MICROCOMPUTERS

A microcomputer is a small, inexpensive computer that contains a single-chip processing unit called a microprocessor. Another name for a microcomputer is personal computer (PC), reflecting the fact that microcomputers are designed to be used by one person at a time. A microcomputer is a general-purpose computer, meaning it can be programmed to perform a wide range of computational tasks, and has low to moderate processing power. Laptop and notebook computers are two types of portable microcomputer.

In contrast to microcomputers, workstations and servers (formerly called minicomputers) are more powerful and more expensive. These systems use more circuitry to implement the central processing unit (CPU) and other subsystems, and have higher capacities for moving and storing information. These midrange computers are designed to support one or two users that have high computational requirements, or several users with moderate requirements. Two still more powerful classes of computers are supercomputers and main-frames. Supercomputers are designed to support the very highest requirements for computational power, while main-frames are designed to support many users simultaneously.

At the other end of the computational spectrum are computing devices with less power than microcomputers. These also use microprocessors to perform computation, but may have limited or no general-purpose programmability and have fewer peripheral devices with which to access and store data. Graphics terminals, network computers, and palmtop computers are examples of such devices.

TYPICAL MICROCOMPUTER SYSTEM

Like all computers, a microcomputer consists of electronic circuitry along with a variety of physical devices used to store, display, and move information from one place to another. Collectively, these components comprise the hardware. Microcomputer hardware consists of three main subsystems: (1) the processor and (2) memory, which comprise the central electronics, and (3) the input/output (I/O) subsystem composed of the peripheral electronics (adapters) and devices (see Fig. 1).

The memory stores information, both programs (code) and data. Programs are sequences of instructions that specify some desired behavior for the computer. In general, that behavior involves moving data into the computer, manipulating it in some fashion, and moving the results back out of the computer. The processor comprises a single integrated circuit (IC), or chip—the microprocessor. It is responsible for fetching instructions out of memory and executing them. The processor instructions specify particular operations to be performed on data held in the processor or in memory. The I/O subsystem provides the means for moving data into and out of the computer, under control of the processor. The processor, memory, and I/O are connected together by busses that provide pathways for the movement of data among the subsystems.

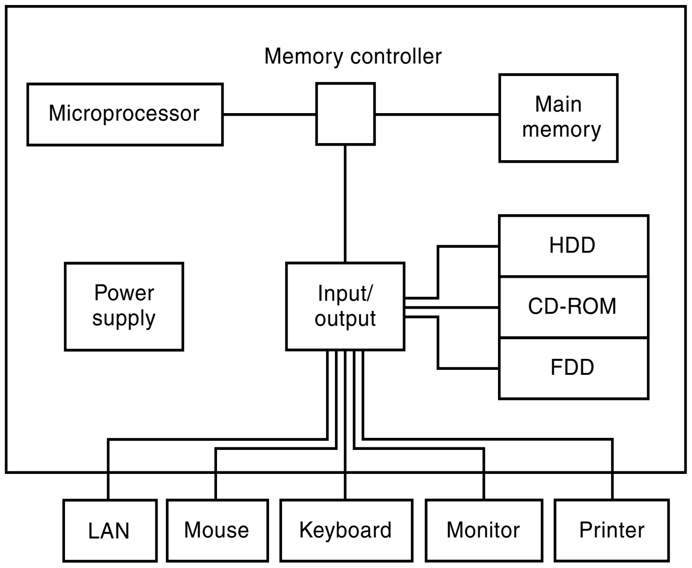

Figure 1. The hardware inside a typical microcomputer system includes the central electronics, the peripheral electronics, some peripheral devices, and the power supply. The central electronics consist of the microprocessor and main memory. The peripheral electronics control the I/O devices. The memory controller is responsible for communications among the subsystems. Devices commonly built into the enclosure include a hard disk drive (HDD), a floppy disk drive (FDD), and a compact disk read-only memory (CD-ROM) player. Other devices connected through external cables include a local area network (LAN), a mouse, a keyboard, a monitor, and a printer.

Stored on peripheral devices and in electronic memory is information, in the form of instructions and data, which control the behavior of the physical components. This stored information is called software. When it is being moved from one place to another or stored, the term data refers to any kind of information, including instructions. When being contrasted with instructions, the term data refers to the information that is manipulated by the instructions.

Inside the Box

Most of the electronics that implement the various subsystems are contained in a single enclosure. These consist of various components, such as transistors, capacitors, resistors, and integrated circuits, mounted on printed circuit boards (PCB) that are attached to one another by connectors and cables. The core electronics—processor, memory controller, standard peripheral adapters—are typically mounted on a single large PCB called the motherboard. Also mounted on the motherboard are several different kinds of connectors, allowing other components to be installed in the system as needed.

For example, memory chips are mounted on one or both sides of small PCBs called single inline memory modules (SIMM) or dual inline memory modules (DIMM), respectively. These memory modules fit into the memory connectors on the motherboard. DIMMs provide a data width that is twice that of SIMMs. By choosing to install cheaper memory modules with low storage capacity or more expensive memory modules with higher storage capacity, the microcomputer can be configured to fit the needs of the user.

Similarly, while the core electronics on the motherboard provides support for basic input/output devices, peripheral adapter cards can be installed in corresponding connectors on the motherboard. These support additional functionality, including graphics displays, local area networks (LAN), hi-fi sound, and external storage devices.

Also packaged in the enclosure is a power supply that develops the required voltages for the different components, fans to dissipate heat from the ICs, and built-in peripheral devices, such as disk drives.

Instruction Execution

Computers use sequential electronic circuits to perform operations as specified by the software. In sequential circuits, a clock signal defines the beginning of each processing cycle. The state of processing, the information associated with the progress of a computation, can only change from one cycle to the next, not within a given cycle. A faster clock allows more computation to be performed in a given amount of time. Clock speed is measured in hertz (Hz), a unit of measure equal to one cycle per second. A microcomputer driven by a 400 MHz (megahertz) clock can change computational states 400 million times each second—once every 2.5 ns.

The basic unit of computation in any particular microprocessor is an instruction. A given microprocessor has a defined set of instructions that it can execute. The overall behavior of the microcomputer is defined by the sequence of instructions—the program—that it is executing. When a program is being executed, its instructions and data are stored in memory. The microprocessor contains circuitry to fetch each instruction from memory, fetch any data needed by the instruction from memory, execute the instruction, and put the results of executing that instruction back in memory. Different instructions take varying amounts of time (numbers of cycles) to execute. An indicator of the relative processing power of two microprocessors within a family (executing the same instruction set) is how many million instructions per second (MIPS) they can execute. To compare microprocessors from different families, execution time for certain standard applications, called *benchmarks,* can be used.

Data Storage

Computers manipulate digital information. A digital representation of a value is discrete, meaning it can take on only a fixed number of possible values. The basic unit of digital representation is the bit, which can have a value of 0 or 1. Combinations of bits can be used to represent larger values. For example, eight bits can be used to represent a value from 0 to 255. Eight bits is a standard unit for representing information in computers, and so has its own name, the byte. Storage capacities for memories and disk drives are usually expressed in megabytes (Mbyte—millions of bytes) or gigabytes (Gbyte—thousand millions of bytes). Transfer speeds for data are usually expressed in Mbyte/s, or in the case where data is transfered serially, a single bit at a time, the unit bits per second (bit/s—often referred to as the baud rate)—or kilobits per second (kbit/s—thousand bits per second) is used.

As it is being manipulated, the information in the computer, both code (instructions) and data, is stored in a variety of ways until needed. The processor itself stores the information for which it has an immediate need in registers. Main memory stores the code and data for the currently active program(s) so that the processor can access it. Main memory also contains the operating system (see below) along with a variety of data structures (organized collections of data) maintained by the operating system to keep track of the overall state of the microcomputer. Programs and data that are not currently active are stored on various peripheral devices, such as disk drives, CD-ROM, and tapes. When needed, these data are copied from the peripheral device to main memory, and if new data is generated, it may be copied from main memory back to a (writeable) peripheral device.

Different storage devices exhibit different combinations of several characteristics that are important to the proper functioning of the microcomputer. First, a storage system may allow only sequential access or it may be a random-access system. In the first case, the individual storage elements can be read or stored only in a particular order, while in the second case any order is allowed. Second, a storage system may be read-only or it may be writeable (read-write). In the first case, the information that is stored can never be changed, while in the second case, new information can replace the current data. Third, a storage system may be volatile or nonvolatile. Volatile memory loses its information when power is turned off, while nonvolatile memory maintains its information in the absence of power.

The memory subsystem is organized hierarchically, using fast, expensive, low capacity devices that are directly accessible to the processor, and successively slower, less expensive, higher capacity devices as that access becomes more remote. Main memory is composed of several different types of IC memory, including two kinds of random-access memory (RAM)—static (SRAM) and dynamic (DRAM)—as well as read-only memory (ROM).

Flow of Information

To be useful, a computer must manipulate data that come from outside the system itself. Similarly, it must be able to make the results of its computations known to the external world. The various systems that provide the data input and output functions to the central system (processor and main memory) are called peripherals. Each peripheral consists of the device itself, which is generally an electromechanical system that originates input, accepts output or stores data, and an adapter, which is an electronic component that allows the processor to control the device.

A basic user interface to the microcomputer is provided by the keyboard and monitor. The keyboard is an input device that allows the user to type information—commands, programs, text, numeric data—into the microcomputer. The monitor is an output device that displays information generated by the microprocessor in a user-readable form. A basic monitor might display only alphanumeric characters in fixed rows and columns on its screen; more typically information is displayed in a graphical form. The monitor itself may be either a cathode ray tube (CRT), like that in a television set, or, particularly for portable computers, it may be a liquid crystal display (LCD) flat panel. Another input device, the mouse, provides a means of pointing to graphical objects displayed on the monitor screen. In addition to the user interface, a hard disk drive (HDD), floppy disk drive (FDD) and compact disk read-only memory (CD-ROM) player are commonly used to load programs into memory.

Microcomputers can be configured with a variety of other peripherals to provide better functionality or performance. For example, alternative pointing devices include joysticks, trackballs, and tablets. Output devices for producing hardcopies (images on paper) of text and figures include printers and plotters. Input devices for capturing image data include scanners and digital cameras. Input/output devices for connecting to other computers include modems and network controllers. Input/output devices for processing sounds include microphones and speakers as well as musical instrument digital interface (MIDI) and other digital audio devices.

Software

The microprocessor gets work done by following sequences of instructions that specify how to access and manipulate particular sources of data to accomplish desired tasks. The term program is used to describe the set of instructions that performs a particular task. The term code is also often used to distinguish instructions from the data they manipulate.

Two main classes of software are system software and application programs. System software includes the base operating system (OS), device driver code that provides an interface between the OS and each peripheral component, library code that serves as an interface between the OS and an application, and the boot code that is responsible for initializing the computer when it is first turned on.

Application programs are designed to perform some particular task for a user. Applications commonly found on microcomputers include programs for word processing and spreadsheets, publishing and presentation, web browsing and e-mail access, bookkeeping and games, as well as accessory and utility programs. Accessories—applications that remain in memory for ongoing use—include clock, calendar, and calculator programs. Utilities—applications that perform maintenance functions—include antivirus and file-compression tools.

To execute an application program, or any other software, it must first be copied from a peripheral device into main memory. The processor is then given the memory address where the first instruction of the application is stored, and program execution begins. The operating system has the task of loading applications, as directed by the user, and then supporting the execution of each application in a number of ways. The OS manages the allocation and security of microcomputer resources such as processor time, memory space, and access to peripherals. It also provides a set of services that allow applications programs to access these resources through simple procedure calls which hide the complexity of the hardware details from the application. In this way, the OS mediates the execution of the application on the particular microcomputer hardware.

MICROCOMPUTER HARDWARE

The microprocessor is the principal component in a microcomputer. All other components are designed to support the efficient operation of the microprocessor. The peripheral subsystem transfers data to and from outside sources to be used by the processor, while the memory subsystem provides a staging area for those data on their way to and from the processor.

Memory Subsystem

The memory subsystem is used to store programs, and the data that are manipulated by the programs, so that the processor can have direct access to them. At any given time, main memory may hold the operating system, including device drivers, dynamic libraries, and tables of configuration and status data, and one or more application programs, including instructions and several areas used to store program data. Whenever the need for main memory space exceeds the available capacity, some contents are copied to backing store (hard disk) temporarily. This costly operation can be minimized by having a large-capacity memory.

The majority of main memory is implemented as random-access memory (RAM), using a technology called dynamic RAM (DRAM). The advantage of DRAM memory is that each unit of storage, or bit cell, is small, and so a high capacity can be achieved with a few ICs. One disadvantage of the small cell is that the stored information must be periodically (dynamically) rewritten into the cell in order to persist. The other disadvantage of DRAM is that it has a slow access time, meaning that there is a significant delay from the time data are requested to the time they are available.

A faster but less dense RAM technology is static RAM (SRAM). This type of RAM is used to implement a smaller-capacity memory called cache memory. Cache memory is placed between the processor and main memory, and holds a copy of some of the information stored in main memory. Since not all of main memory can be cached, some means is needed to decide what should be stored in the cache at any given time. While there are many answers to this question of how to manage the cache, they are all based on the fact that memory access patterns exhibit locality rather than randomness.

For example, if a particular piece of data has recently been accessed, there is a high probability that it will soon be accessed again. This behavior is referred to as temporal locality. Similarly, if a particular piece of data has recently been accessed, there is a high probability that another piece of data stored at a nearby address will be accessed soon. Thus, memory access patterns are said to exhibit spatial locality. Based on locality, the guiding principle for cache management is to retain in cache a copy of any block of data containing an element that has recently been accessed.

Most microprocessors today have a relatively small cache memory on the chip itself. On-chip caches, called level one (L1) caches, range from 8 kbyte to 64 kbyte while main memories are roughly 1000 times larger. In many cases, an additional level of memory is placed between the on-chip cache and main memory. This level two (L2) cache has characteristics somewhere between those of L1 and main (L3 in this case) memory. L2 is slower to access than L1, but faster than L3, and its size may be 10 to 100 times larger than the L1 cache.

Processor Subsystem

The microprocessor chip contains the electronics for the processor and the L1 cache. For the processor itself, there are two main tasks: fetching instructions and data into (and writing data out of) the processor, and executing those instruc-

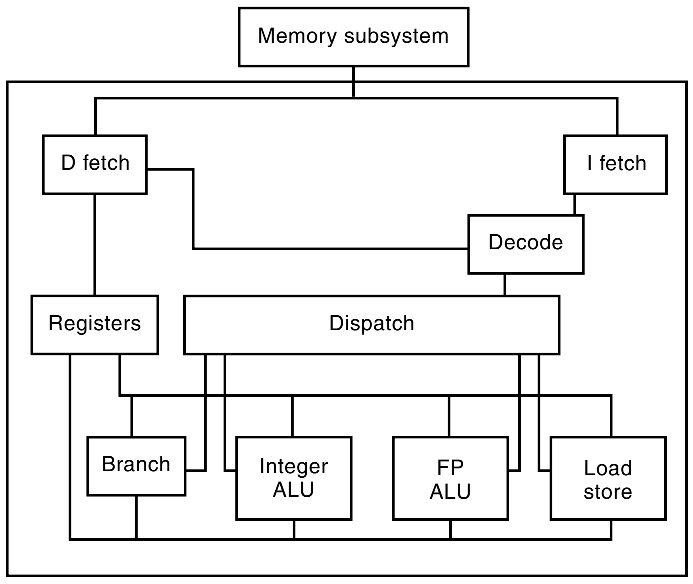

Figure 2. Two main tasks performed by the microprocessor are fetching of instructions and data, and executing instructions. The execution sequence starts with fetching the next instruction from memory (I fetch) then decoding the instruction (Decode) and fetching operand data (D fetch). Once operands are available, the instruction is dispatched (Dispatch) to one of the execution units (Branch, Int ALU, FP ALU, or Load Store). The result is stored back in the registers.

tions. Figure 2 shows the major hardware units in the processor that support these activities.

Registers are storage elements that hold operands and temporary results of computations. These storage elements are referenced in the instructions, and accessed directly by the execution units, providing fast and predictable access times compared to the slower and more variable times required to access memory. In some microprocessor designs operands are required to go through the registers prior to execution, while in other designs operands can be retrieved directly from memory.

Computer memory is organized as an array of storage elements, each of which is identified by its location in the array, referred to as its address. Instructions to be executed in sequence are stored at successive locations in memory. A branch instruction at the end of such a sequence indicates the starting address of the next sequence of instructions to be executed. To execute a given instruction, the following sequence of operations must be performed by the processor: instruction fetch, instruction decode, operand fetch, execution, operand store.

Instruction fetch involves the determination of the next instruction address, followed by a request to memory for that instruction. Once the instruction is in the processor, it can be decoded. Instruction decode involves the determination of the instruction type, and identification of operands (data) that the instruction operates on. The instruction type determines which of the execution units will be used to process the instruction. Prior to executing the instruction, its operands must be made available.

Once all operands are available, the instruction is executed. The execution portion of the processor is generally partitioned into separate computational units corresponding to the different instruction types. For example, fixed-point or integer arithmetic and logical operations would be performed in one unit; floating-point arithmetic, used to manipulate noninteger operands, in another. A separate unit might be used for data movement operations, and another for instructions that change the flow of instructions to another sequence. After the instruction has been executed, the result of any computation is stored back to a register or to memory.

To perform useful work, data from outside the microcomputer must be manipulated. There are two ways for the processor to access peripheral devices. Some microprocessors have instructions specifically for I/O operations. The instruction specifies which I/O device is being accessed and what type of operation is to be performed. If the operation involves a transfer of data, the data are then moved between a register and the I/O device. A second way to perform I/O operations is to allocate a block of memory addresses for use by I/O devices. In this memory-mapped I/O method, each device has one or more control and data registers accessed using an address in the block. A normal instruction that reads or writes memory can then be used to access the I/O device using the appropriate address.

I/O Subsystem

The I/O, or peripheral, subsystem is a heterogeneous system of busses, controllers and devices, whose characteristics vary according to the access times and bandwidth (rate at which data are transferred) requirements associated with different types of input and output devices. A peripheral adapter for each device is attached to the system by a bus, providing a data and command path back to the processor. The adapter controls the operation of the device, and enforces the bus protocol (the rules that define correct use of bus control signals) for transferring data between the device and the central system.

The user interface, consisting of the monitor, keyboard, and mouse, exhibits different bandwidth requirements for input and output. On the high end of the spectrum is the graphics video display. The amount of information displayed at one time on this output device depends on the number of picture elements (pixels) used to fill the screen, and the number of bytes used to represent the color or intensity of each pixel. A 640×480 pixel display that uses three bytes of data per pixel to specify its color requires over 900 kbyte of data per screen image. To support video playback at 30 frames per second, the bandwidth requirement is over 27 Mbyte/s. At the low end of the spectrum are the mouse and keyboard. A user typing on the keyboard at a relatively high rate of 80 words per minute will require a bandwidth of less than 10 byte/s to keep pace with this input device.

Another key I/O device is the hard disk drive. The hard drive is both an input and an output device that stores information on a spinning magnetized disk. It stores programs and data that can be copied into memory for execution and manipulation, and it stores data that have been generated by programs and then copied from memory. Hard drives can also be used to temporarily store data from memory when the memory capacity would otherwise be exceeded. The hard drive is then an extension of the memory hierarchy, and referred to as backing store.

Other I/O devices used to store programs and data include the floppy disk drive, the compact disk read-only memory (CD-ROM) player, and magnetic tape drives. Floppy disks use nearly the same technology as hard disks, except that the magnetized disk is nonrigid. Floppy disks are slower and have less storage capacity than hard disks, but are less expensive and are removable, providing a portable medium for data storage. Removable hard disks having higher capacity and higher cost are also available. CD-ROMs store data optically, rather than magnetically, but otherwise are a form of spinning disk storage. The optical technology prevents the disk from being rewritten with new data, so the CD-ROM player is strictly an input device. Magnetic tape was once used in microcomputers to store programs and data that could then be copied to memory for use. It has a low cost per unit of storage, but is slow and requires sequential access. It is now used for archiving infrequently used data and for hard drive backup—storing a copy of hard drive data in case the hard drive experiences problems.

Some other common peripheral devices found on microcomputers are modems, LAN controllers, sound cards, and printers. A modem uses a serial—one bit at a time—data path to transfer data over phone lines, providing a connection to other computers (and to FAX machines). Data compression is used to achieve a higher bandwidth than phone lines would otherwise support. A LAN controller is used to transfer data over a local area network, such as ethernet or a token ring, also providing a connection to other computers. These network connections allow one microcomputer to share data or to receive services from other computers.

Printers are attached to the microcomputer via a standard interface called a parallel port. Dot-matrix printers represent a low-quality technology that has now been replaced by laser printers and ink-jet printers. Laser printers produce high-quality images at a relatively rapid rate, but are not economical for color printing. Ink-jet printers are slower but support affordable color printing. A scanner is an input device that can be attached to the parallel port to provide a means of capturing image data for display and manipulation.

Various other peripheral adapters are now available to support computationally intensive multimedia processing. Multimedia capabilities include display of 2-D images, 3-D graphics and video clips, along with playback and synthesis of multiple channel music, voice, and other sounds, and two-way audiovisual communication (teleconferencing). Adapters to support these capabilities comprise processing subsystems that may include several megabytes of memory and special purpose processors, such as digital signal processors (DSP) or even an additional microprocessor. The computation performed on-board these adapters is tuned to the requirements of the peripheral task, and reduces the computational load on the microcomputer's CPU.

Busses

A bus provides a pathway for the movement of data from one component to another in the microcomputer. Different types of bus, exhibiting different characteristics, are used to connect the various components, depending on the communication requirements among the components. In general, the choice of what type of bus to use for a particular purpose involves a trade-off between the cost of implementing the bus and its controller, and the amount of data that can be moved in a given period of time.

The number of bits of data that can be transferred simultaneously is called the bus width. Some common bus widths are 1, 2, 4, and 8 bytes. The number of transfers that can be achieved in a specified period of time depends on the clock rate of the bus. If the minimum time between transfers is one clock cycle, then the maximum bandwidth, or transfer rate, is the bus width times the clock rate. For example, a 2 byte wide bus running at 33 MHz would have a maximum bandwidth of 66 Mbyte/s. Bus overhead due to arbitration or collision resolution will reduce the actual bandwidth of a bus.

Associated with each bus is a protocol, or set of rules, that is followed by all devices that share the bus. The protocol is used to determine which device is currently in control of the bus, and what particular function the bus is performing at any given time. A protocol may, for example, define a set of handshaking signals that the devices use to indicate their need to use the bus, or their readiness to receive data. In most cases, there is one device, called the bus controller, that provides a central mechanism for arbitrating requests from the devices sharing the bus.

Figure 3 shows a typical arrangement of busses connecting the components in a microcomputer. Included in the figure are a processor bus, a memory bus, and several I/O busses. Components that connect one type of bus to another are called bridges. The memory controller controls the processor (or system) bus, the memory bus, and the PCI bus. The ISA bridge controls the ISA bus. The SCSI bus has no central controller. The SCSI protocol defines a fixed priority scheme that devices use to arbitrate bus conflicts among themselves. The memory controller in Fig. 3 is also a PCI bridge, providing a path from the processor and memory busses to the PCI bus. Attached to the PCI bus is a SCSI bridge and an ISA bridge.

If an L2 cache is present in the system, it is attached in one of several ways directly to the microprocessor. The microprocessor with or without L2 is connected to the rest of the

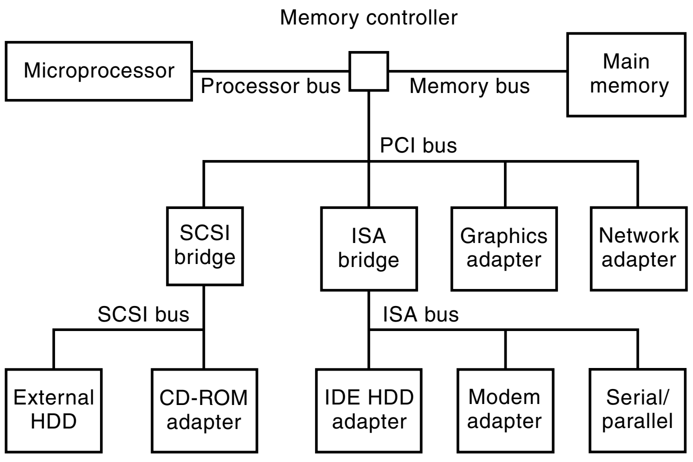

Figure 3. Busses provide pathways for data movement among microcomputer components. The processor bus and memory bus are high-bandwidth busses used within the central system. The PCI bus carries I/O data at moderate rates for devices such as graphics and network adapters. The ISA, SCSI, and IDE busses carry I/O data at a lower rate for slower devices such as the keyboard, modem, printer, and disk drives.

system through the processor bus. This bus carries both the instructions and data needed by the processor to execute applications. To keep the processor busy, the bus must be able to maintain a high rate of data movement. A typical processor bus has a bus width of 8 bytes and a clock speed of 66 MHz (528 Mbyte/s bandwidth), while more recent designs use a 100 MHz bus clock (800 Mbyte/s maximum bandwidth).

Attached to the other side of the processor bus is the memory controller. This component, usually comprising a pair of ICs, is the central arbiter for all data movement within the computer. In addition to the processor, the memory controller connects to the main memory and to the I/O devices. The memory bus connects the memory controller to the system memory. It initiates transfers to read and write memory at a rate that is compatible with the memory type and access time of the particular memory chips being used.

While the processor bus protocol is specific to a particular microprocessor family, it is desirable to define standard I/O busses so that peripheral adapters can be designed to work with any microprocessor. Different I/O device characteristics call for different bus protocols, and so several different bus standards have become generally accepted.

The peripheral component interconnect (PCI) bus is used to connect the central components (processor and memory) to peripherals that have relatively high bandwidth requirements. For example, a graphics adapter would be attached to the PCI bus, as might an adapter for a LAN connection. Connectors on the motherboard allow PCI-compliant adapters to be attached to the PCI bus, to improve the functionality or performance of the microcomputer. Bridges to slower busses are often connected to the PCI bus as well. The standard PCI bus width is 4 bytes, and the clock speed is 33 MHz, so the maximum bandwidth is 132 Mbyte/s.

The Industry Standard Architecture (ISA) bus protocol is older than PCI and supports a lower bandwidth. However, it is easier and cheaper to build an ISA-based adapter, so the ISA bus remains popular for use with peripherals that have only moderate bandwidth requirements. For example, adapters used for the keyboard and mouse, modems, and printers would all be attached to the ISA bus. The ISA bus width is 2 bytes, and the clock speed is 8.33 MHz, but ISA can only transfer data once every 2 clock cycles, yielding a maximum bandwidth of 8.33 Mbyte/s.

Two busses commonly used to connect the various disk drive peripherals to the system are the integrated device electronics (IDE) bus and the Small Computer System Interface (SCSI) bus. IDE provides a relatively cheap interface to hard drives, CD-ROMs, and floppy drives that are contained within the system enclosure. IDE has a maximum bandwidth of 5.5 Mbyte/s. SCSI is more expensive to implement, but it is faster and allows external as well as internal disk drives and other peripheral devices to be attached to the bus. Maximum bandwidth for a SCSI bus is typically 10 Mbyte/s or 20 Mbyte/s, though a wider range of protocols exist.

MICROCOMPUTER SOFTWARE

The information that controls the behavior of a computer is called software. It consists of both instructions and the data used by those instructions for decision-making. Software is often categorized as either an application program or system software. Applications are designed to be run by users to accomplish some task. System software, in particular the operating system (OS), is designed to supervise the execution of applications, and to provide services for those applications. Some programs, such as programming language translators—compilers, assemblers, interpreters—share characteristics of both application and system code.

Application Programs

Microcomputers are most often used by a single user in an interactive mode. Many applications have been developed for microcomputers specifically aimed at this interactive style of computation. For example, what-you-see-is-what-you-get (WYSIWYG) word processors format text as it is input rather than through a postprocessing step. Spreadsheet programs calculate tabular data on-the-fly, providing immediate feedback for testing alternative hypotheses or investigating how a change in one parameter affects the values of other parameters. Image-processing programs allow interactive analysis and enhancement of image data. Media-editing applications support unlimited experimentation with cuts, joins, and special effects to obtain suitable sequences of audio and video streams. Games, educational applications, and desktop publishing programs are also designed around the interactive aspect of microcomputer use. Even applications development itself is well supported through integrated development environments in which program editors, compilers, and debuggers are combined to streamline program development. Of course, noninteractive applications, such as scientific (numeric) programs and data-processing programs—bookkeeping, inventory, database—are also available for microcomputers.

To run an application, it must have space allocated for it in memory for both the instructions and the data that it will use. The program and data are then loaded into memory and linked to (supplied with the actual memory location of) any dynamic libraries that it calls. The processor then branches to the first instruction in the application, and it begins to execute. While executing, if data or instructions are referenced that are not currently in memory, they must be moved into memory. If an application needs data from a hard disk, or prints a message to the screen, or checks to see if a key on the keyboard has been pressed, the corresponding I/O operation must be performed. All of these functions—managing memory space, loading applications, controlling I/O operations, among others—are performed by the processor executing instruction sequences that are part of the operating system (OS).

Operating System

There are several major subsystems in an operating system, including the process scheduler, various resource managers (file system, I/O, memory), and the program loader. An application gains access to resources managed by the OS through calls to dynamic libraries. The OS, in turn, uses device drivers to provide control functions for specific I/O devices. In addition to supporting applications by providing common functions that would otherwise have to be replicated in every application, these OS modules provide security to the system and its users. This is accomplished through the use of certain instructions and certain data areas in memory that can only be accessed by the operating system.

Process Scheduler. The process scheduler determines which of perhaps several available instruction streams, called runnable processes, should be executed next. Early microcomputer operating systems were single-tasking, meaning that there was only one runnable process at any given time. This process was either a command shell in the operating system waiting for user input, or an application that the user chose to run.

More recent operating systems allow nonpreemptive, or cooperative, multitasking. This means that multiple processes may be runnable at any given time, but once the scheduler chooses one to execute, that process executes until it completes. The operating system has no mechanism to stop it.

As with microcomputer hardware, operating systems for microcomputers have evolved and grown more complex, and have inherited functionality from main frames and minicomputers. The most recently developed microcomputer operating systems support preemptive multitasking. This means that multiple processes may be runnable, and that once a process starts to run, it may be suspended by the operating system at any time, to allow another process to run. This capability is particularly important for multiuser systems, where it provides time-sharing of the processor in such a way that each user has the impression that their application is progressing at a steady rate. However, it is also important in a single-user microcomputer, both to support particular styles of programming (multithreading), and to allow efficient and convenient background execution (e.g., spooling), at the same time that one or more interactive applications are running.

Memory Manager. Main memory is physically organized as a one-dimensional array of storage elements, each identified by its order in the array, called its address. All of the information used by the processor to do work, including both instructions and data, must be stored in main memory in order to be accessible to the processor. The memory manager must partition main memory so that each of the different software components that require this resource at any given time have the needed space. Among the different partitions required are those for base OS code and data, for applications code and data, and for dynamic libraries and device drivers.

Today's microprocessors provide hardware support for memory managers to implement a virtual memory. The idea is that the memory manager can behave as if it had a very large memory to work with, and each application has its own memory distinct from that of other applications. This simplifies the memory-management task. However, more space may be allocated in this very large virtual memory than is actually available in the physical memory. The virtual memory system, a combination of microprocessor hardware and OS code, solves this problem by moving information as needed between main memory and backing store. This gives the appearance of having a very large main memory.

Dynamic Libraries. Application programs request operating system services by calling library routines. Each of the services has associated with it an application programming interface (API), which defines the format the application must use to interact with the service. The API provides a level of abstraction between the application and the library. This allows the details of the library software or the hardware involved in the service to change, while the application software remains unchanged.

A library is simply a collection of software functions commonly used by applications. Dynamic libraries, also called shared libraries, are loaded into memory once and retained, so that any application that needs them can access them. Such a library function is dynamically linked to an application that references it when the application is loaded. This dynamic linking reduces the size of the application, and allows the library routine to change without a corresponding change to the application.

Device Drivers. Among the services that an operating system provides to an application program is I/O processing. When an application specifies that a particular data stream is to be written to the display, or that a new file should be created on the hard disk, or the next keystroke should be read in, operating system code is executed to perform the requested function.

The request from the application is abstract, in the sense that it is made independent of which particular device or even class of device will be involved in satisfying the request. The I/O manager has knowledge of different classes of devices, but does not have specific information on how to control every possible I/O device that might be attached to the microcomputer.

The device driver is the piece of code that does have device specific information. When a particular device is installed, the corresponding device driver software is installed as well. When the I/O manager gets a request to perform a particular function on a particular type of device, it passes the request to the appropriate device driver, which turns the request into the correct control sequence for that device.

Booting the Computer. RAM memory, used for most of main memory and caches, is volatile. That is, it loses its information whenever the power is turned off. When a microcomputer is first turned on, main memory has no information in it. In order for the operating system to load a program, the OS must already be in memory. But how does the operating system itself get loaded? In a reference to the expression "picking oneself up by the bootstraps," the process of getting the computer to bring itself to a state where it can run programs is called bootstrapping, or just booting.

The set of instructions for booting the computer, the boot code, is stored in ROM memory, a nonvolatile, nonwriteable form of IC memory. Since instructions in ROM cannot be changed, programs in ROM are often referred to as hard-wired, or hard-coded. Since boot code has this property of being hard-coded software, it is also referred to as firmware.

The boot code performs two functions. First, it checks the hardware to determine that enough of it is functional to begin loading the operating system. In particular, it exercises the basic functionality of the microprocessor, writes and reads the RAM memory to check for data errors, and tests the display adapter, disk drives, and keyboard to verify that they are operational. Second, the boot code loads the operating system. Although loading the operating system can be involved, the boot code itself need only get the process started. Once it locates the device from which the operating system is to be loaded (usually a hard disk, sometimes a CD-ROM, or even the LAN), the boot code loads a program from that device containing information needed to load other pieces of software. These, in turn, may take part in the loading of the rest of the operating system. In this way, the computer "picks itself up by its bootstraps."

EVOLUTION OF THE MICROCOMPUTER

Early electronic computers used vacuum tubes as the switches that implement the calculation and storage circuitry. In the next generation, computers used transistors. Given the sizes of the components, these computers had to be quite large to be capable of doing useful work. Third-generation computers used integrated circuits (IC), consisting of many transistors on a single piece of silicon. At this point, more powerful large computers could be built, but a smaller computer could be built and still do useful work. Fourth-generation computers used higher levels of integration of transistors on single IC chips, referred to as large-scale integration (LSI) and very large scale integration (VLSI). At this point, the entire central processing unit (CPU) of the computer could be implemented on a single chip. Such a chip is called a microprocessor, and the computer that contains it is a microcomputer.

The first microprocessor, the Intel 4004, was introduced in the early 1970s. It had a 4 bit-wide data bus, a 740 kHz clock that required eight clock cycles to execute each instruction, and could address 4 kbyte of memory. Combined with several other chips for memory and I/O, the 4004 was part of the first microprocessor-based computer kit, the MCS-4.

For the next decade, microcomputers evolved from 4 bit and 8 bit hobby kits consisting of a motherboard with chips, some switches and 7-segment displays, to several complete 8 bit microcomputer systems. The Altair 8800, Apple II, and TRS 80 are examples of early microcomputers. These systems generally included a keyboard and a monitor, and could have a floppy disk drive or a printer attached as well. In addition, operating systems, such as CP/M, and programming language translators, such as BASIC, were available for these systems, allowing users to develop applications more quickly and easily. While Intel continued to develop more complex 8-bit and then 16-bit microprocessors, other manufacturers developed their own designs, including TI's TMS1000 and TMS9900, MOS Technology's 6502, and Motorola's 6800 and 68000.

In the early 1980s, IBM announced their PC, a microcomputer based on the Intel 8088 microprocessor (16 bit processing inside the chip, 8 bit bus externally), running the Microsoft disk operating system MS-DOS. To encourage third-party hardware and software vendors to develop products for the PC, IBM published details of its design. This encouraged not only the development of add-on hardware and software, but of PC clones—copies of the entire microcomputer built by other manufacturers.

Over the next decade the market for microcomputers grew rapidly. Dozens of companies introduced complete microcomputer systems and many more developed hardware and software to be used on these systems. During this time there was little standardization, so a hardware adapter or a piece of software had to be developed for one particular microcomputer system. Both functionality and performance improved steadily. Systems based on 16 bit processors replaced 8 bit systems, and 32 bit microprocessors were in development. Hard disk drives were uncommon on early systems, but became more common with capacities growing from 5 Mbyte to 40 Mbyte and higher. Dot matrix printers were replaced by laser and ink jet printers. Modem speeds increased from 300 bit/s to 9600 bit/s. CD-ROM drives became available.

Other developments included networking hardware and software, allowing data and other resource sharing among clusters of microcomputers. Large portable computers and then laptop computers also appeared during this period. In addition, the SCSI, ISA, and EISA bus standards became established, allowing peripherals to be more easily added to a system. Also, user interfaces evolved from primarily text-based to graphics-based. The graphical user interface (GUI) first appeared on microprocessor systems on the Apple Lisa and Macintosh systems, and then later in the decade in Microsoft's Windows operating system.

By the early 1990s, the IBM PC family of microcomputers, based on the 8088 microprocessor and its successors, and the MS-DOS operating system and its successors, had become the dominant microcomputer platform in the industry. As this decade has progressed, further improvements in functionality and performance have been achieved. These include faster modems and CD-ROM drives, higher capacity main memories and hard disks, hardware adapters to speed up display of 2-D and 3-D graphics, and playback and synthesis of sounds, and the availability of scanners and digital cameras for image capture. Another important development has been the emergence of the World Wide Web (WWW), and the availability of browser programs for microcomputers. These allow access to a wide range of information sources, many taking advantage of the multimedia capabilities of today's microcomputers.

The original IBM PC, introduced in 1981, contained a microprocessor running at 4.88 MHz, with 64 kbyte of DRAM for main memory, along with a keyboard, a monitor that displayed text only, and a 160 kbyte floppy disk drive. A 300 bit/s modem and a low-quality (dot-matrix) printer could be added. As performance and functionality increased over the years, the price for a typical system has dropped to about 50% the price of the original PC. In mid-1998, that typical PC would have a microprocessor running at 266 MHz with 32 Mbyte of RAM, along with a keyboard, a graphics monitor, a mouse, a 1.4 Mbyte floppy disk drive, a 4 Gbyte hard disk drive, a 56 kbit/s modem, a CD-ROM drive, and a color printer.

CURRENT TRENDS IN MICROCOMPUTER DEVELOPMENT

The trend toward higher performance—faster cycle times, higher capacities, higher bandwidths—is expected to continue for some time to come. At the same time, there is renewed interest in low-cost computing devices having lower capabilities and capacities for users that do not require the power of current microcomputers.

Performance-Driven Developments

Each of the major computer subsystems—processor, memory, and I/O—are being developed for high performance. The factors driving these high-performance developments are the desire to run current applications more quickly, and to run new applications that have higher computational requirements than could previously be satisfied. For example, such applications as video playback, 3-D graphics, voice input, and teleconferencing, could not have been run on the microcomputers of several years ago. In addition, microcomputers are now being used as servers—systems that manage a particular resource so that other computers (clients) can access them—for more computationally intensive tasks, such as database and transaction processing.

Microprocessor Performance. Since microcomputers use a single-chip microprocessor as the central processing unit, the processing power available to the system is always limited by the size of chips that can be fabricated, and the density of the devices on the chip. As both chip size and density have increased over the years, the larger numbers of available semiconductor devices on a chip have led to increases in both performance and functionality. Often, the increase in circuit count has allowed mechanisms previously used in minicomputers or even main frames to be used in microprocessors. Among the mechanisms used to achieve high performance are pipelining, superscalar processing, out-of-order instruction execution, prefetching, branch prediction, and speculative execution.

One measure of raw microprocessor performance is the number of instructions it can execute in a given period of time, usually expressed in millions of instructions per second (MIPS). This measure is a function of the clock speed in cycles per second, and the number of instructions per cycle (IPC) that can be executed. Improving performance requires that the clock speed or IPC rating (or both) be increased.

Clock speeds have been increasing at a steady rate due to the decreasing sizes of semiconductor devices on silicon chips. Clock speed can be further increased by reducing the amount of computation done on each cycle. This reduction is achieved by using an instruction pipeline. The pipeline consists of a series of processing stages, each stage responsible for only one of the operations needed to execute an instruction. For example, a typical breakdown of instruction execution into stages would include fetching the next instruction from memory, decoding the instruction to determine its type, fetching any operands used by the instruction from memory, performing the specified computation, and writing the results of the computation back to memory.

A given instruction will go from one stage to the next on each cycle, completing the process in five cycles. This is about the same amount of time it would take the instruction to complete if execution were not pipelined. However, after the first instruction completes the first stage, the next instruction can enter that stage. Thus there are five instructions in the pipeline at a time, with one finishing every cycle. Using a pipeline with more stages allows a faster clock, since less work is done at each stage.

IPC can be increased by using superscalar execution. A superscalar processor can execute more than one instruction in each cycle. This is done by fetching and decoding the next two or more sequential instructions, and providing multiple execution units to perform the specified computations in parallel. Each execution unit contains the circuitry for performing one particular class of computation, such as integer arithmetic, floating-point arithmetic, shifting and rotating bit patterns, loading data into registers, and so on. By allowing them to operate independently, and providing more than one of such highly used units as the integer arithmetic unit, two or more instructions that are adjacent in the instruction stream can be executed at the same time.

For a superscalar processor that can execute two instructions per cycle running at 400 MHz, the maximum performance rating is 800 MIPS. There are several reasons why microprocessors do not achieve such maximum performance. One is that there are not enough of the right kind of execution units to process the next two (or more) adjacent instructions. For example, if there is a single floating-point unit, and the next two instructions are both floating-point instructions, they will have to execute sequentially. This problem can be reduced by allowing out-of-order execution of instructions. That is, if an instruction appearing later in the sequence is of a type for which an execution unit is available, and if that later instruction does not depend on any intervening instructions for operands, then the later one can be executed early to avoid IPC degradation.

IPC also degrades because of data dependencies. Two adjacent instructions cannot be executed in parallel if they depend on each other in one of several ways. For example, if an instruction that uses the value in a particular register is followed by an instruction that stores a new value in that same register, the second instruction must not write to the register before the first one reads it. This apparent data dependency, apparent because it is due to a shared register resource, not due to sharing the data value itself, can be solved by reassigning the registers accessed by the two instructions, making use of some additional registers called *rename registers.* The processor must still detect real data dependencies and sequentialize processing to resolve them.

A third reason that maximum IPC is not achieved is that data are not always available when needed. Data that are in the processor registers are immediately available for use. If the data are in memory, there will be a delay to retrieve them. This delay might be one cycle if the data are in L1, several cycles if in L2, and tens of cycles if in main memory. Hardware prefetching of instructions reduces this problem as does software prefetching of data.

Finally, a major source of performance degradation is associated with branch instructions. A branch instruction corresponds to a jump from one sequence of instructions to another. For conditional branch instructions, the branch is taken only if a particular condition is met. If the condition is not met, execution of the current instruction sequence continues. A branch is said to be resolved when it is known whether it will be taken or not.

Many of the performance enhancements described above take advantage of the fact that instructions are executed sequentially. When a branch occurs, this assumption is defeated, and the performance enhancements break down. For example, instructions that have entered the pipeline after the branch instruction must be flushed out and their partial execution discarded if the branch is taken. The pipeline then starts to fill with the first instruction of the new sequence, but nothing comes out of the pipeline for several cycles. This is referred to as a bubble in the pipeline, corresponding to lost instruction throughout.

There are several mechanisms used in today's microprocessors to reduce the degradation caused by branches. First, there is branch prediction. If it is known that a branch will be taken, the target address of the branch can be used to begin fetching the new sequence of instructions. Several sources of information are used for predicting branches, including target addresses of previously taken branches and a history of whether conditional branches have been taken before. This information can be maintained in a table indexed by the branch instruction address. Second, there is speculative execution, involving the execution of one or more instruction streams that may or may not be on the correct execution path, depending on the outcome of upcoming branch instructions. The complexity of such a mechanism comes from the need to undo the effects of any instruction that was speculatively executed and later found to be on a wrong path. The performance gain comes from the fact that as long as one of the paths executed was the correct one, there is no delay due to the branch.

Currently, developments aimed at increasing the computational power of microcomputers are focused on increasing clock speed, increasing IPC using approaches just described, and combining multiple processors in a single system (multiprocessing). In the future, alternative approaches to keeping the processor busy, such as multithreading and the use of a very long instruction word (VLIW) may become popular. Multithreading involves maintaining several independent streams of instructions in the processor so that data dependencies can be reduced and pipeline bubbles from one stream can be filled in by instructions from another stream. VLIW processors use wide instructions to specify multiple operations per instruction that have been determined prior to execution not to be interdependent. These operations can be executed simultaneously to achieve a high level of parallel computation.

Memory Performance. The raw computational power of the processor is not the only factor that determines the overall performance of a microcomputer system, as is clear from the discussion above. If the processor cannot be supplied with enough instructions and data to keep it busy, it will waste many cycles doing nothing. The various components of the memory subsystem are characterized by their capacity, bandwidth, and latency. Because no device exists that optimizes all three of these attributes, the memory system is composed of a variety of components that are combined in such a way that the advantageous characteristics of each component are emphasized.

For example, small and fast caches are used close to the processor to provide low-latency responses to most memory requests, while larger main memory modules are used to provide high capacity. If the cache can be managed so that the requested data are almost always in the cache, the overall memory subsystem appears to the processor as a low-latency, high-capacity storage system. The use of an even smaller and faster cache on the microprocessor chip, and of hard disk backing store at the other end of the memory hierarchy, provide even better latency and capacity, respectively.

While multilevel caches help to alleviate the memory latency problem, main memory latencies have become an ever growing problem in recent years, due to the rapid increases in processor clock speeds. DRAM latencies of 60 ns represent a nearly tenfold improvement over the 500 ns latencies of two decades ago. However, in that time, microprocessor clock speeds have increased from about 5 MHz to over 200 MHz, and will continue to rise quickly for at least the next few years. A processor clock speed of 266 MHz corresponds to a clock period of 3.8 ns. A 60 ns memory latency then corresponds to a 16 cycle delay.

A number of recent developments in DRAM design have been aimed at improving the memory bandwidth that is otherwise degraded by poor memory access time. DRAM chips are organized as two-dimensional arrays of memory cells, each cell storing one bit of information. A memory access consists of first reading the entire row of cells containing the bit of interest, and then choosing the column containing that bit as the data to be transferred. The overall latency in accessing the data is due to the row access time followed by the column access time.

Fast page mode (FPM) DRAM accesses multiple columns once a row has been accessed, reducing the average access time per bit. Extended data out (EDO) DRAM allows overlapping of data transfer with the next memory request to reduce the effective latency. Burst EDO (BEDO) memory allows multiple data transfers per request, reducing the amount of time spent sending addresses to memory. The current trend in DRAM system development is toward the use of synchronous DRAM (SDRAM) memory.

For SDRAM memory, the memory subsystem is clocked at the same frequency as the rest of the system (the microprocessor itself has its own clock that may be some multiple of the system clock frequency). The memory controller puts an address on the DRAM address bus and receives the corresponding data a fixed number of cycles later, with no additional protocol overhead. While today's asynchronous busses typically run at 66 MHz, the first SDRAM busses run at 100 MHz, with higher frequencies expected. What is not yet clear is which of several proposals for synchronous DRAM architectures will become prominent in the coming years.

Peripheral Performance. Early microcomputers connected peripheral adapters directly to the memory or I/O bus on the processor. To support development of peripherals by third parties, standard bus protocols were later defined. The ISA bus, which was introduced in the mid 1980s and standardized in the late 1980s, has a maximum bandwidth of 8.33 Mbyte/s. This is sufficient for connecting keyboard, text-based and low-resolution graphics monitors, modems, printers, and other devices with moderate bandwidth requirements. The PCI bus was developed to support higher bandwidth requirements, such as those of high-resolution and 3-D graphics adapters and high-speed networks. It has a maximum bandwidth of 132 Mbyte/s.

Current trends in peripheral developments are toward both additional functionality and increasing performance. Image rendering for multimedia and 3-D graphics is supported by graphics adapters with on-board microprocessors that can process several hundred million bytes of data per second. Sound cards have DSP chips for real-time processing of multichannel audio signals or synthesis of stereo sounds in virtual worlds. CD-ROM, modem, and network data rates continue to increase. In some cases, current peripheral busses are sufficient to handle the higher bandwidth requirements, but in other cases faster busses are needed. One way to increase bandwidth is to enhance current bus capabilities. For example, a 64-bit wide PCI standard has been defined to double its previous bandwidth. However, as with the memory subsystem, new I/O bus protocols are being developed to significantly increase the data transfer rate.

Cost-Driven Developments

The design of low-cost microcomputers generally involves innovative uses of technology, rather than innovations in the technology itself. The way to achieve low cost is to leave out functionality, reduce capacities, and use parts (such as the microprocessor) that are no longer at the leading edge of technology. The challenge in designing such devices is to find a combination of components that has a significant cost advantage, but also provides a sufficient and balanced functionality and performance to support some useful class of computations.

For example, the network computer (NC) is a low-cost microcomputer designed for users who need only a subset of the functionality available in a PC. For instance, an NC could connect over phone lines or a LAN to the World Wide Web (WWW) or other network resources, allowing the user to browse information, fill in electronic forms, and execute programs that can be downloaded off the network. An NC would not be used, on the other hand, for most applications development, for running computationally intensive applications, or for running applications with large memory requirements. Among the current developments in microcomputer software, the Java programming language is aimed at supporting the NC model of computation.

Reducing functionality further yields a system that can no longer be called a microcomputer. There are a growing number of uses for microprocessor-based systems that contain some of the other components of microcomputers as well. These systems are referred to as embedded, meaning that the computation is done in the service of some fixed control mechanism, rather than being used for general-purpose processing. Examples of systems that incorporate embedded processors include automobiles, microwave ovens, digital cameras, video games, and telephone switches.

The cost constraints of embedded applications encourages higher levels of integration of functions on chips. For example, the integration of the processor, the memory controller, and the L2 cache on one chip has been proposed, as has the integration of the processor and DRAM. Any successful low cost integration of these functions is likely to find its way into future microcomputer designs, particularly in the portable computer segment, where physical space, rather than cost, is at a premium.

BIBLIOGRAPHY

1. A. S. Tanenbaum, *Modern Operating Systems,* Chap. 8, Englewood Cliffs, NJ: Prentice-Hall, 1992.
2. K. Polsson, Chronology of events in the history of microcomputers [Online], 1998. Available http://www.islandnet.com/kpolsson/comphist.htm.
3. J. L. Hennessy and D. A. Patterson, *Computer Organization and Design,* 2nd ed., San Francisco: Morgan Kaufmann, 1998.
4. M. Pietrek, *Windows Internals,* Chap. 6, Reading, MA: Addison-Wesley, 1993.
5. T. Shanley and D. Anderson, *PCI System Architecture,* 3rd ed., Reading, MA: Addison-Wesley, 1995.
6. PowerPC 603/604 Reference Design, Order No. MPRH01TSU-02, IBM Corporation, 1995, available through IBM branch offices.
7. R. White, *How Computers Work,* Emeryville, CA: Ziff-Davis Press, 1997.
8. N. Randall, A RAM primer, *PC Magazine,* **16** (18): 1997.
9. M. J. Zulich, DRAM: The next generation, *Computer Shopper,* June 1997.
10. R. Jain, *The Art of Computer Systems Performance Analysis,* Part I, New York: Wiley, 1991.
11. M. Johnson, *Superscalar Microprocessor Design,* Englewood Cliffs, NJ: Prentice-Hall, 1991.
12. J. Walrand and P. Varaiya, *High-Performance Communication Networks,* San Francisco: Morgan Kaufmann, 1996, Chap. 3.

Reading List

D. Burger and J. R. Goodman, eds., Special issue on Billion-Transistor Architectures, *Computer,* **30** (9): 1997.

Y. Patt, ed., Special issue on Trends in Architecture, *Computer,* **30** (12): 1997.

PETER A. SANDON
IBM Microelectronics Division

MICROCONTROLLER. See MICROPROCESSORS.