Solid state/computers

Demystifying microprocessor design

Big chips require artful compromises, as demonstrated by this design case history

Masatoshi Shima

Stripped of its mystique, designing a high-performance microprocessor like the Zilog Z8000 is like designing any other complex system of digital electronics, except that good engineering practices, instead of being merely desirable, are crucial. Here are the key rules that the designers of the Z8000 followed:

• *Top-down design is essential* (Fig. 1) because once the concepts are frozen in silicon, it takes many months to correct even a minor error. And time is of the essence, because LSI density increases so rapidly each year that nobody wants a design geared to last year's capability.

• *Accurate product cost predictions are more critical than when designing with off-the-shelf ICs.* The LSI analog of IC parts count is chip area, which is much more critical than a conventional bill of materials. Chip price is proportional to yield, but if the chip area is increased, say, 10 percent, the yield may go down by 25 to 50 percent. If the designer waits until the last minute to calculate chip size, his company may wind up with a chip too big to sell profitably.

• *Product definition is crucial in the early design phases.* With a ten-year expected life for the basic design (the life expectancy of the Z8000), a lot of basic analysis and crystal gazing into future application needs must be done by the company's software architecture groups. The senior chip designer must then see if these architectural goals can be realized. He is reasonably free to change the instruction format and to suggest new instructions and drop out old ones, provided he can convince the software architects that the initial overall goals have not been compromised.

• *Second-source availability is vital.* But microprocessor design is process-determined. To insure second sourcing for the Z8000, Zilog chose only a moderately scaled-down MOS process, using 5-6 μm design rules instead of, say, 4 μm, and moderate clock rates of 4 MHz rather than 5 or 6 MHz. The goal was also to achieve higher performance than that in existing 16-bit processors. But to get such performance required very careful planning of the instruction set, the instruction format, and the chip design.

• *Very carefully thought-out methods of communication are needed between members of the design team.* When the Z8000 program started some two and a half years ago, there were very few senior processor designers, few associate designers with any experience, and no specialists in any design phase. The Z8000 design was done by the author; an associate designer, C. N. Patel; an associate layout draftsman; and a senior layout draftsman. Each had to discuss with the three others all phases of the project. Good layout is crucial in keeping down the chip size; there must be no wasted interconnection space, no speed bottlenecks. Optimum power line widths must be correctly specified to insure good noise margins. The designers must provide the draftsmen with rough layouts of critical chip areas and monitor the draftsmen's final layout closely to insure that the propagation time will be as predicted. If those precautions are not taken, the layout may have to be redone.

• *Experience counts in speeding the work flow.* The author was fortunate to have acquired design experience on previous processors, the 4004, 8080, and Z80. This expertise was handed down in an orderly way to the associate designer, both for the present design and for future ones he might undertake. All of the Z8000 detailed circuit design was, in fact, done by Mr. Patel in only six months. As MOS advances toward VLSI and much higher density, the designer's task may well be eased by less need to optimize all the circuits, by a greater number of trained logic circuit and layout designers, and by a library of subsystem designs.

• *All the key rules have to be handled interactively.* The complete specification of instructions and their format, the selection of optimum chip size and process, and the attainment of best chip performance, best use of available skills, and optimum layout cannot be solved separately; they have to be solved together because each greatly affects the others.

Design time vs. performance

As the transistors on microprocessor chips increase each year, it is important to hold their design times within reasonable bounds. By comparing Tables I and II, we see that although the number of instructions multiplied by 3.3 in going from the Z80 to the Z8000, the time it took to design the Z8000 only doubled. The time increase was proportional to the number of transistors on each chip.

It would not be safe to infer, however, that future high-performance 16- and 32-bit microprocessor generations would scale so successfully if one employed the same fine-tuning techniques used with the Z8000. Nor is it clear that even a linear relation between design time and the number of transistors would be acceptable for a VLSI logic chip with several hundred thousand transistors.

The critical goals

Five critical goals were set for the Z8000 to insure a long, cost-effective life. Each unfortunately conflicted with the need to keep the chip size down to one that could be produced economically: a die size of 220 × 220 mil^2 to 250 × 250 mil^2. The objectives were these:

1. A large memory address space with optional segmentation and memory management. The Z8000 chip is offered in two bonding versions. The 40-pin, nonsegmented version uses 16-bit addresses that can directly access 64 kbytes of

0018-9235/79/0700-0022$00.75©1979 IEEE

memory. The 48-pin, segmented or nonsegmented version can access 8 Mbytes of memory, using a 7-bit segment number and a 16-bit offset address. Although segmentation and memory management are provided by outboard chips, the main CPU still requires segment-addressing circuitry and a segment ID bus, and both of these take up chip area, even in the 40-pin versions that don't use segmentation.

2. Abundant resources, including sixteen 16-bit general-purpose registers, eight user-selectable addressing modes, seven main data types, and a large number of distinct instruction sets.

3. Good software regularity, which means that at least the most frequently used instructions can use all addressing modes and data lengths (up to 32 bits in the Z8000). The greater the regularity, the easier it is to write programs with a minimum of instructions, thereby speeding execution of the program.

4. A compact instruction format to insure that frequently used instructions have the fewest bits and that no bits are wasted in the instruction op code, other than for future expansion. The more compact the format, the less use of memory for a given set of program statements and the faster they will execute.

5. High-speed execution of frequently used instructions. This requires maximum effective use of the memory bus for such instructions. Bus use of 100 percent would mean that the processor spent all its time accessing memory and wasted none figuring out what to do next. At 100 percent use, execution speed is determined by RAM access speeds, not by the processor. The Z8000 has a bus-use factor of 80 to 85 percent for typical instruction mixes and up to 90 percent if jump instructions are excluded. The author's Z80 design, by way of comparison, had a bus-use time of only 65–70 percent.

It is because of bus use, by the way, that with the design of the Z80, the original 8008 instruction format had been developed to its fullest. All additional instructions would require two or three 8-bit instruction fetches, a poor use of memory bus bandwidth. The next generation had to change to a 16-bit format to get higher throughput.

To get a rough idea of how the five critical goals affect chip size, suppose the Z8000 instruction set were less regular than it is but still usable for low-end, 16-bit minicomputer applications. Only 14 000 transistors would have been needed instead of the present 17 500, and the chip size would have been 215 × 215 mil² instead of its present 238 × 256 mil². Such a processor would not, however, have had the additional features of the Z8000—namely, 32-bit operations, segmented memory, and task switching between supervisory and normal operating modes.

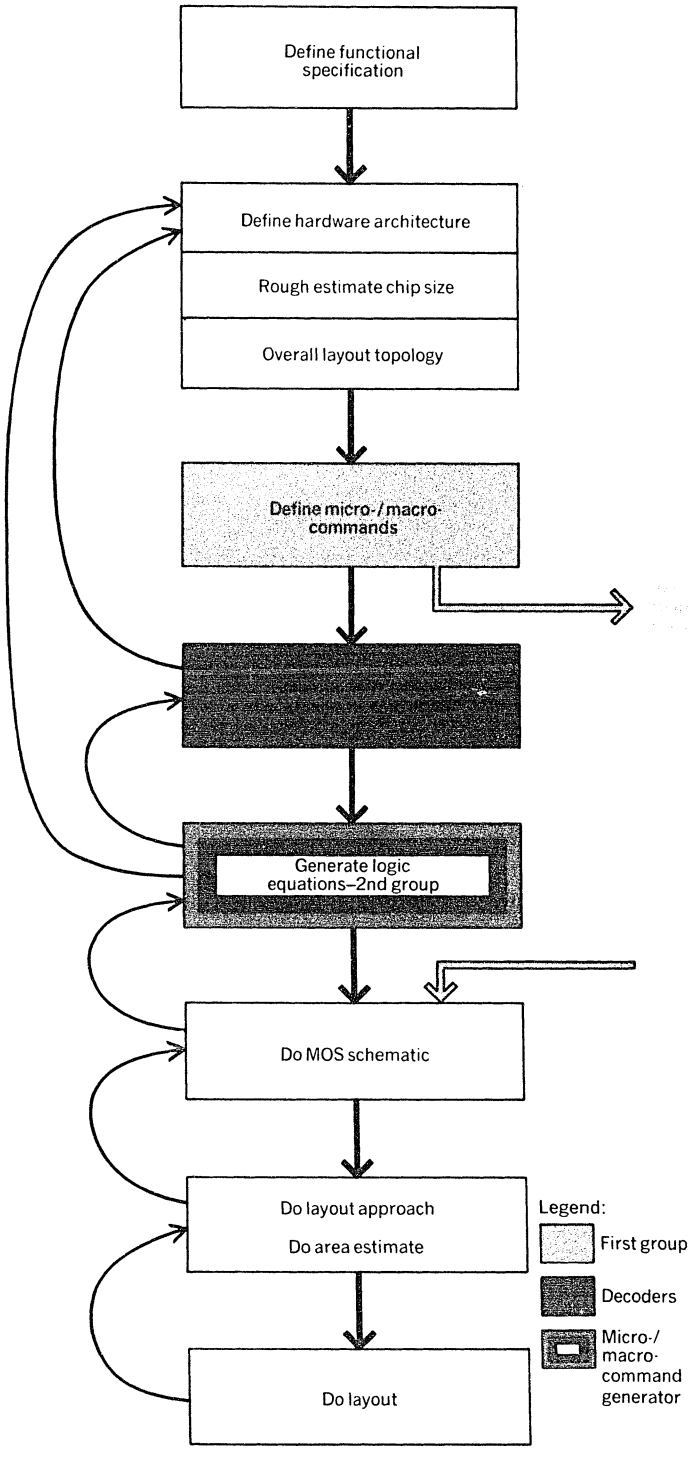
Looking at chip size vs. performance goals in another way, we find that if the same instruction set of the production Z8000 were designed to run 25 percent slower with the same LSI process, the chip area could have been reduced 20 percent. This substantial saving would have come about through reduction of device sizes needed for high speed, reduction of large transistor areas, reduction of wide power-line bus widths, and by leaving out the following extra circuitry that gives the Z8000 added speed:

- Four-bit, sliced carry look-ahead logic in the 16-bit arithmetic unit.
- The look-ahead instruction decoder.
- The redundant primary instruction decoder.
- Several different types of internal data and control busses. Had these been multiplexed, with a subsequent loss in speed, there would have been a substantial reduction in the chip area needed for subsystem interconnection.
- Many circuits used to minimize the number of clock cycles per instruction. But more complex logic does not always mean more area. The goal is to find the circuits of least area that meet the chip's performance objectives.

Taking the first step

The first step in creating any microprocessor is to specify the kinds and the total of distinct instructions that will be

[1] The top-down design cycle for the Z8000 microprocessor. In many cases it is necesary to go backward as a check against design or layout error. Color-key is the same as in Figs. [3] and [7].



needed, the degree of instruction regularity required, and the compactness of the instruction format.

The next step is to decide on the bit organization of the instruction format—that is, the number of bits in each instruction and the arrangement of bits for the data, address, and control fields. An important consideration is how much space to leave for future instructions. The logic complexity of the chip—and hence its speed and size—depends critically on the complexity of the format.

Some regularity and some compactness can be sacrificed to save chip area or to decrease the number of clock cycles for the instruction. The yardsticks for measuring the degree of regularity and compactness are by both static and dynamic checks of the frequency of instructions, including all register contents. Clearly the instruction DECREMENT AND JUMP ON NON-ZERO may appear infrequently measured statically but very frequently dynamically if the loop is repeated often.

Performance or throughput is proportional to code compactness—the less the memory-code space, the faster the execution. So for the Z8000 every effort was made to conserve compactness. Some regularity was also sacrificed to compactness, in the sense that the less frequently used instructions are not very regular. For example, ROTATE/SHIFT instructions are used only in the register mode.

If regularity alone had been considered, the ideal instruction bit length of the Z8000 would have been 21 bits. This would have permitted a completely regular instruction set and would also have resulted in assembler programs of minimum length, since with that 21-bit word length, each bit and each bit field would have served only one purpose. Each bit location would have been concerned with some addressing mode, op code, data length, source or designation field, but not with any combination of these.

Since 21-bit words obviously don't store very efficiently in byte-organized memories, the next simplest choice would be both 2- and 3-byte formats. The 2-byte group would be reserved for the most frequently used addressing modes of REGISTER, DIRECT and IMMEDIATE.

The variable-byte format would have resulted in very tight instruction packing in memory, plus good regularity, but the author felt that performance would have suffered and chip size would have increased, compared with what could be done with other formats.

The chief chip-design problem with variable-byte instructions occurs during a JUMP RELATIVE command, which may happen with a frequency of 10 to 16 percent in dynamic checks of typical program mixes. The processor does not know the number of bytes in the next instruction, and this number is needed to update the program counter correctly. Instruction-prefetch circuitry, not used in the Z8000, would have helped save time in those jumps whose next instruction location had been guessed correctly but would have been a

I. Design parameters for the 8080, Z80, and Z8000

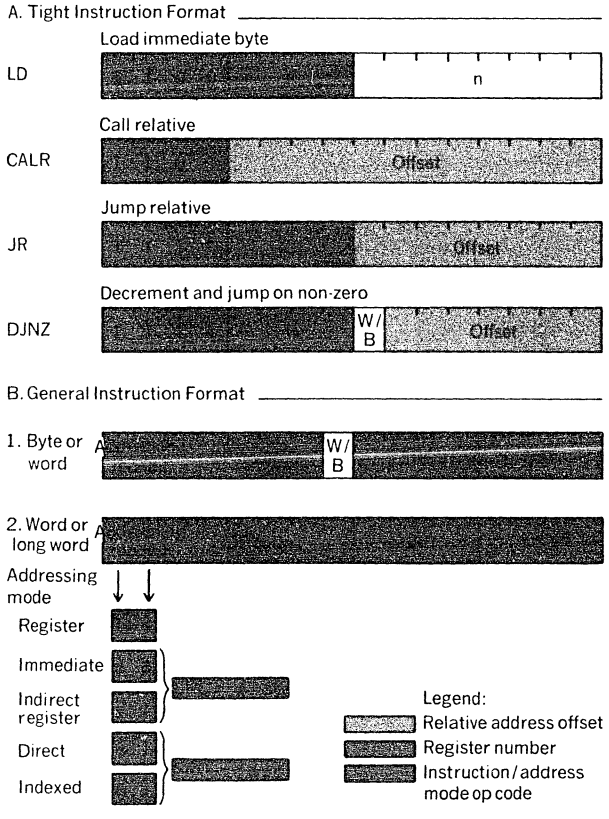
	8080	Z80	Z8000
Date of Initial Production	1974	1976	1979
Technology	N-channel silicon gate (high voltage)	N-channel silicon gate (low-voltage depletion load)	Scaled-down, N-channel silicon gate
Transistor area with scale factor defined as 1.0 for 8080	1	0.70	0.44
Instructions	65	128	414
Chip size, mm²	22.3	27.1	39.3
Total transistors	4800	8200	17 500
Transistors used to implement instruction decoders and microcode generators with "random-logic" wiring	1800	3700	8600
Percentage of total area	31	36	40
Percentage of usable area	37	44	47

II. Man-months of design spent for the Z80 and Z8000

	Z80	Z8000	Percentage of Total Time Spent by Z8000 Chip Designers
Functional definition	5	15	40
Logic/circuit design	7 S*	11 S	100
Layout design	8 S 10 A†	8 S 30 A	26
Test program debut chip characterization	6 S	10S 3 A	65

* S-senior personnel † A-associate personnel

[2] The three most frequently used Z8000 instruction classes and their formats.



hindrance in those cases where the wrong direction had been prefetched. The Z8000, for example, can make a JUMP RELATIVE in six clock cycles, which is substantially less than can be achieved with instruction prefetch techniques.

The Z8000 does have an instruction decode look-ahead used to speed instruction addressing and execution operations. Since this works only on the instruction fetched, not the one ahead of it, wrong guesses are avoided.

The even-byte alternative

The author's alternative to variable-byte instructions was even-byte, or word-aligned, instructions that were mostly one or two words long. These still provide good memory packing density, regularity, and high performance for a given clock speed, provided the instructions used most often also use the fewest words.

An advantage to these word-aligned formats—one that saves chip area—is that an instruction decode look-ahead is very easy to lay out under internal bus metal lines with a minimum of extra circuitry.

The first question to be raised about the word-aligned format is how much compactness is lost by giving up single-byte instructions and relative-address, one-byte offsets. It is true that a few Z8000 instructions, like INTERRUPT DISABLE and

A 'don't-do-it-yourself' home project

Given a pencil, 30 D-size drawing sheets, a desk calculator, occasional use of a computer and some five to 20 years of spare time, a reader who followed Masatoshi Shima's design plans for the Z80 and Z8000 microprocessors might design and optimize his own high-performance microprocessor. (By optimization is meant minimum chip area for maximum chip program throughput.) Such a do-it-yourself approach, however, would not solve the semiconductor industry's search for new designs to put on VLSI chips (see 'VLSI, some fundamental challenges," *Spectrum*, April 1979, pp. 30-37).

Fine-tuning the chip design for optimum performance—as practiced by Mr. Shima, who was also the Intel 4004 and 8080 chip designer—may prove too time-consuming for future high-performance 16-or 32-bit microprocessor chips. Still, Mr. Shima feels it will continue to be a good approach for the 8-bit microprocessor and for simple, high-speed 16-bit microprocessors.

One saving grace of future high-density VLSI may prove to be its comparative freedom from this need for time-consuming optimization of the precise execution speed of each instruction. The degree of this freedom depends on the application. At one extreme, where projected sales volume is a few thousand per year—for example, when VLSI is used to duplicate the architecture of a large mainframe—it will probably be cheaper to go for an automated design with deliberately wasted chip area rather than spend the extra man-years refining the design. At the other extreme, for general-purpose microprocessors with potential sales of several million per year, some form of human optimization will continue to be necessary to save chip area.

Present computer-aided design (CAD) programs for high-volume microprocessors leave a lot to be desired in chip-area saving. For example:

- One problem with present CAD is computer time. All the internal delay times of a complex chip can be precisely modeled by computer today, but the programming and computer execution time is too expensive for anyone to try it, except for critical paths. Simplifying the delay-time calculations in an orderly way, as done by Mr. Shima, may prove necessary.
- Some future CAD programs may prove as effective as humans, or even more so, at the final design stages and for layout. But at the early conceptual stages unavoidably subjective decisions will probably always be necessary. In the design of a new processor architecture, there is at present no extensive theory that will predict the pipelining and parallel processing needed for a given performance. There are a few programs for computer modeling of different processor architecture types, but they are not widely used at the commercial semiconductor houses.
- The intermediate stages of design, between conception and detailed layout, lend themselves in theory, but not yet in practice, to machine optimization. The difficulty is one of topology. While the number of transistors to accomplish most tasks can be minimized by use of Boolean logic equations, the area they occupy cannot be so minimized: A topologically disorganized collection of transistors will require more interconnect area, and hence more total area, than a well-organized set. Moreover top-level decisions to speed a particular section increase the transistor area in that section and require a decrease in area elsewhere. A partial solution, suggested by Mr. Shima, is to codify in some way the present expertise of the semiconductor house designers and draftsmen, so each new design team doesn't have to reinvent the wheels of the last one.

Designers of the future, as pointed out by Prof. Carver Mead of the California Institute of Technology, will be designing more "machines within machines," such as microprogram controllers and intelligent memories inside microprocessors. So topology optimizing of the future will go on at even more levels that those described by Mr. Shima. Professor Mead and his associates are exploring theoretical solutions for such complex organizations.

But until VLSI chip designers have more high-yield silicon area than they can possibly fill with useful structures, there are lessons to be learned from Masatoshi Shima's hand-optimized approach.

The basic Shima chip design of the Z8000, by the way, differs significantly from that of the Intel 8086 (see "New options from big chips," *Spectrum*, March 1979, pp. 28-34). The Intel approach used an instruction prefetch technique, perhaps well-suited to Intel's variable-byte word lengths, high-speed clock process, and memory-segmentation scheme. The Z8000 design, on the other hand, is tailored to a slower MOS process and clock and a different segmentation scheme, and it uses multiples of 16 bits for each instruction rather than the 8-bit multiples of the 8080. It does not have an instruction prefetch mechanism.

The reader should not conclude from this, however, that the Z8000 is slower than the 8086. The Z8000 uses only three clock cycles instead of Intel's four to access memory, and Mr. Shima feels that the Z8000 will execute an average instruction mix more quickly than the 8086, although that question is not addressed quantitatively in the accompanying article.

The designer has included a wide variety of optimizing circuits to speed the Z8000 performance. For example, the microprocessor uses random logic for control instead of the microprogram approach of the Intel 8086 or the Motorola 68000. The random-logic approach, at least as used by Mr. Shima, is recognized as the ultimate fine-tuning technique for best performance with minimum chip area. But both Mr. Shima and designers throughout the industry regard it as too time-consuming for the design of 16-bit and 32-bit microprocessor structures that are more elaborate than the Z8000.

—Robert Sugarman, Associate Editor

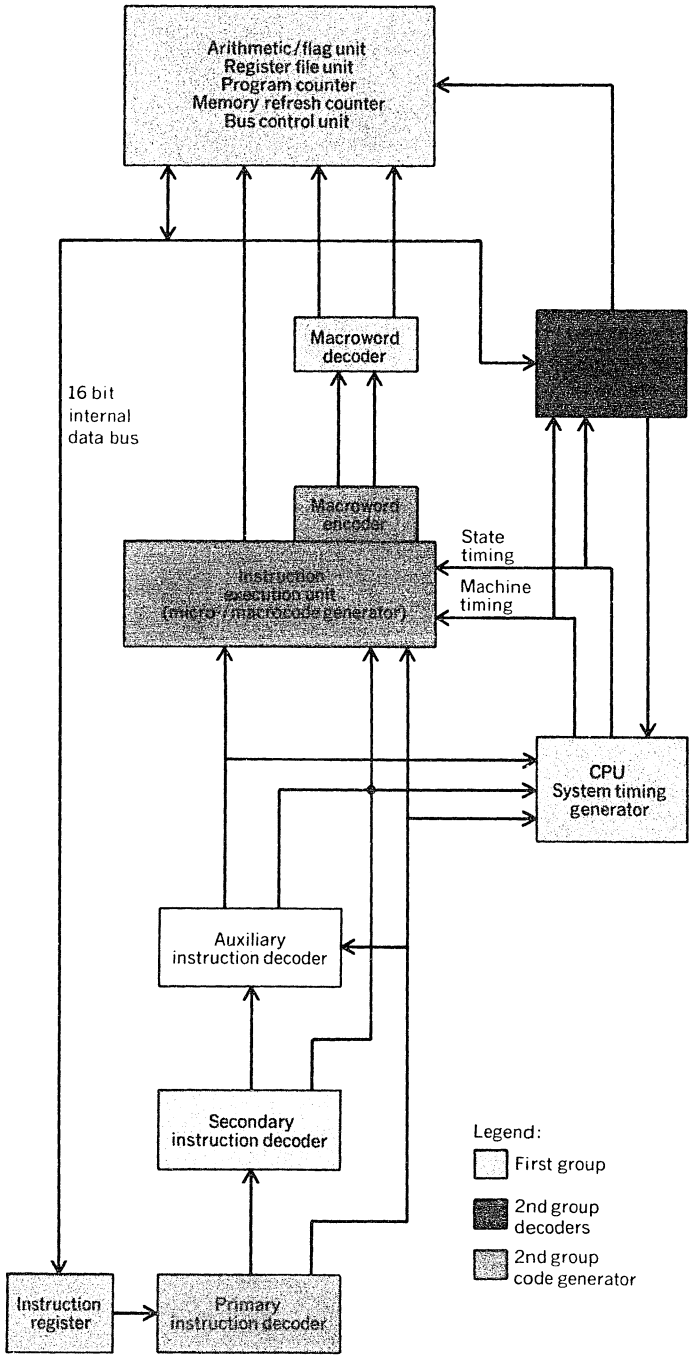
SET and RESET FLAGS, might be done in one byte, but these instructions are infrequently used.

For general address relocation, a one-byte offset relative to the program counter might have been added. But a one-byte offset is often inadequate for data relocation, especially when the programs are designed for separate program and data areas. Rather than use such an offset, selected frequently used instructions are allowed relative offsets in their instruction fields. Relocation of the other instructions and of data is done by an external memory management chip.

One-byte primitives might have been added to assist in block manipulation of characters in memory, but these were unnecessary as more sophisticated two-word preprogrammed block-transfer and string-manipulation instructions were judged to give greater performance.

The major reason, however, why one-byte instructions wouldn't work in the Z8000 was the decision to use 16 rather than eight accumulators or general-purpose registers. Eight could have created programming bottlenecks. Further, the Z8000 avoids another bottleneck that hampered previous generations of microprocessors like the Z80: It abandons object code compatibility by avoiding the restricted, implied or dedicated registers used in the 8080. For increased regularity, none of the Z8000 registers, with one exception, is ever implied in an instruction and none has special restrictions on its use. The extra op code required to handle the 16 registers in this way made one-byte register operations impractical.

[3] The Z8000 subsystem architecture. Color-key is the same as [1] and [7].



Obstacles to the first proposal

The first 16-bit format proposed by the author as an alternative to variable-byte instructions had two disadvantages: the one-byte address-offset length, which followed the instruction in the relative addressing mode, and the constant byte of data, which followed the instruction for the immediate addressing mode.

To eliminate those one-byte additions, the "tight instruction" format was introduced. It was employed only for the most frequently used one-word instructions: less frequently used instructions, like AND IMMEDIATE, were excluded for lack of op-code space. The tight instructions are JUMP RELATIVE, LOAD IMMEDIATE BYTE, LOAD IMMEDIATE WORD, DECREMENT AND JUMP ON NON-ZERO, and CALL RELATIVE.

The organization of the tight group, as well as two next-most-frequent classes of instructions are shown in Fig. 2. These formats clearly do not have absolute regularity, but the resulting loss in programming efficiency is not high statistically. The major frequency grouping is as follows:

- The five address modes used most often are permitted for most two-operand instructions. In this case only two bits are reserved in the op code for the address modes, so the extra code bit needed for five modes is obtained by determining whether the source field is all zeros, as shown in the figure. For one-operand instructions, four addressing modes and any data length are permitted.
- LOAD and STORE instructions require three more addressing modes—BASE ADDRESS, BASE INDEX, and RELATIVE ADDRESS—while a LOAD ADDRESS provides for more sophisticated addressing schemes.
- LONG WORD instructions are used less frequently than WORD or BYTE instructions and single-operand instructions less frequently than double-operand types.

All these careful distinctions of frequency of use, which made for compactness and preserved statistical regularity, were critical to the chip design phase. For best statistical performance, the instructions used most often were designed to be executed with the fewest clock cycles.

The circuitry area that would have been required for a variable-byte format was instead used in two other ways:

1. To increase the efficiency of the memory bus for frequently used instruction types and addressing modes.
2. To condense time-consuming software subroutines into simpler instructions.

Some of the simpler instructions that replace conventional subroutines are:

- Long-word division and multiplication.
- LOAD and STORE MULTIPLE.
- INCREMENT/DECREMENT BY N.

• High-speed 888 kbytes-per-second interruptible, preprogrammed block and string operations.

• A limited number of memory-to-memory operations, like STORE IMMEDIATE, PUSH IMMEDIATE, and COMPARE IMMEDIATE.

The decision not to use variable-byte architecture is typical of the complex decisions facing chip designers at the first stage of design, so it is worth discussing in a little more detail. As already mentioned variable-byte code requires instruction-prefetching for best performance. This, in turn, requires a lot of pipelines and an instruction buffer to keep up the efficiency. It also requires a clock of higher speed—say, to 6 MHz—than the clock used in the Z8000. The reason for the faster clock is that the buffer and the instruction execution unit must run asynchronously. This may cost a clock cycle if the execution unit cannot detect in time whether the buffer is empty. A short clock cycle minimizes the time lost.

Prefetch is also useful if the processor takes a long time to calculate an address, since the next instruction may be fetched in that time. The Z8000, however, is optimized for short address calculation and execution time for the most frequently used address modes and instructions.

Detailing the system architecture

Having settled on the overall architecture—in this case, word-aligned instructions, no prefetch, and no instruction pipelining—the designer can now go down one step in the design hierarchy and decide on the subsystem architecture. The subsystem may be divided into two groups: the execution and the instruction-implementation circuitry. The execution group includes:

• Arithmetic and flag section.
• Register files.
• Program counter.
• Dynamic RAM memory refresh counter.
• External address, data, and control busses and their controllers.
• Interrupt handling controller.
• CPU mode-selection unit, for selecting segmented or nonsegmented memory operation and system or normal modes.

The instruction-implementation group includes:

• Instruction register.
• Look-ahead instruction decoder.
• Primary decoder.
• Second decoder.
• Auxiliary decoder.
• Instruction-execution unit.
• CPU timing generator.

The first group's circuit design and layout won't change, even if the second group is changed drastically—say to go from random-logic to microprogram control. This is fortunate because it permits the two designs to proceed independently. The draftsmen, for example, can lay out the first group while the designers start on the second group, as was done for the Z8000. Also, instruction types and formats can be changed without effect on the first group.

But there are design tradeoffs between the two groups, so both have to be studied carefully before signing off on the design of the first group. The Z8000, for example, uses both macrowords and microwords for controlling the CPU. Macrowords, which control a group of functions, like arithmetic commands, take less complex circuitry in the implementation or control section but more circuitry in the execution section.

First-group architecture

A number of design tricks can be used to speed the execution group. In the Z80 design, for example, the author was

Partial state diagram for an ADD instruction

Instruction	IF 1				EX 1		
	T1	T2	T3	T4	T1	T2	T3
ADD data in SRC to data in DST		PC = PC + 2 B/W ALU →DST FLG	Data in →IR Flag transfer: CJ →CYF OVJ →OVF ZJ →ZF SJ →SF	SRC →DATA out (address)		B/W DST →ATM CLR CY_0	B/W DATA in →TMP

A

B

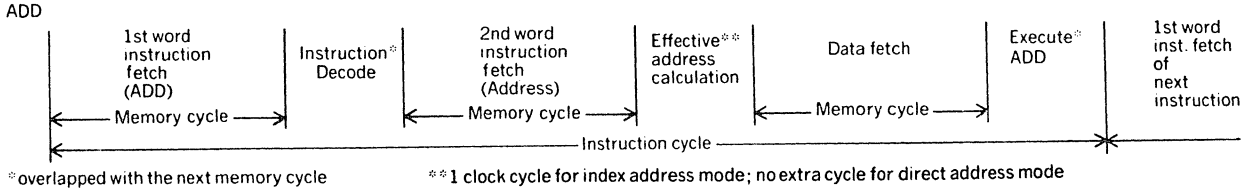
Logic equations for two microcommands used in the ADD

Micro or macro-level command	T	IF	EX	Instruction
Load internal bus to TMP	3		1	ADD; SUB and many other instructions
Send DST to ATM	2		1	ADD; SUB and many other instructions

Legend:
ALU unit has 3 operands: ALU = ATM ± TMP ± CY
IF = Instruction fetch from memory
EX = Execute
DST = Destination
SRC = Source
IR = Instruction register
B/W = Byte or word

[4] The state table (A) must be converted to microcommands (B) used by all the instructions. A portion of one state table is shown for an ADD instruction.

[5] An ADD instruction shows how cycles can be buried within memory cycles.



*overlapped with the next memory cycle

**1 clock cycle for index address mode; no extra cycle for direct address mode

able to isolate various sections with MOS switches. This allowed several to work in parallel. In particular, the following operations could all be done simultaneously:

- Data transfer between arithmetic unit and accumulator in the register file.
- Flag transfer between temporary flags and the flag register in the register file.
- Loading of instructions into the instruction register from the data bus.
- More complex logic for incrementing of the program counter.

Unlike the Z80 unfortunately, whose process was fast compared with its expected performance, the Z8000 process was very close to its expected performance, so the delay time incurred by MOS switching, as used in the Z80, could not be tolerated.

But it was important for the arithmetic section of the Z8000 to perform a complete operation every clock cycle, including testing the final result for condition flags, like CARRY or ZERO. So to save time, arithmetic and flag-testing operations were done with separate circuits, each operating in a time-overlapped mode. Separating the two functions physically also made for a cleaner layout.

Some other interesting parallel circuitry was added to the Z8000 without incurring a time delay penalty.

To allow 32-bit operations with 16-bit register files, for example, the 16-bit files were double-ported with two data lines, one for the most significant word and the other for the less-significant word. The data lines connected each bit of the register, with each data line from one register group connected to every other register.

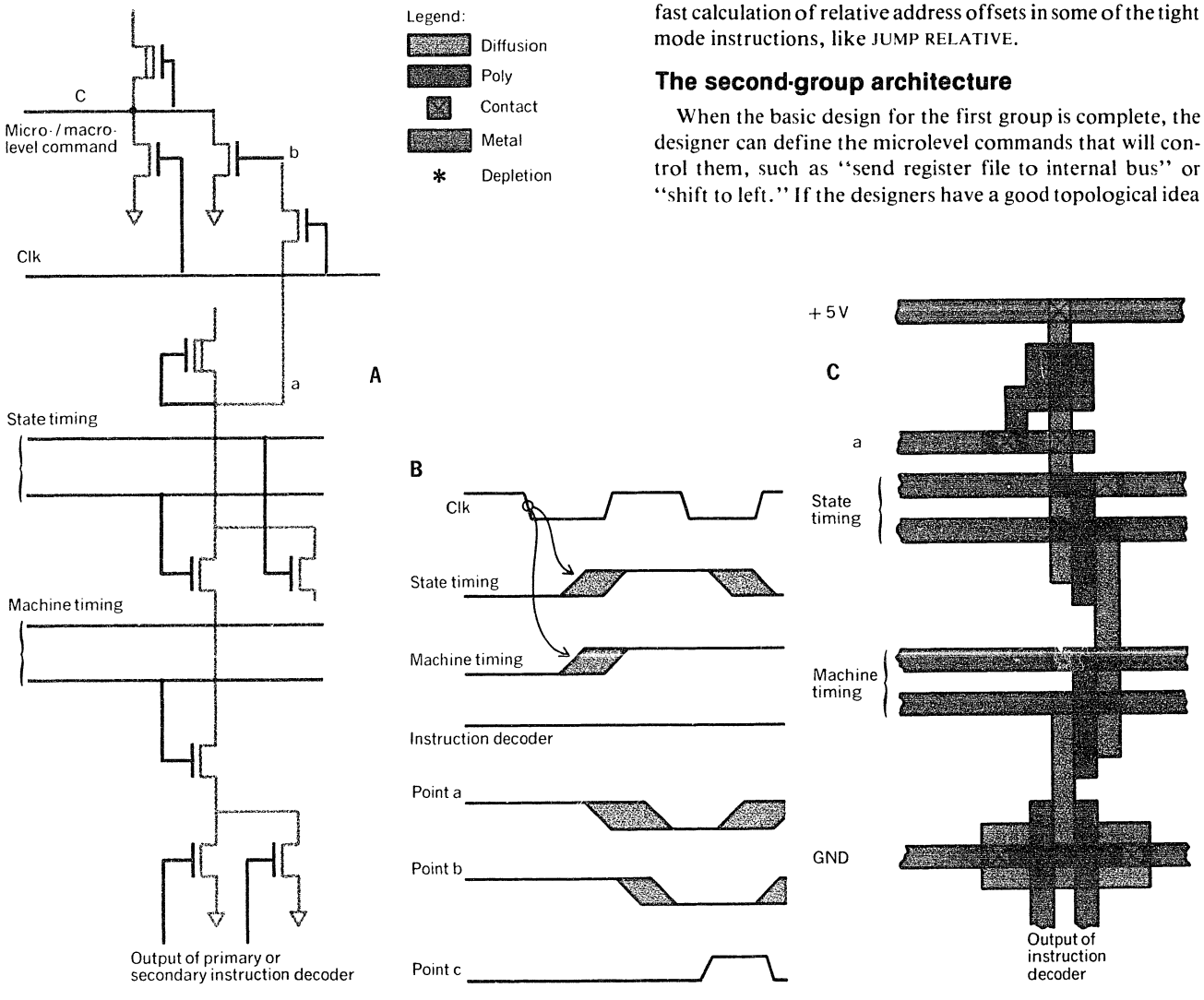
There are a number of advantages to this dual porting:

- The 32 bits can be read at once. When the register file is used in segmented mode as the memory pointer, the most-significant 16 bits of the address—which are, by definition, the segment number—can be sent to a memory-management chip, while the least-significant 16 bits go out on the address bus. Also, the segment number can be sent out one clock cycle earlier than the 16-bit offset. Thus little time need be lost in calculating the physical address in the memory-management chip.
- The most-significant 16 bits of data can be read while the less-significant 16 bits are written into the file from the internal data bus.
- Sixteen 16-bit registers can be laid out in more compact form than can eight 32-bit registers.

Another speedup hardware circuit added to the Z8000 was a shifting mechanism between the register files and the internal data bus. This shifter saves clock cycles for 32-bit data manipulation. For example, 16-bit unsigned multiplication can be done in 48 clock cycles. Another shifter is used between the arithmetic unit input and the internal data bus for fast calculation of relative address offsets in some of the tight mode instructions, like JUMP RELATIVE.

The second-group architecture

When the basic design for the first group is complete, the designer can define the microlevel commands that will control them, such as "send register file to internal bus" or "shift to left." If the designers have a good topological idea



of how much and what kind of space will be required by the execution group, its layout can be started by the draftsmen while the designers start on the instruction-implementation group.

But it is important to know the topology of each subsystem before any layout is started. An irregular layout—metal lines not running straight across a subsystem, for example—means longer interconnect lines, particularly for the internal data-control bus, and hence wasted chip area. In addition a lot of time is lost just checking the layout for continuity. The Z8000 layout is cleaner than the Z80's because the author learned from the latter that he had to spend a larger proportion of the total design time just trying to visualize the final layout.

The design rules for the process determine the gate density for the chip and hence its performance for given total area. Size is restricted in one direction by the width and separation of the metal lines; in the other direction size is determined by the gate-channel length, diffusion and poly width and separation, as well as by other geometric rules. If the designer has sufficient experience, he can choose logic that he knows will take the minimum layout space, and he can provide size estimates to the draftsmen.

Generating microcode commands

Random logic, used for the Z8000 instruction-implementation unit, could have been replaced by extensive multilevel program logic arrays (PLAs). But ROM-like organizations, implying a matrix of many transistor nodes, were inappropriate to high performance with the chosen MOS process. The large capacitive loading of the nodes would have increased internal delay times significantly.

Instead the logic was implemented with an extension of the minimal-node designs used for the 8080 and Z80; it requires less chip area and only about a third the layout time of "conventional" random logic. Because of its topological regularity, it also allowed a good estimate of chip area prior to layout.

The technique is to split the instruction implementation unit into three major sections (Fig. 3):

1. The instruction op-code decoders. These generate static commands but do not carry timing information. Since speed is not too critical here, these sections use extensive ROM-like decoding.

2. An instruction-execution unit, which uses NAND timing gates to quantify these static levels into micro- and macrowords at appropriate time slots. Timing is critical in this section, and the state-variable equations are not very complex; so the NAND gates are more effective than ROMs in minimizing the number of transistor nodes and the chip area, while maintaining good timing accuracy. The basic time slot is a single clock cycle. The CPU executes instructions by stepping through a set of machine cycles, such as MEMORY READ or WRITE, or internal operations. Up to eight machine cycles are used either for instruction fetching or execution. Each machine cycle uses a minimum of three clock cycles, with one to five clock cycles added if necessary for data manipulation. The instruction-state diagram of Fig. 4 shows how an instruction may be executed with microcommands at the proper state and machine cycles.

3. To supervise all the timing options, a CPU system timing generator provides, where necessary, up to eight state and eight machine options to the NAND gates. The decoders provide state information to the timing generator to tell it the number of fetch machine cycles, execute cycles, and state cycles in each machine cycle of the instruction to be executed.

More on priority and timing

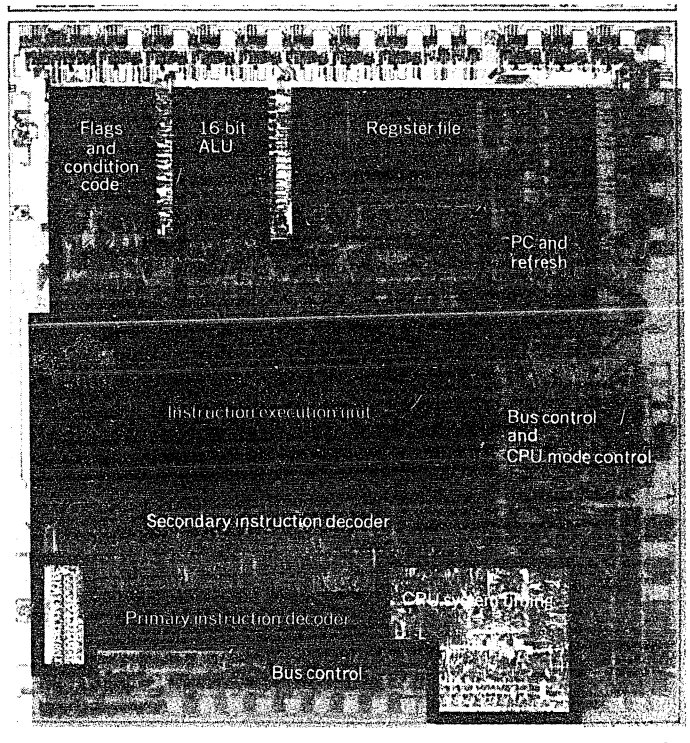
The microcode commands (microcommands) needed for instruction execution were grouped into four sections, each with a separate decoder, according to priority. Instructions with the highest priority were executed as soon as possible, those with the next highest priority after a one-clock cycle delay, the next highest priority after a two-clock cycle delay, and so on. Because there was no instruction prefetching mechanism, it was crucially important to generate the first microcommand as quickly as possible after a new instruction was loaded into the microprocessor. In order of their priority, the four decoders are the look-ahead, primary, secondary, and auxiliary units.

As an example of assigned priorities based on frequency of use, two-operand instructions were judged more important than one-operand; REGISTER and DIRECT were the most important addressing modes; and the five main addressing modes were more important than the remaining three.

The look-ahead instruction decoder operates on frequently used instructions, while the instruction is still being stored in the instruction register. DIRECT and INDEX addressing modes, for example, use the look-ahead decoder. As a consequence, the Z8000 uses no extra clock cycles to decode such a mode and to decide whether it has a short or long ad-

[7] The major subsystems of the Z8000 are shown overlaid on a photograph of the chip. The overlap map shows a relatively large portion of chip area devoted to bus and CPU mode control. The photograph shows a clean layout, with little space wasted in "meandering" interconnections. The NAND gates used in the instruction execution unit are not visible at this magnification. Color-key is identical to that in Figs. 1 and 3.

Legend:
Orange–First group
Dark blue–Primary instruction decoder; 2nd group
Light blue–Secondary instruction decoder; 2nd group
Red–Instruction execution unit; 2nd group



dress offset. In most such instructions, moreover, the data-manipulation time is fully overlapped with the fetch time of the first word of the next instruction. An example of the overlap for an ADD instruction is given in Fig. 5.

The decoder outputs go to the NAND gates in the instruction-execution unit. Since over 700 three-input NAND gates are used in the execution unit and CPU system-timing generator control unit, it is worth discussing the NAND gates in more detail.

The three-input NAND gate is shown in Fig. 6A. Its timing diagram is in 6B and the layout in 6C. One NAND input selects one of eight clock cycles per machine cycle, the second input one of eight machine states per instruction, and the third input an output of the instruction decoder. Each state and machine timing signal is generated at the falling edge of the clock.

The gate output (point A of 6A) is transferred to point B while the clock is high, and it is stored there by the gate capacity until the next clock cycle occurs. Then, when the clock goes low, the microcommand is generated. Because the clock pulse resynchronizes the output, the NAND gates can introduce a big time delay up to one clock cycle.

This comparatively large time skew allows a moderate amount of NAND gate node capacitance so the gates can be laid out perpendicular to the state and machine-timing metal lines.

The Z80 and 8080 also used NAND gates but in a less complex way, since these processors, having less complex instructions, had fewer state and machine cycles.

The instruction implementation execution unit of the Z8000 also generates some macrocommands.

Typical microcommands are used for arithmetic-unit register control, shift control, and register-file control for selecting byte or words. These macrocommands are encoded on a control bus that is connected to the execution unit. They are then decoded in the arithmetic and register-file sections of the chip.

Going down the design ladder

Two of the most difficult steps in designing the control logic of the Z8000 were converting instruction-state diagrams to the logic equations and then converting the equations to an MOS schematic. The greatest problem was to avoid leaving something out or making an incorrect interconnection—mistakes that would be costly to correct.

Elaborate precautions were taken when the MOS schematic for the control group was complete. To check for mistakes, for example, the design team went through the tedious exercise of converting the schematic back to logic equations and the logic equations back to state diagrams.

The instruction-state diagrams list what microcommands have to be given for each instruction during every clock cycle and for each fetch and execute machine state. Figure 4 shows some instruction states and their conversion to microcommands.

Estimating speed

The key point in converting the logic equations to the MOS schematic is to keep the schematic as close as possible topologically to the final layout (Fig. 7). If this is done, the designer can then estimate the internal delay times by calculating the loaded capacity for each logic node. For the Z8000, the designers prepared tables listing the critical capacities for each important node. Estimates were made of the probable layout, line lengths, and spacings and overall topology—for example, what and how many metal lines would be placed between two power busses. Then, using computer-generated circuit simulation for a matrix of preassigned capacities, the designers extrapolated the results with a pocket calculator and could predict the delay times with about 90 percent of the accuracy of a total computer simulation. This simple procedure worked because even the largest network used for computer delay simulation can be divided into several basic elements, such as invertors, flip-flops, and drivers.

But good communication remains essential between the layout draftsmen and the designers. The designer must know the particular layout procedures used by each draftsman and must be sure that critical propagation delay paths are laid out as anticipated. As a further check, when the layout is completed, it must be rechecked for proper interconnections and node capacitance where critical.

After computer digitizing of the layout to make a mask, and after a few production dies have been obtained, the logical operations are rechecked. This also permits a characterization of performance vs. clock speed and temperature.

For die checkout, the author used a microprocessor-controlled checkout system that included 1 k of RAM memory for fast dumping of the microprocessor's contents. With this system, the state of all registers, flags, and the address/data control bus at each machine cycle can be modified and stored on a floppy disk, displayed on a CRT or printer and later converted into a test pattern for functional testing. This technique saved at least three to four months of debugging time. ◆

For further reading

For further details of the Z8000 and other 16-bit microprocessors, the following reading may be beneficial:

"Two versions of 16-bit chip span microprocessor, minicomputer needs," Masatoshi Shima, *Electronics*, Dec. 21, 1978, p. 81.

"Architecture of a new processor," Bernard Peuto, *IEEE Computer*, February 1979, p. 10.

"New options from big chips," James McKevitt and John Bayliss, *IEEE Spectrum*, March 1979, p. 28.

"Computers: our 'micro-universe' expands," Robert Sugarman, *IEEE Spectrum*, January 1979, p. 32.

Masatoshi Shima received a B.S. in chemistry from Tohoku University, Japan. Now at the Intel Corp., he formerly was manager of high-end microprocessors at Zilog Inc., where he was responsible for the chip design and development of the Z8000 16-bit microprocessor. While an employee at Busicom Corp. in Japan, he worked with Intel on the development of the MCS-4 microcomputer family, which included the 4004 microprocessor. Then, as supervising engineer for Intel, he was responsible for the detailed design and development of the microprocessor CPU and production tester for the 8080 system and supervised the development of several 8080 support chips. At Zilog he was also responsible for the design and development of the Z80 family and for the detailed design and development of the Z80 CPU and production tester.