BYTE

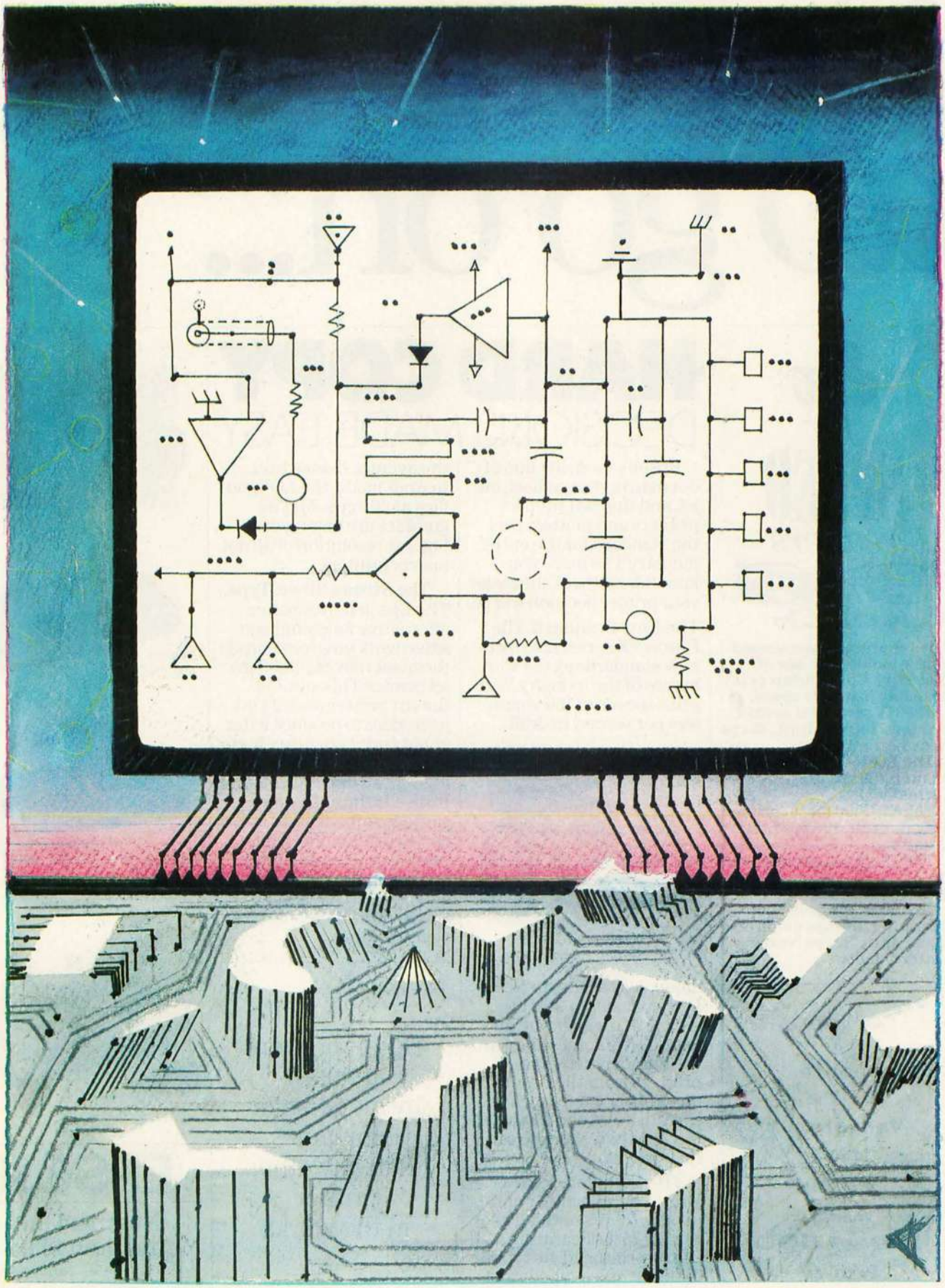
THE SMALL SYSTEMS JOURNAL

AUGUST 1985 VOL. 10, NO. 8

$3.50 IN UNITED STATES
$4.25 IN CANADA / £2.10 IN U.K.
A McGRAW-HILL PUBLICATION
0360-5280

THE AMIGA from Commodore

DECLARATIVE LANGUAGES:
Prolog, Hope, FP



Features

IN EARLY 1984, officials from a start-up company called Amiga showed journalists prototypes of a new personal computer. The prototypes used a Sage 68000-based machine as a CPU. Big steel boxes performed the special graphics and sound functions that Amiga planned to implement in silicon. The graphics were spectacular—fast enough to support animation. The audio output not only produced music but used stereo to enhance animation. Sound shifted from the right speaker to the left as a ball bounced across the screen. Everyone wondered if Amiga could really reduce all the power in the prototypes to silicon chips.

Late in 1984 and early in 1985, venture capital firms became wary of new entries in the crowded personal computer market increasingly dominated by IBM. Serious doubt arose about whether Amiga would be able to get capital to manufacture its machine. Many of us feared that the exciting machine we had seen in prototype would never become a product. We were delighted when Commodore acquired Amiga and saved this technically outstanding machine from oblivion. Gregg Williams, Jon Edwards, and Phillip Robinson give an in-depth look at the technology that makes the Amiga the most advanced and innovative personal computer today.

Steve Ciarcia's Z8 controllers are running all sorts of devices throughout the world. This month, Steve introduces a new controller that is bus-compatible with his Z8 products. The BASIC-52 computer/controller has an 8K BASIC in ROM and so is easy to program. Steve will be developing applications for the BASIC-52 in the months ahead.

BYTE's readers appreciate the 32032 microprocessor from National Semiconductor because of its outstanding architecture and its raw power. Those who want to buy a complete 32032 system can now get systems such as the Elite Computer Systems Expert 32 (see What's New, May, page 464). Those readers who want to move a 32032 into an existing box can complete the DSI-32 32032 coprocessor board for the IBM PC described in this issue. Five authors from Definicon Systems tell about the hardware and software that will give many of us our first taste of 32-bit microcomputing.

The August Programming Project lists Pascal code that can do context-free parsing of arithmetic instructions, which converts them to executable form. The code generates what amounts to a TI-style calculator. In the process of building the calculator in software, you learn a lot about the roots of the parser in linguistic theory of context-free grammars. Jonathan Amsterdam wrote the parser in highly portable Pascal code.

PRODUCT PREVIEW

The AMIGA PERSONAL COMPUTER

Its speed and colorful graphics come from a 68000 and sophisticated custom chips

Editor's note: The following is a BYTE *product preview. It is not a review. We provide an advanced look at this new product because we feel it is significant. A complete review will follow in a subsequent issue.*

THERE ARE TWO ways to get work done inside a computer: do it in software or do it in hardware. The first way gives you unlimited flexibility; the other, speed. The Apple Macintosh does almost everything in software—and,

(continued)

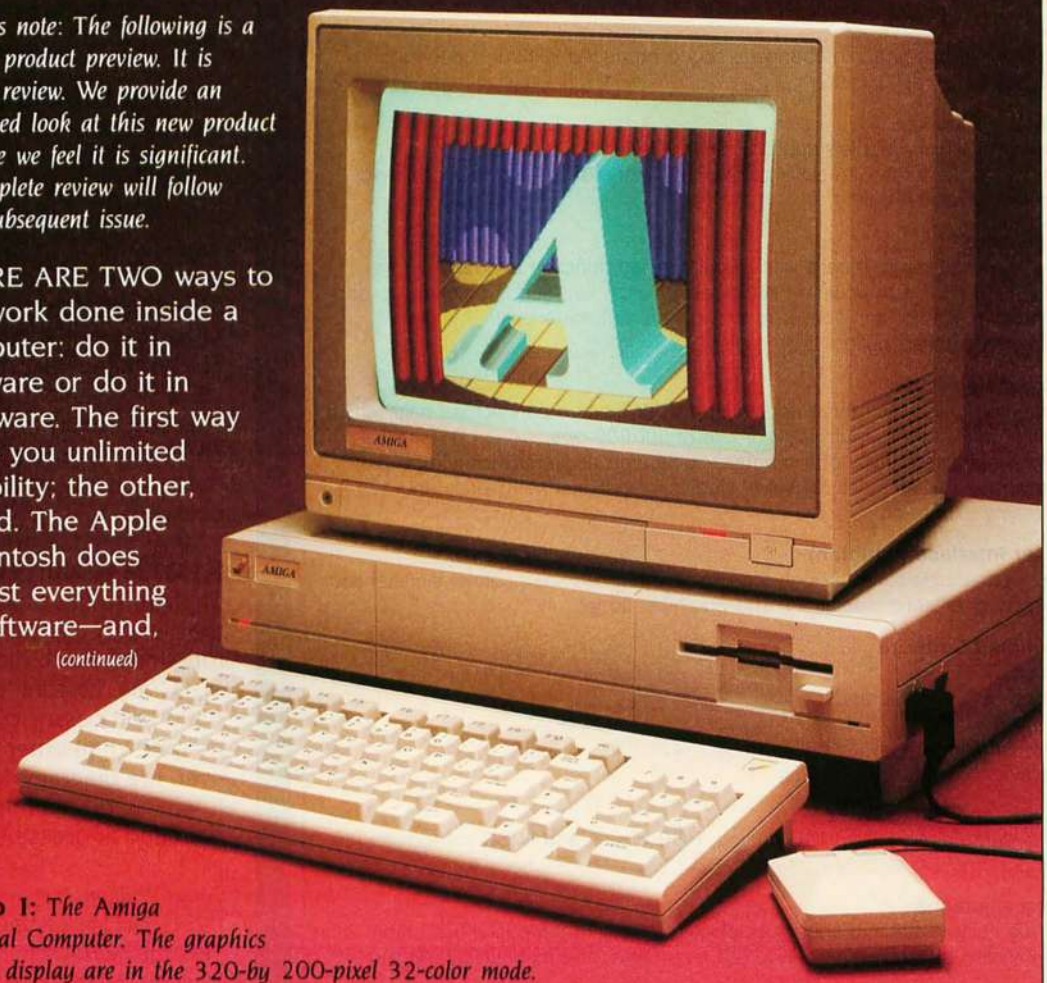


Photo 1: *The Amiga Personal Computer. The graphics on the display are in the 320-by 200-pixel 32-color mode.*

PHOTOGRAPHED BY MICHAEL CARR

BY GREGG WILLIAMS, JON EDWARDS, AND PHILLIP ROBINSON

IN BRIEF

Name
Amiga Personal Computer

Manufacturer
Commodore International
1200 Wilson Dr.
West Chester, PA 19380
(215) 431-9100

Price
$1295

Microprocessor
Motorola 68000, a 32-/16-bit microprocessor (32-bit internal data path and registers, 16-bit external data bus) running at 7.15909 MHz

Main Memory
256K bytes dynamic RAM, user-expandable to 512K bytes; machine's design allows for maximum of 8.5 megabytes

ROM
192K bytes of ROM containing multitasking, graphics, sound, and animation support routines

Graphics
Five modes (320 by 200 pixels, 32 colors; 320 by 400, 32 colors; 640 by 200, 16 colors; 640 by 400, 16 colors; sample-and-hold mode); independent horizontal and vertical scrolling of dual playfields; eight hardware sprites; colors chosen from a palette of 4096 colors

Sound
Four independent audio channels; sound produced without supervision of 68000

Floppy Disk
Built-in 3½-inch double-sided disk drive. Disks hold 880K bytes in 160 tracks, each with eleven 512-byte sectors; drive hardware can read an entire track at a time

Keyboard
Detached 89-key keyboard with calculator pad, function and cursor keys; keyboard returns row/column keycodes for each key, sends both key-up and key-down signals; can sense up to two keys simultaneously; 8-key type-ahead buffer

Expansion Ports
Disk port onto which three additional disk drives can connect via daisy chain; serial port with maximum transfer rate of 500,000 bps; programmable parallel port normally configured as Centronics-compatible; expansion bus includes full set of signals for optional peripherals and memory expansion

User Interface (Intuition)
Supports multitasking through the use of virtual terminals; allows simultaneous display of different resolutions and graphics modes

Bundled Software
AmigaDOS
Voice Synthesis Library
ABasiC
Tutorial (Mindscape)
Kaleidoscope (Electronic Arts)

Audio and Video Ports
Two stereo audio jacks; RGB analog, RGB digital, and NTSC composite output

Miscellaneous
Three custom chips to control graphics, audio, and peripheral I/O; chips connected by 19-bit register-address bus; two-button mechanical mouse

Optional Peripherals
3½-inch 880K-byte disk drive; RGB analog color monitor; 256K-byte memory expansion module; 300/1200-bps modem; MIDI interface; frame grabber

not coincidentally, people want Apple to increase the Mac's speed, add color, and lower its price.

Commodore has just introduced a computer that promises these improvements, and it does so by doing many things in hardware. At $1295, the Amiga Personal Computer (see photo 1) promises lightning-fast desktop-metaphor graphics in color and twice as much memory and disk storage as the Macintosh for several hundred dollars less than the Macintosh (about $900, but you'll have to buy a monitor or television set for the Amiga). It also has an expansion bus and a whopping 192K bytes of sophisticated 68000 code in ROM (read-only memory) that extends the multitasking, graphics, sound, and animation capabilities of the Amiga hardware.

System Description

The Amiga is summarized in the In Brief section on this page. It has no slots for expansion cards, but Commodore later intends to offer a box that connects to the expansion connector to add several expansion slots. (It is theoretically possible to add 8 megabytes of memory in this way.) The Amiga's disk operating system will also be able to look at the expansion box, determine what peripherals are present, and configure itself accordingly, regardless of the box's contents.

System Architecture

The Amiga has a unique architecture that is only partially described by a functional block diagram (see figure 1). Three custom chips relieve the 68000 processor of many tasks that tie it down in other computers. However, the diagram does not show the finely tuned sharing of the system's data and address buses, the 25 DMA

Gregg Williams is a senior technical editor at BYTE, *and Jon Edwards is a technical editor. They can be reached at* BYTE, POB 372, Hancock, NH 03449. *Phillip Robinson is a West Coast senior technical editor at* BYTE. *He can be reached at* BYTE Magazine, 425 Battery St., San Francisco, CA 94111.

(direct memory access) channels that do many data-movement-intensive operations without tying up the 68000, or the multiprocessing routines in ROM that allow the Amiga to orchestrate a variety of tasks. In the following sections we will look at the key elements of the Amiga's system architecture.

THE CUSTOM CHIPS

The three custom chips that control DMA, graphics, sound, and I/O (input/output) (see photo 2) were designed by Jay Miner, who is best known for his design of the custom chips in the Atari 800 series computers. Although we will discuss them in depth by function, here is a simple breakdown:

- The "animation custom chip" actually contains several miscellaneous functions. It is the "traffic cop" that controls DMA. It contains the Copper, a coprocessor that can directly control the other chips in relation to the video beam, and the Blitter, a device that quickly draws lines, fills areas with a given color, and manipulates rectangular blocks of pixels.
- The graphics custom chip, which manipulates the visible display, permits up to two independent bit-mapped images and eight sprites (which are images that can be moved easily around the screen, "under" or "on top of" the bit-mapped images).
- The peripherals/sound custom chip contains four channels of sound, the disk controller, an interrupt controller, and the interfaces for the serial port and the mouse/joystick port.

INTERRUPTS AND DMA

In the Amiga, all the peripherals are interrupt-driven—that is, the 68000 is not tied up constantly *polling* them to see if they have new data; instead, the 68000 gets data from the peripheral only when the peripheral sends an interrupt signal. The peripherals/sound chip receives interrupt-request signals from one of 15 sources (e.g., the disk drive or a sound channel), translates the request to one of six interrupt levels supported by the 68000 (the seventh is reserved for future use), and sends the interrupt signal to the 68000.

The 68000 shares the address and data buses with 25 channels of DMA, the registers and logic of which reside in the custom chips. Amiga's DMA is fast for two reasons: first, the fact that each device has its own DMA channel decreases the overhead associated with a DMA operation; second, many DMA operations are interleaved with 68000 bus access in a way that makes the DMA transparent to the 68000 (see below for details).

When DMA occurs between memory- and custom-chip registers, the use of the 19-bit register-address bus (see figure 1) makes the transfer twice as fast. By putting the memory address on the address bus and the register address on the register-address bus, the DMA circuitry causes the data value to move directly from the memory address to the register. This occurs twice as fast as DMA would via the 68000, which would first read the data into itself and then write the result to the register.

LIBRARIES AND DEVICES

System software (much of it in the 192K bytes of ROM) contains *libraries*.

(continued)

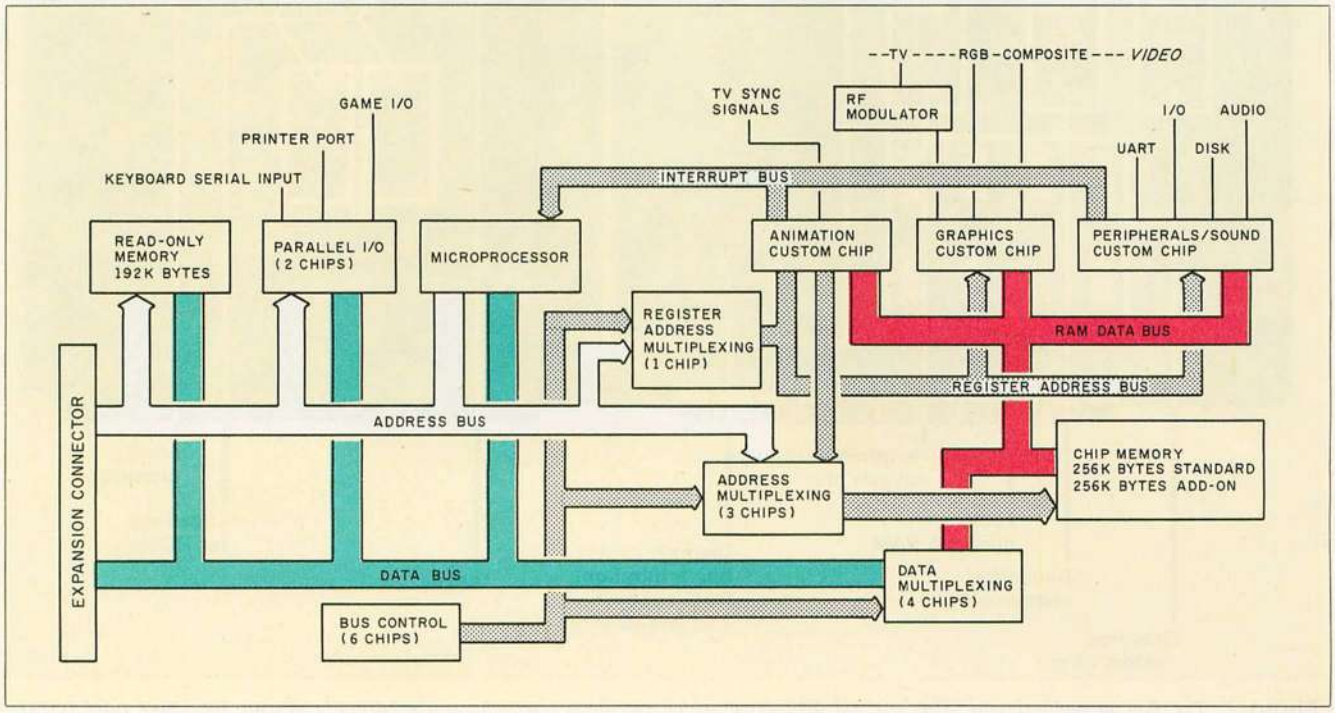


Figure 1: *A block diagram of the Amiga Personal Computer.*

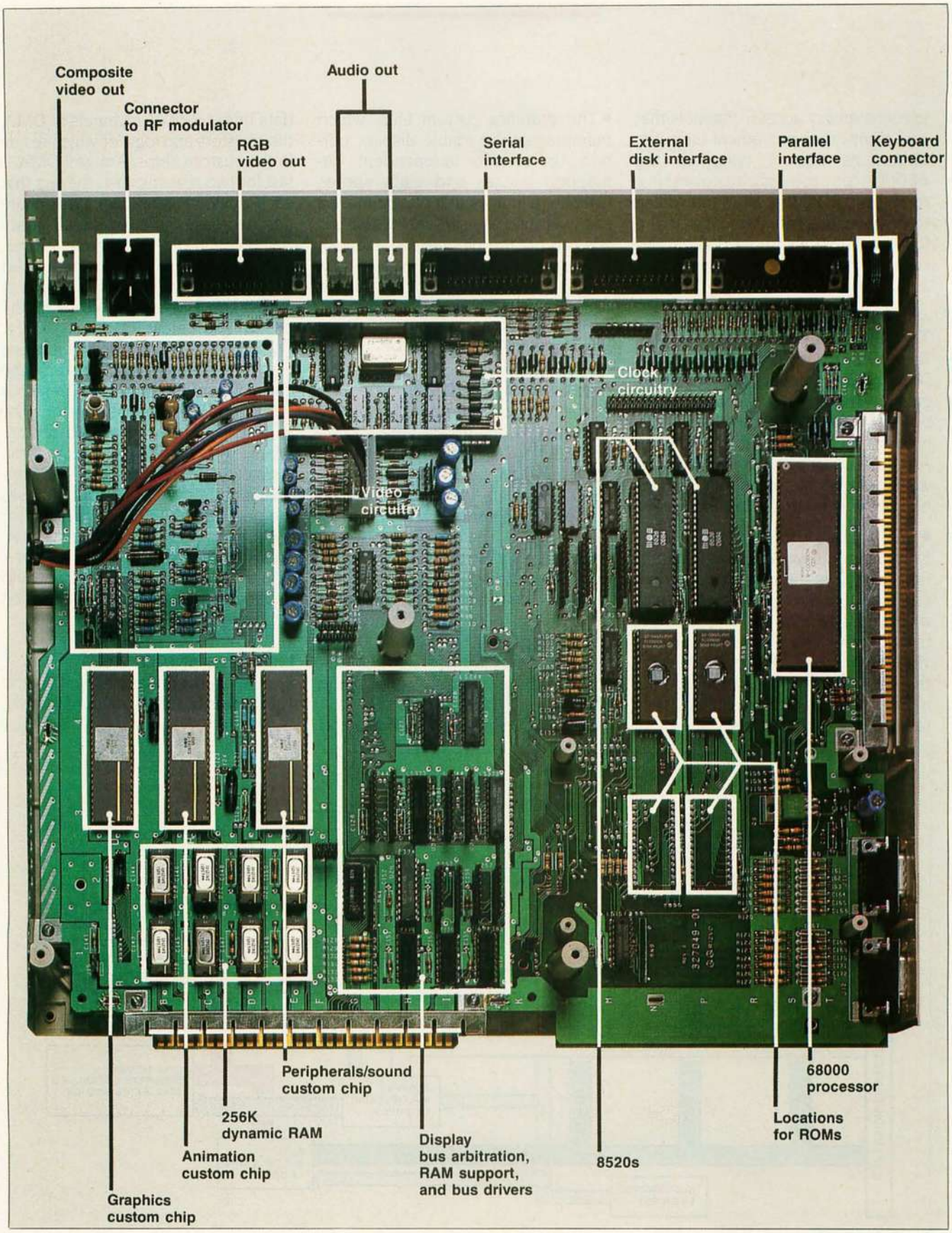


Photo 2: *The Amiga motherboard. The internal disk drive, which has been removed, would normally obscure the lower right corner of the motherboard. The power supply (not shown) is to the left of the motherboard.*

a predefined way of organizing useful routines so that they can be accessed with maximum flexibility. Libraries can be resident or transient and can be used at any memory address (when they're in RAM [random-access read/write memory]). Both routines and data can always be called via a 68000 indirect reference with offset; this allows you to write code using a library routine without knowing that library's address at compile time. (In fact, all the code in the system can be referenced knowing only one fixed address in the machine, and even that address is supplied to any machine that needs it.) A *device* is an extension of the library concept that allows software to access I/O devices (both present and future) in a uniform way.

THE EXEC ROUTINES

The Exec system is a collection of reentrant optimized 68000 ROM routines that perform many functions vital to the operation of the Amiga. It includes routines that create and manipulate lists and queues, schedule tasks by priority, handle interrupts, organize device I/O, control memory use, and perform other functions.

An important data structure in the Amiga is the *list node*. The list node is a block of data with pointers to the predecessor and successor nodes in the list it's in, two 8-bit type and priority fields, and an associated block of data. A *list* is a doubly linked chain of list nodes and items, started by a header that points to the first and last nodes. Exec contains several routines that let you do things like create a new list, insert a list item into its proper place in a queue, and remove a node from a list.

Another important set of routines allows you to manipulate *tasks*. A task is a unit of work that shares the Amiga with other tasks in a way that varies with both the type and priority of the task. (All the current tasks are held in a queue and are executed by decreasing priority.) Most programs and operations reside in the Amiga as tasks.

The task priority field, which contains a number between −128 and 127, determines the order in which

The Exec routines perform many functions vital to the operation of the Amiga.

tasks will execute. Tasks with identical numbers share the Amiga in time slices of preselected duration. A task with higher priority preempts the current task and begins executing. Because the system saves a task's states, registers, and stack area, a task can resume at any time. More important, programmers do not have to make allowances for other tasks that may be running concurrently—while a task is active, it "thinks" that it has full unrestricted access to the 68000.

SHARING THE SYSTEM BUS

Consider that the Amiga can simultaneously read the disk, play four channels of audio, and show 16-color low-resolution bit-plane graphics and eight sprites with virtually no slowdown of the 68000 processor. This is possible largely because of the way various subsystems share the bus.

The Amiga's 68000 runs at 7.15909 MHz, while its memory runs at twice that speed. Most of the instructions in the 68000 alternate between using the bus and doing internal calculation. In this situation, the memory can run at its top speed and still leave every other bus cycle free.

The bus sharing takes place in subdivisions of the time the electron gun takes to draw one line of pixels and do a horizontal retrace, approximately 63 microseconds (μs). This divides into approximately 226 memory-access cycles of 280 nanoseconds (ns) each. The Copper, Blitter, and 68000 access memory on the even cycles (0, 2, 4, . . .); the odd cycles (1, 3, 5, . . .) are reserved for four cycles of memory-refresh DMA, three cycles of disk DMA, four cycles of audio DMA (enough for four channels), 16 cycles of sprite DMA (enough for eight sprites), and 80 cycles of bit-plane DMA (enough to show a 16-color low-resolution image). The DMA circuits on each chip "know" when their slots occur on each horizontal line and automatically initiate the DMA transfer without involving the 68000.

In many cases, the Copper and the Blitter aren't active, leaving the 68000 running at full speed. (Actually, some instructions need the bus at odd times; if the bus isn't available, the 68000 will insert wait states until the bus-arbitration PAL [programmed-array logic chip] signals that the bus is free by asserting the 68000's $\overline{\text{DTACK}}$ line. This happens more frequently as the custom chips demand more of the bus's cycles.)

Several things modify this bus sharing. If you use more than four bit planes of low-resolution display, or more than two high-resolution bit planes, the bit-plane DMA will steal some memory cycles from the 68000. Both the Copper and the Blitter have higher priority than the 68000 and will get the cycles they need first. If the Blitter senses a memory-bus request by the 68000, it will halt within a few cycles to let the 68000 use the bus; then it will again take over the bus and continue. This gives the 68000 some cycles even when the Blitter is running. If you set an internal "Blitter priority" bit, however, the Blitter steals *all* the cycles it needs from the 68000. Even this is not as bad as it sounds; whenever any of the above items steals cycles, it still performs its function faster and more efficiently than the 68000 could have.

MULTITASKING

The Amiga is multitasking—that is, it can work on more than one thing at a time. At a low level, for example, this means that the Amiga can move sprites, read from the disk, and play music at the same time. At higher levels, several programs can run simultaneously in overlapping windows.

The Amiga's multitasking ability comes from several features we've already discussed: the interrupt structure and the Exec multitasking routines in ROM. Interrupts, which are

(continued)

routed through and prioritized by the peripherals/sound chip, initiate task switching. For example, when a peripheral signals its need to do I/O, the interrupt goes through the peripherals/sound chip and causes the peripheral's interrupt routine to execute (assuming that no interrupt of higher priority is running). The interrupt routine either handles the peripheral's need immediately or notifies a task to do so, then the routine ends. In both cases, the Amiga then calls the task rescheduler, which ensures that the appropriate task has the chance to use the system.

THE COPPER

The Copper is a coprocessor inside the animation chip that runs its own program. The execution of this program is tied to the progress of the electron beam as it draws the video display. Because of this capability, the Copper is most often used to control the graphics and sound parts of the custom chips, thus relieving the 68000 of the same task. The Copper reads its instructions from memory and uses DMA to write from its program (in memory) to the registers in itself and the other two custom chips. (According to Jay Miner, this is not so strange if you look at the three chips as "one big custom chip.")

The Copper's instruction set has only three instruction types: *move* immediate data to a register, *wait* until the electron beam passes a given position, and *skip* past the next instruction if the electron beam is past a given location. The beam-position values are accurate to the exact line vertically and to 4 low-resolution pixels (or 8 high-resolution pixels) horizontally.

The Copper's versatility can be extended by clever use of its registers. For example, you can get the Copper to jump to a given instruction by causing the new address to be placed in the Copper's internal "program counter." By setting bit 15 of the INTREQ (interrupt request) register, the Copper can cause a level-6 interrupt, which should lead to a more complex 68000 routine that will service the situation that caused the interrupt.

One important aspect of the Copper is that, while it is waiting for the electron beam, it is off the system bus and does not tie up any resources. This is in contrast to many systems that tie up their processors while waiting for a given beam position. Because of the Copper, the 68000 is never tied up for several milliseconds waiting for a display-related event.

The Copper can handle many basic system functions without the intervention of the 68000. For example, it can refresh certain bit-plane and sprite values that must be restored at the beginning of each frame. It can also change the color palette in midscreen (giving you more than 32 colors on the screen), change the graphics mode (saving memory), and update the display memory without glitches by changing an image *after* the electron beam has drawn it for the current frame.

The Copper programs give the maximum amount of control over the video display and events of that

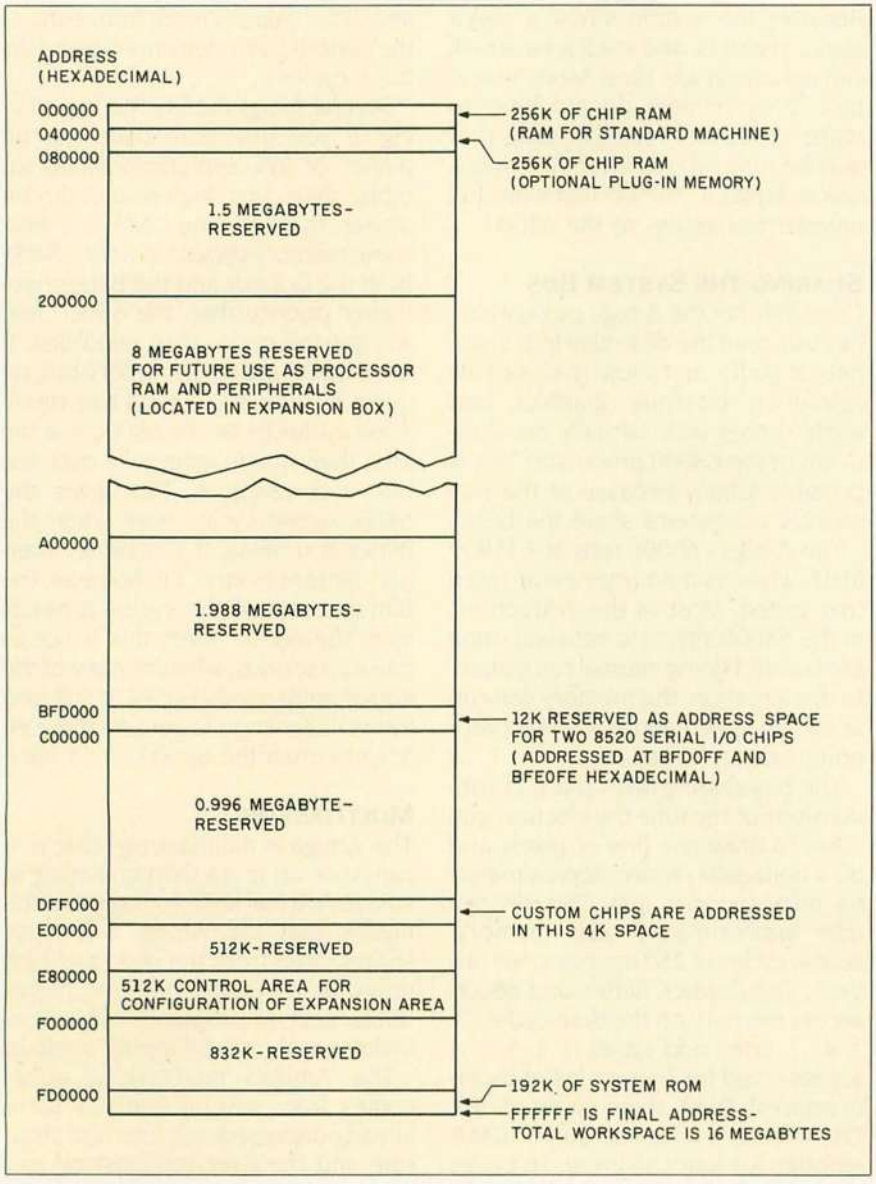


Figure 2: *The Amiga memory map.*

periodicity, but most programmers will not create them directly. Many of the ROM routines that accomplish high-level tasks manipulate Copper programs to get their work done.

Memory Space

The first 512K bytes of memory is called the *chip memory* (see figure 2 for a memory map). Any function performed by the custom chips—bit-plane and sprite images, Copper programs, and other data (covered below)—must be in this memory area.

Of course, in the standard 256K-byte Amiga (or the expanded 512K-byte version), the chip memory is also used for everything else a computer needs RAM for. Commodore/Amiga may announce an expansion box at a later date that can accommodate various peripheral cards and up to 8 continuous megabytes of memory. Normal programs and data should be placed there, leaving the display memory free for its specialized uses.

Graphics

The Amiga's graphics are, in a word, breathtaking—in both their quality and their speed. The machine's major graphic components are the playfield, the sprites, the Blitter, and the animation and text routines.

The Playfield

A *bit map* is an area of memory that the computer interprets as a rectangular array of pixels (dots); most computers have some bit-mapped graphics capability. Many machines form different colored pixels by grouping two or more adjacent bits in the bit map. The Amiga, however, uses only one bit per pixel in its bit map (this is called a *bit plane*) and "stacks" separate bit planes together to get different colors (see figure 3). (The colors available are not "hard-wired" into the machine but are specified in a *color-register table*, also known as a *color palette*.) An image created by multiple bit planes is called a *raster*. The *playfield* is the bit-mapped graphics display that comprises most of the Amiga's video display.

The Amiga can stack up to five bit planes to get a maximum of 32 colors. The color-register table contains 12-bit values that can specify any of 4096 different colors. Therefore, the Amiga can draw images that use any 32 of these 4096 colors.

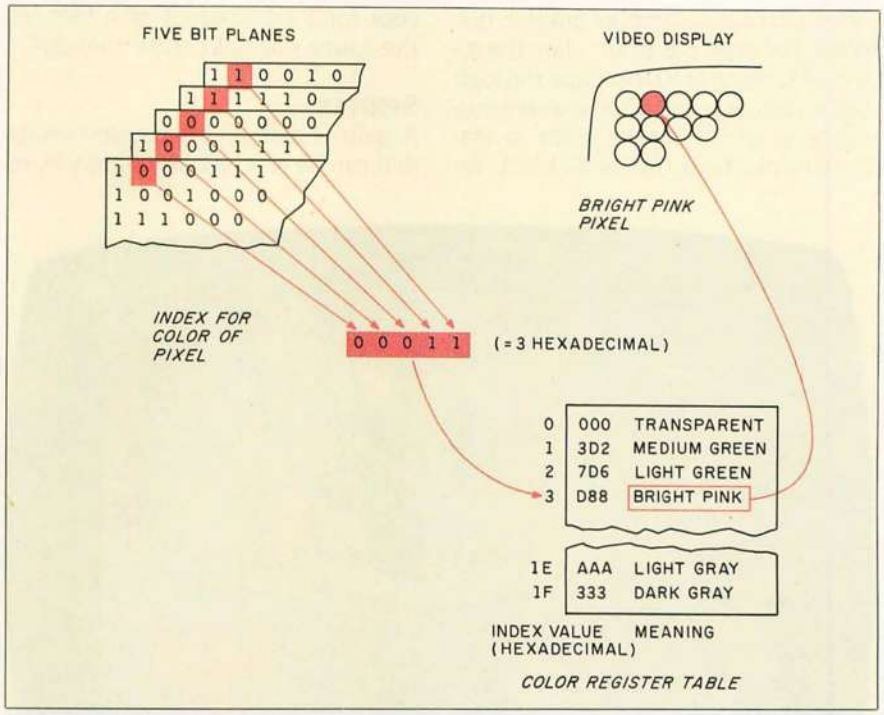


Figure 3: *Amiga playfield graphics. The bits from a given position in each bit plane combine to create an index into the color-register table. The selected entry in the color-register table determines the color of the pixel.*

The Amiga has five bit-mapped resolutions. Four of them come from two horizontal resolutions (320 pixels per line, low resolution, and 640 pixels per line, high resolution) times two vertical resolutions (200 visible lines per screen, *noninterlaced* frame, displayed every 1/60 second, and 400 visible lines per screen, *interlaced* frame, displayed in two passes every 1/30 second). These can take anywhere from a minimum of 4000 bytes (for a 320- by 200-pixel image) to 32,000 bytes (for a 640- by 400-pixel image). Photo 3 shows an example of the 320 by 200 mode.

The fifth mode, called *hold-and-modify*, uses six bit planes in a way that can simultaneously display all 4096 colors on screen. In this mode, the top 2 bits of a pixel control the interpretation of the bottom 4 bits, which may represent either a color-register table value for that pixel or a modification to one component of the previous pixel's color. Using hold-and-modify, you can display all 4096 colors on an analog RGB (red-green-blue) color monitor.

A playfield image can be much larger, both horizontally and vertically, than the screen area used to display it. By manipulating several register values, you can scroll an image horizontally, vertically, or both, with very little effort. (When the total image is wider than its displayed part, the last pixel on one line and the first pixel on the next are not adjacent and are separated by a fixed number of bytes. The Amiga makes use of *modulo registers* to make the manipulation of two such bytes as fast and as simple as if they were contiguous.)

Another display option is called the *dual-playfield mode*. When you use this mode, up to six bit planes are divided into two separate images of up to three bit planes each, with one image having priority over the other. This

(continued)

often simplifies complex graphic displays. For example, to simulate the effect of looking at a landscape through binoculars, you can scroll a wide landscape playfield "underneath" a stationary playfield that is all black except for a transparent area that lets the lower playfield show through.

SPRITES

A *sprite* is a small bit-mapped image that can be repositioned simply by redefining the horizontal and vertical values for its upper left corner; sprites are independent of the playfield and appear to be over or under each other and the playfield(s) according to a specified priority.

The Amiga has eight hardware sprites, each of which can have three colors (sprites are two bit planes deep, and each 2-bit pixel translates to three colors plus transparency). Amiga sprites are 16 low-resolution pixels wide by any height. Each pair of sprites shares a different three-color color-register table (for example, sprites 0 and 1 share color registers 17, 18, and 19, sprites 2 and 3 share 21, 22, and 23), allowing the eight sprites to use up to 12 colors. Adjacent sprites (0 and 1, for example) can be *attached*, meaning that their four bit planes are combined; an attached sprite pair can then use color registers 17 through 31 to display up to 15 colors.

As happens often in the Amiga, complexity underlies apparent simplicity. A sprite is actually a 16-bit value with a specified horizontal displacement for the current line of the video display. In *manual mode*, you are responsible for creating the sprite's image on a line-by-line basis (few people will use this mode directly). In *automatic mode*, however, you activate the sprite's DMA circuitry, which looks to a data structure that contains the line-by-line position and shape of the sprite and draws it automatically. In addition, you can redefine the sprite indefinitely while the electron beam creates the video display. The sprite DMA circuitry accepts a list of sprite position and shape-definition words and draws them as long as the bottom line of one occurrence and the top line of the next are separated by at least one video line (note that this is *without* intervention of the Copper).

Photo 3: *Robocity, an example of Amiga graphics in the 320- by 200-pixel 32-color mode.*

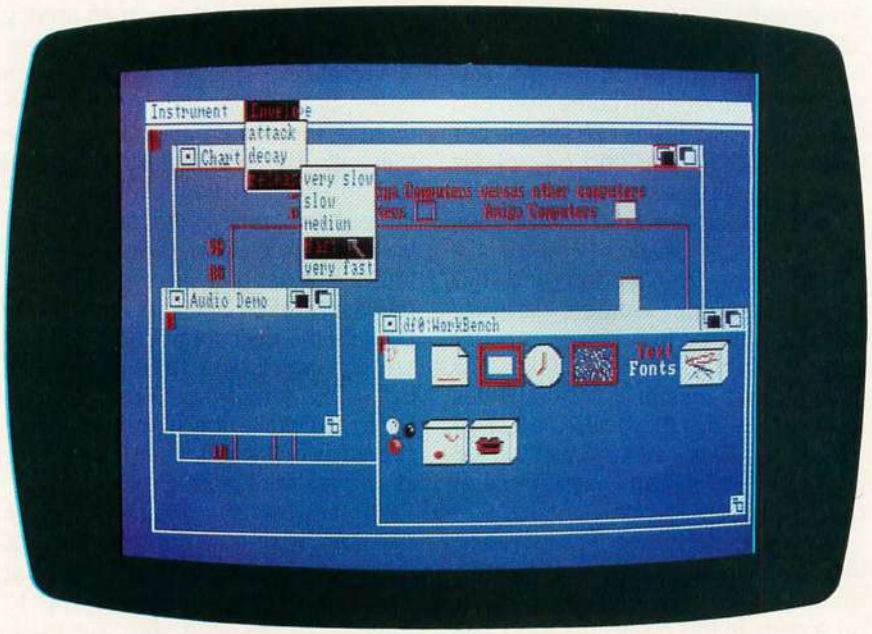


Photo 4: *The Workbench display. This is an example of the 640 by 200 mode.*

THE BLITTER

The Blitter is an area of the animation chip that controls a DMA channel dedicated to drawing lines and manipulating rectangular areas of the playfield. Its name comes from an earlier

(continued)

term, *bit-blt*, which means "bit-mapped block transfer." Miner calls it a *Bimmer*, for "bit-mapped image manipulator," because of its extended capabilities, but "Blitter" is used exclusively in the Amiga's documentation.

When manipulating blocks of an image, the Blitter (when properly set up) takes care of a number of "housekeeping" tasks that, in other computers, tie up a lot of the processor's time. These include: masking out the bits just outside the image that belong to the same memory word as the desired bits; shifting the image several bits horizontally to match the word alignment of the destination; and filling an area bounded on the left and right by two nonhorizontal single-pixel lines (this is the basis of its area-fill capability).

The Blitter distinguishes itself from other bit-blt devices by its ability to combine up to three source areas in one of 256 ways to become the destination area. (If we call the sources A, B, and C and their inverses $\overline{A}$, $\overline{B}$, and $\overline{C}$, these combine in eight ways: ABC, AB$\overline{C}$, A$\overline{B}$C, . . . , $\overline{A}\overline{B}\overline{C}$. There are 256 possible combinations of these eight terms.)

When being used to draw lines, the Blitter can draw lines as 1s, 0s, or a specified pattern; it can also draw single-bit-wide lines, which are needed to bound an area to be filled.

In both its line-drawing and area-manipulating operations, the Blitter must have a moderate amount of "housekeeping" calculations done first. Given the speed and simplicity of the resulting operation, the setup calculations are not an unreasonable overhead; however, you can deal with the Blitter on a higher level using some graphics routines in ROM.

Animation Routines

The animation routines that are part of the Amiga's ROM form the basis for the most sophisticated color animation the personal computer market has ever seen. One of the demonstrations we saw, Robocity, showed five cartoon characters roaming across the screen. The resolution was very good—only when you looked closely could you see the "jaggies" that proved you weren't looking at a hand-drawn cartoon.

The basic element in the animation subroutine is the GEL, *a graphics element.*

Animation is accomplished through a few subroutine calls that draw a linked list of things needing to be animated. The basic element in the animation subsystem is the *GEL*, or graphics element. There are four types of GELs: VSprites, BOBs, AnimComps, and AnimObjs.

VS*prite* stands for "virtual sprite." A VSprite is a data structure in memory, closely tied to a hardware sprite, that is managed by the animation routines. By letting the routines manage the mapping of VSprites to hardware sprites, you can (with some limitations) define more than eight VSprites and let the routines keep track of the details automatically. VSprites can also be clipped to display themselves only within a certain horizontal slice of the display.

BOB stands for "Blitter object." A BOB is an image that acts like a sprite, but the animation routines use the Blitter to "paste" the image onto the playfield and (optionally) restore the image that was "underneath" the BOB. A BOB is defined by the combination of a BOB data structure and a VSprite data structure, both of which point to each other. One advantage of a BOB over a VSprite is that a BOB is drawn into a playfield—this means it can be of any width and it can have as many colors as the playfield (up to 32). BOBs can also be clipped to appear only in a certain rectangular window.

An *AnimComp* is an animation component, one part of an *AnimObj*, an animation object. If your AnimObj is a figure of a man walking, its AnimComps will probably include BOBs for a torso, a head, two arms, and two legs. Each AnimComp includes several views of the same object (e.g., arm bent, arm straight) with an associated time that must elapse before progressing from one view to the next. Once all this is assembled, repeated calls to the Animate routine substitute new views (as determined by their timer constants) into the linked list of GELs before drawing the items in the list.

You can do *sequenced drawing animation* by specifying a series of views that describe a repeated motion and by specifying an offset to add to the object's position each time the routines cycle from the last view to the first. For example, take the example of a cat walking two steps to the right in six views so that view 1 appears natural when it is shown after view 6. By specifying the correct horizontal offset to the right (which gets added every time the image cycles back to view 1), the Animate routine will automatically draw the six views in the correct order and position to make the cat appear to walk across the entire width of the screen.

Alternatively, you can have the Animate routine do *motion-control animation*, in which the next position of a BOB is automatically calculated from its current position and four *x*- and *y*-axis velocity and acceleration values. (You can also do this with a "ring" of BOB views that cycle as in sequenced drawing animation.)

Another routine, DoCollision, detects two types of collisions, GEL-to-GEL collisions and boundary collisions (collisions of GELs with rectangular boundary windows); the routine then executes a given collision-handling routine from a table of 16 possible routines. GELs can be coded so that only certain types of collisions register (useful in a game, for example, to detect missile-target collisions but not missile-missile collisions).

Text

The Amiga treats text as a special kind of graphics. Fonts are described

(continued)

by a Text Font (TF) data structure that allows the creation of either monospaced or proportional characters of any height. To save room with larger fonts, a font may define anywhere between 1 and 255 characters. Two fonts, Topaz 8 and Topaz 9, are in the Amiga ROM. The first gives 40 characters per line in normal resolution, 80 in high resolution; the second gives 30 and 60 characters per line, respectively. Additional fonts may be loaded into and removed from RAM as needed.

The Amiga uses the ROM routine TxWrite to draw a given message to a given location. The text can be drawn in one of two user-definable "pen" colors and in one of three drawing modes: JAM1, an overstrike mode; JAM2, a mode that draws both the character in one color and the "white space" behind it in another; and Complement, which inverts every pixel that corresponds to a pixel of the character being drawn.

As in the Apple Macintosh, fonts may be modified by combining any of several styles: underline, italic, boldface, and extended. However, unlike the Macintosh, the Amiga text-drawing routine looks for a separately defined font that contains the needed style(s). If this fails, a future revision of the text-drawing routine may try to modify the existing "normal" version of the font (this is the only way of achieving font styles in the Macintosh).

Audio Hardware

The Amiga includes four hardware channels of sound that are largely controlled by DMA circuitry, independent of the 68000. Audio-controlling routines in part of the Amiga's ROM extend these capabilities, allowing you to work with the Amiga's sound capabilities at a higher conceptual level and to manipulate the sound channels "on the fly" without "glitching" the output.

The four channels of sound, numbered 0 through 3, are converted to analog signals, filtered through a low-pass filter, and mixed into two separate output signals, one combining channels 0 and 3, the other, channels 1 and 2. The filter begins to attenuate frequencies between 5.5 kHz and 7.5 kHz and effectively eliminates any higher frequencies. This eliminates much *aliasing*, which is distortion that occurs when a signal that was sampled too infrequently is played back.

Fonts may be modified by any combination of several styles: underline, italic, boldface, and extended.

The sound channels can be controlled directly by the 68000, which gives you complete control over the sound but keeps the 68000 from doing other work. In most cases, you can get the sound you need by letting the DMA channels produce the sound from a table of values (called a *sound table*) that describe one or more cycles of the needed waveform.

In the Amiga, each audio DMA channel includes registers that give the channel's loudness, point to a 16-bit-wide table of sound-table bytes (the values are fetched a word at a time and must be stored on even byte boundaries), and establish the time that must elapse before the next sound byte is sent out. This last is a *period register*, which contains a value that is decremented every 279 ns; the next value from the sound table is sent out when the counter reaches zero, and the register is reset to its original value. When the pointer to the sound table reaches its last value, the pointer is reset to the start of the table. In this way, the audio channel continues to produce the given waveform without supervision until it is explicitly turned off.

Sound channels 0 through 2 can be *attached* to the channels directly above them to modulate the output of the higher channel. When a channel is attached, the 16-bit words that make up its sound table are not interpreted as two 8-bit sound values. Instead, the data words are interpreted as volume or period values for the current value in the channel being modulated (i.e., the volume value will determine the current loudness of the channel, and the period value determines how much time passes before the channel sends out the next value in *its* sound table). You can manipulate these values to cause either amplitude modulation, frequency modulation, or both.

Audio Software

The ROM contains three kinds of routines. The first, channel-allocation routines, allow you to allocate, use, and discard a channel without keeping track of which channel it is. If you have more than four "virtual" channels open, the four with the highest priorities are mapped to actual hardware audio channels.

Second, the DMA-control routines control the way the audio DMA channel manipulates the hardware audio channel via the various registers and the sound table. In addition, you can cause the channel to send a user-specified signal bit to an existing task (which may then trigger some event) when the sound channel has played a given number of repetitions of the sound table; this allows tasks to manipulate the Amiga based on the sound channel's activity.

Third, the envelope-generator routines automate the task of varying the amplitude envelope that determines how slow or fast a note changes volume when it is played. To use these routines, you must create a table of four slope/destination values that describe an ADSR (attack, decay, sustain, release) envelope. (The ADSR envelope tells you how fast the note gains volume as soon as it starts, what its maximum value is, how fast it decays once it reaches that value, on what level it remains as long as the note is sustained, and how fast it returns to zero once the note is released. You can draw such an envelope with four line segments; the

(continued)

Amiga defines the ADSR envelope by giving the slope and destination *y*-axis values for each line segment.) As with the audio DMA, the software involved can be told to send a signal bit to a given task when the envelope is completed.

One potentially significant piece of code is a library of text-to-speech routines that is included with the standard Amiga computer. These are transient routines that are loaded from disk to memory when needed; they are capable of "speaking" normal English text in a variety of pitches and rates via one of the sound channels. We heard the routines and found their output to be heavily inflected but understandable even with our eyes closed (a test that many text-to-speech algorithms fail).

INTUITION

Intuition, the user interface of the Amiga, sits on top of the disk operating system and provides the icon-oriented, mouse-based, desktop-metaphor interface popularized by the Apple Macintosh. Intuition complements the architectural philosophy and the graphics capabilities of the computer by managing a complex windowing system and providing access to multitasking capabilities.

Intuition allows programs to execute, each in its own window, simultaneously. Each program opens a *virtual terminal* that has access to all the system resources. Even though multiple programs can execute simultaneously, only one can accept input and display its menu bar. You can select which program does this by clicking on its window; this window will also display special command messages from the system. Different programs can share the video display, or a single program can create several virtual terminals.

To support the simultaneous display of different resolutions and graphics modes, Intuition uses *screens*, which are rectangular areas that occupy the full width of the video diplay. Screens have predefined resolutions, color palettes, and height and contain one or more windows. A bar at the top of each screen identifies the screen.

All screens have pull-down menus. Pressing the right mouse button, which generally summons a menu, transforms the screen bar into a menu bar (a strip containing the names of the menus that apply to the currently active window). The screen bar also contains two boxes that, when clicked with the left mouse button (generally responsible for selecting things), move the screen to the top or bottom of the stack of screens. You can select menu items in the conventional way, although there are several new features. Pull-down menus, for example, can have up to two levels (see photo 4). Menus can contain options that, when selected, persist until other, mutually exclusive choices are made. Programs may allow you to use command-key/letter combinations to select commonly used menu items.

Programmers have considerable flexibility in designing the menus. For example, menus can appear in multicolumn format and contain graphics. Menu items can, when selected, be marked with checks, and they can automatically display command-key/letter alternatives.

Windows, which appear within screens, can support all of the Amiga's graphics, text, and animation features. Since Intuition opens application programs in windows, applications must specify their graphics, text, and color requirements by selecting or creating an appropriate screen. Intuition will support as many screens and windows as can fit in memory, but only one window and, by extension, one screen can receive input at a time. As a virtual terminal, programs need not know if they are active; they can continue to process data as long as they don't need any external input.

You can activate a window either by moving the on-screen pointer inside it and clicking the mouse button or by moving an icon into it. Closing a window causes the last activated window to be reactivated. Windows can include any of several features, including vertical and horizontal scroll bars, title bars, window-dragging areas (used to drag the window to a new position), depth arrangers (which move the window to the top or bottom of a stack of windows), sizing boxes (which allow you to change the window's size), and close boxes (which close a window).

Intuition supports *backdrop windows*, which open behind all other windows and cannot be moved, sized, or depth-arranged. The application program is entirely responsible for maintaining its contents, and normal windows appear on top of it. A graphics program, for example, may use a backdrop window as the primary drawing area and call a normal window to show you a palette of colors from which to choose.

Programmers can specify whether an application will refresh its window when partially covered and uncovered, or whether memory must be allocated to save the concealed portions of the window. A third choice, *super bit map*, reserves enough memory to store an image larger than the windowing system will display. Intuition automatically adjusts and displays as much of the super bit map as it can. Programmers can use this technique to create windows whose contents scroll. They can also determine where windows will appear, what color to use when drawing the border and text, whether the window will have a border, and whether to include a window title.

REQUESTERS, ALERTS, AND GADGETS

Requesters are pop-up information boxes that wait for either keyboard or mouse input from you. Normally, you will have to click the left mouse button over an "OK" area before continuing, although you may be able to switch to a different window (the requester will still be there when you return to the first window). With a single call, programmers can attach requesters to a window or to the double click of the mouse button.

Programmers have access to predefined system requesters, like the "Please Insert Disk XXXX" requester.

(continued)

To use a custom requester, however, the programmer must specify things like gadgets (discussed below), borders, requester text, and, if desired, hand-designed bit-mapped images.

Alerts are special screens that carry absolutely crucial information. They differ from requesters in that no screen or window can obscure them, and users must act immediately on the information before proceeding. *Recovery alerts* require immediate responses; *dead-end alerts* tell users that the system has crashed.

Screens, windows, requesters, and alerts all use *gadgets*, which are input devices that attach to windows, requesters, and alerts. *System gadgets* include window-sizing gadgets, window-/screen-dragging areas, depth arrangers, and close boxes.

Programmers can design their own gadgets by specifying border shapes and colors, describing the select box of the gadget, providing gadget text, supplying a memory buffer for the gadget response, and defining how the gadget will behave.

In addition to system gadgets, programmers can select among Boolean, string, integer, and proportional gadgets. *Boolean gadgets* are true/false devices that return a value only when selected. *String gadgets* return a string from the keyboard. *Integer gadgets* return integer values. *Proportional gadgets*, which return a value proportional to their positions on either the horizontal or vertical axis (or both), are similar to scroll bars on the Macintosh. A programmer can customize the appearance of the *knob* (the element that slides along the axis of movement) to something different from the default rectangular shape.

The Workbench

Intuition includes Workbench, an iconic, window-based command interface. The Workbench area is a four-color screen with 640- by 200-pixel resolution. It is both a *screen* on which disks will open and application programs will run and an *application* that keeps track of Workbench objects and displays information using Intuition windows. The Workbench automatically opens when you enter a disk containing it. By opening the Workbench library, programmers can access Workbench functions to create and manipulate the Workbench and Workbench objects.

In the Workbench, users can open and close disks, tools, projects, drawers, the clipboard, and the trash can. Opening a *tool* (Amiga's term for an application program) creates a window on the current screen. Tools create *projects*—files associated with the tool. (A document file, for example, is the project of a word-processing application.) Opening a tool automatically opens a window that lists the names of available projects. Opening a project icon automatically opens the tool associated with it.

Workbench also supports *extended selection*, a method of selecting multiple items that will be operated on in the order they were selected. For example, you can select a word processor and three projects (documents); the word processor will then work on the projects in the order in which they were selected.

Drawers are Workbench icons that contain tools, projects, and other drawers; when opened, they display their contents as icons in a window. To add an item to the drawer, either drag the item's icon into the window of an opened drawer or drop it over a closed drawer's icon. You can delete an item by moving its icon over the trash can, a special drawer in each disk drawer that contains deleted objects.

The *clipboard* is a special object that lets you transfer data between tools (programs). The clipboard stores the last text, graphics, or data cut from a project as a RAM-based file (disk-based if the clipping is too large for memory). By using the clipboard, you can quickly transfer information between tools or projects.

Programmers can also design custom screens, in which they can specify things like the screen size and position, the number of colors available, the screen titles, and the default font.

(continued)

The Workbench also contains a program called Preferences that lets you set things like the maximum time for two clicks to be considered a double click, the monitor type, the speed with which keyboard keys repeat, the interval before they begin to repeat, and the presence of optional peripherals, including printers, modems, and touchpads.

The Preferences program can also give you access to a command-line interface (CLI), which allows you to get work done via typed-in commands. The CLI, which opens as a window under Workbench, will not be heavily documented in the standard manuals, and you will normally not see the icon associated with it. The CLI uses commands that are similar to those of Microsoft's MS-DOS. It can, for example, examine directories, run programs, and redirect input and output; in essence, it gives programmers access to the operating system that is "underneath" Workbench.

CAVEATS

This product preview is unusual in that we looked at the Amiga in an earlier state than we usually do for other product previews. We feel justified in doing this for two reasons: First, the hardware *was* in its final state (the custom chips were working on the production-version motherboard, although the PROM [programmable read-only memory] chips did not contain the final version of the ROM code); second, the Amiga should be announced by the time you read this, and we feel that the technology used here is noteworthy. BYTE will print a formal review of the Amiga as soon as we can get our hands on a finished machine.

We wrote this product preview after two days with the Amiga engineering staff, much study of four volumes of technical documentation and several user manuals, and subsequent telephone conversations. At the time we saw the machine, neither the ROM code nor the operating system had been "frozen," which limited the amount of software we could see to the Workbench user interface, several demonstration programs, and an early version of the Graphicraft drawing program.

All the screen shots in this product preview came from working (though still unfinished) software, but most of what we've written about the Amiga's software came from the documentation or the engineers. According to Commodore/Amiga, the BASIC that will be bundled with the system will have extended graphics and sound capabilities driven by calls to the ROM routines. Table 1 gives a list of products for the Amiga that we learned of from their respective manufacturers.

Table 1: *This is a list of the announced hardware and software for the Amiga.*

Hardware

20-megabyte hard disk,
  20-megabyte tape backup,
  multifunction card,
  2400-bps modem (**Tecmar**)
Laser disk,
  Color digitizer,
  Genlock peripheral—allows computer's
  display to overlay an external video signal (**Commodore**)

Software

Pascal,
  Linkage Editor,
  Overlay Loader,
  Macro Assembler (**Metacomco**)
Turbo Pascal (**Borland International**)
Logo (**The LISP Company**)
Propaint,
  Business Graphics,
  Graphicraft,
  Animation (**Island Graphics**)
Enable/Write (**The Software Group**)
Textcraft (**Arktronics**)
Musicraft (**Commodore**)
Harmony and four-octave music keyboard,
  Pitchrider (**Cherry Lane Technologies**)
C Compiler (**Lattice**)
General Ledger,
  Accounts Receivable,
  Accounts Payable (**Chang Laboratories**)
7 Cities of Gold,
  One on One,
  Archon,
  Adventure Construction Set,
  Pinball Construction Set,
  Skyfox,
  Financial Cookbook,
  Deluxe Music Construction Set,
  Black Knight,
  Video Construction Set,
  Return to Atlantis (**Electronic Arts**)
Communications package (**Software 66**)
Welcome Aboard,
  Print Shop,
  SynCalc,
  Mindwheel (**Broderbund**)
Keyboard Cadet,
  The Halley Project (**Mindscape**)
All **Infocom** Interactive fiction products

CONCLUSIONS

We were impressed by the Amiga's detail and speed of the color graphics and by the quality of its sound system. The interlocking features of the Amiga—its custom chips, multitasking support, multiple DMA channels, shared system bus, display-driven coprocessor, system routines in ROM, etc.—point to a complexity of hardware design that we have not seen before in personal computers. (It's interesting to note that the Macintosh's complexity is in its software and that, according to several third-party developers who have used both computers, the Macintosh is harder to program.) The synergistic effect of these features accounts for the speed, quality, and low cost of the Amiga.

We are also very excited about the inclusion of the text-to-speech library in the Amiga. This means that *any* Amiga program can potentially create voice output, something that has never been common in personal computers because it was never, until now, a standard feature.

The hardware looks good—we have seen it work—but we saw very little software actually working (a painting program, the Workbench "desktop," and a few demonstration programs). However, we think this machine will be a great success; if that happens, the Amiga will probably have a great effect on other personal computer companies and the industry in general. ■