Ease into 16-Bit Computing:

Get 16-Bit Performance from an 8-Bit Computer

Steve Ciarcia
POB 582
Glastonbury CT 06033

Stopping for coffee at the local doughnut shop has become a morning ritual. I am quite capable of making coffee at home, but I am not what you would call a "morning person." Even though I have culinary talents that include the preparation of eggs Benedict and strawberry crepes, it had better be evening when you request them around our house.

This morning started out like any other. I pulled my car into the doughnut shop's parking lot only after carefully examining all the potential hazards. I carefully avoided the broken glass, the beat-up 1962 Chevy and the large black van with a "Tax the Rich!" bumper sticker.

After entering the shop, I sat down and spread my reading material, the latest issue of BYTE, on the counter. As my coffee and bran muffin were delivered, I could not help but overhear the conversation of two other people at the counter.

"Dave, have you been reading any of the magazines lately? It looks like everyone is going 16-bit crazy."

"I've read a lot of descriptive articles, but I suppose it'll take a while before we see any real hardware."

"Actually, I'm a little hesitant to just jump on the bandwagon. My 8085 works just fine."

"I know what you mean, Ed. The Z80 system I built from scratch is still cranking along. I'd like to do something with the 16-bit chips, but I sure don't want to throw out my 8-bit system."

"What about building a small system to experiment with? Didn't I see an article a few months ago on a single-board 8086?"

"Yeah, I remember. It was in BYTE. Wasn't it written by that guy who lives around here someplace, in his cellar or something?"

Upon hearing that last statement, I nearly choked on my muffin. I thought it would be prudent to remain anonymous until I learned whether or not they enjoyed the article. I carefully closed the magazine and placed it face down on the counter.

One way to ease yourself into the world of 16-bit computers is with the Intel 8088. This microprocessor is an 8086 on the inside with an 8-bit data bus on the outside.

"Maybe, but anyway, the article wasn't too bad," said Ed. I'm sure they didn't hear the sign of relief from across the counter. Then he continued, "But it just seemed like a larger computer than I have time to build. It's obviously oriented toward guys who don't have any other development system. I'd prefer a minimal hardware configuration to start with. If I want large programs, I'll run a macroassembler on my 8085 system, write the object code into an EPROM, and then plug it into the test board."

"Eliminating all the keys and displays will help, but how small a computer can we end up with and still be 16-bit? You'll need 16-bit address and data buses, and what's 1 K words of memory—four chips? All the EPROMs I know are 8-bit output. That means at least two of them."

"Wait a minute," said Ed. "I didn't say I had all the answers. The minimal configuration may be twenty chips, but isn't this closer to something we could afford to experiment with?"

This was the perfect opportunity to express my point of view concerning the things that I write and consult about. "Excuse me," I said. "I couldn't help but overhear your conversation. Had you considered using an 8088?"

The two young men looked up at me, paused, and harmonized, "An 80 what?"

"I know a little about microprocessors. Have you considered using an Intel 8088?"

"Is it 16-bit?" asked Dave.

"Well, yes and no," I replied. "It uses an 8-bit data bus, but, internally it's an 8086. Essentially it's an 8-bit chip that's completely 8086-software-compatible."

Should they listen to this doughnut and coffee philosopher? "That sounds tremendous, but won't it still require quite a few chips to make an operational computer?"

I sensed that this was a good time for my exit. Staying any longer would involve my designing a computer for them on the back of a napkin. Ordinarily I probably would have stayed, but I had just completed a similar task in my latest article, so I decided to let them wait a few more weeks. I rose to leave, carefully rolling up the copy of BYTE, cover page inside, and stopped behind them on my way out. "My recollection is that while four chips is a possibility, a five-chip computer is quite a reality. I've even seen how a BASIC interpreter could be written to run on it. In case you're interested, the next issue of BYTE has an article all about it."

I excused myself to attend an important meeting. As I opened the door I heard, "Thanks, I'll look forward to reading it." They watched me intently as I drove out. I could only speculate on their final conversation.

The 16-Bit Generation

The exciting items in microcomputing these days are the 16-bit microprocessors made by companies such as Intel (the 8086), Zilog (the Z8000) and Motorola (the M68000). All of these devices, although they differ in internal architectures, commonly claim to have compressed the power of a minicomputer within a single chip of silicon. Most notably are the 16-bit data bus and increased addressing space. A 20-bit address can directly address a megabyte of memory.

There seems to be little doubt in the minds of microcomputer-system designers that the 16-bit processors are the wave of the future. Already some major manufacturers are designing the new processors into intelligent terminals, word-processing systems, and other equipment. The day when this revolution within a revolution will affect the personal and small-business computer marketplace is not too far away.

But if it is obvious that the 16-bit machines will be the trend of future product technology, it is equally obvious that it is relatively difficult for the designer to make a leap from the 8-bit world of the 8080, Z80, 6800 and 6502 to the emerging 16-bit world. The 16-bit instruction sets are more complex. The 8086, for instance, has a repertoire of some 133 instructions, as compared to seventy-eight for the 8080. Simply because of the larger range of memory that can be addressed and because of address segmentation, addressing of memory is more advanced. Also, the register set is more complicated, and the types of operands with which the processor can work are more extensive.

As complex as the 8086 or any other 16-bit microprocessor is from a software viewpoint, it is in the design of hardware circuits to work with the 16-bit processors where the real complexities arise. Peripheral interfaces and existing hardware systems are generally based on an 8-bit data bus. When your whole design is built to make efficient use of an 8-bit data bus, converting to a 16-bit architecture is not a simple matter of replacing the processor. This incompatibility dictates substantial design changes to take advantage of the new 16-bit microprocessor.

A Gradual Approach to 16-Bit Computing

There is an alternative to converting abruptly to 16-bit architecture. Look at photo 1 and observe the Intel 8088 microprocessor. This device uses an 8-bit data bus, so all of your present hardware system components will work with it from the standpoint of getting information between the processor and the peripheral-support devices or memory, but the 8088 features a common internal architecture and complete software compatibility with the 16-bit 8086 processor.

As a result, the 8088 provides an excellent way for designers, engineers, hobbyists, and students to ease into the world of 16-bit computing. Its 8-bit-compatible bus structure makes it the logical choice for upgrading 6800, 6502, Z80 and 8080 designs to 16-bit capability without

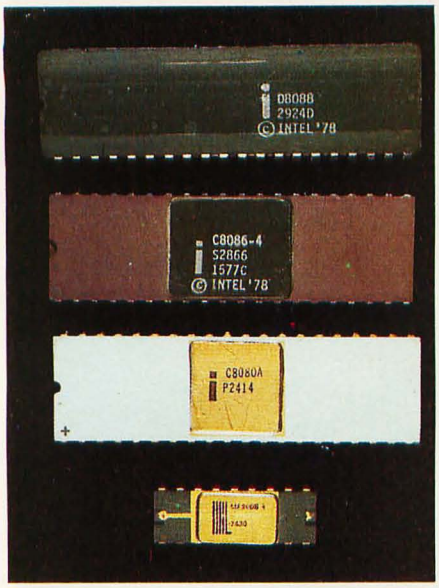

Photo 1: *An exhibit of advancing microprocessor technology. Here are four integrated circuits produced by Intel Corporation. From bottom to top, we have the 8008, the first 8-bit general-purpose microprocessor; the 8080A, one of the breed of 8-bit devices that helped ignite the microcomputing boom; the 8086, the advanced 16-bit processor; and the 8088, the subject of this article—a component that contains 16-bit computing capability in a package that can communicate with the outside world through an 8-bit data bus.*

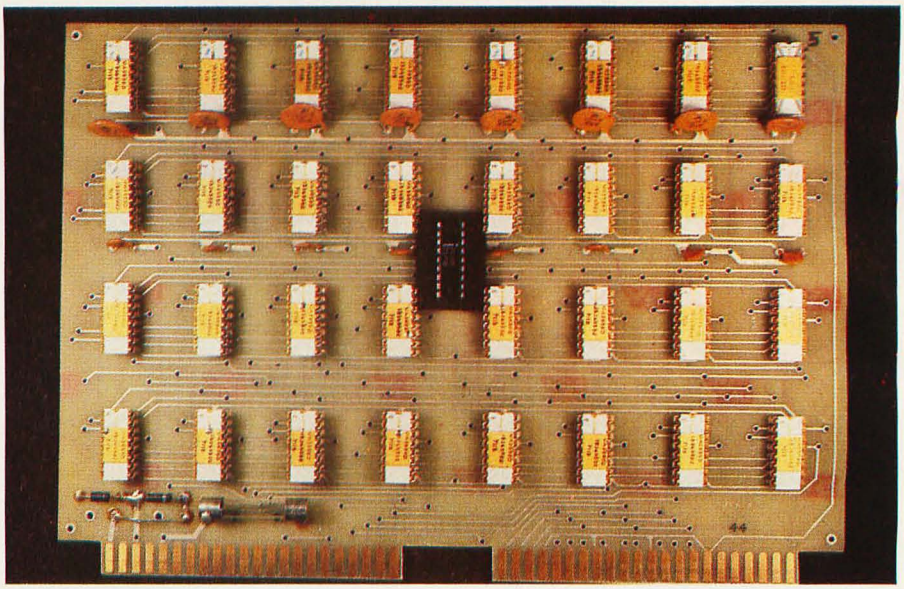

Photo 2: *An exhibit of advancing memory technology. The single black integrated circuit at the center can replace the entire board of components. The center component is the Intel 8185 1 K-byte static programmable memory. The board is a 1 K-byte memory board from a Scelbi 8B microcomputer system, which used the 8008 microprocessor (circa 1975).*

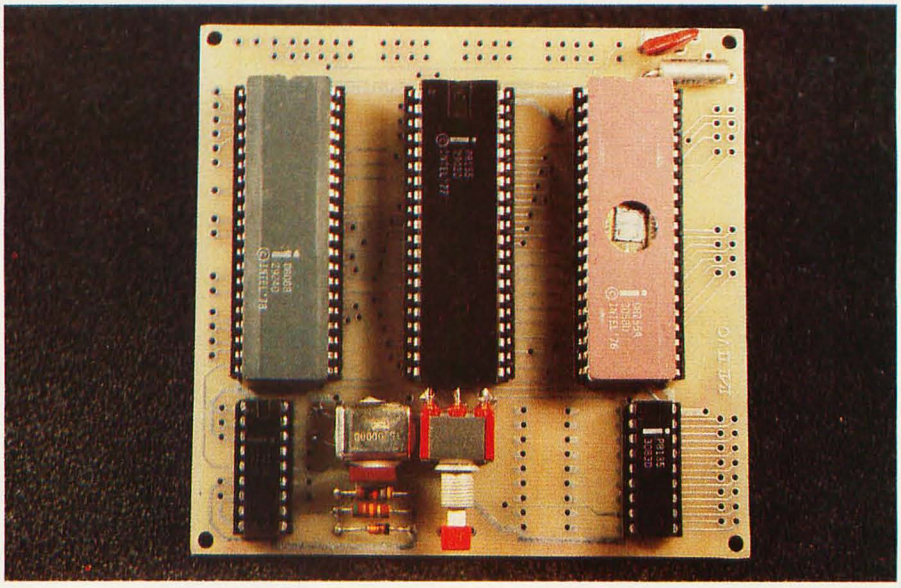

Photo 3: *Using the 8088 and other components of kindred technology, it is possible to build a functional microcomputer system with only five integrated circuits. Part 2 of this article (in the April 1980 BYTE) will present more detailed information about this system.*

alteration of existing 8-bit hardware.

The 8088 can be used in projects such as a low-cost system that employs multiplexed peripherals such as the 8155, 8755A and 8185. Or, fully expanded, it forms a system that allows a full megabyte of address space and compatibility with the 8086 family of coprocessors and multiprocessors.

This two-part article is designed to give you a glimpse of the 8088. This month in Part 1, I shall attempt to familiarize you with the instruction set of the 8088 and the hardware of a microcomputer that is made from an 8088 and only four other integrated circuits. The power of this five-chip circuit will be emphasized by illustrating, among other examples, how it can be configured to support a multi-user Tiny BASIC.

Architecture of the 8088

Anyone comparing the internal architectures of the 8088 and the 8086 processors will realize that they are identical. Even though I have previously discussed the 8086, a brief explanation of this architecture is necessary since the capabilities of our five-chip computer depend directly upon it. However, if you wish to read a more detailed description, you should refer to a previous Circuit Cellar article, "The Intel 8086" (November 1979 BYTE, page 14).

A diagram of the internal structure of the 8088 is shown in figure 1. The 8088 contains two logical "units", the bus-interface unit (BIU) and the execution unit (EU), and a 4-byte instruction queue.

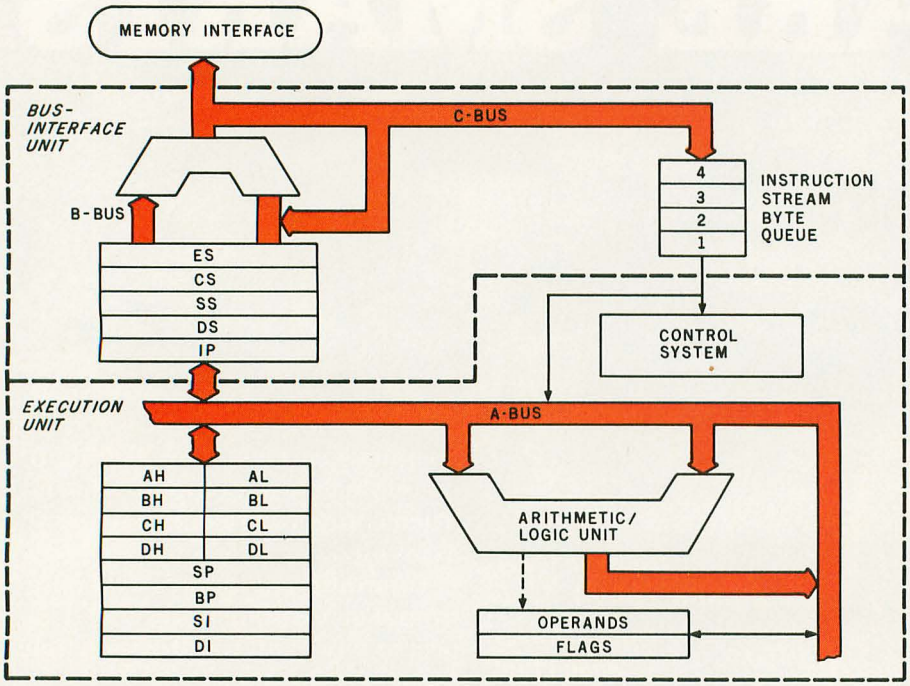

Figure 1: *Diagram showing internal operational principles of the 8088 microprocessor. The 8088 (and the 8086) use a pipelined architecture that increases performance by overlapping instruction execution with memory-fetch operations. The 8088 can directly execute any 8086 software.*

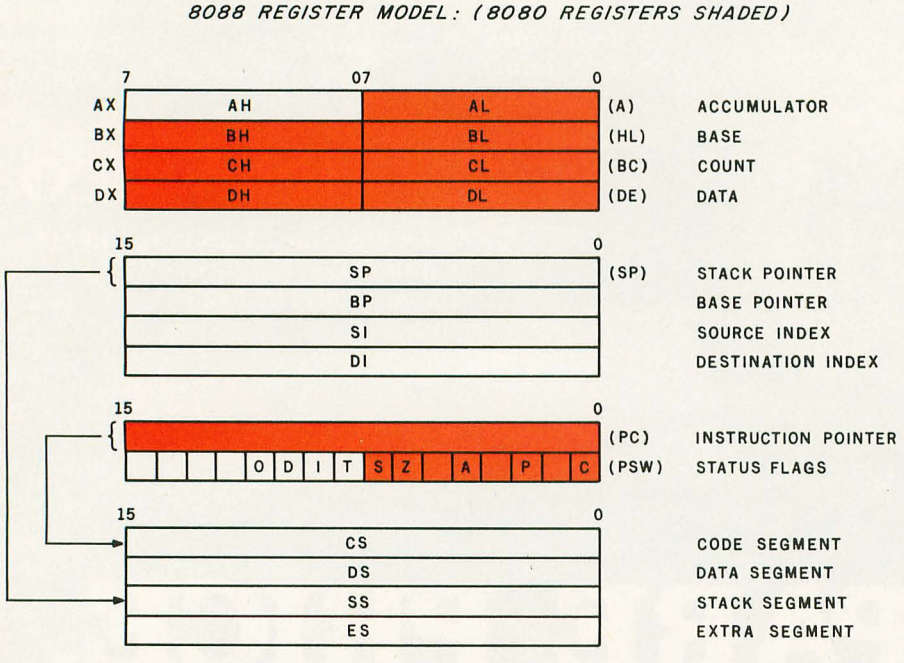

Figure 2: *The 8088 contains fourteen 16-bit registers. The shaded registers are those common to the 8088 and the 8080.*

The *execution unit* is where the actual processing of data takes place inside the 8088. It is here that the familiar arithmetic logic unit (ALU) is located, along with the registers used to manipulate data, store intermediate results, and keep track of the stack. The execution unit accepts instructions that have been fetched by the bus-interface unit, processes the instructions, and returns operand addresses to the bus-interface unit. The EU also receives memory operands through the bus-interface unit, processes the operands, and then passes them back to the bus-interface unit 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 bus-interface unit carries out this assignment in two basic ways:

- by fetching instructions *before* they are needed by the execution unit, storing them in the instruction queue
- by taking care of all operand fetch and store operations, address relocation, and bus control (These actions of the bus-interface unit leave the execution unit free to concentrate on processing data and carrying out instructions.)

Figure 2 summarizes the 8088 register set. 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 *H*igh and *L*ow bytes, include the accumulator (AX), base (BX), count (CX) and data (DX) registers. The AX register may 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 of the other three general registers (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), source index (SI), and destination index (DI)

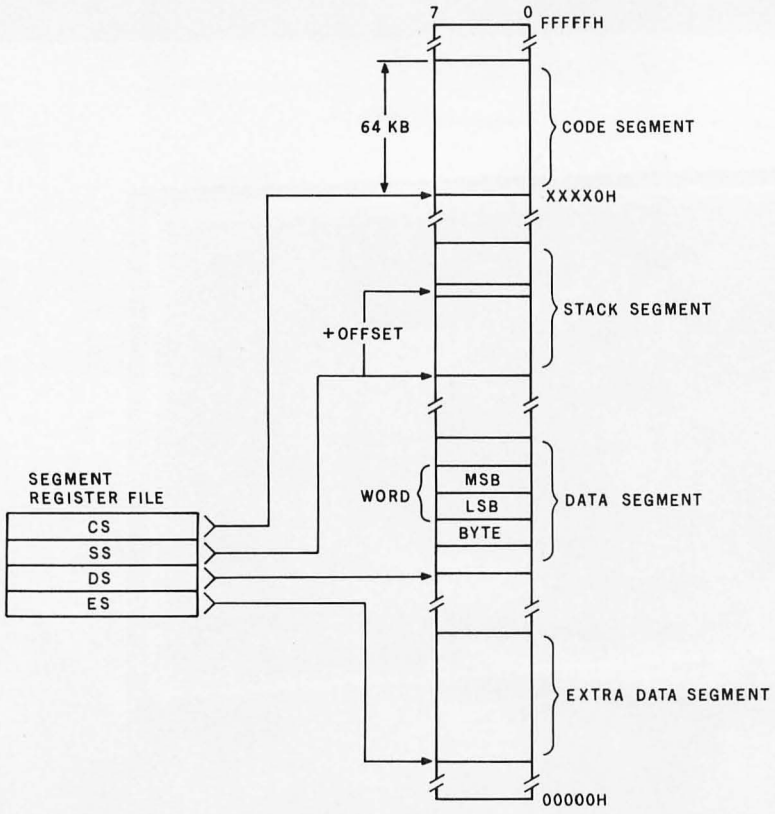

Figure 3: *Memory organization. The 8088 uses a memory-segmentation technique to address up to 1,048,576 bytes (1 M byte) of memory. The user can use attributes of the memory-addressing system to dynamically relocate a program anywhere within the entire address space.*

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 arithmetic and logical operations of the 8088.

The 8088 uses memory segmentation to address this large memory space efficiently. At any one time, the 8088 can deal with memory as a set of four 64 K-byte segments. The total memory is organized as a linear array of 1,048,576 bytes, addressed as hexadecimal 00000 to hexadecimal FFFFF. The 8088 creates a 20-bit address by combining a 16-bit offset and a segment boundary value stored in one of the segment registers. Figure 3 demonstrates 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 is added to a 16-bit offset address, forming a 20-bit address. The memory is thus divided into a maximum of four 64 K-byte segments that are active at any single time. The *code segment* of memory is where instructions are stored, the *stack segment* of memory is where the pushdown stack is located, the *data segment* is where data to be operated on is found in memory, and the *extra segment* is an additional 64 K-byte data area.

When fetching an instruction from memory, the location accessed is given by a 20-bit address that is the sum of two numbers. The first number is the value of the 16-bit instruction pointer. The second number is a 20-bit value that is the 16-bit code-segment register with four low-order zero bits appended. This forms the 20-bit address required to specify any location in the megabyte-sized address space.

In the case of a memory-reference operation for a transfer of data, the absolute memory address referenced by a given memory-access instruction is calculated by adding the given 16-bit address to the base address. The base address is given by the contents of the data-segment or extra-segment register and is followed by four low-order zero bits.

In the case of a stack operation, the memory location referenced is similarly offset from the value contained in the stack-segment register.

The 8088 has both *relative* and absolute branch 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, the program segment can be relocated dynamically anywhere within the megabyte address space. A program is relocated in the 8088 simply by moving the code, updating the value of the code-segment register, and resuming execution.

Small System Applications

The 8088 can be used in a broad range of applications, from systems requiring use of a minimum number of components to systems requiring maximum performance. The component-count-sensitive applications include point-of-sale terminals and simple controllers, which require that system cost be kept low, but need substantial processing power. A big reason for this design flexibility is the ability of the 8088 to operate in a minimum-hardware mode.

The minimum-mode, multiplexed configuration, as shown in figure 4, is an effective way of building a powerful system around the 8088, while using the smallest number of parts. The processor is connected in the minimum mode by wiring its Mn/Mx pin in the high-logic state (at V_{cc} potential). The multiplexed bus is directly compatible with the Intel 8085A-family peripheral components (8155, 8355, 8755A, and the new 8185).

A four-chip system can be designed using the following components: an 8088 microprocessor; an 8284 clock generator; an 8155 memory, input/output (I/O), and timer device; and an 8755A EPROM and I/O device. A fifth component, the 8185, is a simple

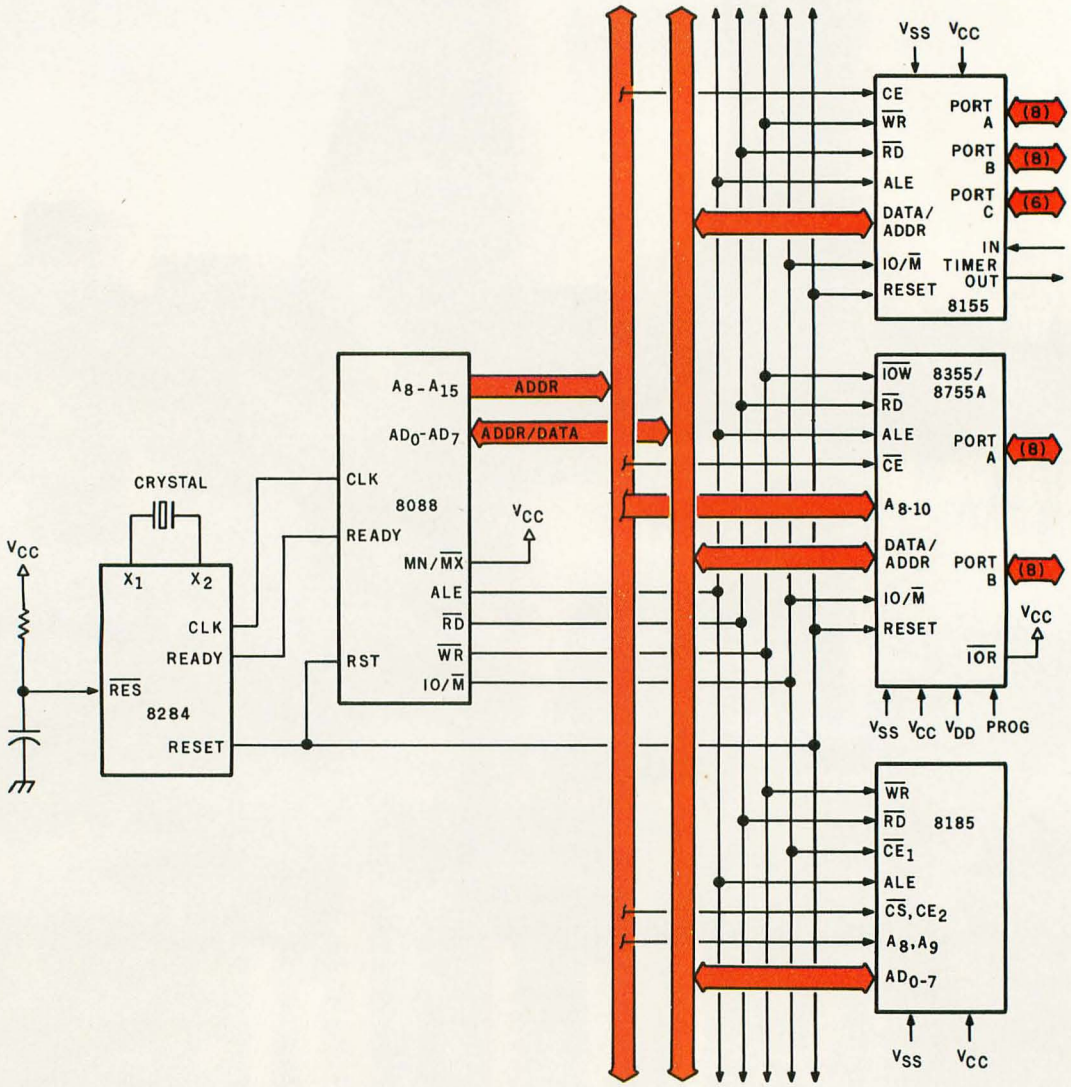

Figure 4: *When used in the minimum mode (MN/ $\overline{MX}$ line held high), the 8088 interfaces directly with the multiplexed address and data components in the 8085A-support family to form a functional microcomputer system using only five integrated circuits. Detailed information concerning this circuit will be given in Part 2.*

addition to the system and provides an extra 1 K bytes of user memory.

In the minimum-mode configuration, the 8088 provides all necessary bus-control signals, including $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$ and ALE. It further provides HOLD and HLDA (hold-acknowledge) signals to allow direct-memory-access (DMA) data transfer, INT and $\overline{INTA}$ to interface the 8259A interrupt controller, and $\overline{DEN}$ and DT/$\overline{R}$ to control transceivers on the data bus.

The power of the 8088 can be extended in large-system applications by wiring it into the maximum-mode configuration. However, a discussion of maximum-mode features is beyond the scope of this article.

The 8088 Instruction Set

A complete discussion of the 8088's instruction set is also beyond the scope of this article. Rather than attempt it, I shall concentrate on some specific features of the 8088 instruction set that facilitate the specific application discussed next month in Part 2 of this article. These features include extended arithmetic instructions, direct use of ASCII-encoded data, multiprocessing features, string-manipulation instructions, and table-translating aids. The 8088 instruction set includes single-instruction multiplication and division instructions, along with five different types of addition and seven types of subtraction operations.

These multiply and divide instructions greatly facilitate "number crunching." This numerical ability saves much time in such applications as data sampling, signal processing, and scientific calculation. Not only are fewer machine instructions needed to perform a given task, with corresponding savings in memory usage and execution time, but the versatility of the instructions and the

Text continued on page 30

Listing 1: *An example of the efficiency of the 8088 and 8086 instruction set. This short routine accepts input of five values from an input port, and then calculates and sends a running-average value to an output port. Compare this listing with listing 2.*

```
                XOR     BX, BX          ;CLR BX
                MOV     CX, 5           ; Set loop counter

Average         INC     BL              ;Increment data counter
                IN      AL, Port #      ;Input data
                ADD     BH, AL          ;Update running total
                MOV     AL, BH
                DIV     BL              ;Divide running total by
                                        ;data counter.

                OUT     Port #, AL      ;Output running average.

                LOOP    Average         ;Return unless fifth pass
                                        ;is completed.

                HLT
```

Listing 2: *A routine that performs the same task as the routine given in listing 1. This code, however, was written for the older 8080 processor. As you can see, it is longer and more tedious to write.*

```
                MVI     H,00            ;Clear H register
                MVI     E,00            ;Clear E register

Average         INR     E               ;Increment data counter

                MOV     C, H
                IN      A, Port #       ;Input data
                ADD     H               ;Add data to running total

Divide          XRA     A               ;Clear accumulator
                MOV     B, A            ;Clear B register
                MOV     L, A            ;Clear L register
                MVI     C, 80           ;Initialize bit counter

Loop            MOV     A, C            ;Shift B and C as
                RAL                     ;a 16-bit unit—
                MOV     C, A            ;one bit left
                MOV     A, B
                RAL
                MOV     B, A

                CMP     E               ;Compare data
                JC      Next            ;counter (divisor) with
                                        ;dividend; if divisor is larger,
                                        ;bypass subtract.
                SUB     E               ;Divisor is smaller; subtract.
                MOV     B, A
                MOV     A, D            ;Set current bit of
                ORA     L               ;L to 1
                MOV     L, A

Next            MOV     A, D            ;Shift D right and check carry
                RRC
                JNC     Loop            ;If no carry, return for next bit.

                MOV     A, L            ;Outport running average
                OUT     Output #

                MVI     A, 05           ;Return unless fifth pass is
                CMP     E               ;completed.
                JNZ     Average

                HLT
```

Text continued from page 26

ability of the 8088 to deal with several types of data remove the usual necessity of handling messy conversions from one type of data representation to another and back again.

Two program listings demonstrate the saving of effort. Listing 1 gives the 8088 code for the skeleton of a subroutine that accepts data from a specified input port and calculates a running average of the values entered. The same subroutine section coded for the older 8080 microprocessor is shown in listing 2.

Direct Use of ASCII and Decimal Data

The direct use of unpacked binary-coded decimal (BCD) or ASCII-encoded data in a microcomputer has a number of obvious advantages. Since many I/O devices present data to the processor in American Standard Code for Information Interchange (ASCII) format and expect responses in the same format, microcomputer-system designers have for years faced the necessity of putting their input and output through a translation process (usually involving a table look-up operation) before processing the input or responding with output.

With the 8088's instruction set, such manipulation is no longer necessary. All four mathematical instruction types (add, subtract, multiply, and divide) provide for ASCII adjustment of the accumulator contents by a single instruction. This feature is obviously of great use in everyday microprocessor applications. Equally interesting (and useful) are the two instructions that adjust the results of addition and subtraction to *packed* decimal form.

Table-Translating Aid

Despite the availability of single instructions to convert accumulator contents from one type of data representation to another, it may still be necessary from time to time to translate data by means of the traditional look-up table. This might, for example, be necessary if the data is being received or transmitted in EBCDIC (Extended Binary-Coded-Decimal Interchange Code) rather than in ASCII form.

Listing 3: *A segment of 8088 code that translates characters from Extended Binary-Coded-Decimal Interchange Code (EBCDIC) to American Standard Code for Information Interchange (ASCII) form. The 8088 instructions for manipulating and translating strings of characters are put to good use.*

```
        MOV     SI, FFFE    ; Source index register contains start of EBCDIC Buffer
        MOV     BX, 0100    ; B register points to translate table
        MOV     DI, ASCBUF  ; Destination index points to ASCII buffer
        MOV     CX, 528     ; C register contains length of buffer
        CLD
        JCXZ    EMPTY       ; Skip if input buffer empty
NEXT:   LODS    EBOBUF      ; Get next EBCDIC character
        XLAT    TABLE       ; Translate to ASCII
        STOS    ASCBUF      ; Transfer ASCII character to buffer
        CMP     AL, EOT     ; Test for EOT character
        LOOPNE  NEXT        ; Continue if no EOT received (CX decrements first)
        .
        .
        .
EMPTY:  (Program continues)
```

The XLAT (ie: translate) instruction allows the user to define a 256-byte table of correspondence and then to reference any point in the table very easily. The base address of the table is placed in the BX register and the index (ie: table position) is stored in the accumulator. Then the single instruction code XLAT is used to refer to the proper point in the table, pick out the translation, and store the result in the accumulator.

This is useful particularly when data that has been entered from a port comes into the accumulator for disposition or transfer. If you are dealing with a stream of incoming characters in EBCDIC format, for example, the translation proceeds thusly. You begin by storing the beginning memory address of your 256-byte translation table in the BX register. If you set up the table so that the base address of the table corresponds to an incoming EBCDIC value of 00, the next address to an incoming value of 01, etc, all you must do is simply accept a byte of data and execute the XLAT instruction.

This simple procedure lets us obtain the correct translation of that byte into the proper format for handling by the 8088 or some other processor. A MOV instruction will then store the result of translation until it is needed; the translation process can then be repeated with the next incoming byte. Setting up the necessary instruction sequence requires one instruction: a MOV to the BX register of the base address of the table. The loop for handling the translation requires only three basic instructions: the input instruction, XLAT, and MOV.

String-Manipulation Instructions

Since typical computer applications often deal with strings of characters consisting of letters, numbers, and special symbols, easy-to-use string-manipulation instructions are a welcome enhancement to 8-bit processors. The 8088 addresses this need by providing five powerful primitive string operators that may be preceded by a single-byte repetition prefix.

For a byte-for-byte or word-for-word comparison of two data strings (as you might use in verifying the accuracy of data loaded into memory from a mass-storage device, for example), the 8088 offers the CMPS instruction. This also allows termination of a program segment upon occurrence of a predetermined equality or inequality condition, as well as automatic incrementing or decrementing.

You can scan through a string of data for an occurrence or for an absence of occurrence of a specific string or character by using the SCAS instruction. This operation subtracts the byte or word operand in memory (or elsewhere) from the accumulator and changes the logic state of the flags; it does not, however, return a result. Again, decrementing or incrementing is automatic.

The STOS instruction allows you to fill a string of arbitrary length with a single value (eg: a string of zeros or nulls for a floppy disk initialization routine), once more with automatic incrementing or decrementing of a predetermined count.

Putting Some Things Together

Let's take a quick look at a small but powerful example that employs both the string manipulation and the XLAT instructions to solve a very practical problem.

You are designing an input routine that must translate a buffer filled with EBCDIC characters into ASCII form, continuing the transfer until one of several possible EBCDIC characters is received. The transferred ASCII string should be terminated with an EOT (end-of-transmission, hexadecimal value 04) character. Assume that the buffer starts at hexadecimal memory location FFFE, the table to translate the EBCDIC form to ASCII begins at hexadecimal location 0100 and the CX register is to contain a value giving the length of the buffer containing EBCDIC characters. The buffer may, of course, be empty.

The small 8088 program segment shown in listing 3 accomplishes this task in a small number of instructions and handles a great deal of overhead work with little effort or concern on the part of the system designer and programmer.

By now you should have an understanding of the power of the 8088 microprocessor. Even in a minimal-mode, five-component circuit, our little computer will have the following attributes:

- 5 MHz 8088 8-bit processor (completely 8086 software-compatible)
- 1280 bytes of static user memory
- 2048 bytes of erasable, programmable read-only memory (EPROM)
- 38 parallel I/O lines
- a 14-bit counter/timer
- power-on reset and nonmaskable interrupt.

Next month, in Part 2, we will deal with some key features of the 8088 which make it particularly suited to multiprocessing situations. We will investigate the operating system of a multi-user, Tiny BASIC language system on our minimal-configuration computer.■

These figures are provided through the courtesy of Intel Corporation.