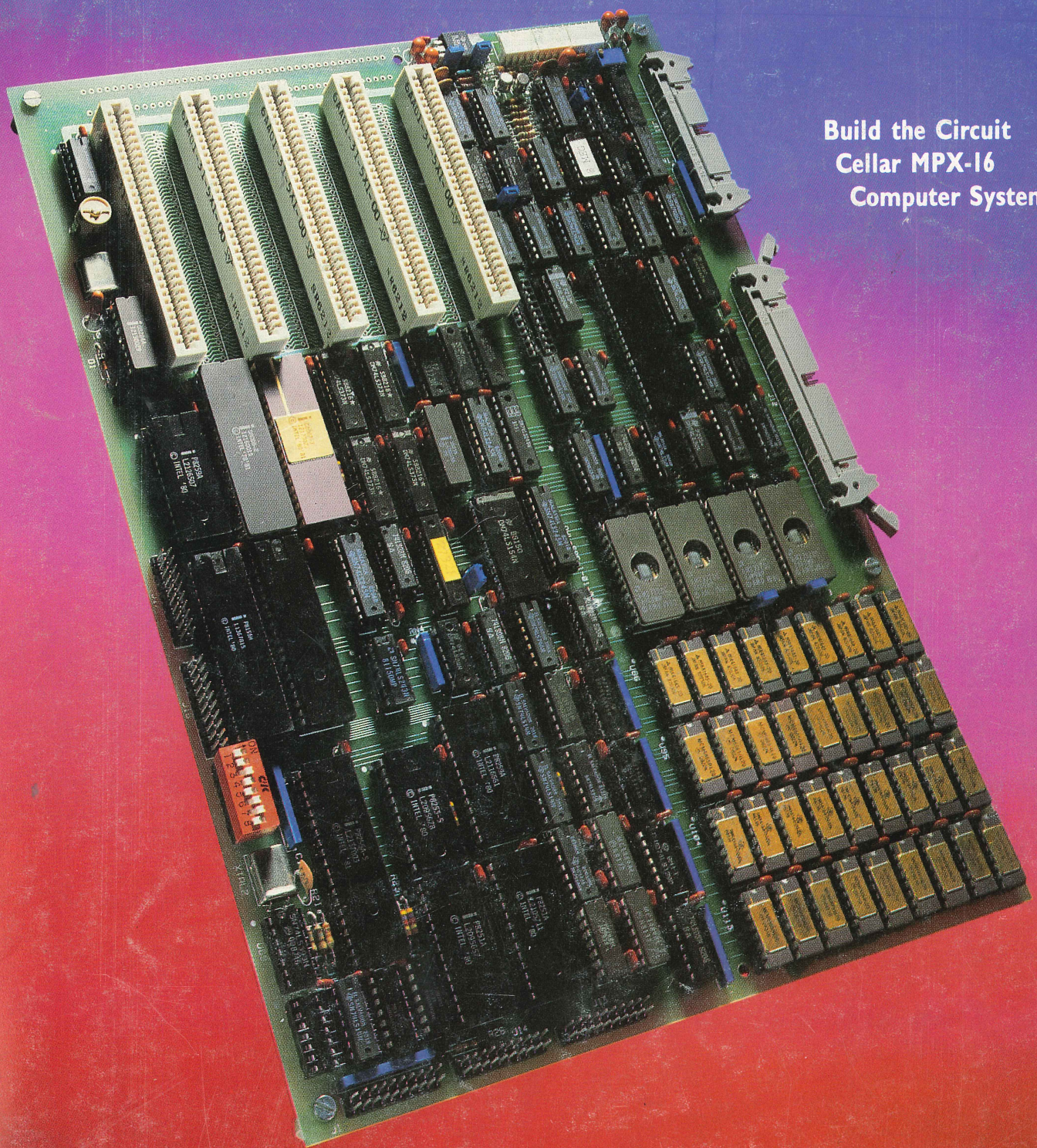
BYTE®

the small systems journal

NOVEMBER 1982 Vol. 7, No. 11
$2.95 in USA
$3.50 in Canada/£1.85 in U.K.
A McGraw-Hill Publication



Build the Circuit Cellar MPX-16 Computer System

GRAPHICS

Build the Circuit Cellar MPX-16 Computer System

Part 1

Any peripheral device designed to be installed in the IBM Personal Computer can be plugged into this 8088-based system.

Steve Ciarcia
POB 582
Glastonbury, CT 06033

Let's see. What's next? A computer-controlled bird bath? An early-warning radar transponder? How about a satellite-tracking system? Something simple.

After this series of articles is over, I am going to write about uncomplicated construction projects for a while. You'd think that after doing 50 or so projects over the past four years I'd have learned to recognize when uncontrolled invention was getting the upper hand, as it did in this month's project.

I was caught up in the fervor that resulted from the introduction of the IBM Personal Computer. As I had already written two articles on the Intel 8088 microprocessor used in the IBM machine, I quickly decided to jump on the bandwagon and purchase the first IBM PC (as it's called by its owners) that I could get my hands on. I've found myself in agreement with the prevailing opinion that the IBM PC is a solid design and well supported, but it's relatively expensive to upgrade.

The design of the MPX-16 had to be a team effort.

Somewhere along the way I had the absurdly ambitious idea of presenting a Circuit Cellar construction project on building a full computer system based, like the IBM PC, on the Intel 8088 microprocessor. (After all, I've done many microprocessor projects before.) And somewhere further along the way I decided to do it.

Design Concepts

Certain questions had to be addressed, of course. Should I try for a 10-chip design or splurge and make it 20 chips? What kind of expansion-bus scheme should the system use? What about supporting software? Could I design a small 8088-based computer and call it a development system?

The initial stages of design moved very quickly, and in a few weeks I had put together a prototype of a 64K-byte 8088-based trainer or development system. It was a compact design with limited input/output (I/O) capability but with relatively little expansion potential, lacking an expansion bus. It could have served well as a Circuit Cellar project. However, owning a so-called development system has come to mean that you are on your own: you won't get much support for either software or hardware. If the project was to have any real significance, support had to be available, and the burden of providing support would have been mine.

IBM and IBM Personal Computer are trademarks of International Business Machines Corporation.

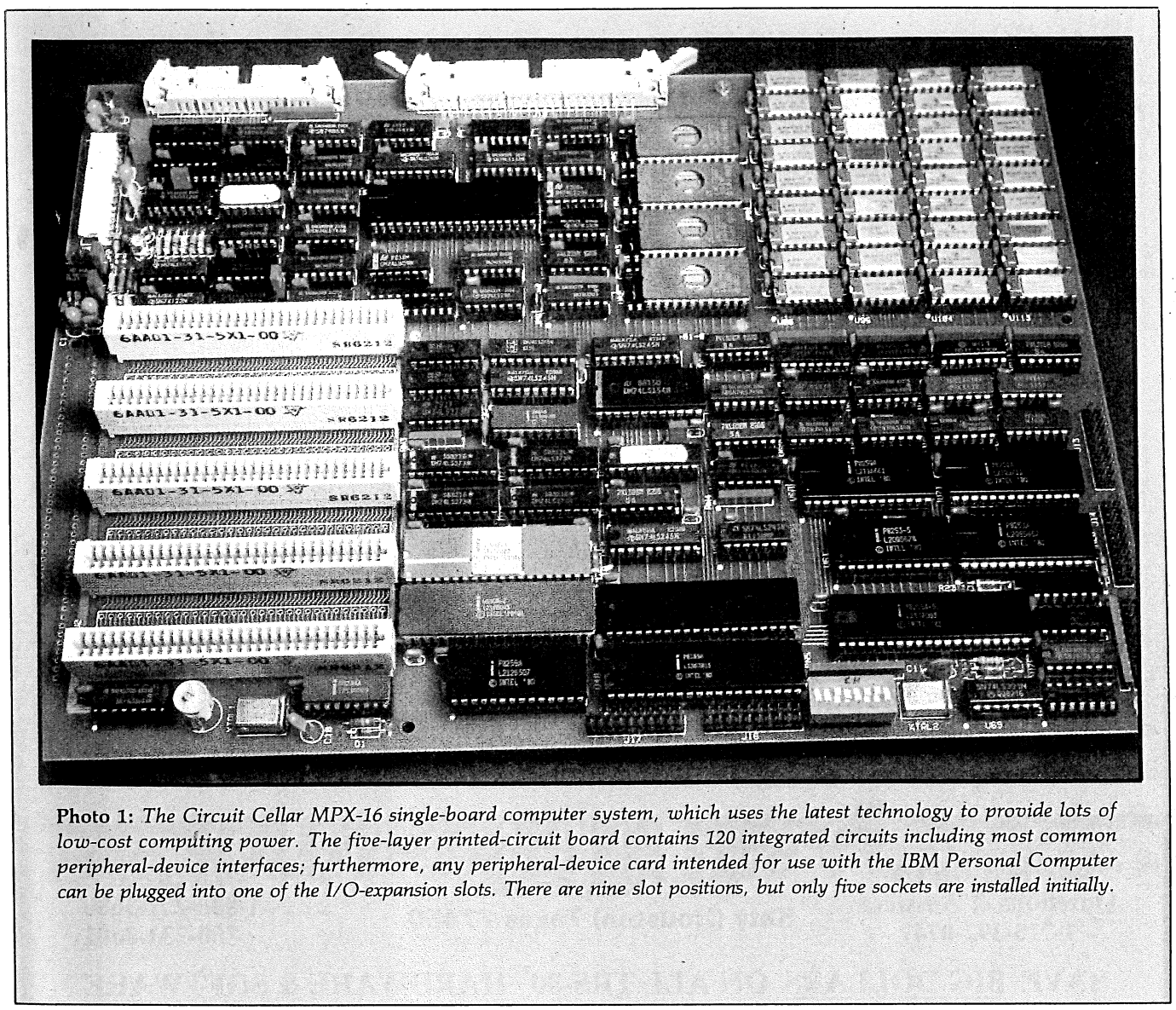


Photo 1: *The Circuit Cellar MPX-16 single-board computer system, which uses the latest technology to provide lots of low-cost computing power. The five-layer printed-circuit board contains 120 integrated circuits including most common peripheral-device interfaces; furthermore, any peripheral-device card intended for use with the IBM Personal Computer can be plugged into one of the I/O-expansion slots. There are nine slot positions, but only five sockets are installed initially.*

There was only one answer: compatibility. The project would have to be compatible with peripheral-device expansion boards and software designed for some popular computer system. The logical choice, given that I intended to use an 8088, was to make my project compatible with the I/O-expansion bus of the IBM Personal Computer. (The only real alternative was the S-100 bus, but somehow in my fervor the IBM PC route seemed more natural. S-100 fans should look up reference 1.) Consequently, IBM PC memory and I/O-expansion boards, available from numerous sources, could be used to expand this new computer.

But in making this choice, I opened Pandora's box. I was already committed to producing the article, but making my little prototype bus-compatible with the IBM PC was like fitting the *Queen Mary* into a bathtub. Scratch one prototype; start thinking about the "system board."

Ten minutes later, I realized that this would have to be a team effort. I would need assistance in developing the design, the documentation, and the software, so I enlisted help from a few friends and other engineers to form the design team.

At that point, team (or rather, committee) dynamics came into play. If you give a committee 3 square inches of empty space on the printed-circuit board, they'll want to increase performance by packing 10 more integrated circuits into it. Essentially that's what happened to my little trainer board. Not only would the resulting system be bus-compatible with the Personal Computer, but it would overcome some of the expansion weaknesses of the IBM machine by incorporating many peripheral devices as part of the basic design. Instead of a board that could be expanded into a system, this new computer would be a complete system that had been shrunk to fit on a single board.

Design Characteristics

The result of our effort is called the Circuit Cellar MPX-16 Computer

1. 5-MHz Intel 8088 main processor
2. optional Intel 8087 numeric coprocessor
3. 256K-byte on-board-user-memory capacity, with parity
4. two RS-232C serial input/output ports
5. three parallel input/output ports
6. on-board controller for either 5¼-inch or 8-inch single- or double-density floppy-disk drives (up to four)
7. supports the CP/M-86 operating system directly, with BIOS in EPROM
8. nine expansion slots (five connectors provided), bus-compatible with IBM Personal Computer
9. sockets for 64K bytes of 24- or 28-pin EPROM
10. four independent DMA channels
11. sixteen levels of vectored interrupts

Table 1: *Features of the MPX-16 computer system.*

System, shown in photo 1. Consisting of a single 9- by 12-inch five-layer printed-circuit board containing 120 integrated circuits (ICs), the MPX-16 is completely compatible with the expansion bus of the IBM Personal Computer and contains the following features: provision for an optional Intel 8087 math coprocessor, 256K bytes of RAM (random-access read/write memory), serial and parallel I/O ports, floppy-disk controller, expansion slots, and support for Digital Research's CP/M-86 operating system. (A more detailed list of features appears in table 1.)

The MPX-16 constitutes a complete, single-board computer system, using the latest technology to provide lots of low-cost computing power. It is designed to utilize all the expansion peripherals that are available for the IBM machine, and because it has so many capabilities built in, you don't have to use up expansion slots for simple jobs like interfacing a printer. Programmers, however, will undoubtedly want more memory. To meet this demand, additional memory boards can be plugged in to provide the system with one full megabyte of user memory. A hard-disk drive can be added easily, and an 8087 mathematics coprocessor can be inserted to multiply the system's raw computing power by a factor of 10 to 100.

The MPX-16 is designed initially to use CP/M-86, but it will ultimately accommodate Microsoft's MS-DOS and any other software that does not use unique features of the IBM Personal Computer. The greatest difference is this: as a stand-alone system, the MPX-16 communicates with the user through a serially interfaced display terminal instead of through a memory-mapped video display and separate keyboard. The BIOS (basic input/output system) module of CP/M-86 is contained in a set of EPROMs (erasable programmable read-only memories) on the board.

The MPX-16 is almost complete on a single board. In addition, you need merely a power supply, a serial terminal, and one floppy-disk drive. To start operation, you just turn on the power, insert a CP/M-86 disk, and start the bootstrap operation. For the sake of appearance, though, you may

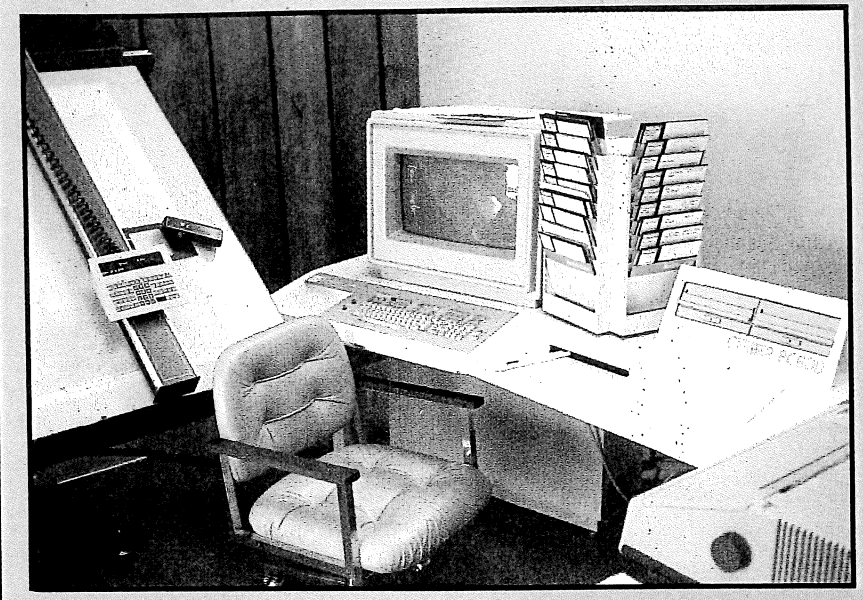


Photo 2: *The Gerber Scientific Instrument Company PC-800 CAD (computer-aided design) machine used to lay out the MPX-16's five-layer printed-circuit board.*

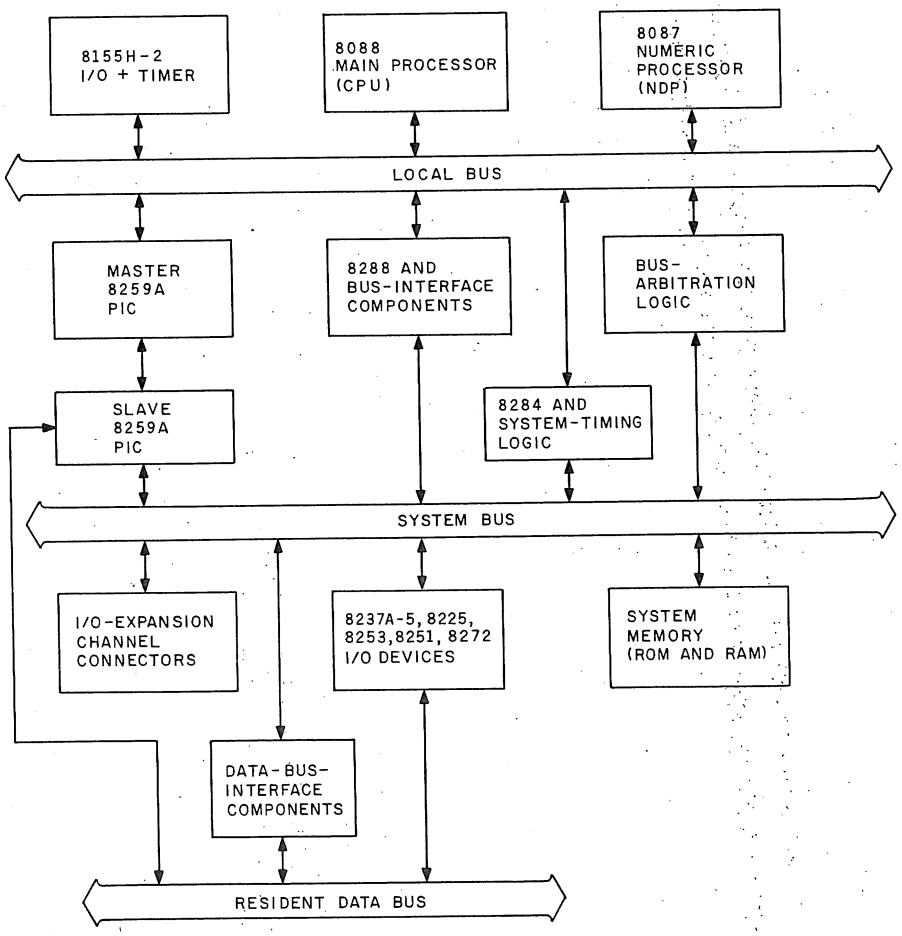


Figure 1: *Simplified, high-level block diagram of the Circuit Cellar MPX-16 computer system. The abbreviation "PIC" stands for "programmable interrupt controller."*

want to wrap up the whole thing in a suitable enclosure; one should be available by the time you read this.

Pragmatic Considerations

Obviously, it is impossible to describe the construction of such a powerful computer in detail in a single article. Even dividing it into three parts, as I plan, will be a difficult task; it will take us three months to print schematic diagrams of the entire computer in the magazine. I'll try to be as explicit as I can concerning how the circuitry works, but you must understand from the outset that this is no beginner's project.

The condensation of information here is counterbalanced by the support available from The Micromint, where you can get assembled and tested systems, blank printed-circuit boards, and complete documentation containing all the circuit diagrams plus much more detail than can be included in these brief articles.

Finally, before I start the details, I'd like to say something about the MPX-16's circuit board. Printed-circuit boards are available for building most recent Circuit Cellar projects, and this project is no exception. The only departure from the norm this time is in the complexity of the board.

The MPX-16 contains 120 IC packages. To keep its size manageable, we had to use a multilayer printed-circuit board instead of the relatively simple double-sided boards used in smaller-scale projects. With the aid of a Gerber Scientific Instrument Company PC-800 CAD (computer-aided design) machine, shown in photo 2, we eventually arrived at a 9- by 12-inch board with five layers of connecting traces. This is significant because multilayer boards cost about 10 times as much as standard double-sided boards. But even with an expensive circuit board, I believe that the MPX-16 has unbeatable performance for its cost.

MPX-16 Overview

The functional organization of the MPX-16's onboard components is illustrated in two levels of detail. Figure 1 shows a simplified, high-

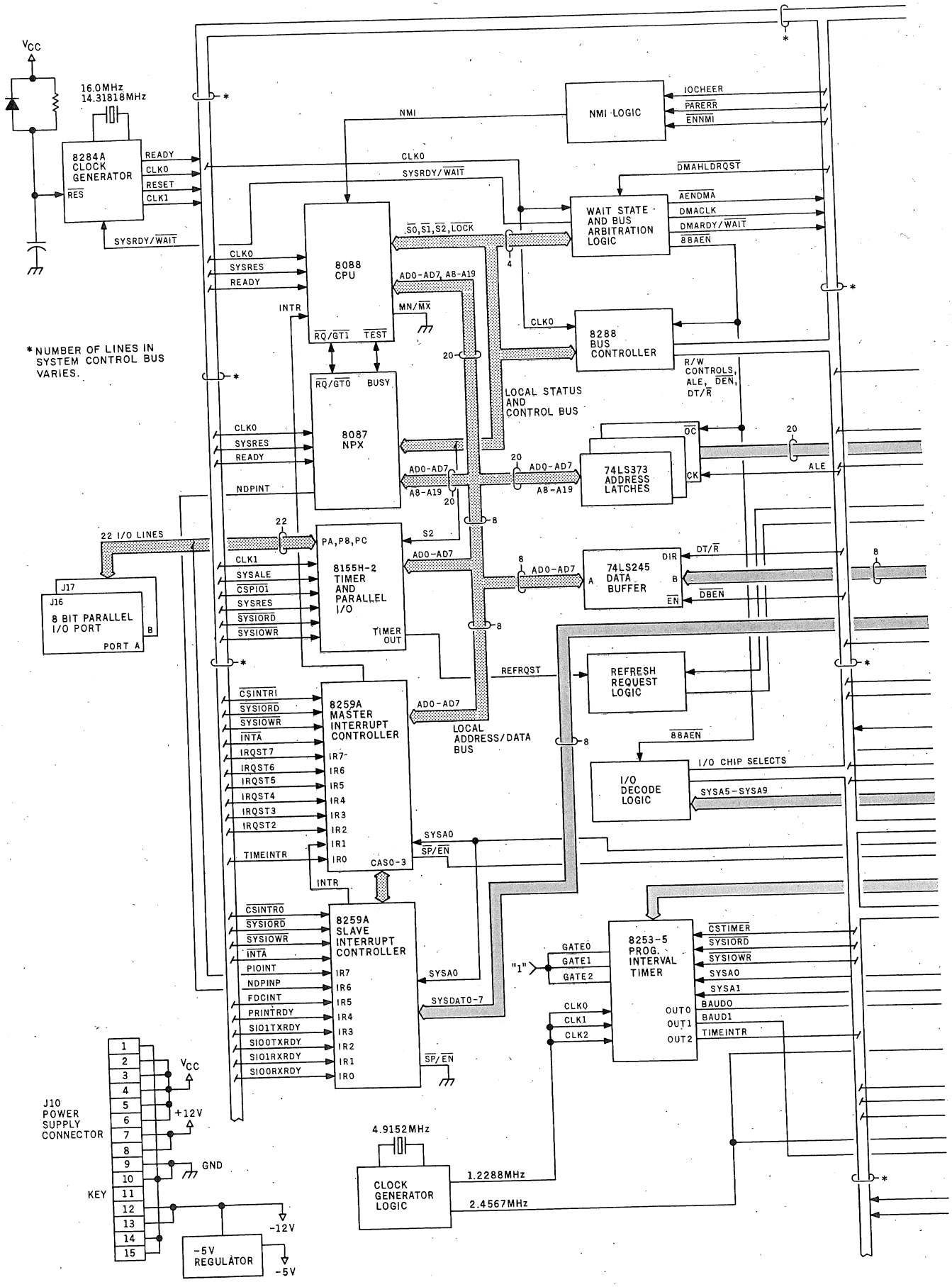
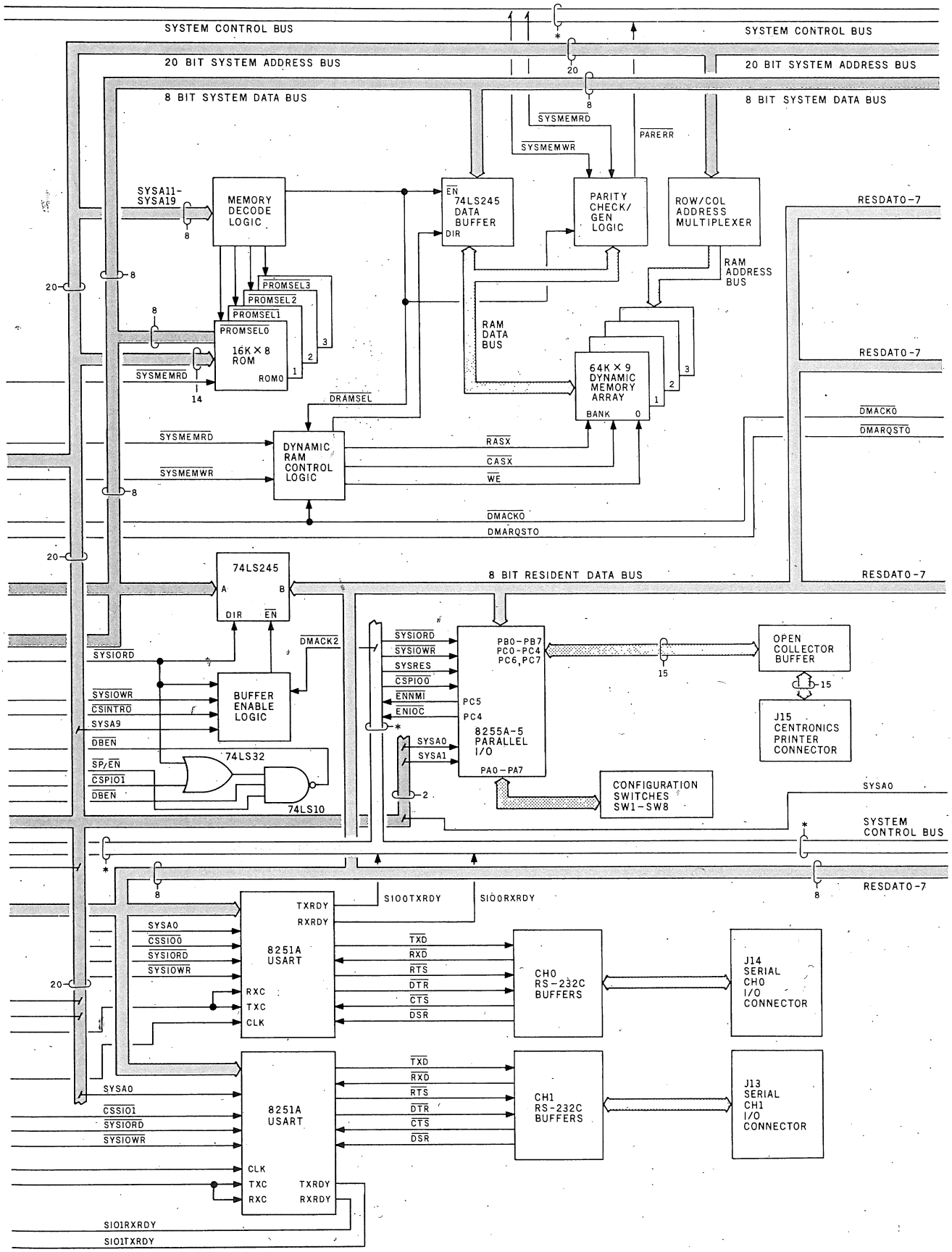


Figure 2: *Complete, detailed flow diagram of the MPX-16 system. (The diagram is continued on page 86.)*


SYSTEM CONTROL BUS
20 BIT SYSTEM ADDRESS BUS
8 BIT SYSTEM DATA BUS
SYSMEMRD
SYSMEMWR
PARERR
SYSA11-SYSA19
MEMORY DECODE LOGIC
EN 74LS245 DATA BUFFER DIR
PARITY CHECK/ GEN LOGIC
ROW/COL ADDRESS MULTIPLEXER
RESDAT0-7
RAM ADDRESS BUS
RAM DATA BUS
PROMSEL3
PROMSEL2
PROMSEL1
PROMSEL0
16K X 8 ROM
ROM0
64K X 9 DYNAMIC MEMORY ARRAY
BANK 0
DRAMSEL
DYNAMIC RAM CONTROL LOGIC
RASX
CASX
WE
DMACK0
DMARQST0
74LS245
8 BIT RESIDENT DATA BUS
SYSIORD
SYSIOWR
SYSRES
CSPIO0
ENNMI
ENIOC
PB0-PB7
PC0-PC4
PC6,PC7
PC5
PC4
8255A-5 PARALLEL I/O
PA0-PA7
SYSA0
SYSA1
OPEN COLLECTOR BUFFER
J15 CENTRONICS PRINTER CONNECTOR
CONFIGURATION SWITCHES SW1-SW8
DMACK2
BUFFER ENABLE LOGIC
CSINTR0
SYSA9
DBEN
SP/EN
CSPIO1
74LS32
74LS10
SIO0TXRDY
SIO0RXRDY
8251A USART
TXRDY
RXRDY
TXD
RXD
RTS
DTR
CTS
DSR
RXC
TXC
CLK
CSSIO0
CSSIO1
CH0 RS-232C BUFFERS
CH1 RS-232C BUFFERS
J14 SERIAL CH0 I/O CONNECTOR
J13 SERIAL CH1 I/O CONNECTOR
SIO1RXRDY
SIO1TXRDY

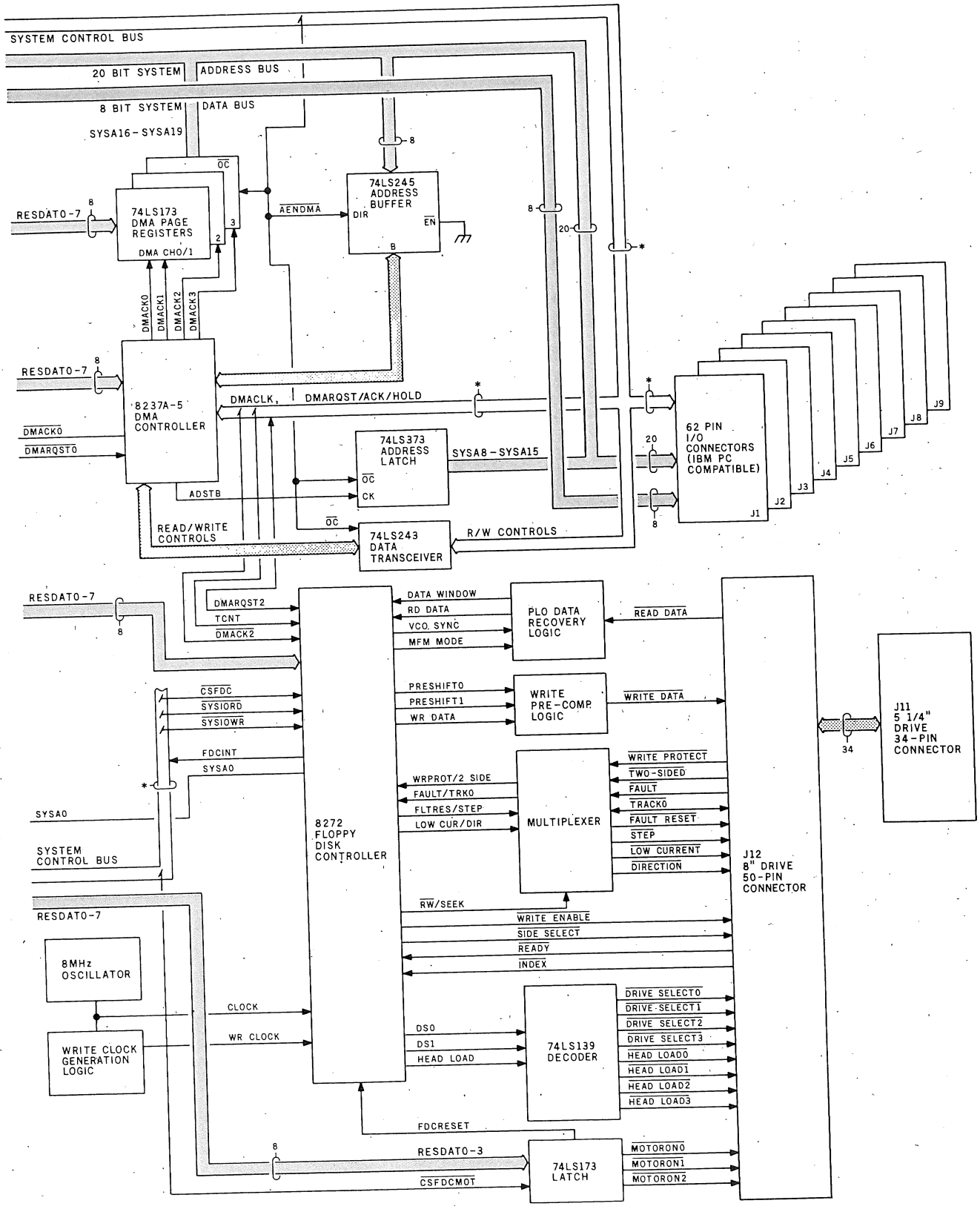


Figure 2: *Continued from page 85.*

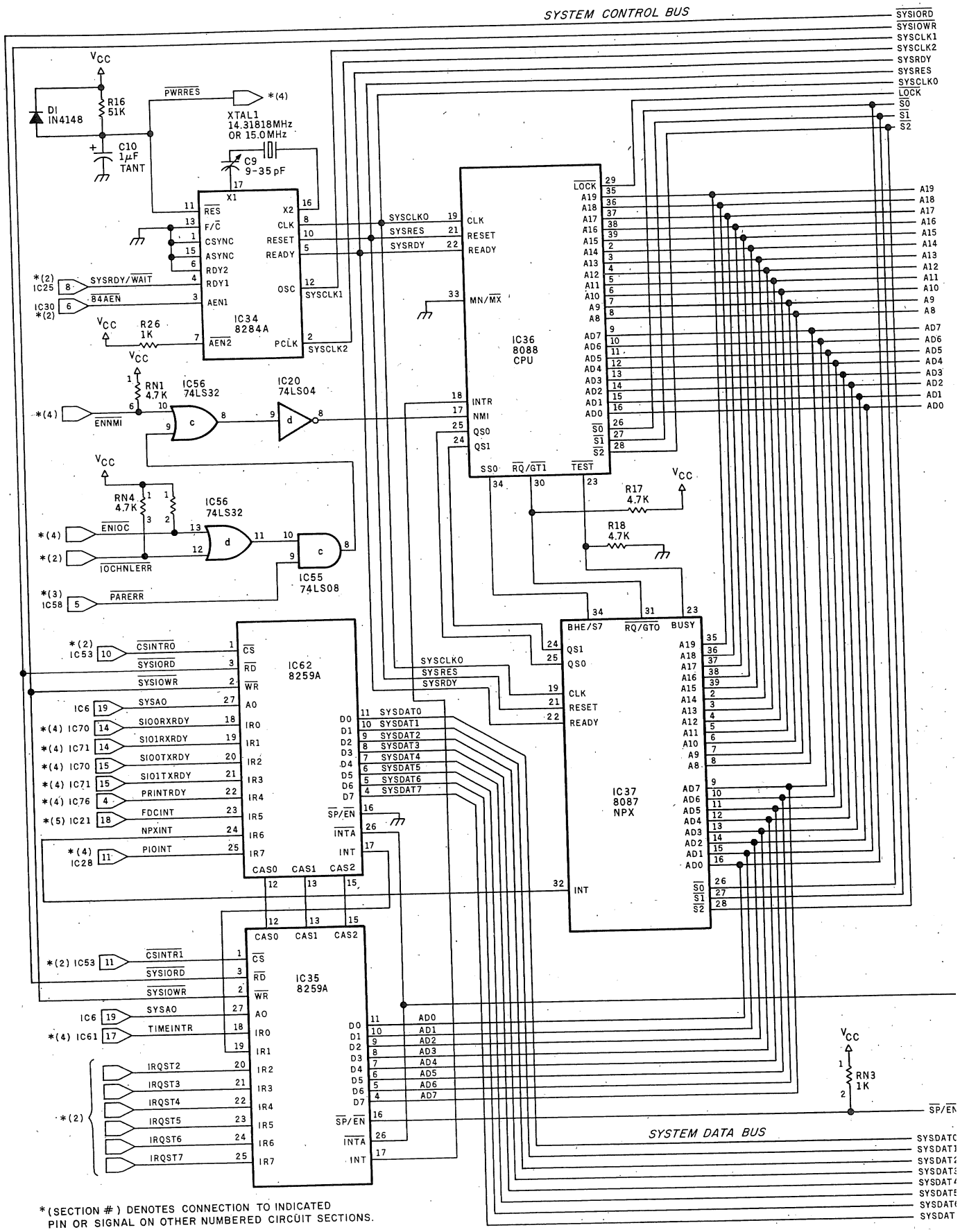
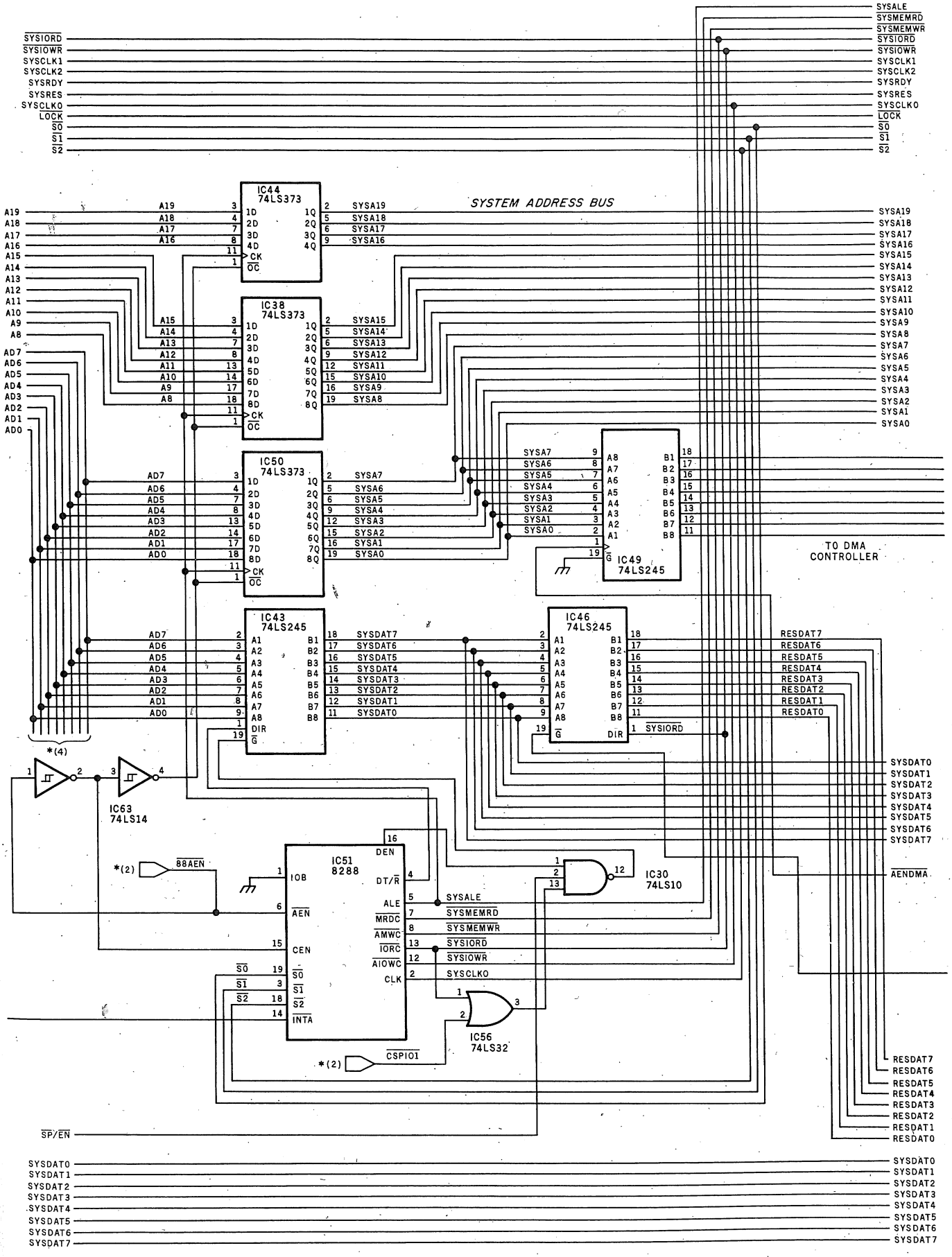


Figure 3a: *Half of section 1 of the schematic diagram of the MPX-16; the second half of section 1 appears as figure 3b on the next t pages. Sections 2 through 5 of the schematic will appear in December's and January's articles. Many connections to other sections the schematic are indicated in this figure by the notation *(n), where n is the section number; IC numbers in the other sections*



given where appropriate. Here are shown the main processor, numeric coprocessor, interrupt controllers, clock generator, bus controller, bus latches, bus transceivers, and miscellaneous components. A table of power connections will be published in the December 1982 BYTE.

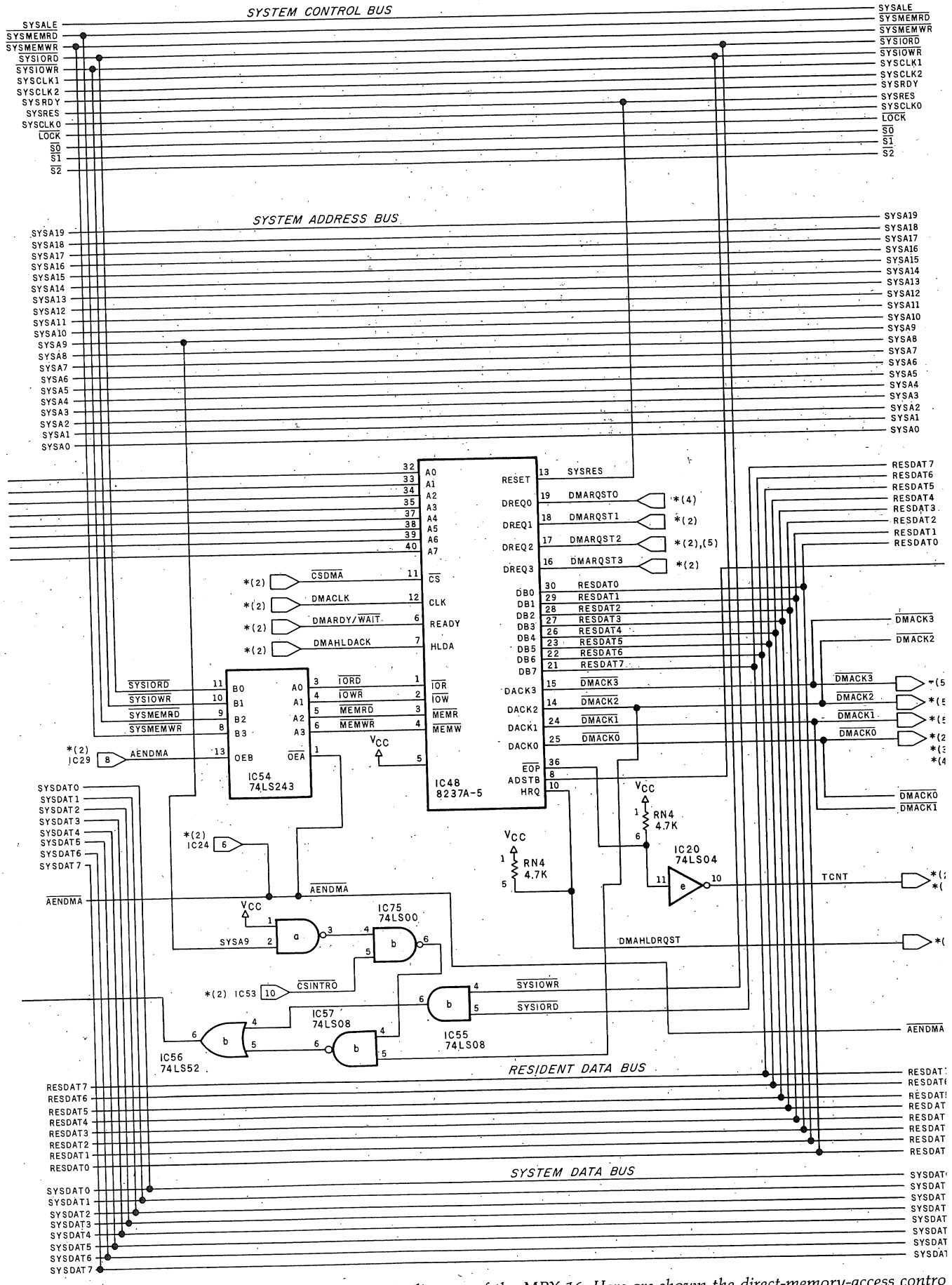
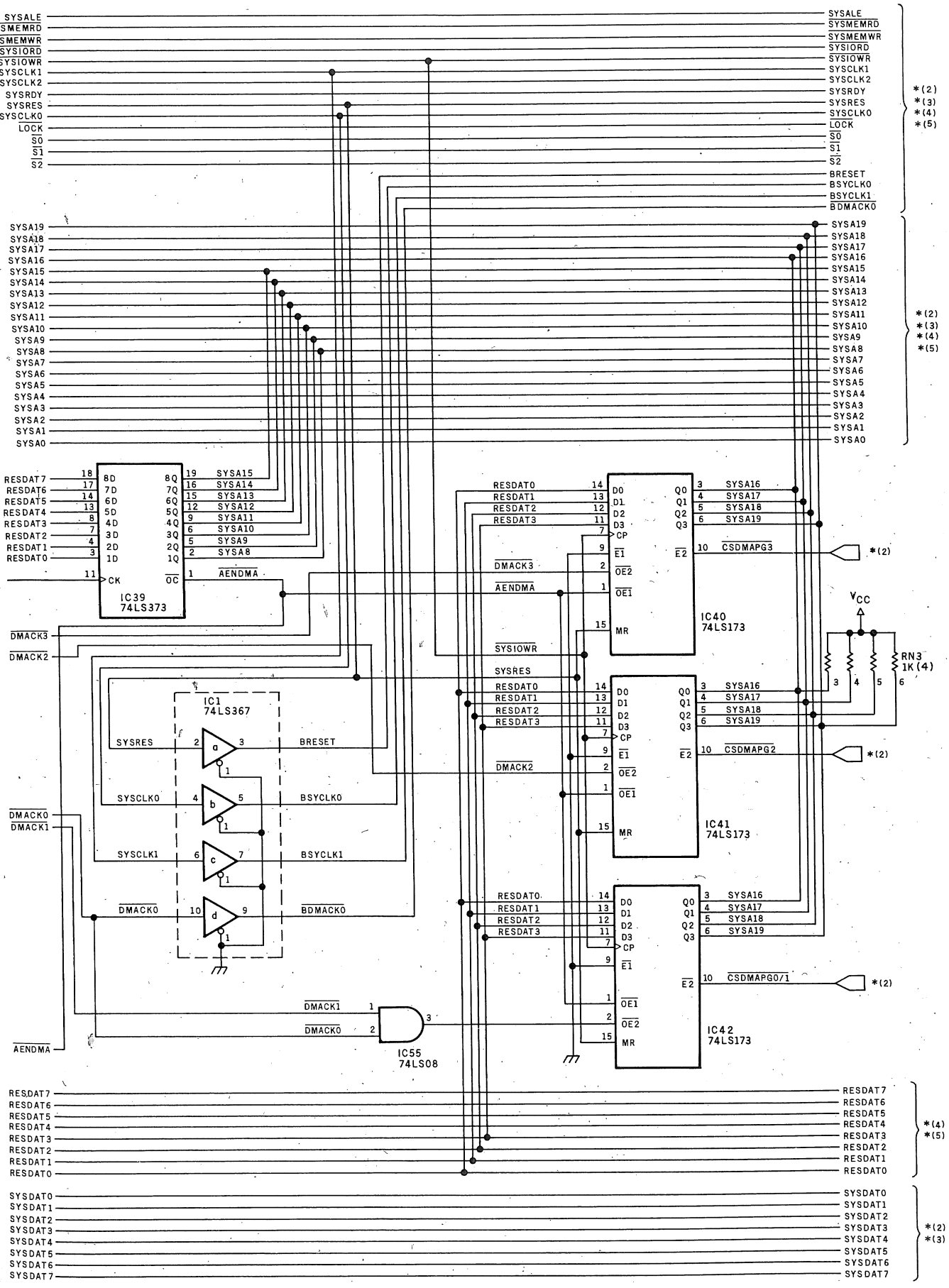


Figure 3b: *Second half of section 1 of the schematic diagram of the MPX-16. Here are shown the direct-memory-access contro control-signal latches, registers, and transceivers; and various logic gates. Note the large number of bus lines for addresses, data, control signals. Sections 2 through 5 of the schematic will appear in December's and January's articles. Many connections to o*



*sections of the schematic are indicated in this figure by the notation *(n), where n is the section number; IC numbers in the other sections are given where appropriate.*

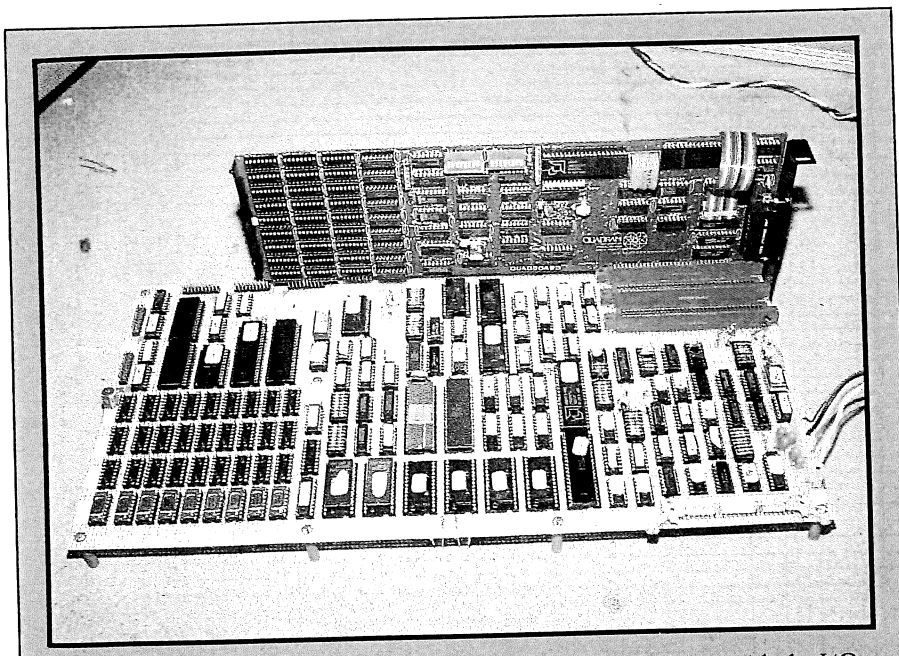


Photo 3: *Prototype of the MPX-16 being tested for compatibility with the I/O-expansion bus of the IBM Personal Computer. A Quadram Quadboard (an expansion card for the IBM PC that contains 256K bytes of memory, a serial port, a parallel port, and a real-time clock) is inserted in one of the MPX-16's slots. It works!*

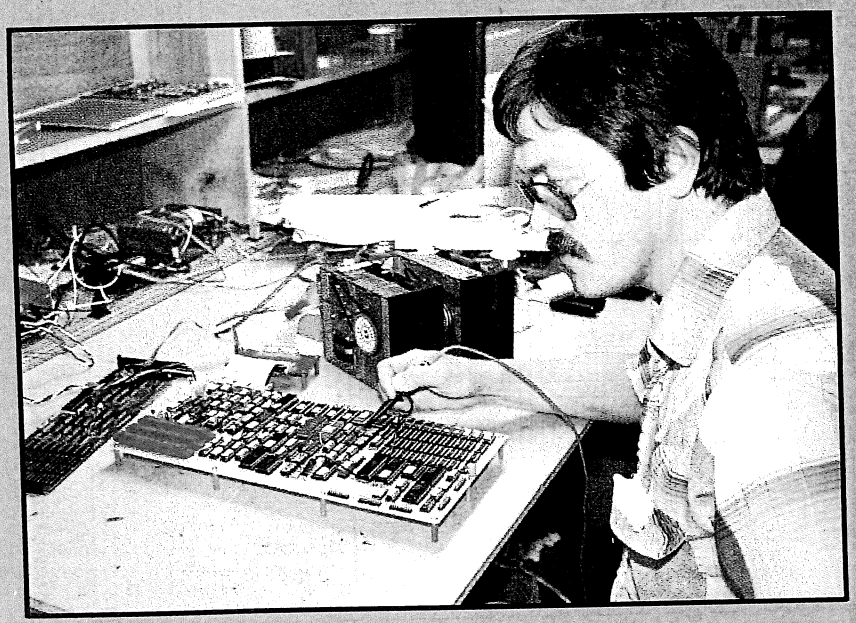


Photo 4: *Engineer Jim Norris of Owl Electronic Laboratories tests the MPX-16 prototype.*

level block diagram, while figure 2 on pages 84, 85, and 86 contains a full flow diagram for all parts of the system.

We'll look at each constituent subsystem separately, beginning with the processor and coprocessor; arrangement of data, address, and control signal buses; clock signals; the NMI (nonmaskable interrupt); and the DMA (direct memory access) subsystem. Section 1 of the schematic diagram, which appears as figures 3a and 3b on pages 88, 89, 90, and 91, contains most of these subsystems, although I do mention some things that will show up in schematic-diagram sections to be published in parts 2 and 3 of this series.

Intel 8088 Processor

The new 16-bit microprocessors are more powerful than their 8-bit predecessors. Not only do they operate at faster speed, but the 16-bit chips manipulate numerical quantities in larger chunks, directly address more memory, and offer the programmer expanded instruction sets. But along with the greater capability comes a new set of computer-design considerations.

An alternative to complete commitment to 16 bits is embodied in the heart of the MPX-16: the powerful Intel 8088 microprocessor. The 8088 uses a 16-bit internal architecture and instruction set and possesses a 1-megabyte memory-addressing capability and a 64K-byte I/O-addressing capability, but communicates through an 8-bit external data bus (sort of like putting its data flow through a funnel). The 8088 has a common internal architecture and complete software compatibility with the pure-16-bit Intel 8086 microprocessor. As a result, the 8088 provides an excellent way for designers, engineers, hobbyists, and students to ease into the world of 16-bit computing by taking advantage of its 8-bit-compatible bus structure.

The 8088 can be used in low-cost systems that employ a few multiplexed-bus support chips such as the Intel 8155 (2K-bit static RAM with I/O ports and timer), 8755A (16K-bit EPROM with I/O ports), and 8185

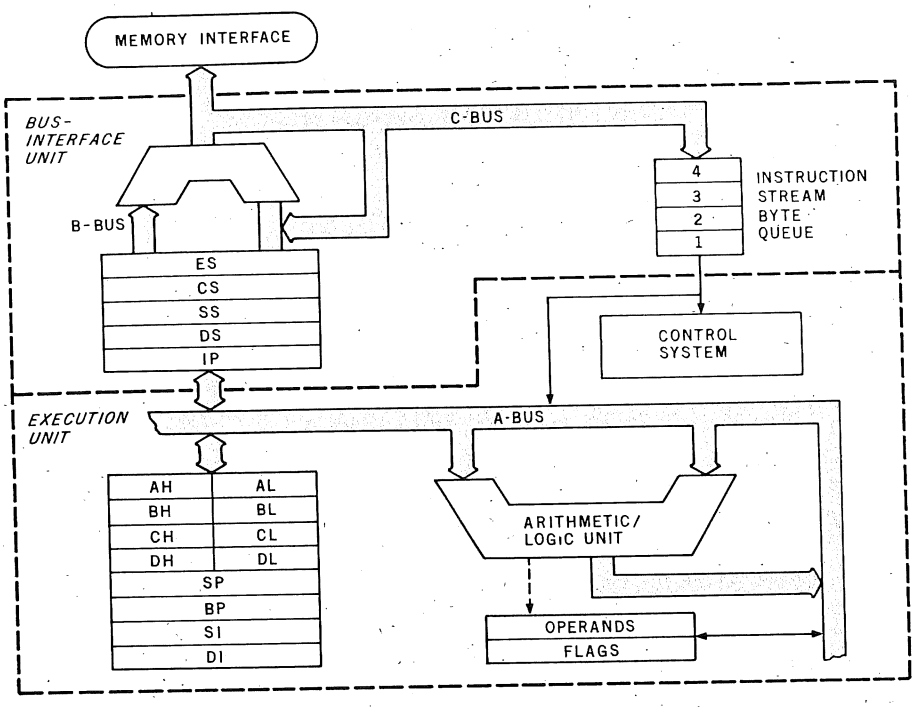


Figure 4: *Functional block diagram of the Intel 8088 microprocessor. Its pipelined architecture, shared with the 8086, increases speed by overlapping the execution of instructions with memory-access operations.*

(1K-byte static RAM). That was the approach I took in my previous article series (see reference 2, listed on page 114). But the power of the 8088 can best be exploited when it serves as the nucleus of a fully expanded system, using its full address space and coprocessing capabilities.

The 8088 microprocessor can be set up to interact with other components in the system in either the maximum or minumum mode. Certain control and status signals differ between the two modes. The selection is made by connecting the MN/$\overline{MX}$ pin on the 8088's package to ground or to +5 V (volts). In the minimum mode, the 8088 functions as a standalone processor, interacting with peripheral devices somewhat like the 8-bit 8085 processor. In the maximum mode, other integrated circuits perform certain specialized functions such as bus control, numeric data processing, and input/output control. In the MPX-16, the 8088 is configured for maximum-mode operation.

Architecture of the 8088

The internal architectures of the 8088 and 8086 processors are identical. A diagram of their internal structure is shown in figure 4. The 8088 contains two logical functional divisions—the bus-interface unit (BIU) and the execution unit (EU)—with a logical pipeline between them that provides an instruction queue.

The 8088 uses *instruction queuing* to increase computing speed. A 4-byte instruction queue holds contents of the four bytes in memory that consecutively follow the instruction being performed by the execution unit. These four bytes of instructions or data are brought into the processor before they are to be executed; therefore, when the EU is ready to execute the next instruction, frequently it or the data required is contained already in the queue. Only when the EU needs to access nonconsecutive addresses (or during a few combinations of especially fast-executing instructions) will time be consumed for memory fetches. By not tying up the memory bus as often as its nonqueuing 8-bit predecessors, the 8088 makes the bus available for use by other powerful support devices. The overall result is increased efficiency and faster processing.

The *execution unit* is where the actual processing of data takes place inside the 8088. It is here that the familiar arithmetic and logic unit (ALU) is located, along with the registers used to manipulate data, store intermediate results, and keep track of the pushdown stack. The EU accepts instructions that have been fetched by the BIU, then processes the instructions. It next returns operand addresses to the BIU, processes the operands, and then passes them back to the BIU for storage in memory.

The role of the *bus-interface unit* is to maximize bus-bandwidth utilization (that is, to speed things up by making sure that the bus is used to its full capacity). The BIU carries out this assignment in two basic ways: first, by fetching instructions before they are needed by the EU and storing them in the instruction queue, and second, by taking care of all operand fetch and -store operations, addre

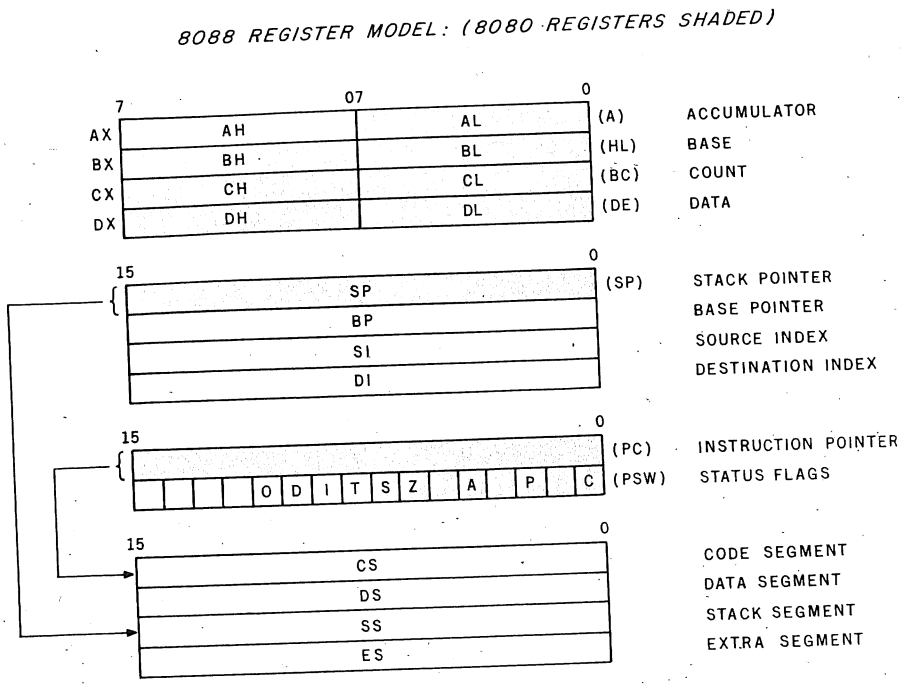


Figure 5: *Programmer's model of the 8088's fourteen 16-bit registers. The shaded registers are the 8080-register subset, that is, the registers that are common to the 16-bit 8088 and its 8-bit predecessors.*

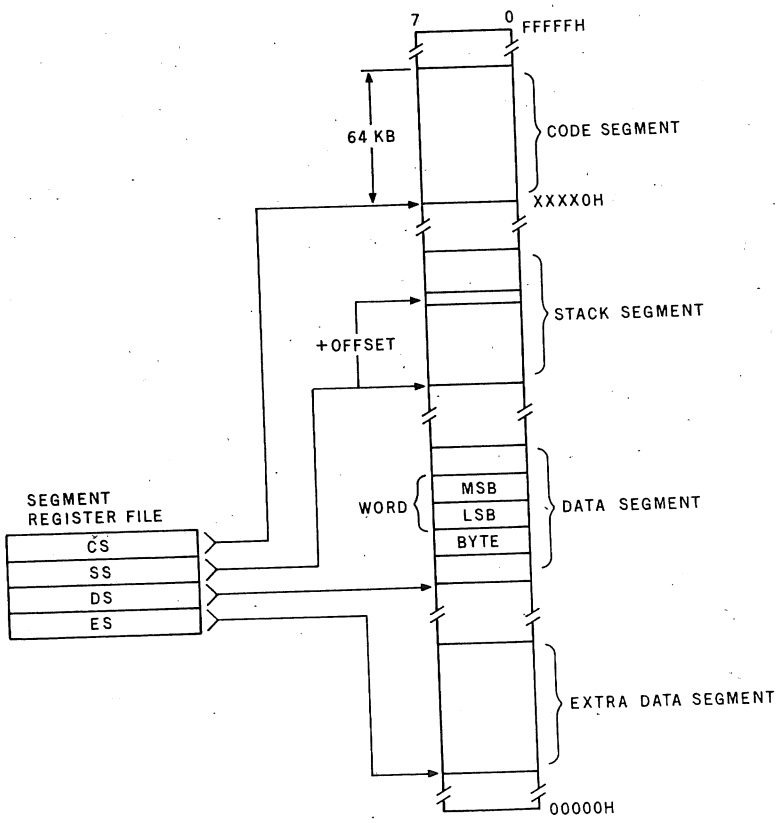


Figure 6: *Memory organization in the 8088. Memory segmentation is used to address up to 1 megabyte (1,048,576 bytes) in segments of 64K bytes. The 8088 creates a 20-bit address by combining a 16-bit offset value with a segment-boundary value stored in one of the segment registers.*

relocation, and bus control. (These actions of the BIU leave the EU free to concentrate on processing data and carrying out instructions.)

Figure 5 shows the programmer's model of the 8088's fourteen 16-bit registers. The shaded registers are the 8080-register subset, that is, the registers that are common to the 8088 and its 8-bit predecessors.

The general registers, also called the HL group because they can be subdivided into high and low bytes, include the accumulator (AX), base register (BX), count register (CX), and data register (DX). The two bytes in any of the general-purpose registers can be operated on separately; for instance, the AX register can be addressed as a 16-bit register, AX, or the high-order byte can be addressed as the register AH and the low-order byte as AL. The same holds true for BX, CX, and DX.

Another group of registers is the pointer and index (or P and I) group. This set contains the stack-pointer (SP), base-pointer (BP, an extra pointer into the stack), source-index (SI), and destination-index (DI) registers. Generally speaking, these registers hold offset addresses used for addressing within a segment of memory. They can also participate, along with the general-register group, in the arithmetic and logical operations of the 8088.

The 8088 uses *memory segmentation* to address this large memory space efficiently; it deals with memory as a set of four 64K-byte segments simultaneously defined (possibly overlapping) within the memory-address space, which is organized as a linear array of 1,048,576 bytes, addressed as hexadecimal 00000 through hexadecimal FFFFF. The 8088 creates a 20-bit address by combining a 16-bit offset value with a segment-boundary value stored in one of the segment registers. Figure 6 shows how this works.

Each of the 16-bit segment registers, the code-segment (CS) register, the stack-segment (SS) register, the data-segment (DS) register, and the extra-data-segment (ES) register, contains a value that can be combined with the 16-bit offset address speci-

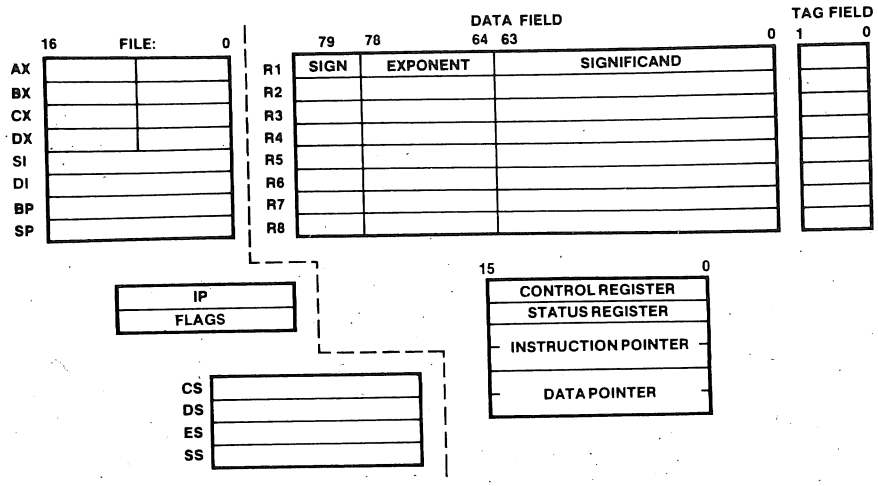


Figure 7: *Programmer's model of the 8088/8087 coprocessor combination. The 8087 adds eight 80-bit registers to the architecture and 68 operations to the instruction set. The 8088/8087 combination can operate on BCD (binary-coded decimal) numbers up to 18 digits long without round-off errors and perform arithmetic on 64-bit integers. (Figure provided courtesy of Intel Corporation.)*

fied by the instruction operand to form the 20-bit address. For instance, the 16-bit value in the code-segment register first has four low-order zero bits appended; it is then added to the low-order 16 bits of the offset address. When the 8088 fetches an instruction or data byte from memory, it comes from the location at the absolute address thereby formed.

The memory is thus divided into four segments: the code segment, where instructions are stored; the stack segment, where the pushdown stack is located; the data segment, where data to be operated on is found; and the extra segment, a 64K-byte data area assignable for any data-storage use. Which code-segment register is used to form the address varies according to what processor instruction is being executed.

The 8088 has both relative and absolute control-branching instructions. When all branch instructions within a given segment of memory are specified in relation to the instruction pointer and the program segment does not modify the value of the code-segment register, that program segment can be relocated dynamically anywhere within the entire address space simply by moving the code, updating the value of the code-segment register, and resuming execution.

The 8087 Numeric Processor

The Intel 8087 numeric processor extension (NPX) is an integrated circuit designed for use with the 8086 or 8088 (serving as the central-processing unit, or CPU) to form a high-performance numeric-data-processing system (called the iAPX 86/20 NDP or iAPX 88/20 NDP in Intel jargon). Its use is optional in the MPX-16.

The 8087 is designed to coordinate its functions with other processors in a coprocessing or multiprocessing environment. As a coprocessor, the 8087 adds 68 machine instructions to the system; these operate on its eight 80-bit floating-point registers, which function alongside the 8088's register set. The 8087 is designed to handle very large numbers; its internal temporary-storage format for floating-point quantities is 80 bits: 1 bit for sign, 15 bits for exponent, and 64 bits of mantissa. A programmer's model of the resulting architecture is shown in figure 7.

Capable of executing arithmetic, trigonometric, exponential, and logarithmic instructions, the 8087 conforms to the proposed IEEE (Institute of Electrical and Electronics Engineers) floating-point standard

and provides 100 times the performance of the 8088 working alone. It's a real number cruncher.

The 8087 uses its own instruction queue to monitor the 8088's instruction stream, operating only on those instructions intended for it. When a numeric instruction appears in the code, it is treated as an "escape" from the normal sequence by both the 8088 CPU and 8087 NPX; the NPX processes it while the CPU finishes its current task. Concluding that, the CPU either does nothing (if the NPX requires no further data) or it calculates an address and reads a byte of memory which is used by the NPX (the CPU ignores this value in its own computations).

The 8087 is only an extension processor and cannot run by itself. It needs a separate CPU to operate the data, address, and control buses, which provide it with instructions and operands. Once the NPX has started its operation, the CPU may continue executing the main program while the NPX "crunches." This parallel operation of the NPX and CPU can continue until the NPX needs to reference memory. Only then will the processor give the NPX access to the bus (the main processor may, however, continue to process instructions from its instruction queue). A special request/grant line, $\overline{RQ/GT0}$, is used to pass control of the buses shared between the NPX and the CPU. The relationship between the CPU and the NPX is similar to the master/slave scheme used in less complicated computers, while the protocol is somewhat like hold and hold-acknowledge signals, although more complicated. (Additional processors or coprocessing devices can be attached to the NPX/CPU combination through another signal line, $\overline{RQ/GT1}$, although no provision for this has been made in the MPX-16.)

The amount of time that the processor actually waits to get back on the bus is very small. If it were not for a few stolen memory cycles, the coprocessor's operation would be essentially invisible to the host processor; it's a small price to pay for the great increase in performance for numeric computation. As a comparison, even though it's quite a powerful microprocessor, the pure-16-bit 8086 takes about 20 milliseconds to compute a square root, using a floating-point subroutine. Eliminating the subroutine and using the 8087 instead, the result can be calculated in less than 40 *micro*seconds (the speed-up is similar for the 8088). Such speed is an undeniable asset to high-level languages such as BASIC and Pascal. They not only run faster, but the memory space devoted to floating-point subroutines is saved.

MPX-16 Bus Structures

The MPX-16 system supports two major signal-bus structures, the processors' local bus and the global system bus, as you can see clearly in the simplified block diagram of figure 1 and somewhat less clearly in the detailed diagram of figure 2. Most of the signals in the MPX-16 pass on one or more of the several buses.

The *local bus* is shared by the 8088

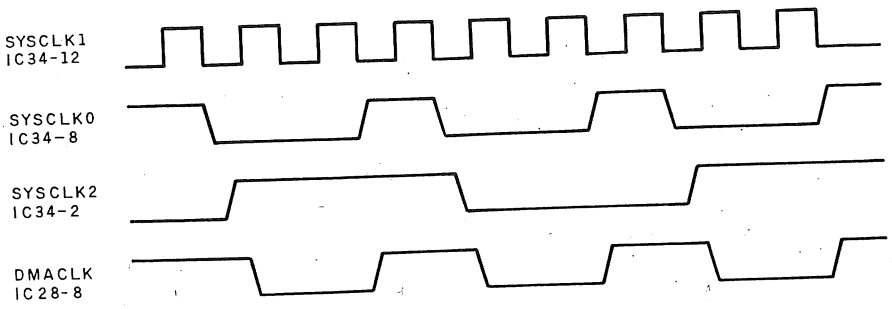
CPU and the 8087 NPX (if it is installed), either of which can be the local-bus master. The system bus can be driven either by the local bus under the control of the 8288 bus controller or by the 8237A-5 DMA controller.

The local bus consists of 8 multiplexed address and data lines (AD0 through AD7), 12 address lines (A8 through A19), and 3 status and control lines (S0, S1, and S2), which are connected with the global system bus through three-state buffers. Several other signals, including system clock signals and reset lines, are directly common with the system bus. The local bus can be controlled by either the CPU or the NPX, both of which have on-chip arbitration logic to determine which processor has control of the bus.

The CPU acts as a host processor to the NPX coprocessor. For example, when the 8087 NPX requires use of the local bus to return the result of an operation, it notifies the CPU by placing a series of handshaking signals on its bidirectional request/grant arbitration line $\overline{RQ}/\overline{GT0}$. A ready/wait control line is used to lengthen bus cycles on the local bus, which may be necessary to meet the access-time requirements of slow memory and peripheral devices, or to accommodate a DMA cycle already in progress on the system bus.

The *system bus* consists of 20 system address lines, 8 bidirectional system data lines, and several system control lines. The system data bus drives the system-board memory arrays and the I/O-expansion connectors and is buffered again to produce a "resident" data bus to which most of the on-board peripheral devices are attached. The system control bus consists of all timing signals, bus-cycle-control signals, interrupt-request lines, DMA-request/acknowledge lines, and system-bus-arbitration-control lines.

Control of the system bus is determined by a sequential-logic system-bus-arbitration circuit. The bus is always being controlled either by one of the two coprocessors via the local bus and the 8288 bus controller (with the $\overline{88AEN}$ control line active), or by the 8237A-5 DMA controller (with $\overline{AENDMA}$ active). The simple bus-arbitration circuit isolates the local bus from the system bus whenever system-bus access is given to the DMA controller for direct access to memory by one of the peripheral devices. For the DMA controller to gain access to the system bus in response to its HOLD request, a "locked" 8088 instruction (which must have continuous bus access for the 8088) must not be in execution, and the local bus must be in an idle state. The $\overline{LOCK}$ signal is also active during interrupt-acknowledge sequences, preventing the occurrence of a DMA cycle in the middle of the acknowledge sequence. Since neither of the coprocessors is involved in this bus-request/grant-arbitration sequence, a low input to the RDY1 line on the 8284 clock generator is used to force continuous wait states to be inserted in the local-bus timing cycle



	TYPICAL CLOCK PERIODS (ns)							
	SYSCLK0		SYSCLK1		SYSCLK2		DMACLK	
	CLK_L	CLK_H	CLK_L	CLK_H	CLK_L	CLK_H	CLK_L	CLK_H
15.0MHz CRYSTAL	133.3	66.7	33.3	33.3	200	200	105	95
14.31818MHz CRSTAL	139.7	69.8	34.9	34.9	209.5	209.5	111.5	98

Figure 8: *Timing diagram showing the relationship of the four major system clock signals. The table shows clock-high and clock-low periods for both the 14.31818-MHz and 15.0-MHz crystals in nanoseconds.*

until access to the system bus has been restored to the local bus.

Reset and Clock-Generator Circuits

The power-on reset pulse and all major system clock signals for the MPX-16 system and I/O-expansion slots are generated by an Intel 8284 clock generator and driver (IC34 in figure 3a). The 8284 is designed to provide the optimum clock signal, with a 33 percent duty cycle, at the voltage levels and transition times required by the 8088 CPU (IC36) and the 8087 NPX (IC37).

The 5-MHz 8088 used in the MPX-16 must operate with a clock rate between 2 and 5 MHz. (The 8088-2 version can run at 8 MHz.) The standard MPX-16 operates at a frequency of 4.77 MHz, which is derived from a 14.31818-MHz crystal oscillator. This crystal frequency provides compatibility with IBM Personal Computer color-graphics adapters, which use the 14.31818-MHz OSC clock output of the 8284 to produce a 3.58-MHz color-burst signal. The variable trimmer capacitor C9 is used to make minor adjustments in the clock frequency. An optional 15-MHz crystal can be substituted to operate the MPX-16 at its maximum clock rate of 5 MHz.

The 8284 divides the 14.31818-MHz oscillator frequency by 3 to provide the 33-percent-duty-cycle CPU clock, SYSCLK0. This clock signal is used by many parts of the MPX-16, including the 8087, the 8288 bus controller, the system-bus-arbitration circuit, and the I/O-expansion channels. SYSCLK0 is also used to provide a clock signal (DMACLK) for the 8237A-5 DMA controller. Some Schmitt-trigger inverter sections (IC24, which will appear next month in section 2 of the circuit) lengthen the level-high duration of SYSCLK0 so that the clock requirements of the 8237 will be met. Deriving the DMACLK signal from SYSCLK0 has the obvious advantage of maintaining synchronization between the local bus masters (the 8088 and 8087) and the alternate system-bus master, the DMA controller (the 8237).

In addition to the processor clock signal, the 8284 provides a peripheral-device clock signal, which is one-half the frequency of the processor clock and has a 50 percent duty cycle. The oscillator clock, SYSCLK1, is not used on the MPX-16 circuit board but is routed to the I/O-expansion connectors.

The peripheral clock, SYSCLK2, is used to drive the timer input of the 8155H-2 component (IC47, which will appear in January in section 4 of the circuit). The 8155's timer output is used to generate periodic memory-refresh requests for the dynamic memory on the system board, using the DMA controller (which we'll discuss further presently). The relationship of the four major system clock signals is illustrated in figure 8, which also contains a table of clock-high and clock-low periods for both the 14.31818- and 15.0-MHz crystals.

The 8284 clock generator is also used to generate the power-on reset pulse, SYSRES, which is active high. When power is first turned on, the rising supply voltage activates the Schmitt-trigger input pin $\overline{RES}$ on the 8284, which has approximately 0.25 V of hysteresis; as a result, the SYSRES pulse remains active until a voltage level of 1.05 V is reached on the $\overline{RES}$ input. The resistance/capacitance time constant set by R16 and C10 provides the necessary minimum reset pulsewidth of 50 μs (microseconds). The 1N4148 diode D1 provides a discharge path for C10 when power has been removed.

Nonmaskable-Interrupt Logic

The nonmaskable interrupt (NMI) input of the 8088 CPU is used for handling parity errors in the system-board memory and I/O-channel errors, which are typically also parity errors occurring in expansion-memory modules.

Although the NMI signal is nonmaskable once it gets to the 8088, logic is provided to externally mask the signals that would normally generate an NMI, if desired. Two input lines, PC4 and PC5, of the 8255A-5 PPI (programmable peripheral interface, IC60, to appear in section 4 of the circuit) are used as active-low enable signals. The $\overline{ENNMI}$ signal either enables or prevents NMI signals from reaching the 8088.

One source of interrupts is the $\overline{PARERR}$ signal, which is generated by the system-board circuit that calculates parity values for memory and detects errors. The second source of interrupts is the $\overline{IOCHNLERR}$ signal, which comes from the I/O-expansion slots. The latter signal can be masked by the $\overline{ENIOC}$ control line in such a

way that only system-board parity errors will cause an interrupt. The NMI input of the 8088 CPU is edge-sensitive but must remain active-high during two consecutive CPU clock cycles to guarantee recognition of the interrupt.

The NMI condition is predefined to be a "type 2" interrupt for the 8088, the highest-priority hardware interrupt. Since the vector (control-branching) location has been pre-assigned to locations hexadecimal 00008 through 0000B, no interrupt-acknowledge sequence is needed in a program.

DMA Controller and Bus Arbitration

Direct memory access has long been known as a way to improve the performance and I/O speed of a computer system by allowing I/O devices to directly transfer data to or from system memory without processor intervention, but until recently it has rarely been found in microcomputer systems. However, more widespread use of DMA has been made possible by semiconductor manufacturers, which have developed new ICs that make DMA much more easily provided. The MPX-16 employs one such integrated circuit, the Intel 8237A-5 DMA controller (IC48 in figure 3b on page 90).

Four independent channels of 20-bit-address direct memory access are supported by the MPX-16 system. Two of the DMA channels are available on the I/O-expansion bus to support high-speed data transfers between external peripheral devices and memory. A third channel, used by the floppy-disk-drive controller, is connected to the I/O-expansion bus for compatibility with the IBM Personal Computer.

The fourth DMA channel is used to provide the periodic refresh signal for the on-board dynamic-memory array, as well as any expansion memory boards, in which each row address of the dynamic-memory chips must be accessed. During system initialization, the TIMER OUT output of the 8155H-2 is set up to trigger a dummy DMA transfer approximately every 15 μs. The DMA channel is pro-

grammed for a memory-read cycle; it automatically increments the row-address counter for memory after each refresh cycle.

When no DMA requests are pending, the DMA controller is in an idle state (S1) and can be programmed by the CPU. If a DMA channel requests service for a peripheral device and that channel has been enabled by the system software, the DMA controller sends the signal DMAHLDRQST (DMA hold request) to the system-bus-arbitration circuit and enters the active state S0. The 8237 remains in the S0 state until it has received the signal DMAHLDACK (DMA hold acknowledge) from the bus-arbitration circuit, indicating that it has been granted control of the system bus.

At this time the system-bus-arbitration circuit isolates the local bus from the system bus by activating the control signal $\overline{88AEN}$. When this signal becomes inactive again, the 8288 bus controller (IC51) places the system-bus command-line buffers into a high-impedance state and disables the 74LS245 data transceiver IC43. In addition, the $\overline{88AEN}$ signal places the system-bus-address latches, IC50, IC38, and IC44, into a high-impedance state so that the local-bus master can drive the local bus during a DMA cycle without affecting operations on the system bus.

After one system-clock cycle following the arrival of the hold-acknowledge signal, the $\overline{AENDMA}$ control signal from IC24 enables the DMA bus-interface components. One of the 74LS373 latches, IC43, drives system address lines SYSA8 through SYSA15. The eight low-order address lines, SYSA0 through SYSA7, are driven by lines A0 through A7 on the 8237 through a 74LS245 transceiver, IC49 (shown in figure 3a on page 89). The data-flow-direction input of IC49 is controlled by the $\overline{AENDMA}$ signal such that data flow is from IC49 *to* the system address bus when a DMA transfer is in progress.

During processor memory transfers, the $\overline{AENDMA}$ signal is high, and address information flows *from* the system address bus through IC49 to the 8237. The four high-order system-bus-address lines, SYSA16 through SYSA19, are driven by three 4-bit latches (IC40, IC41, and IC42). These latches are loaded by either the operating system or application software and allow each DMA channel to operate in a separate 64K-byte section of memory if desired. Since DMA channel 0 is used for memory refresh and only the eight low-order address lines are significant, the latch for DMA channel 1 is used to drive the upper four address lines for both channels 0 and 1. The three address latches are enabled when both the $\overline{AENDMA}$ signal is active-low and the appropriate acknowledge signal is active.

Once the transfer of a single byte has been completed, the DMA controller turns off the DMAHLDRQST line. As a result, the DMAHLDACK signal goes inactive almost immediately. On the next clock cycle, the system-bus-interface components and 8288 bus controller are reactivated by a low state on the $\overline{88AEN}$

line, and the DMA bus-interface components are disabled by a high state on the $\overline{\text{AENDMA}}$ line. After the $\overline{\text{88AEN}}$ signal goes active-low, the 8288 does not drive the control bus until at least 105 ns (nanoseconds) and not more than 275 ns have elapsed, if a local-bus master has a bus cycle pending. A section of the 74LS10 three-input NAND gate IC30 and some flip-flops (in section 2) guarantee this by delaying the $\overline{\text{84AEN}}$ signal by two clock periods from the time the $\overline{\text{88AEN}}$ line goes low.

The DMAHLDRQST signal goes inactive after the transfer of each byte, even if the channel requesting service has not dropped the request. This provides at least one machine cycle between successive DMA transfers.

To Be Continued

Since it may take you a month to digest this much information, I'll stop the first installment of this series here. (Besides, I don't want to take up the whole magazine, though I could easily do it in describing this complex project.)

Next Month and Thereafter:

In Part 2, I'll concentrate on the MPX-16's memory section, interrupt logic, and I/O-expansion bus (including a detailed definition of each signal). The third installment will discuss the serial and parallel I/O ports, floppy-disk-drive controller, and operating-system BIOS, plus any other facts needed to summarize the project. ■

To receive a complete list of Ciarcia's Circuit Cellar project kits available from the Micromint, circle 100 on the reader service inquiry card at the back of the magazine.

Editor's Note: *Steve often refers to previous Circuit Cellar articles as reference material for each month's current article. Most of these past articles are available in reprint books from BYTE Books, McGraw-Hill Book Company, POB 400, Hightstown, NJ 08520.* Ciarcia's Circuit Cellar, Volume I, *covers articles that appeared in BYTE from September 1977 through November 1978.* Ciarcia's Circuit Cellar, Volume II, *contains articles from December 1978 through June 1980.* Ciarcia's Circuit Cellar, Volume III, *contains the articles that were published from July 1980 through December 1981.*

Acknowledgments

Thanks to Jim Norris, George Martin, and Linda Spencer of Owl Electronic Laboratories for their contributions to the design.

Thanks to Mark Dahmke and Gordon Heins for their help with the documentation.

Thanks to Bill Morello and his staff at Techart Associates for their careful work in drawing the schematic diagrams.

References

1. Cantrell, Thomas. "An 8088 Processor for the S-100 Bus." Part 1, September 1980 BYTE, page 46; Part 2, October 1980 BYTE, page 62; Part 3, November 1980 BYTE, page 340.
2. Ciarcia, Steve. "Ease into 16-Bit Computing." Part 1, March 1980 BYTE, page 17. Part 2, April 1980 BYTE, page 40. Reprinted in *Ciarcia's Circuit Cellar, Volume II.* Peterborough, NH: BYTE Books, 1981, page 171.
3. Ciarcia, Steve. "The Intel 8086." November 1979 BYTE, page 14. Reprinted in *Ciarcia's Circuit Cellar, Volume II.* Peterborough, NH: BYTE Books, 1981, page 120.
4. *Component Data Catalog.* Santa Clara, CA: Intel Corporation, 1981.
5. *iAPX 88 Book.* Santa Clara, CA: Intel Corporation, 1981.
6. *iAPX 86,88 User's Manual.* Santa Clara, CA: Intel Corporation, 1981.