data set testing
11	Unassigned
12	Secondary data carrier detect
13	Secondary clear to send
14	Secondary transmitted data
15	Transmit signal element timing
16	Secondary received data
17	Receive signal element timing
18	Unassigned
19	Secondary request to send
20	Data terminal ready (DTR)
21	Signal quality detector
22	Ring indicator
23	Data signal rate select
24	Transmit signal element timing
25	Unassigned

The RS232 serial port generally transmits ASCI characters. The relationship between the number code (7-bit binary) and the ASCI

TABLE 6.4.3 IBM PC 9-Pin Signals

Pin	Description
1	Data carrier detect (DCD)
2	Received data (R × D)
3	Transmitted data (T × D)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready (DSR)
7	Request to send (RTS)
8	Clear to send (CTS)
9	Rind indicator (RI)

characters is shown in Table 6.4.4. There is also an 8-bit code, including several foreign language characters.

The standard printer interface on desktop computers transmits data in parallel, rather than serial. Figure 6.4.7 shows the basic printer cable's connector. There are two data rate links, one an 8-bit data link from the computer to the printer and the other an 8-bit data link from the printer to the computer. Control signals in this case reflect the handshake between the computer and the printer.

Recently, there has been an emphasis on serial data communications technologies capable of handling a far wider range of peripherals than can be handled by the low data rate of RS232 (and its extensions). USB and Firewire are two examples, both allowing connections to not only modems and printers but also to cameras, hard disk drives, and so on. These newer serial data technologies provide high data rates and allow a single serial data port to be connected to a multiplicity of peripherals. These newer

TABLE 6.4.4 The ASCII Code

DECIMAL	HEX	ASCII	DECIMAL	HEX	ASCII	DECIMAL	HEX	ASCII	DECIMAL	HEX	ASCII
0	00	NUL	32	20		64	40	@	96	60	*r*
1	01	SOH	33	21	!	65	41	A	97	61	a
2	01	STX	34	22	″	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	’	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	–	127	7F	DEL
127	7F	DEL									

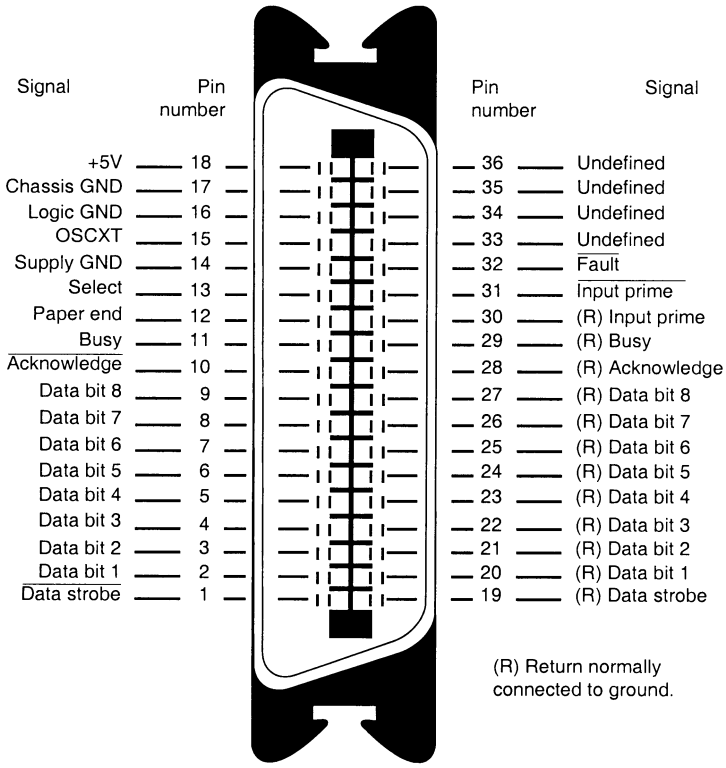


FIGURE 6.4.7 The Centronics parallel printer interface using a 36-pin connector.

serial data communication technologies provide significantly smaller connectors and smaller diameter cables, well suited for today's laptop computers but also providing improved performance for desktop computers.

Disk Interfaces

The evolution of interfaces to hard disk drives illustrates the major impact that standardization plays in the area of computer systems. Early computer manufacturers provided both the computer and the hard disk drive, using a proprietary interface requiring that the disk drives be obtained from the same manufacturer as the computer. In those earlier computers, most of the functional tasks required to read data from and write data to a hard disk drive was managed by a disk controller located in the computer itself, the disk drive having little built-in intelligence. As VLSI advanced, it became possible to place much of the functionality needed to manage the reading and writing of data in the hard disk drive unit itself. Once this responsibility was separated from the computer, hard drives could be purchased from a number of manufacturers, so long as the cable connection between the computer and hard drive was satisfied. This drove the evolution to interface and cable standards to connect hard drives to computers, allowing the disk drive market to emerge and provide drives for virtually any computer with the standardized interface.

Standard disk drive interfaces developed to support the Intel-based PC included the ESDI (Enhanced Small Device Interface) and IDE (Integrated Device Electronics). The SCSI (Small Computer System Interface) standard (pronounced "scuzzy") emerged for general computer use, not tied to any particular computer platform. The standards have had to adapt to the remarkable increases in disk drive capacity and data transfer rate,

leading to the highly sophisticated hard disk drives in use today. For example, the SCSI standard has passed through several generations, successive generations providing higher data rates and less bulky connectors. These interfaces are used not only for hard disk drives but also for tape drives, CD drives, and DVD drives.

These disk drive interface standards generally require rather bulky cables, limiting their extension to other peripherals such as digital cameras. The need to support a wide range of external devices has stimulated the development of less bulky but high data rate interfaces such as USB (modest data rate) and Firewire (high data rate), signaling a new generation of standardization of cables and connectors for personal computer use as the variety of peripherals expand.

BIBLIOGRAPHY

Antonakos, J. L., "The Pentium Microprocessor," Prentice Hall, 1997.

Ayala, K., "The 8086 Microprocessor: Programming and Interfacing the PC," West, 1995.

Bakoglu, H., and T. Whiteside, "RISC System/6000 Hardware Overview," IBM RISC Technologies, IBM, 1990. SA23-2619.

Bartee, T., "Computer Architecture and Logic Design," McGraw-Hill, 1991.

Carter, J. W., "Microprocessor Architecture and Microprogramming," Prentice Hall, 1995.

Gibson, V., "Microprocessors: Fundamental Concepts and Applications," Delmar, 1994.

Gilmore, C., "Microprocessor Principles and Applications," McGraw-Hill, 1989.

Goody, R., "Intel Microprocessors," Glencoe/McGraw-Hill, 1992.

Hall, D., "Microprocessors and Interfacing," 2nd ed., Glencoe/McGraw-Hill, 1991.

Hennessy, J. L., and D. A., Patterson, Computer Architecture: A Quantitative Approach, 2nd ed., Morgan Kaufmann, 1996.

Hester, P., "RISC System/6000 Hardware Background and Philosphies," IBM RISC Technologies, IBM, 1990.

Horwath, R., "Introduction to Microprocessors Using the MC6809 or MC6800," McGraw-Hill, 1992.

Kleitz, W., "Digital and Microprocessor Fundamentals," Prentice Hall, 1990.

Leventhal, L., "Microcomputer Experimentation with the Intel SDK-86," BBS College, 1987.

Mazidi, M., and J., "The 80 × 86 IBM PC and Compatible Computers," Vol. I and II, Prentice Hall, 1995.

Mazur, H., "The History of the Microcomputer—Invention and Evolution," *Proc. IEEE*, Vol. 83(12), pp. 1601–1608, 1995.

Mueller, S., "Upgrading and Repairing PCS," 4th ed., Que, 1994.

O'Connor, P., "Digital and Microprocessor Technology," 2nd ed., Prentice Hall, 1989.

Pack, D. J., and S. F., Barrett, "68HC12 Microcontroller," Prentice Hall, 2002.

Putnam, B. W., "Digital and Microprocessor Electronics: Theory, Applications, and Troubleshooting," Prentice Hall, 1986.

Ronen, R. A., Mendelson, K. Lai, S-L. Lu, F. Pollack, and J. P., Shen, "Coming Challenges in Microarchitectures and Architecture," *Proc. IEEE*, Vol. 89 (3), 2001, pp. 325–340.

Smith, A. J., "Cache Memories," ACM Computer Surveys, Vol 14(3), pp. 473–530, 1982.

Thompson, A., "Understanding Microprocessors: A Practical Approach," Delmar, 1994.

Treibal, W., and A. Singh, "The 8088 and 8086 Microprocessors," Prentice Hall, 1992.

Uffenback, J., "The 8086/8088 Family, Design, Programming and Interfacing," Prentice Hall, 1987.

Urganiak, K., "Experiments for the Intel 8088," Delmar, 1994.

Valvano, J. W., "Embedded Microcomputer Systems," Brooks/Cole, 2000.

Yeager, K., "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, Vol. 16(4), pp. 28–40, 1996.