Simplifying microcomputer architecture

J. Weisbecker

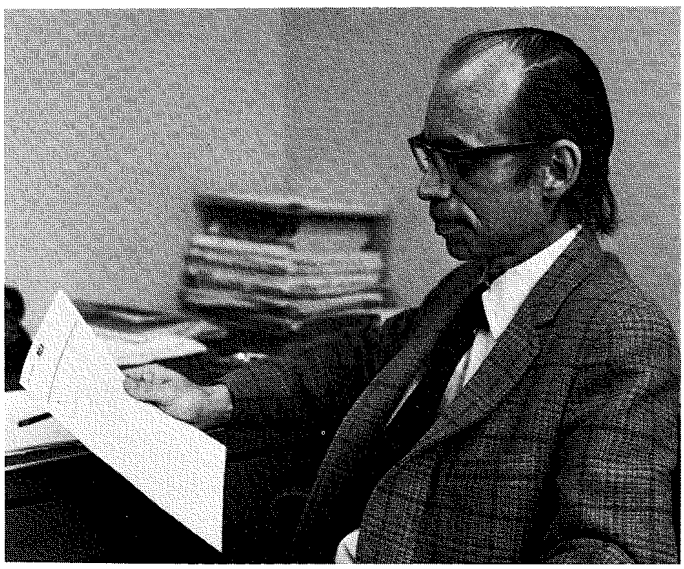
In 20 years computer hardware has become increasingly complex, languages more devious, and operating systems less efficient. Now, microcomputers afford the opportunity to return to simpler systems. Inexpensive, LSI microcomputers could open up vast new markets. Unfortunately, development of these markets may be delayed by undue emphasis on performance levels which prohibit minimum cost. Already promised are more complex next generation microcomputers before the initial ones have been widely applied. This paper discusses these points and describes a simplified microcomputer architecture that offers maximum flexibility at minimum cost.

LARGE SCALE INTEGRATION (LSI) of semiconductor devices has finally become a practical reality. The long-awaited revolution in electronic products appears to be at hand. The basis of this revolution is the ability to provide complex electronics at greatly reduced prices. Major cost reduction opens up entirely new markets and is as significant a development as the invention of the vacuum tube or transistor.

The four-function electronic calculator represents the first wave of the revolution. Further new markets will emerge with the ability to provide complete stored program computers at a fraction of current minicomputer costs. A number of microcomputer chip sets have already been announced.[1,2] We can expect a proliferation of microcomputer types and products based on them over the next several years. Unfortunately, old habits are hard to break and we can also expect to see increased emphasis on performance instead of cost.[3] This could easily obscure the fact that many major new markets will depend primarily on absolute cost.[7,8]

Consumer, educational, small business, and communications markets are prime targets for truly low-cost microcomputer based products. The architecture described in this paper was developed to satisfy the requirements of these potential new markets. Practical, stand-alone systems (including input/output device control and memory) requiring as few as six LSI chips are feasible with this architecture. Such systems have been breadboarded and programmed. Based on this experience, the microcomputer described appears to satisfy the requirements of a much wider range of applications than originally intended. It also appears to be simpler than most existing microcomputers. It is estimated that this new architecture compares favorably with the complexity of current four-function calculator chips. This simplicity is expected to provide significant production advantages. Improved yields and decreased testing costs are anticipated.

Since LSI improvements are permitting ever larger numbers of devices per chip, there are definite long-term advantages in minimizing microcomputer complexity. If the microcomputer is prevented from growing in complexity as the device per chip ratio improves, more of the system can be pulled back into a single chip. For example, small systems in which both memory and microcomputer are provided on a single chip become feasible resulting in added, long-term cost-reduction potential. This approach is ruled out when increased device per chip ratios are used to provide more complex microprocessors.



Joseph A. Weisbecker, LSI Systems Design, Solid State Technology Center, RCA Laboratories, Princeton, New Jersey, graduated from Drexel University in 1956 with the BSEE. Prior to joining RCA Laboratories in 1970, he was active in various product planning positions within RCA's Computer Systems Division. He has made major contributions to RCA computer development since 1953. During a three-year sojourn with a smaller company, he developed a number of customized data terminals. Mr. Weisbecker holds 19 patents with others pending. He is a member of IEEE and Eta Kappa Nu, and he has received an RCA Laboratories Achievement Award. His children have had a computer at home for several years which reinforced his interest in low-cost, widely available microcomputer systems.

Reprint RE-19-5-10
Final manuscript received November 26, 1973.

Design philosophy

Minimum system cost is the primary goal. To achieve this goal, an architecture is required that is both simple and flexible. Simplified computer architecture has received relatively little attention in the literature. Prior approaches toward simplified computers appear to be incompatible with microcomputer application goals.[4,5,9]

The architecture that was finally developed evolved from examining proposed applications. Another approach would have started with a more or less conventional minicomputer architecture and instruction set. This latter approach was discarded due to fundamental differences in minicomputer and microcomputer applications. It was also felt that a minicomputer starting point would not yield the simplest architecture.

Since a single-chip microcomputer promises minimum cost, the architecture was constrained to a 40-pin interface. Smaller microcomputer interfaces tend to require extensive multiplexing of interface signals which adds demultiplexing logic external to the microcomputer chip. This increases system cost.

An 8-bit parallel (or byte) architecture was chosen. This yields maximum performance consistent with interface pin constraints and is compatible with input/output requirements. One and four-bit organizations unduly restrict the range of potential applications. Sixteen or more bits exceed single-chip pin constraints or impose the need for multiplexed word transfers.

Since continued memory cost reduction is anticipated, techniques using memory to reduce hardwired logic complexity are heavily relied on. It is also apparent that many microcomputer applications will fall into the intelligent buffer category. For these reasons, direct memory addressing capability of up to 64K bytes is provided. Random-access memory (RAM) and read-only memory (ROM) can be mixed in any combination via a common memory interface. This is a distinct simplification over an architecture that provides separate RAM and ROM interfaces. The common RAM/ROM interface also enhances flexibility. System simplicity results since a single LSI chip containing both ROM and RAM segments will suffice for many applications.

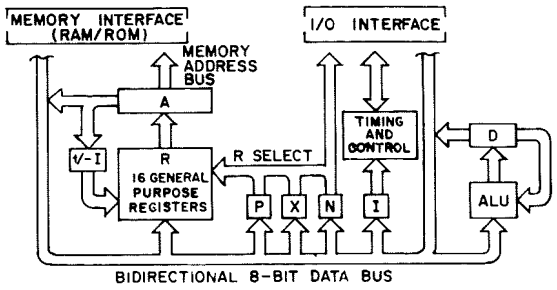


Fig. 1 — Microcomputer architecture.

While low memory costs can be expected, very low-cost systems will result only from minimizing memory capacity requirements. A unique architecture was devised which uses an 8-bit instruction format. This permits compact programs and subroutines. Useful systems requiring 1024 bytes or less of memory are possible with this format.

Random control logic uses chip area less efficiently than register/memory arrays. For this reason a simple, fixed cycle, microinstruction set was developed to reduce required control logic. The user has the option of programming directly in microcode, using a set of subroutines stored in memory (ROM/RAM), or a combination of these approaches. Sets of subroutines stored in memory are equivalent to applications-oriented macroinstructions and can readily be provided where ease of programming is important. On the other hand, many systems will utilize the microcomputer as a substitute for special purpose logic and can be programmed directly and efficiently in microcode. This approach retains most of the advantages of a microprogrammed computer but eliminates much of the specialized, hardwired sequencing and control logic usually associated with microprogrammed systems.[6] Simple, short-subroutine-calling byte sequences provide flexible macroinstruction definition.

Whether used as a component of larger systems or as a freestanding computer, the microcomputer architecture requires efficient, flexible, input/output capability. This is provided via programmed byte transfers and a built-in direct memory access (DMA) channel. Programmed input/output byte transfer instructions provide maximum flexibility for input/output selection, control, and data transfer. The DMA channel facilitates high-speed I/O block transfer, TV display refresh, and initial program loading with a minimum of external logic. While the inclusion of a DMA channel adds negligible complexity to the microcomputer architecture, it greatly simplifies system design, leading to reduced overall cost. In addition to the two basic I/O modes, four uncommitted flag lines are provided for activation by external devices. These flags can be tested as required by program. A flexible program interrupt capability also exists. Individual reset and load lines minimize external initializing logic.

The overall design philosophy consisted of developing a simple, flexible, microcomputer architecture which satisfies a wide range of potential applications at minimum cost. Performance levels were chosen to satisfy large-volume applications without overkill. The resulting architecture can be implemented initially on one or two chips using slow MOS technology.

Instruction execution times in the range of 4 to 8 μs are anticipated with LSI technologies that approach current TTL speeds. Experimental work has demonstrated that this performance level is adequate for most anticipated applications.

Microcomputer architecture

Fig. 1 illustrates the microcomputer architecture. "R" represents an array of sixteen, 16-bit general purpose registers. (This is essentially a 16x16-bit RAM.)

Registers P, X, and N are three 4-bit registers. The contents of P, X, or N select one of the 16 R registers. Register R(N) will be used to denote the specific R

register selected by the 4-bit hex digit contained in the N register. R0(N) denotes the low-order 8 bits (byte) of the R register selected by N. R1(N) denotes the high-order byte. The contents of a selected R register (2 bytes) can be transferred to the A register. The 16 bits in A are used to address an external memory byte via an 8-bit multiplexed memory address bus. The 16-bit word in A can be incremented or decremented by "1" and written back into a selected R register.

M(R(N)) refers to a one-byte memory location addressed by the contents of R(N). This indirect addressing system is basic to the simplicity and flexibility of the architecture.

Register D is an 8-bit register that functions as an accumulator. The arithmetic logic unit (ALU) is an 8-bit logic network for performing binary add, subtract, logical "and," "or," and "exclusive or" on two 8-bit operands. One operand is the bus byte and the other is contained in the D register. The D register can also be shifted right by one bit position. Add, subtract, and shift operations set a 1-bit overflow register (not shown) which can be tested by branch instructions.

The I is a 4-bit instruction register. Four-bit operation codes are placed in I and decoded to control instruction execution. Bytes can be read onto the common data bus from any of the registers, external memory, or input/output devices. A data bus byte can, in turn, be transferred to a register, memory, or input/output device.

The operation of the microcomputer is best described in terms of its instruction set. A one-byte instruction format is used as shown in Fig. 2.

Each instruction requires two machine cycles. The first cycle causes an 8-bit instruction to be fetched from external memory and placed in the I and N registers. This is written as M(R(P))→I,N. The 4-bit digit in register P selects R. R(P) is then transferred to A and used to address memory. While waiting for the

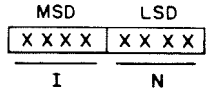


Fig. 2 — Eight-bit instruction format.

Table I — Instruction set for the microcomputer.

Hex digit	Operation
	Register Operations
[1]	Increment R(N) by 1
[2]	Decrement R(N) by 1
[8]	Transfer R0(N) to D
[9]	Transfer R1(N) to D
[A]	Transfer D to R0(N)
[B]	Transfer D to R1(N)
[C]	Transfer D0 to R00(N)
	Memory Operations
[4]	Load D from M(R(N)) and Increment R(N)
[5]	Store D in M(R(N))
	Miscellaneous Operations
[0]	Idle
[3]	Branch
[6]	Input/output byte transfer
[7]	Interrupt control
[D]	Set P to value in N
[E]	Set X to value in N
[F]	ALU operations

memory access, A is incremented by "1" and replaces the original contents of R(P). The most significant digit of M(R(P)) is placed in register I and the least significant digit is placed in register N. At the end of an instruction fetch cycle, I and N always contain the 8-bit instruction originally addressed by the current program counter [R(P)], and this program counter has been incremented to point to the next memory byte in sequence. Note that any R register can be selected as the current program counter by merely changing the digit in register P. Multiple program counter systems and simple branch and link operations are readily achieved with this approach.

The next machine cycle always causes the instruction contained in registers I and N to be executed. This fixed two-cycle, fetch-execute sequence simplifies control logic and permits program interruption or DMA cycle stealing to occur only between instructions. This results in even further control simplification. Because the operation code in register I is limited to 4 bits, only 16 instruction types need be decoded. The 16 possible operations specified by the hex digit in I are listed in Table I.

The first group of instructions permits selecting any 16-bit general purpose register (R) and incrementing or decrementing it. Upper or lower halves of selected R registers can be copied into D or set from D by these instructions. Operation "C" permits the least significant 4 bits of D to be set into the least significant 4 bit positions of any R register. This facilitates table-lookup operations using 4-bit digit arguments.

The two basic memory operations permit loading D from memory and storing D in memory. Used in combination with the register operations, selected general purpose registers can be set or stored. Instruction "4" is of particular interest. When N equals P, this instruction permits a byte to be retrieved directly from the program stream and placed in D. Since R(N) is the program counter, incrementing it maintains program counter integrity. A three-byte sequence serves to set a one-byte constant into any R register. This technique is normally used for initialization of R registers.

The last group of operations provides a variety of functions. The idle state can be entered via program or a reset line provided in the microprocessor interface. The idle state waits for externally generated program interrupts or DMA requests. The branch instruction performs a test specified by the value in N. As a result of this test, the next byte in memory, as addressed by R(P), is either skipped or placed in the lower half of R(P). This latter action causes a branch within the current 256-byte memory segment. Tests specified by N include zero in D, the states of four externally activated flags, and the status of the ALU overflow register. Two instructions, "D" and "E", permit the current digit in the P or X register to be modified. The "D" instruction provides the ability to change program counters at any point in a program. For example, "D4" would immediately change the current program counter to R(4). Branch and link operations are thereby facilitated. The "E" instruction permits changing the ALU operand or input/output byte address pointer. Instruction "F" permits 8-bit ALU operations. N designates the specific ALU operation to be performed. One of the operands comprises the byte contained in D. The other operand comes from memory. The second operand can be addressed by either R(P) or R(X) as specified by N. The result of ALU operations always replaces the original byte in D.

Instruction "6" permits byte transfers between memory and input/output devices via the common byte bus. The value of N specifies the direction of the byte transfer. M(R(X)) can be sent to an input/output device or any input/output device byte stored at M(R(X)). In the former case R(X) is incremented permitting X to be set equal to the current P value. The digit in N is made available externally during execution of the input/output byte transfer instruction. This digit code can be used by external I/O device logic to interpret the common bus byte. For example, specific N codes might specify that an output byte be interpreted as an I/O device selection code, a control code, or a data byte. Other N codes might cause status or data bytes to be supplied by an I/O device. In small systems the N code can directly select and control I/O devices.

Four flag lines that can be activated by I/O devices are provided. These can be used as general purpose I/O device status indicators (byte ready, error, etc.) These flag lines are tested by the branch instruction. Two interface lines control the built-in DMA channel. An I/O device can activate either an input or an output byte request line. At the end of the execution of the current instruction, a DMA channel cycle will occur causing the requested memory I/O device byte transfer to occur. R(0) is used for addressing memory during DMA cycles and is automatically incremented by one, following each byte transfer. Once initiated, blocks of data can be efficiently transferred between an I/O device and memory, independent from normal program execution.

A program interrupt line can be activated at any time by external means. At the end of the current instruction, an interrupt cycle will occur. During this cycle, X and P are placed in an 8-bit temporary storage register (T); P is then set to 1 and X is set to 2. Normal fetching and execution is then resumed. Activation of the interrupt line therefore causes a branch to the instruction stream addressed by R(1). R(2) should point to a memory area used by the interrupt routine to store the state of the machine for subsequent return from interrupt. Instruction "7" with N equal to 8 stores the contents of T in the memory location specified by R(X). It is a "save state" type instruction. If N is O, instruction "7" causes M(R(X)) to be placed in P and X. R(X) is incremented and an interrupt mask bit is reset. This instruction provides a "return after interrupt" function. The interrupt mask bit inhibits further responses to external activation of the interrupt line. This mask is always set by an interrupt permitting multiple interrupts to be queued under program control.

Programming considerations

Since the instruction set of this microcomputer differs considerably from that normally encountered, some comments relative to programming are in order.

A major difference between this architecture and more conventional organizations lies in the complete separation of operation codes and memory addresses. Conventional instructions have one or more addresses associated with each operation code. This system utilizes a limited table of memory addresses contained in a set of general purpose registers. These registers may also be used for program counters and data storage. Their use is entirely controlled by program, with the exception of R(0), R(1), and R(2). This structure is basic to the simplicity and flexibility of the architecture. It also permits the use of a short, 8-bit instruction format resulting in compact coding.

It has long been realized that storing a full memory address with each operation code is inefficient since a small number of unique memory addresses are repeated many times throughout a program. Various abbreviated addressing schemes have been used to permit more compact programs. These are almost always offered as optional alternatives to providing a full address in each instruction. Here we must always obtain a memory address from the limited, current set in the 16 general-purpose registers. We might intuitively suspect that this would be an unduly restrictive approach. Surprisingly, it turns out to be relatively easy to write programs and is highly efficient relative to the amount of memory used. A variety of programmers, from those who have only used high-level languages to engineers with limited programming experience, have had little difficulty in adapting to this architecture.

A number of programs have been written for a breadboard version of the microcomputer with a variety of input/output device attachments. This experience has validated the flexibility and efficiency of the architecture. For example, a four-function calculator program was found to require only 1024 bytes of memory, most of which could be ROM. This included provision for keyboard input; operands up to 30 digits; TV display refresh storage; 5x7 digit font conversion table; push-down stack; and algorithms for signed, *n*-digit decimal add, subtract, multiply, and divide. An interpreter for a simple, decimal, tutorial language was written in less than 600 bytes. A number of game and/or educational programs require well under 1000 bytes of memory. Many small business and communications systems programs are possible with 2000 to 4000 bytes of memory.

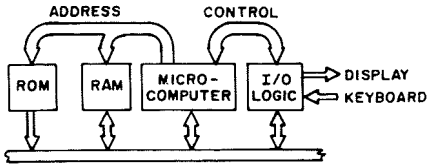


Fig. 3 — Calculator system.

While the instruction set initially appears quite limited, it should be kept in mind that each operation requires only one byte of storage (ROM or RAM). Short sequences of these microinstructions readily provide macro-operations.

Apparent weaknesses in the architecture are the limited branch capability (within 256 bytes) and the lack of a hardwired program stack for multilevel nested subroutines. These apparent oversights are the result of a deliberate design philosophy which eliminates special purpose logic for those functions which are performed easily by subroutines. The architecture permits a flexible subroutine "call" and "return" system requiring less than 70 bytes of memory. This includes a push down stack for nested subroutines. By providing this system in software (or firmware) it can be tailored to individual applications.

Where extensive programming effort is required, a set of applications-oriented subroutines is easily developed. These subroutines constitute a user-oriented macroinstruction set for minimum effort programming. This technique has proved extremely useful in an experimental small

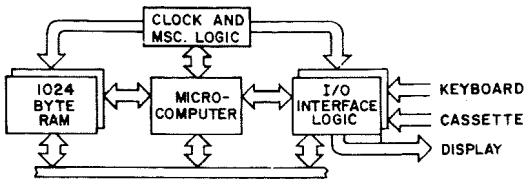


Fig. 4 — Six-chip, stand-alone system.

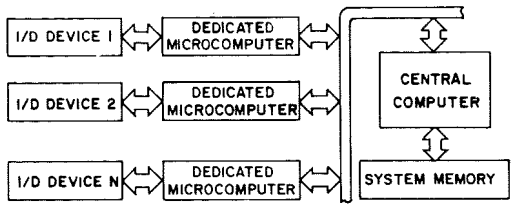


Fig. 5 — Dedicated multi-microcomputer system.

business system.

For microcode programming, an assembly language has been developed. This approach simplifies machine language coding considerably. An interactive simulator greatly facilitates initial program debugging. Both of these microcomputer software support systems are readily modified to run on existing time-sharing systems.

In general, the simplified microcomputer presents no difficulty in programming. It provides a simple set of short, easy to understand microinstructions that do not require high skill levels to use. For specific applications, tailored macroinstructions are readily provided via a flexible subroutine handling system.

Typical systems

Several systems using the microcomputer can be outlined. Many others are, of course, possible.

Fig. 3 indicates a possible microcomputer-based calculator. ROM and RAM might be provided on one chip resulting in a basic three-chip calculator. Functions could easily be added with ROM increments. This type of system could also provide a programmable calculator.

Fig. 4 illustrates a stand-alone system which might require only six LSI chips total.

It is assumed that 4x1024-bit memory chips will be available within the next several years. Subsequent LSI improvements could further reduce the chip count. Use of a small keyboard, audio cassette,[10,11] and CRT display might reduce system cost to a few hundred dollars. Such a system could have wide application in consumer and educational markets. This system, with more memory, hardcopy output, and low-cost floppy disk (or magnetic bubble bulk storage), would provide the basis for a wide range of inexpensive, turnkey, small business systems.

Fig. 5 illustrates a large computer system in which each I/O device is controlled by a dedicated microcomputer providing an intelligent buffer, as well as a replacement for special purpose logic. RAM, ROM, microcomputer, I/O device and central computer interface circuits could readily be provided on a small set of LSI chips. The microcomputer DMA feature is extremely useful for high-speed block transfers in this type of system. Downline loading of the microcomputer memory can immediately change its mode of operation. Off-line editing and maintenance is provided free. This type of large-scale system approach will become more popular in the future as microcomputer costs decrease.

The performance level of the simplified microcomputer described is more than adequate for the above types of systems as well as many others.

Conclusions

Much current microcomputer development effort appears to be directed toward improved performance. There is, however, a need for simple, minimum cost structures that will satisfy large-volume applications which do not require minicomputer performance levels. These microcomputers must also be organized to reduce total system cost. One such microcomputer architecture has been developed. It promises low cost, together with minimum external memory and system logic requirements. Hopefully, microcomputers of this class will accelerate the development of major new markets.

Currently high input/output device costs might be used as an argument against minimizing microcomputer cost. This is extremely short-sighted. The availability of ten-dollar microcomputer chips will, by itself, exert considerable pressure on the development of compatible low-cost I/O and bulk storage devices. Even now there are many potential new products that demand minimum cost microcomputers of the type described.

Because of its flexibility and potential for low-cost systems, RCA is currently developing a COS/MOS-LSI version of this microcomputer — COSMAC, SOS versions are also being investigated for applications requiring higher instruction execution rates. Both implementations are expected to find wide application in a variety of future products.

Acknowledgments

The following people have devoted considerable effort toward evaluating and developing software for the microcomputer described here: S. Heiss, A.R. Marcantonio, J.T. O'Neil, A.D. Robbi, P.M. Russo, R.O. Winder, and C.T. Wu. Without this effort, validation of the architecture would have been impossible.

References

1. Lapidus, G., "MOS/LSI Launches the Low-Cost Processor," *IEEE Spectrum* (Nov. 1972) pp. 33-40.
2. Sideris, G., "Microcomputers Muscle In," *Electronics* (March 1, 1973) pp. 63-64.
3. Hoff, M.E., Jr., "Applications for Microcomputers in Instrumentation," 1973 IEEE Intercon Papers, Session 21/1.
4. Hitt, D.C., Ottaway, G.H., and Shirk, R.W., "The Minicomputer—A New Approach to Computer Design," *Proc. of* 1968 *Fall Joint Computer Conference*, pp. 655-662.
5. Conn, R.W., "The Dinkiac I," *Proc. of* 1971 *Spring Joint Computer Conference*, pp. 1-9.
6. Reyling, G., Jr., "LSI Building Blocks for Parallel Digital Processors," 1973 IEEE Intercon Papers, Session 21/3.
7. "Backer Sought for Information Center," *Electronics* (Mar. 29, 1973) pp. 33-34.
8. "Personal Lifetime Computer Foreseen," *Electronics* (Sept. 11, 1972) pp. 40-42.
9. *KENBAK-1 Computer Programming Reference Manual*, KENBAK Corp. (April 1971).
10. "Putting Data on an Ordinary Audio Recorder," *The Electronic Engineer* (May 1972) p. DC-9.
11. Wolf, E., "Ratio Recording for Lower Cassette Recorder Cost," *Computer Design*, (Dec. 1972) p. 76.