

"MICROPROCESADORES EN DISEÑO DIGITAL"

DR. PAUL M. RUSSO
RCA Research Center
Princeton, N.J.

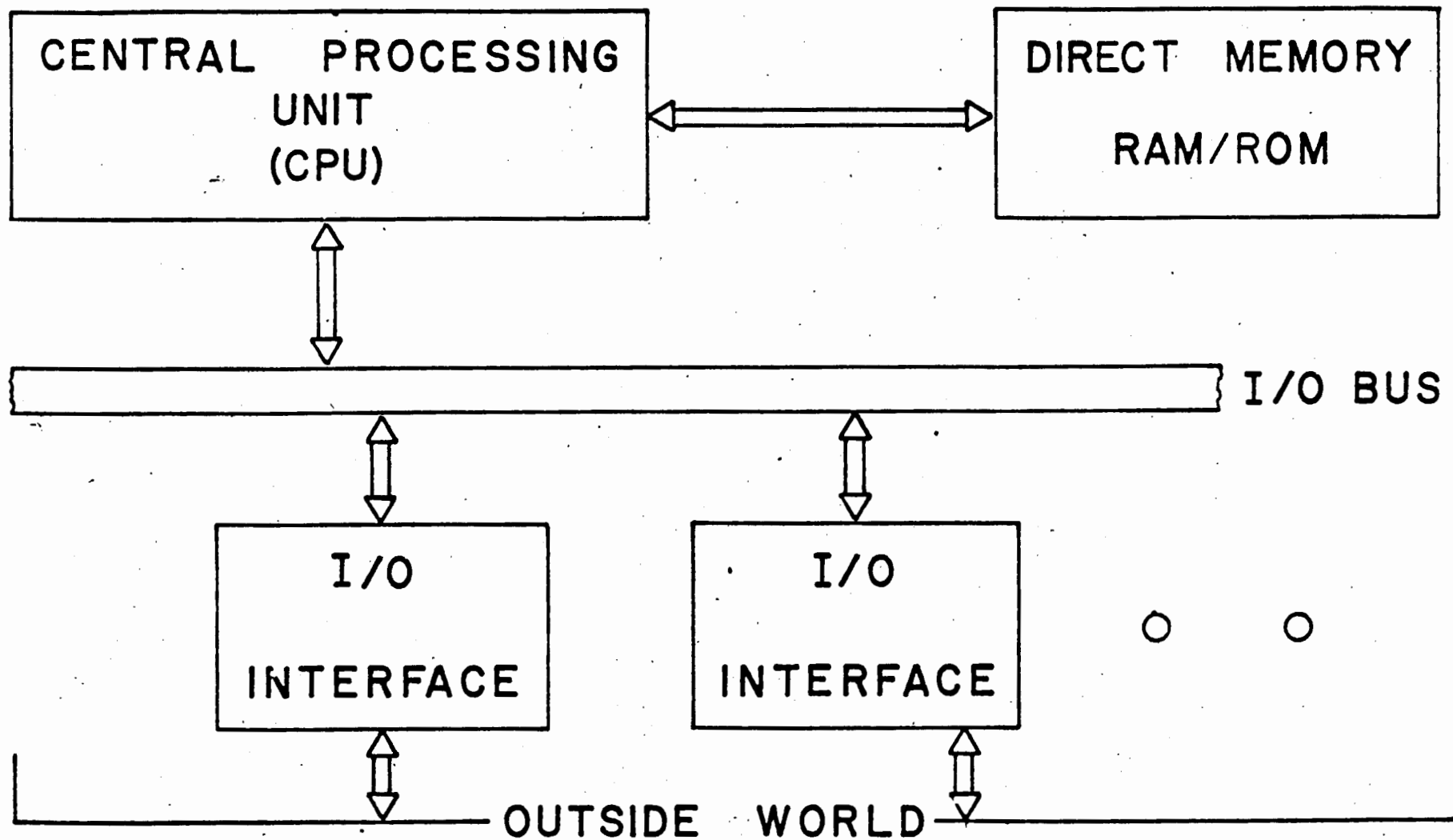
Curso Corto; Sección de Ingeniería Electronica
DESFI, UNAM

COURSE

- I
 - INTRODUCTION
 - BASIC PROCESSOR OPERATIONS
- II
 - DETAILED DESCRIPTION OF "COSMAC"
- III
 - COMPARISON WITH OTHER μ P'S
- IV
 - I/O STRUCTURES
- V
 - APPLICATIONS
 - HOME/SCHOOL
 - DATA COMMUNICATIONS
 - PROCESS CONTROL/AUTOMATION
- VI
 - MULTI-MICROPROCESSORS

INTRODUCTION & BASIC PROCESSOR OPERATIONS

- WHAT IS A μ P
- μ P APPLICATIONS
- HOW TO DESIGN WITH μ P
- STATUS



COMPUTER SYSTEM

I/O DEVICES

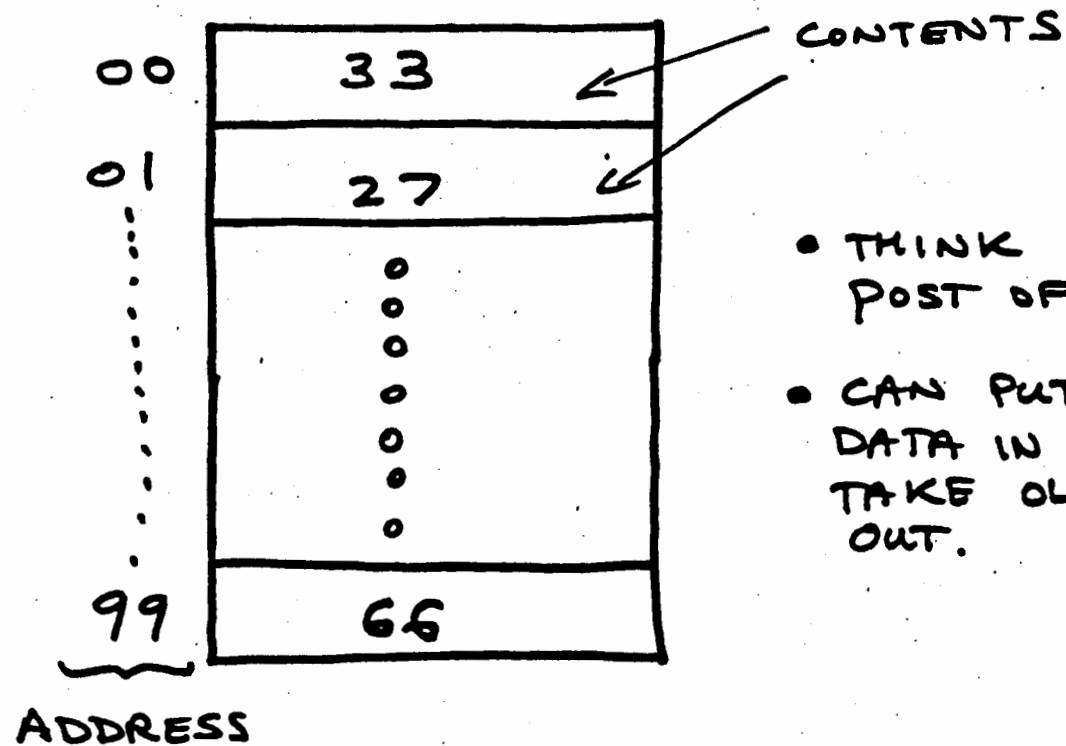
- TTY'S
- CRT'S
- PRINTERS
- DISPLAYS
- DISCS
- MAGNETIC TAPE
- SENSORS
- RELAYS
- ANALOG DEVICES
 - A/D
 - D/A

MEMORY : NEEDED BY ANY COMPUTER

- ADDRESS
- INFORMATION AT ADDRESS

MEM. CONTENTS

- DATA
- INSTRUCTIONS



- THINK OF POST OFFICE
- CAN PUT NEW DATA IN OR TAKE OLD DATA OUT.

MICROPROCESSOR: CPU ON 1 CHIP

TECHNOLOGY LIMITS: CHIP AREA →
LIMITS FUNCTIONS

PACKAGE → LIMITS
PINS

MULTI CHIP SETS (EG, BIPOLAR SLICES
OR LSI-11 TYPE SETS)

ARE NOT TRUE MICROPROCESSORS.

THEY ARE MINICOMPUTER BUILDING
BLOCKS.

EVERY COMPUTER : - FETCH INST. FROM MM

MACHINE
CYCLES

- EXECUTE INST.

- FETCH NEXT INST. FROM
MM

- EXECUTE IT

COSMAC: 2

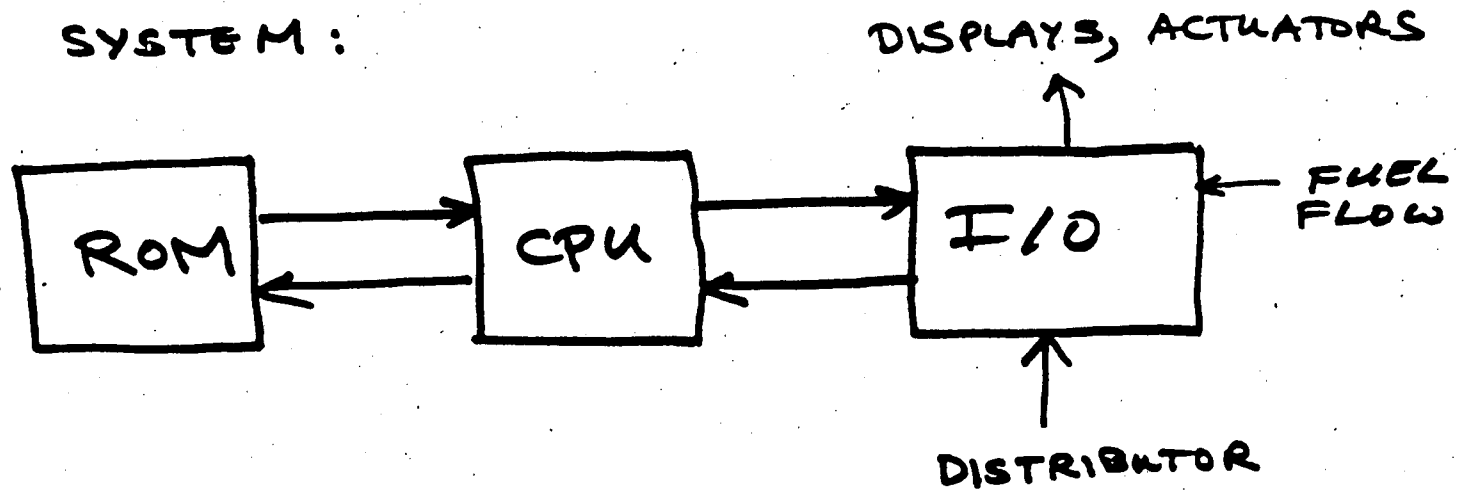
MACHINE CYCLES

PER INST.

A "PROGRAM" DETERMINES
SEQUENCE OF INSTRUCTION TO
BE EXECUTED.

EXAMPLE OF MICROCOMPUTER :

CAR SYSTEM :



GET : OPTIMAL FUEL MIXTURE
: STATUS — MPG (Km per Litre)
— SPEED
— ALARMS
— ETC

MP APPLICATIONS

- REPLACE MINIS (50,000/YEAR)
- REPLACE HARDWIRED LOGIC (CONTROLLERS)
 - TERMINALS
 - INSTRUMENTS
 - PROCESS CONTROL
- NEW PRODUCTS
 - CARS
 - HOME
 - SCHOOL
 - ETC.

ADVANTAGES OF USING ERP

- COST
- FLEXIBILITY
 - CUSTOMIZE IN SOFTWARE
 - NEW FUNCTIONS
- MAINTENANCE
- RELIABILITY

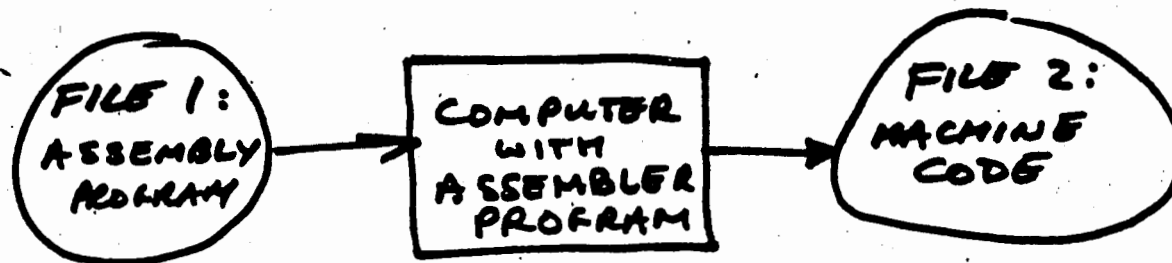
AIDS TO μ P SYSTEM DESIGN

- LEARNING AIDS — MICROTUTOR
 - DEVELOPMENT AIDS — MICROKIT
— INTELLEC
(WORK WITH TERMINAL)
 - CROSS-ASSEMBLERS
 - SIMULATORS
 - DEBUGGERS
- } AID
PROGRAM
DEVELOPMENT

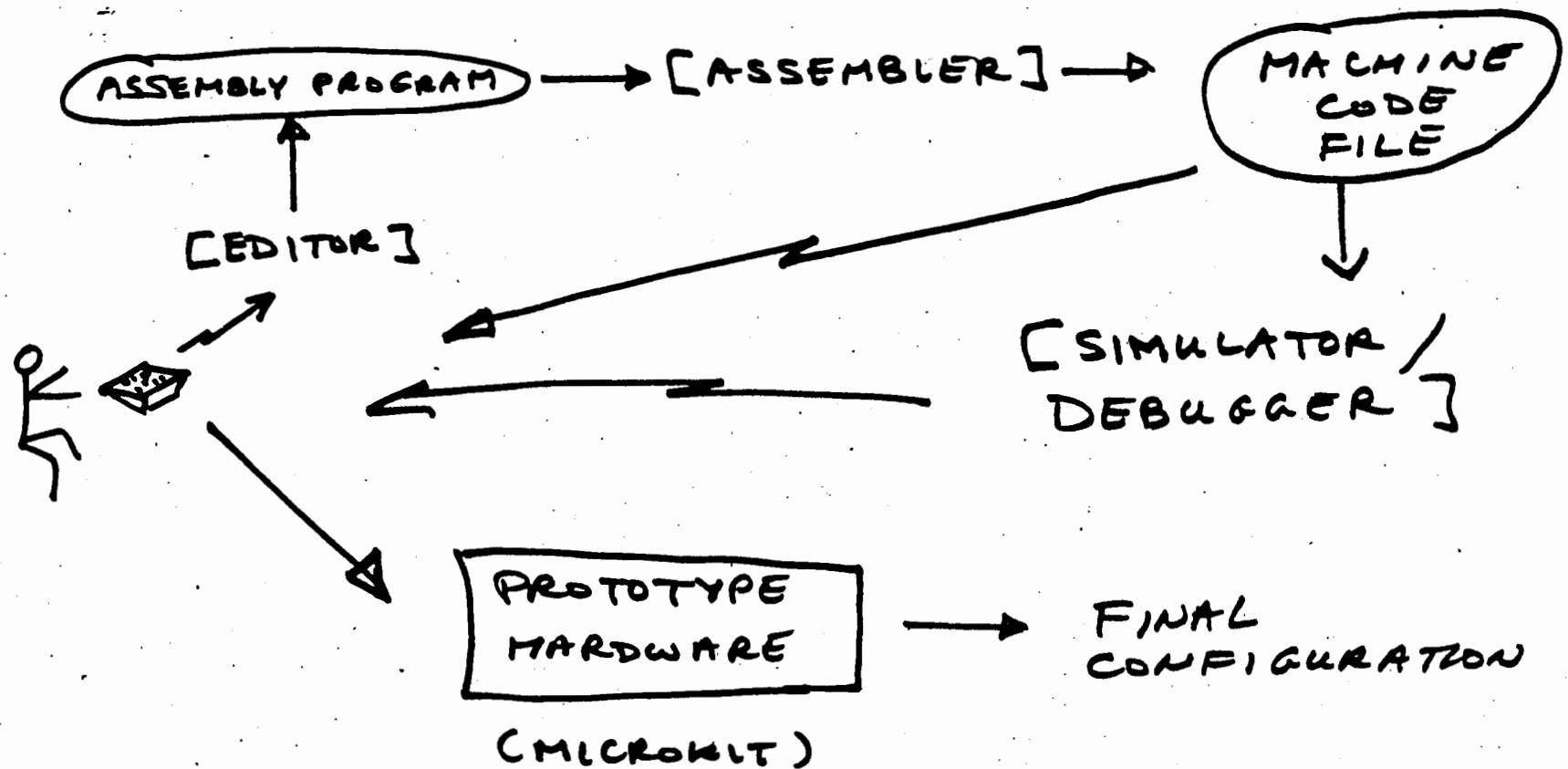
ADVANTAGES OF ASSEMBLY LANGUAGE

- NAMES FOR INSTRUCTIONS & LOCATIONS IN MM
- EASY TO WRITE & CHANGE
- CAN USE MORE POWERFUL COMPUTER TO TRANSLATE ASSEMBLY PROGRAM INTO MACHINE CODE

EG



TOTAL SYSTEM DESIGN



BINARY

EG.

$$\begin{array}{cccccc} 0, & 1 & & & & \\ & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ & 1 & 0 & 1 & 1 & 0 \end{array} = (22)_{10}$$

$$16 \times 1 + 8 \times 0 + 4 \times 1 + 2 \times 1 + 1 \times 0 = (22)_{10}$$

BASE = 2

HEXADECIMAL

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C,
D, E, F

$$A = 10$$

$$B = 11$$

$$C = 12$$

$$D = 13$$

$$E = 14$$

$$F = 15$$

BASE = 16

$$\begin{aligned} \underline{\underline{EG:}} \quad & \begin{array}{c} 16' \quad 16^0 \\ (F3)_{16} = (243)_{10} \end{array} \\ & 15 \times 16 + 3 \times 1 \end{aligned}$$

2 HEX DIGITS = 1 BYTE = 8 BITS

$$\begin{aligned} \underline{\underline{EG:}} \quad AC &= \underbrace{1010}_{A} \underbrace{1100}_C = 172 \\ & \underbrace{\hspace{1.5cm}}_{1 \text{ BYTE}} \end{aligned}$$

BINARY

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

DECIMAL

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

HEX

0 1 2 3 4 5 6 7 8 9 A B C D E F

COMPUTERS USE BINARY INTERNALLY

HEX NOTATION IS HUMAN CONVENIENCE

RAM

RANDOM ACCESS MEMORY

- USUALLY MEANS READ/WRITE MEMORY

ROM

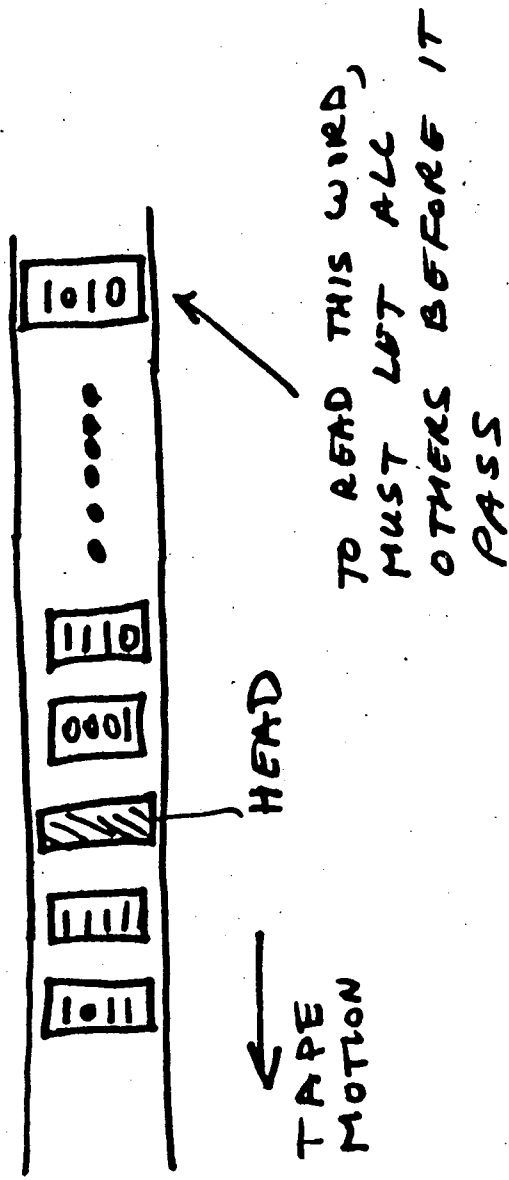
READ ONLY MEMORY

- ALSO RANDOM ACCESS
- CANNOT WRITE TO IT

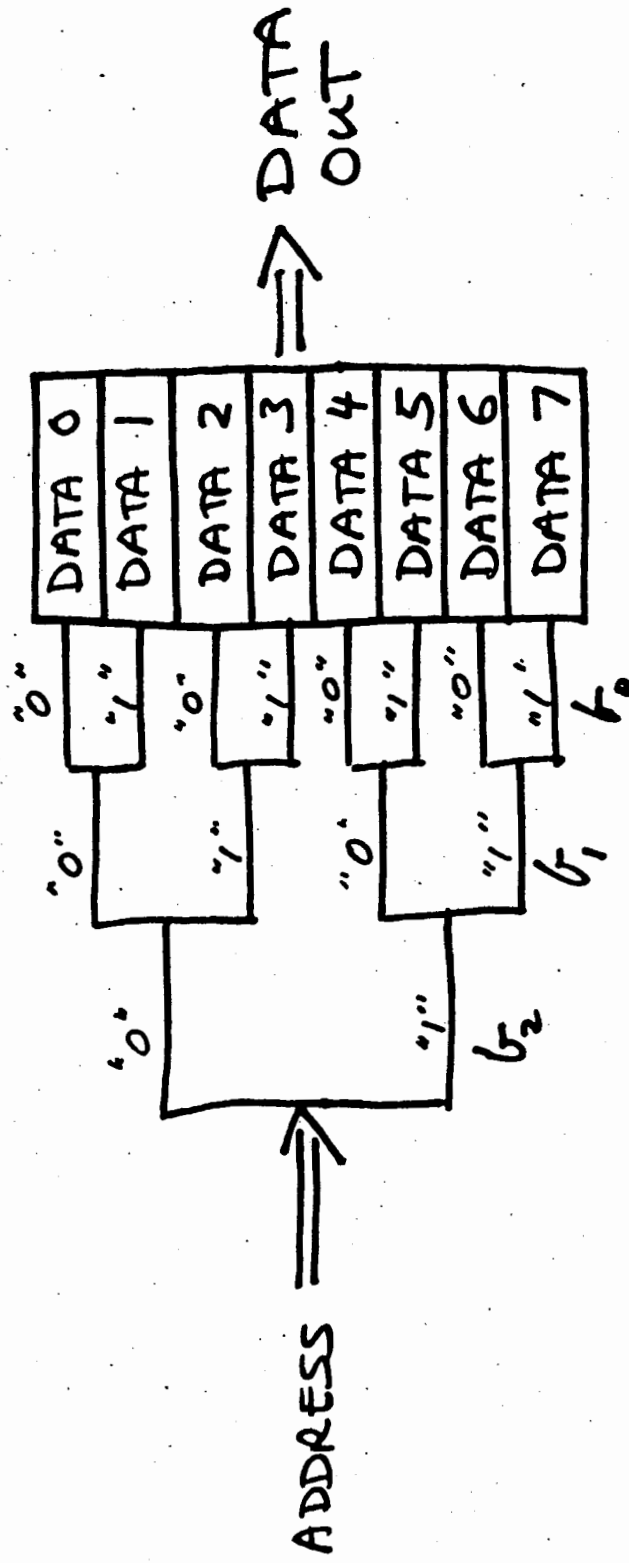
WHAT DOES RANDOM ACCESS MEAN?

- CAN ACCESS ANY WORD WITH SAME SPEED

EG. SERIAL MEMORY (TAPE)



FOR RAM, DECODE ADDRESS IN PARALLEL



101 \Rightarrow DATA 5

3 BIT ADDRESS $b_2 b_1 b_0$

001 \Rightarrow DATA 1

MEMORY SPEC: - ADDRESS WIDTH

\Rightarrow HOW BIG, HOW MANY LOCATIONS

eg, 16 BIT ADDRESS

$\Rightarrow 2^{16}$ OR 65,536

LOCATIONS

(64K)

- WORD WIDTH

4, 8, 12, 16, 32 BITS

4, 8, 12
MICROS

\nearrow

LARGE CPAs

MINIS

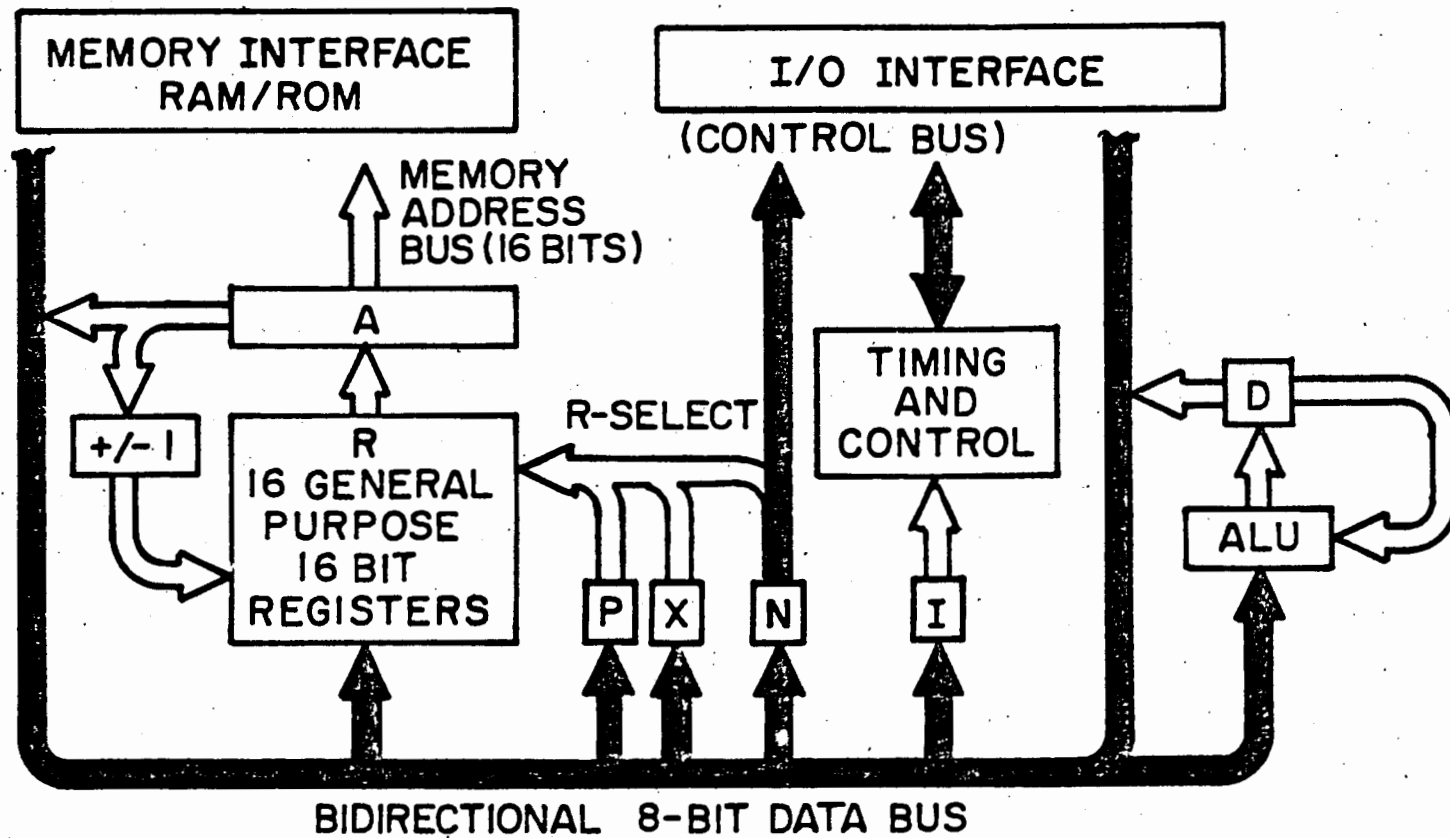
(IBM)

8 BIT WORD IS MOST COMMON

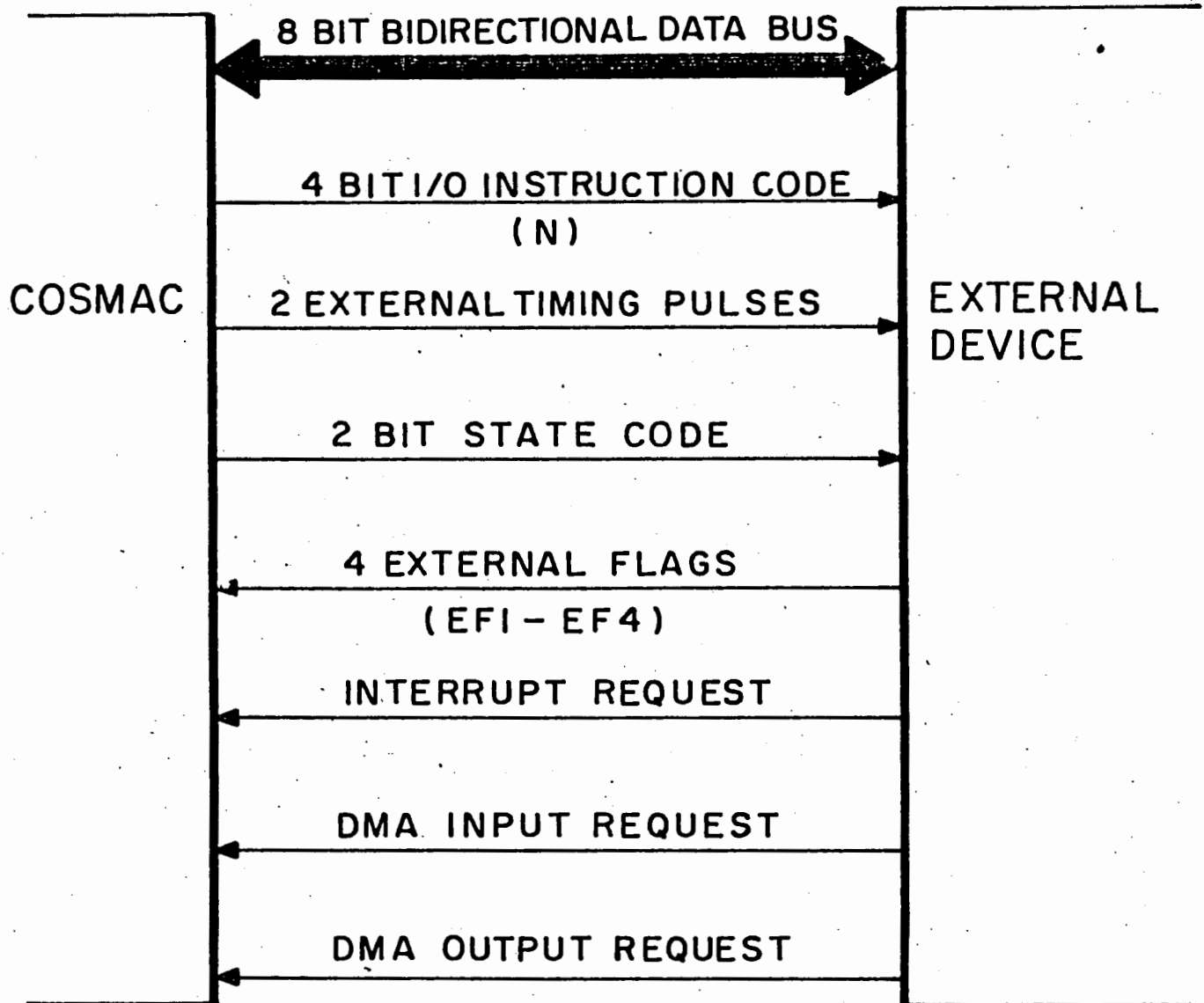
FOR MICROS COSMAC, INTEL 8080, MOTOROLA 6800)

COSMAC

- 8 BIT DATA BUS (8 BIT "WORD SIZE")
- 16 BIT ADDRESSING (64K)
- EVERY INSTRUCTION IS 1 BYTE LONG
(COMPACT CODE)
- EACH INSTRUCTION NEEDS TWO MACHINE CYCLES, FETCH and EXECUTE
- 2 CHIPS, BUT WILL BE ONE CHIP SOON



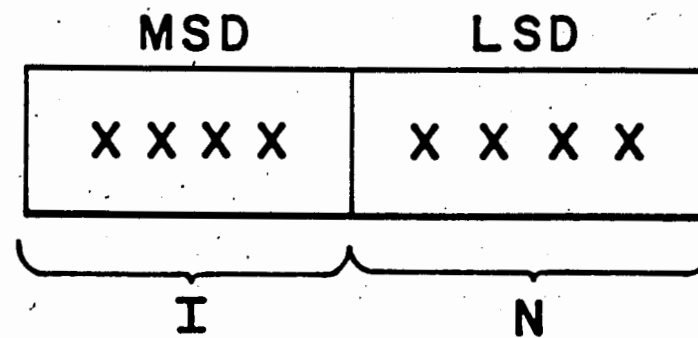
COSMAC MICROPROCESSOR ARCHITECTURE



COSMAC I/O INTERFACE

COSMAC ARCHITECTURAL ADVANTAGES

- ★ EXTERNAL CPU TESTABLE FLAGS
- ★ BUILT-IN DMA CHANNEL
- ★ SINGLE-PHASE CLOCK
- ★ CMOS TECHNOLOGY

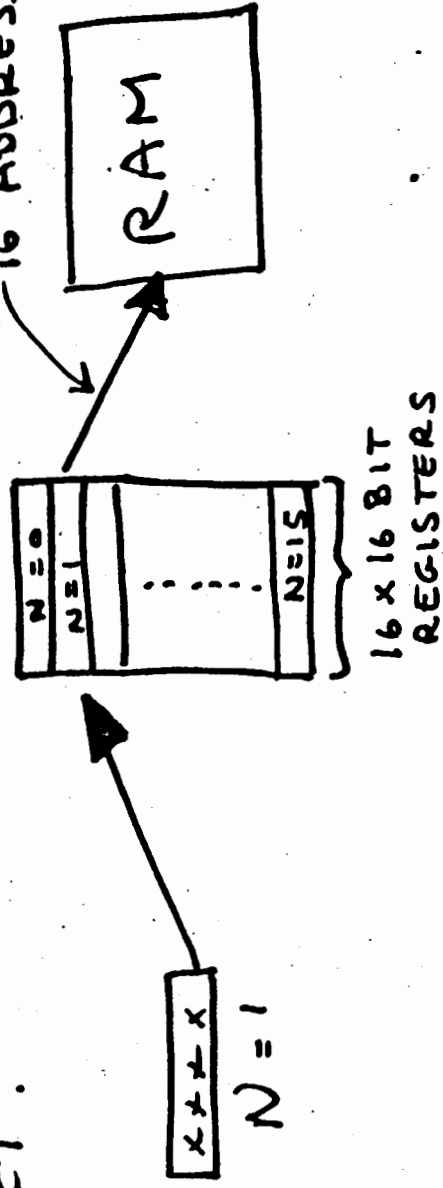


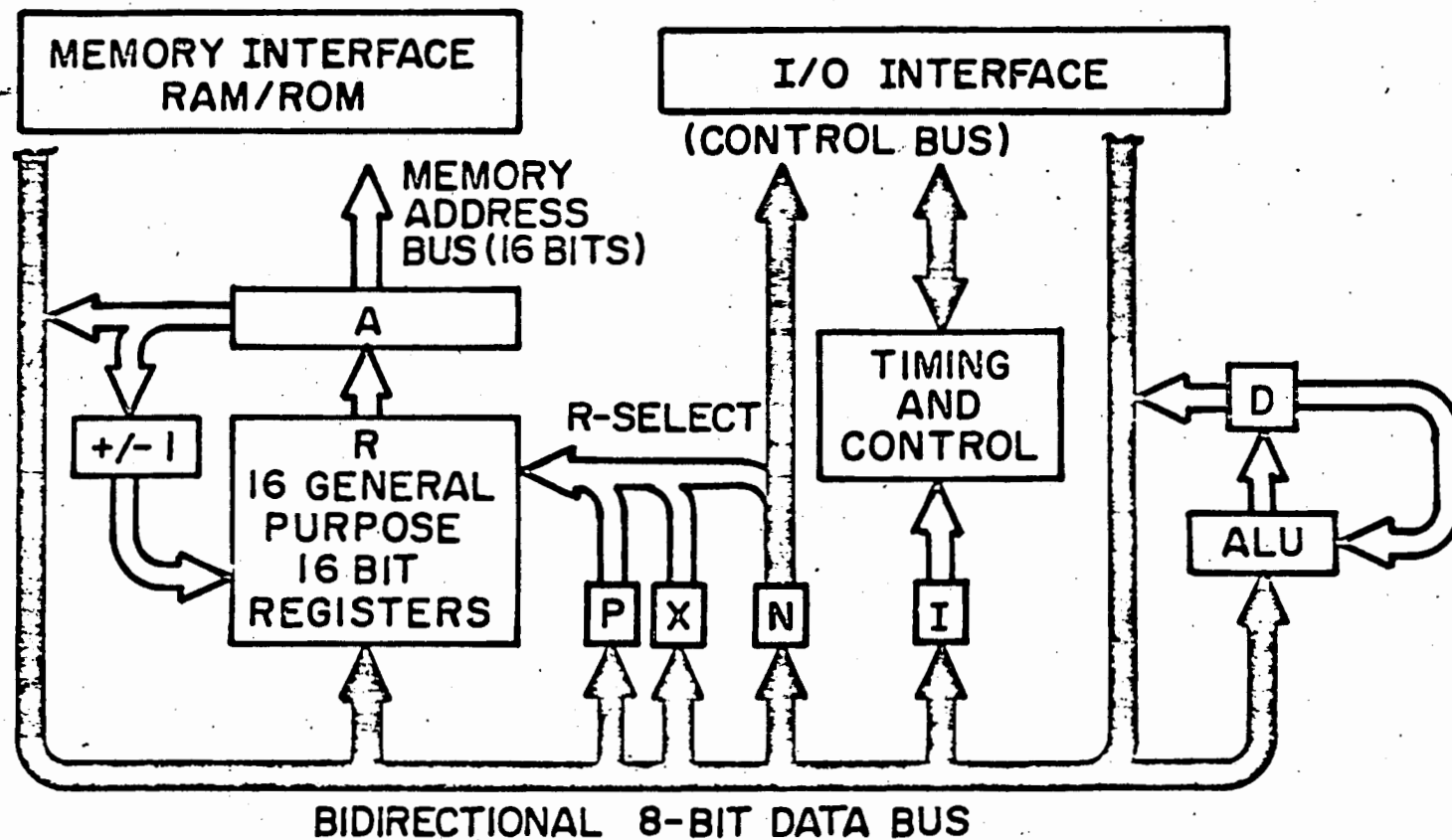
ONE BYTE INSTRUCTION FORMAT

I, 4 BITS, SPECIFIES INSTRUCTION
CLASS

N, 4 BITS, MODIFIES INSTRUCTION

ALL ADDRESSING IN COSMAC IS
INDIRECT.





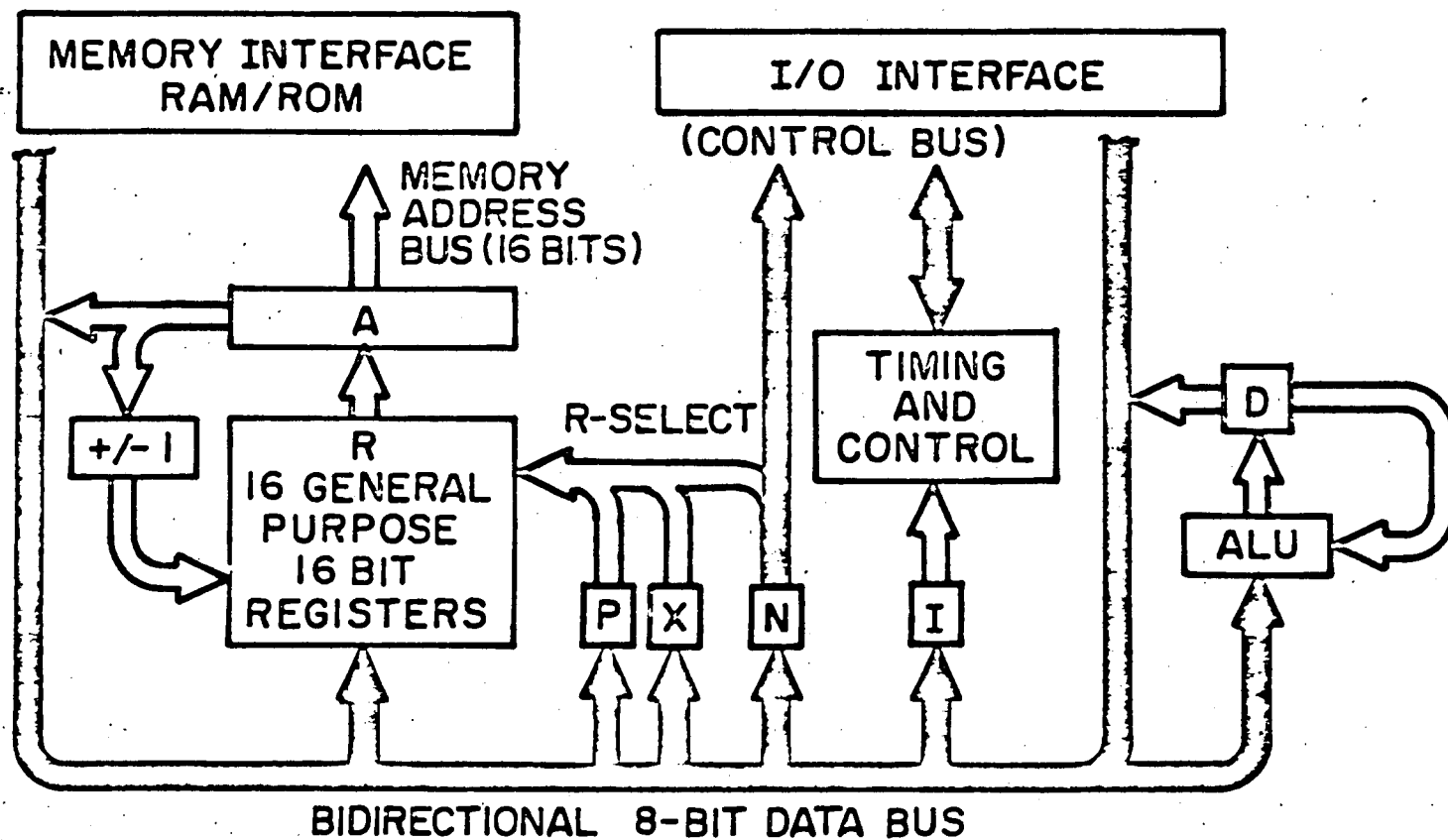
COSMAC MICROPROCESSOR ARCHITECTURE

P, 4 BITS, SELECTS PROGRAM COUNTER
(PC)

X, 4 BITS, SELECTS DATA POINTER

PC ALWAYS POINTS AT NEXT INSTRUCTION
TO BE FETCHED FROM MEMORY

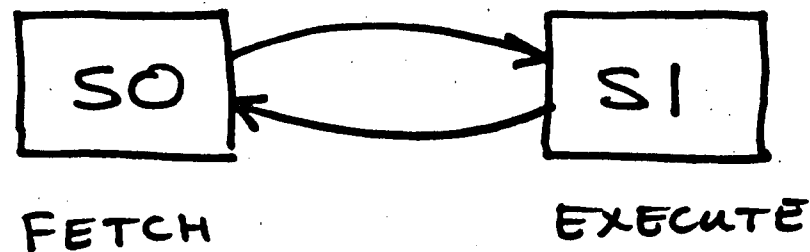
ANY OF 16 R'S CAN BE PC,
DATA POINTER, ETC.



COSMAC MICROPROCESSOR ARCHITECTURE

4 BASIC MACHINE STATES

S0	FETCH	}	NORMAL PROCESSING
S1	EXECUTE		
S2	DMA		
S3	INTERRUPT		



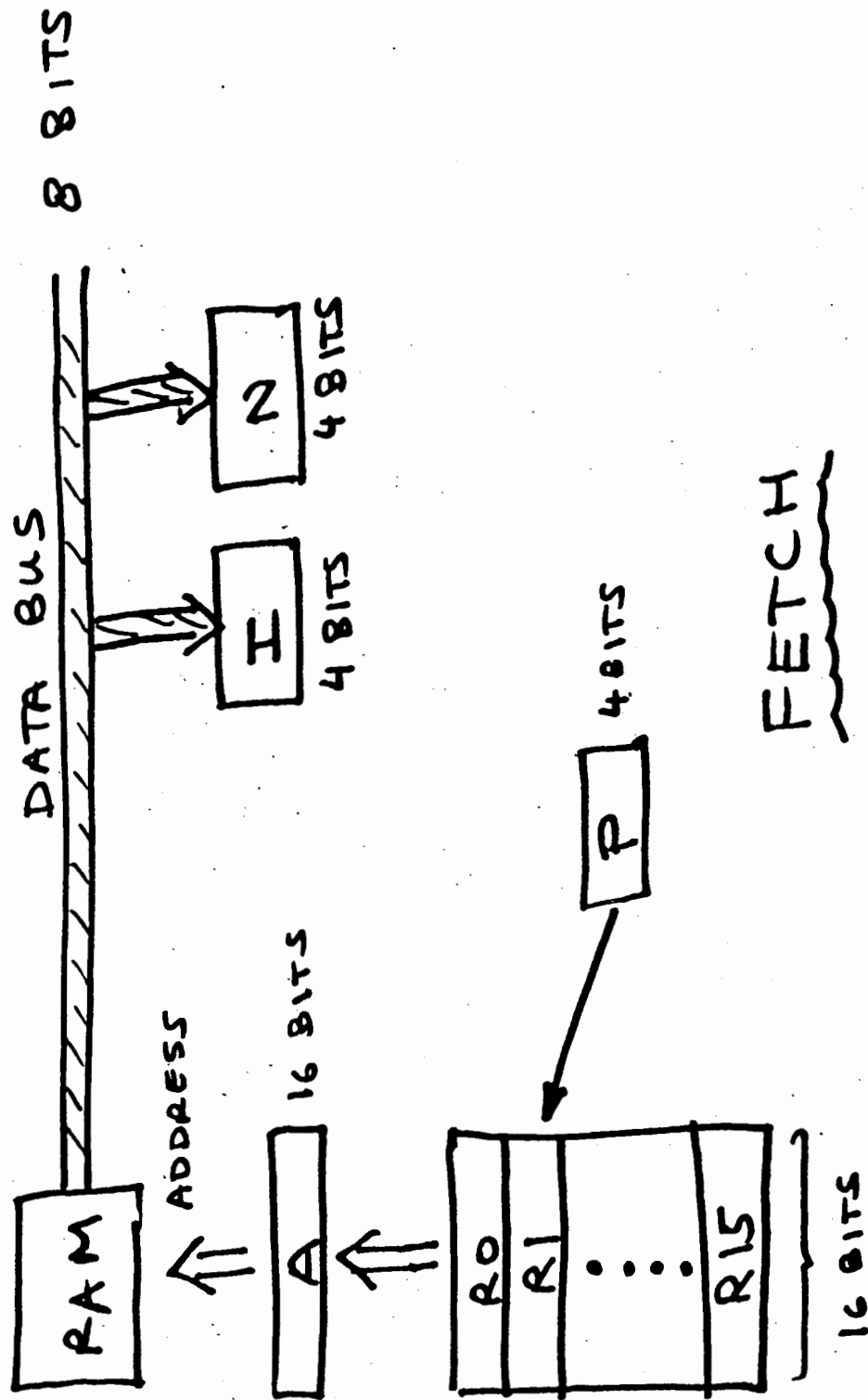
NORMAL
PROCESSING
STATE
DIAGRAM

INTERNAL FETCH CYCLE - SO

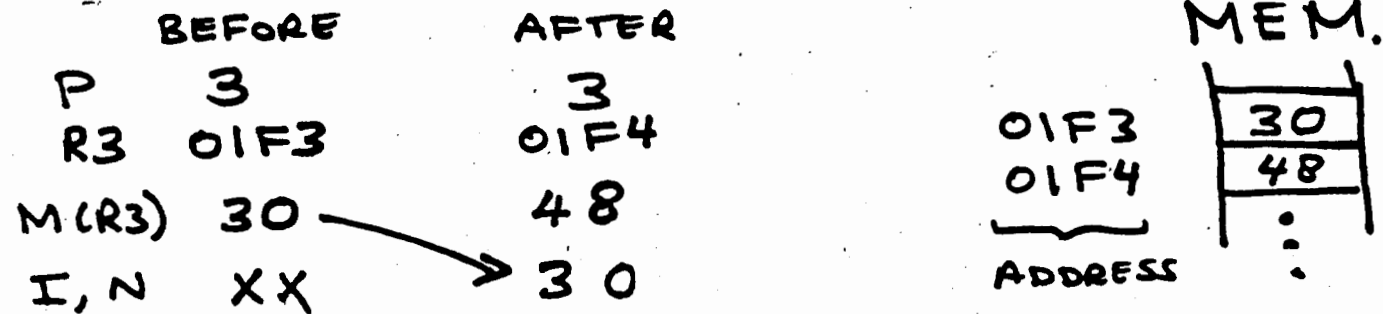
P SELECTS R(P) (PROGRAM COUNTER)
R(P) CONTENTS \rightarrow A (ADDRESS REGISTER)
& R(P) INCREMENTED

M(A) \rightarrow DATA BUS

DATA BUS \rightarrow I, N



FETCH EXAMPLE



NOTE: COSMAC CHIP ADDRESS BUS IS
8 BITS. 16 BITS SENT OUT
SEQUENTIALLY.

MICROTUTOR ONLY HAS 256 BYTES OF
MEM. — SO IGNORE UPPER 8 BITS.

EXECUTE CYCLE - S1

I, N DECODED TO PERFORM
DESIRED INSTRUCTION EXECUTION

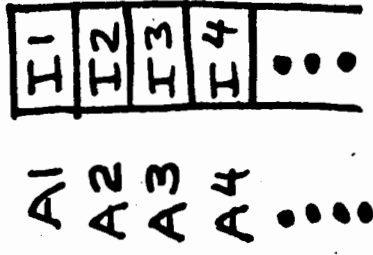
WHAT IS A PROGRAM

EXECUTION

SEQUENCE

I1
I2
I3
I1
I4
...

MEMORY



MICROTUTOR

- ⇒ • 256 BYTES RAM (IGNORE UPPER 8 BITS OF ADR)
 - ⇒ • 2 HEX DIGITS DISPLAY (OUTPUT)
 - 8 SWITCHES (1 BYTE) (INPUT)
 - LOAD SWITCH (RUN / LOAD)
 - CLEAR BUTTON (SET $R0 = 0000$, $P = 0$)
($I = 0$, $N \neq 0$)
 - ⇒ • IN BUTTON (MEM. LOAD OR PROGRAM RUN I/O)
 - START BUTTON (START EXEC. OR STEP THROUGH MEM.)
- * FOR COSMAC
ALWAYS START
EXECUTION AT ADDRESS 0001

COUNTER PROGRAM FOR TUTOR,

RECALL : 256 BYTES OF MM

⇒ IGNORE UPPER 8 BITS OF
MEMORY ADDRESS.

WANT: DISPLAY COUNT, STARTING WITH 00
ADD 1 TO COUNT WHENEVER "IN"
IS PRESSED

PROGRAM STEPS (FLOW DIAGRAM)

INITIALIZE



DISPLAY COUNT ----- OUTPUT



WAIT FOR "IN" ----- TEST FLAG
AND BRANCH WHEN
SET



ADD 1 TO COUNT --- ALU FUNCTION



UNCONDITIONAL BRANCH

NEED: OUTPUT INST. (DISPLAY)

TEST FOR "IN" INST.

ADD INST.

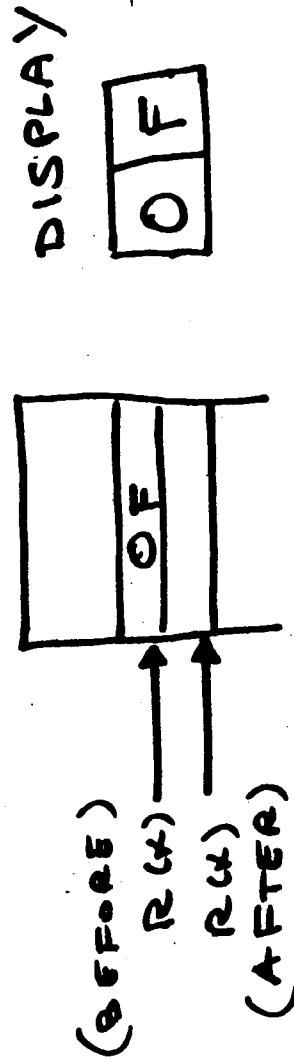
BRANCH INST.

LOOK AT THESE IN DETAIL !

OUTPUT INSTR.

TUTOR

60 M(R(x)) \Rightarrow DISPLAY , R(x) + 1 \rightarrow R(x)



IN GENERAL, "6" OP CODE \Rightarrow I/O

6N IS I/O INSTR.

60-67 OUTPUT $M(R(x)) \rightarrow BUS$
 $R(x)+1 \rightarrow R(x)$

68-6F INPUT $BUS \rightarrow M(R(x))$

SEX INSTR. (SET X REG.)

E.G. E3 MAKES R3 BECOME X REG.

IF $X = P$, THEN HAVE "IN LINE" DATA.

E.G. EO✓ SET $P = X$

TUTOR 3, 60✓ $MC(RX)) \rightarrow$ DISPLAY, $R(X)+1 \rightarrow R(X)$

$P = 0$ F7

XX✓

XX

DISPLAY

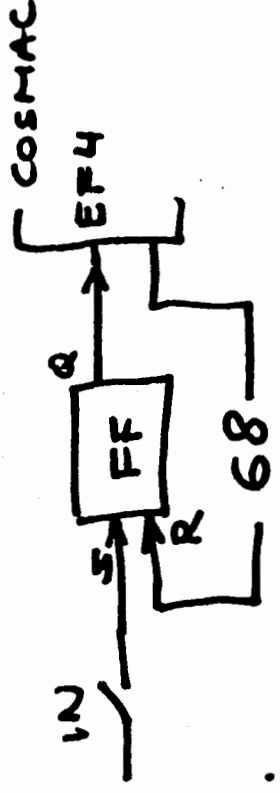
F	7
---	---

FETCH GO INC R0

EXEC GO INC R0 ($P = X$), SO SKIP "F7"
DATA

TEST FOR "IN"

WHEN "IN" PRESSED, LOGIC SETS A
FF TO "1". FF TIED TO
EF4 FLAG OF CPU.



CPU HAS "BRANCH" INSTR.

32 }
"BRANCH" — CONDITION

E.G. 30 } UNCONDITIONAL
XX } BRANCH TO XX

3F } BRANCH TO XX
XX } IF "EF4 = 0", OTHERWISE
YY EXECUTE NEXT INSTR. (YY).

NEED ADD INSTR.

M.OP.D \Rightarrow D, DF \leftarrow 1 BIT DATA FLAG

.OP. = +, -, LS, OR, AND, XOR, ETC.

FN
ALU INSTR.
TYPE.

E.G. FC XX ADD XX TO D
XX AND STORE RESULT
IN D REGISTER.
IF CARRY, SET "DF"

LOAD D, RETRIEVE FROM D

4N LDA M(R(N)) \rightarrow D, R(N)+1 \rightarrow R(N)
LOAD SPECIFIES MM ADDRESS

5N STR D \rightarrow M(R(N))
STORE SPECIFIES MM ADDRESS

F8 LDI M(RCP)) \rightarrow D, RCP)+1 \rightarrow RCP)
LOAD IMMEDIATE

A_N }
CODE
PLO
(PUT LOW)
IDENTIFIES
REGISTER

$D \rightarrow \overbrace{R(N) \cdot 0}$
LOWER 8 BITS OF $R(N)$

8_N }
CODE
G-LO
(GET LOW)
IDENTIFIES
REGISTER

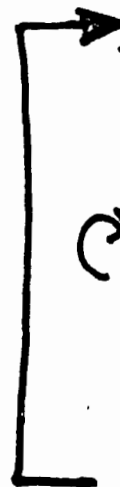
$R(N) \cdot 0 \rightarrow D$

MM

CODE

00
01
02
03
04
05
06
07
08
09
0A
0B
0C
0D
.
.

00
F8
00
A3
53
E3
60
23
3F
08
FC
01
30
04



} STORAGE BYTE FOR OUTPUT
} PUT 00 IN D REGISTER

D → R(3).0 (PUT 00 IN R(3).0)
D → M(R(3)) R3 POINTS
X REG. IS R(3) AT 00

M(R(X)) → DISPLAY, INC. R(3)
DEC. R(3) (POINTS AT 00 AGAIN)
BRANCH TO 08 IF EF4 = 0
(WAIT FOR "IN")

} ADD "01" TO D, STORE RESULT
} IN D

} BRANCH TO 04

RECALL,

R(0) = 0000

P = R(0)

} WHEN EXECUTION
} BEGINS

I. GENERAL

A. Turning it ON

Figure 1 shows what MICROTUTOR looks like in case you don't have one. If you do have one, plug the memory card into the first socket (M). Plug the COSMAC microprocessor into the middle socket (P). The component (bumpy) side of these cards should face the rear. Don't apply power until the M and P cards are in unless you enjoy replacing integrated circuits.

Plug the power pack cord into the back of MICROTUTOR to turn it on. Pull the cord plug out to turn it off. If the red display lights don't come on when you plug in the power you are the proud owner of what is technically known as a lemon.

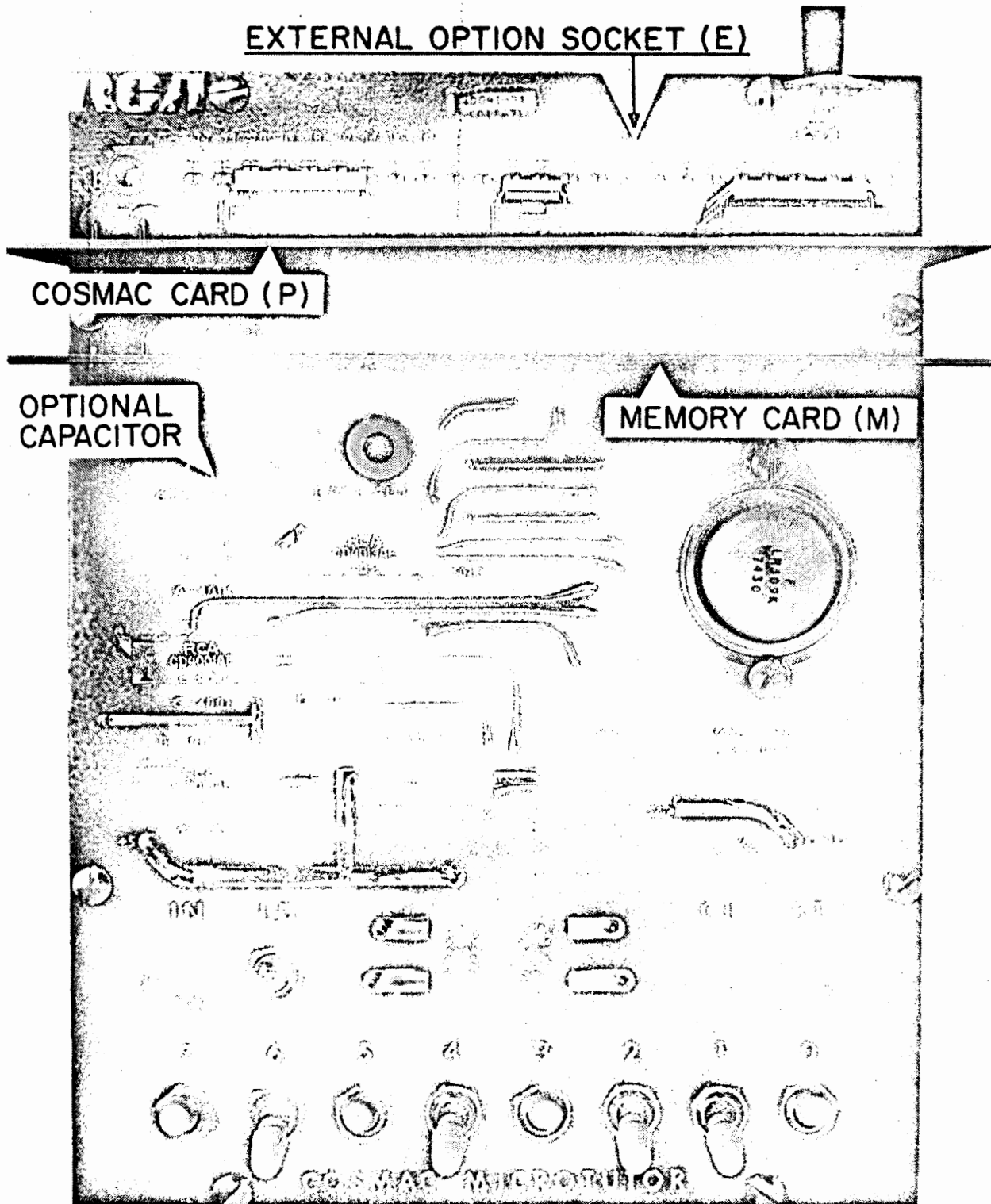
B. Bits, Bytes, and Hex Digits

Before a stored program computer can run, it must have a program stored in its memory. Before storing a program in the MICROTUTOR memory, some basic definitions should be stored in your memory. Familiarity with binary notation (bits) is assumed. If this is a rash assumption, please correct the obvious gap in your otherwise outstanding educational background before proceeding.

A byte is a group of 8 bits. The COSMAC microprocessor (along with many others) uses 8-bit bytes (or words). These 8 bits are labeled 0-7 corresponding to the eight MICROTUTOR byte input switches as shown below:

SWITCH	X	X	X	X	X	X	X	X
BIT NO.	7	6	5	4	3	2	1	0

A byte can be divided into two 4-bit digits (D1 and D0). The high order digit (D1) comprises bits 7-6-5-4, while D0 comprises bits 3-2-1-0. Each 4-bit digit can be represented by a single HEX symbol as follows:



MICROTUTOR LAYOUT
FIG. 1

BINARY	HEX	DECIMAL EQUIVALENT
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

The byte "01011011" can now be described as "5B" in HEX notation. Press the MICROTUTOR CL (Clear) switch and flip the LD (Load) switch up. Set "01011011" into the input switches (setting input switches up for "1" and down for "0"). Press IN and the byte will be displayed in hex form as "5B". Change the input switches and press IN (with LD up) to convert other binary numbers to HEX.

HEX is a base 16 numbering system which was developed by an obscure group of bald headed, 16 toed programmers. It is only through the development of notational jargon such as bit, byte, and HEX that we can maintain our superiority over the average citizen. HEX notation will be used throughout this manual unless noted otherwise.

C. Down Memory Lane

Whoever said that memory is fundamental to a stored program computer has been long forgotten. Just in case he (or she) was right, MICROTUTOR is provided with a memory. The MICROTUTOR memory can store up to 256 bytes in locations numbered consecutively from 00 to FF. The number of a stored byte location is called its memory address. The notation M(4A) is used to specify the memory byte located at address 4A. For example, four bytes might be stored in the first four memory locations as follows:

ADDRESS (M)	BYTE
00	E2
01	27
02	51
03	F6

M(02) would then represent the memory byte, addressed by 02, which is 51. The ability not to confuse memory addresses with the bytes stored at those addresses separates programmers from normal people.

When a byte is stored in memory at a specified location or address it replaces the byte previously stored at that location. When a byte is fetched or read from memory, a copy of it remains stored. (This is analogous to recording or playing back magnetic tape.)

D. Getting It In (Program Loading)

Computer instructions are individual binary codes or bytes stored in memory. Each byte specifies an individual computer operation such as store an input byte, add two bytes, display an output byte, etc. A sequence of such instructions is called a program. The computer obtains each instruction, as required, from memory and performs the specified operation.

The following is a short COSMAC MICROTUTOR program code that can be stored in memory:

ADDRESS (M)	INSTRUCTION BYTE (CODE)
00	00
01	F8
02	00
03	A3
04	53
05	E3
06	60
07	23
08	3F
09	08
0A	FC
0B	01
0C	30
0D	04

Flip LD (Load) up. This tells MICROTUTOR that you want to load a program. Push CL (Clear) and loading of the following sequence will start at memory address 00.

You are now ready to load your first program. Do not become nervous or excited as this will lead to mistakes. Set the input switches to 00 (Binary 00000000) and push IN. 00 is displayed and stored at address 00. MICROTUTOR also advances its load memory pointer to 01 so that your next input byte will be stored at memory address 01. This ability to anticipate your next move led to the early belief that computers were giant brains. (This belief was later shattered by the discovery of the first program bug.)

Proceeding, set the input switches to the next instruction byte, F8 (Binary 11111000), and push IN to store it at memory address 01. Continue loading the rest of the program bytes into memory until the last instruction (04) has been stored.

E. So You Made A Mistake

You can check memory with LD up. Push CL to return to memory address 00. Now push ST and the byte stored at address 00 will be displayed. Push ST again and the next byte (F8) will be shown. Continue pushing ST to check that all bytes in the program are properly stored.

Checking the program in memory is generally skipped by those of us who don't make errors. Some programmers actually prefer the challenge and added fun of trying to run an improperly loaded program. If you are only interested in getting a program to run, include the checking step.

If one of your memory bytes is wrong you can loudly blame it on the computer or quietly change it to the right value. To change a byte, leave LD up and push CL. Repeatedly push ST until the byte just before the one you want to change is displayed. Set the input switches to the byte code you want to substitute for the wrong one and push IN. The new byte will replace the wrong one in memory and be displayed. Pressing ST will resume memory stepping for correction of a subsequent error in the byte sequence.

F. Pushing the Start Button

With the above program properly stored in memory you are ready to run. LD should be down. Always push CL to start the program at the beginning. (Starting programs at the end only works for backward programmers.) After pushing CL push ST (Start). The program is now running.

Unfortunately nothing spectacular happens when this program is running so you'll just have to take our word for it. If you are a doubting Thomas type, you can verify that the program is running by pushing IN. Each time you press IN the program adds 1 to the display. This hex counter program only required 14 bytes. You really couldn't expect anything too exciting, could you?

To stop the program, press CL. Now nothing happens when you press IN, does it? To restart the program, press ST. This program remains stored in memory until you disconnect power or load a new program.

Want to become a NIM game hustler? Find a friend to play with. (If you don't have a friend, you are well on your way toward becoming a professional programmer.) Start the computer (00 displayed). You and the other player take turns. On each turn add 1, 2, or 3 to the displayed hex number (press IN 1, 2, or 3 times). The first player to reach 10 (decimal 16) wins. If you graciously let the other player have the first turn you can always win (unless he cheats). We'll leave the how as an exercise for the reader.

Later on a program which plays this game against you (and always wins) will be described. In the meantime, this program can be used to illustrate a major advantage of computers. This advantage lies in the ease with which operation can be changed. For example, changing the 01 byte at M(0B) to 02 will increment the display by 02 each time IN is pushed. Substituting an FF instruction for the FC at M(0A) will decrement the display each time IN is pushed.

The next section provides some more programs to load and play with before getting down to the nitty-gritty details of hardware and programming.

SPECIAL FEATURE:

A Simplified Microcomputer Architecture

Joe Weisbecker
RCA Laboratories

Copyright 1974 by The Institute of Electrical and Electronics Engineers, Inc. Reprinted, with permission, from *COMPUTER*, March 1974.

Introduction

The motivation behind this work has been the view that for 20 years computer hardware has become increasingly complex, languages more devious, and operating systems less efficient. Now, microcomputers afford some of us 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. We are already promised 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. Design philosophy, programming considerations, and typical systems are also discussed. Experience with breadboard versions of this architecture has verified its usefulness over a surprisingly wide range of potential applications.

Overview

Large scale integration 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 here 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 6 LSI chips are feasible with this architecture. Discrete TTL and COS/MOS breadboards of such potential 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, however, when increased device per chip ratios are used to provide more complex microprocessors.

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 because of fundamental differences in minicomputer and microcomputer applications. The most obvious mini-micro difference is cost. Because of their higher cost, minicomputers must generally be structured to compete on a free standing cost/performance basis with a high degree of upward expansion capability and the need to support relatively efficient operating systems, etc. Simple, inexpensive microcomputers will generally be imbedded in special-purpose products. They do not require the same degree of upward expansion or an excessive preoccupation with cost/performance ratios. (Who cares if a \$20 microcomputer chip is efficiently utilized?) Used to replace custom logic, the microcomputer can be designed to put more burden on the programmer. Software (in many cases) will be developed on larger computers. Because of such differences it was 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, and that 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, heavy reliance is placed on techniques using memory to reduce hardwired logic complexity. 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. RAM and 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. A single LSI chip containing both ROM and RAM segments will suffice for many applications — and that results in system simplicity.

While low memory costs can be expected, very low cost systems will only result 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 of memory or less are possible with this format.

Random control logic uses chip area less efficiently than register/memory arrays. For this reason a very 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 employing 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 free-standing computer, the microcomputer architecture requires efficient, flexible, input/output capability. This is provided via programmed byte transfers and a built-in direct memory access channel. Programmed I/O byte transfer instructions provide maximum flexibility for I/O selection, control, and data transfer. The direct memory access 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 direct memory access channel adds negligible complexity to the microcomputer architecture, it greatly simplifies system design, and that leads 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 microseconds 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

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

Figure 1 here.

P, X, and N are three 4-bit registers. The contents of P, X, or N select one of the 16 R registers. 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 1-byte memory location addressed by the contents of R(N). This indirect addressing system is basic to the simplicity and flexibility of the architecture.

D is an 8-bit register that functions as an accumulator. The 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 one bit position. Add, subtract, and shift operations set a 1-bit overflow register (not shown) which can be tested by branch instructions.

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 I/O devices. A data bus byte can, in turn, be transferred to a register, memory, or I/O device.

The operation of the microcomputer is best described in terms of its instruction set. A 1-byte instruction format is used as shown in Figure 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 $M[R(P)] \rightarrow I, N$. The 4-bit digit in P selects R. R(P) is then transferred to A and used to address memory. While waiting for the 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 I and the least significant digit is placed in 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. It should be noted 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 I and N to be executed. This fixed 2-cycle, fetch-execute sequence simplifies control logic and permits program interruption or direct memory access cycle stealing to occur only between instructions. This results in even further control simplification. Since the operation code in I is limited to 4 bits, only 16 instruction types need be decoded. The 16 possible operations specified in the hex digit in I are listed below:

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)

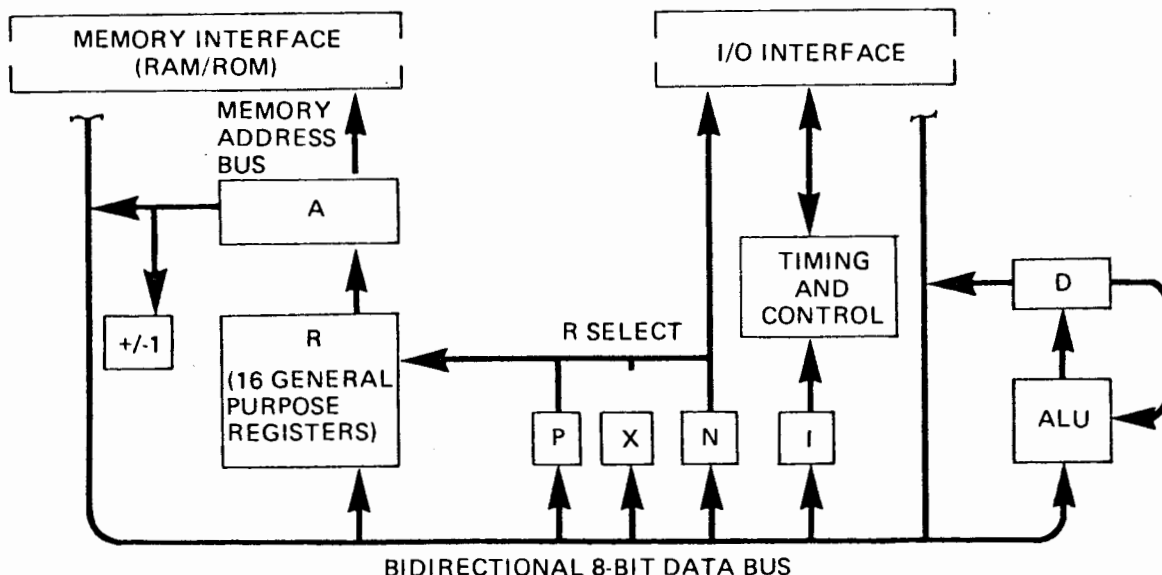


Figure 1. Microcomputer Architecture

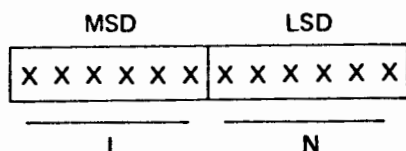


Figure 2. Eight-Bit Instruction Format

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

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 $N=P$, 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 3-byte sequence serves to set a 1-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 direct memory access 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 I/O 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 I/O 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 I/O device or an I/O 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 I/O 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 direct memory access channel. An I/O device can activate either an input or an output byte request line. At the end of execution of the current instruction, a direct memory access channel cycle will occur, causing the requested memory-I/O device byte transfer to occur. Contiguous direct memory access cycles are permitted for fast-burst transfers. $R(0)$ is used for addressing memory during direct memory access cycles and is automatically incremented by 1 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 are 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 0, 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, which in turn results 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 highly efficient relative to memory usage. 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 bread-board version of the microcomputer with a variety of I/O 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.

While the instruction set initially appears 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. If this system is provided 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 business system.

For microcode programming, an assembly language has been developed. This approach considerably simplifies machine language coding. An interactive simulator greatly facilitates initial program debugging. Both of these microcomputer software support systems are readily modified to run on existing timesharing 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. Figure 3 indicates a possible microcomputer-based calculator.

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

Figure 4 illustrates a stand-alone system which might require only 6 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. With more memory, hardcopy output, and low cost floppy disk or magnetic bubble bulk storage, this system would provide the basis for a wide range of inexpensive, turnkey, small business systems.

Figure 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 direct memory access feature is extremely useful for high-speed block transfers in this type of system. Down-line loading of the microcomputer memory can immediately change its mode of operation. Off-line editing and maintenance are 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.

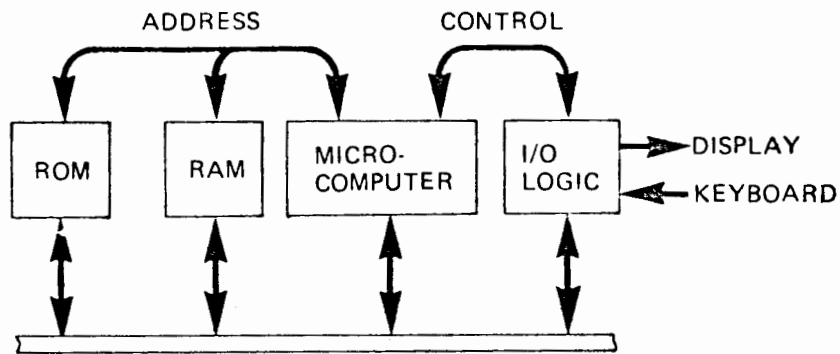


Figure 3. Calculator System

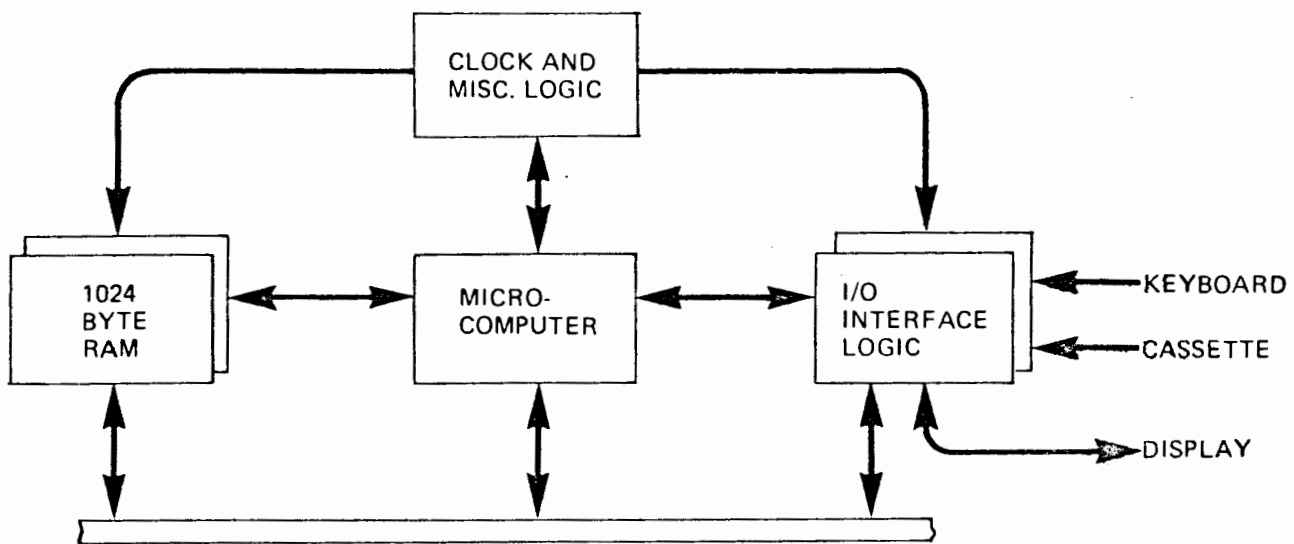


Figure 4. Six-Chip Stand-Alone System

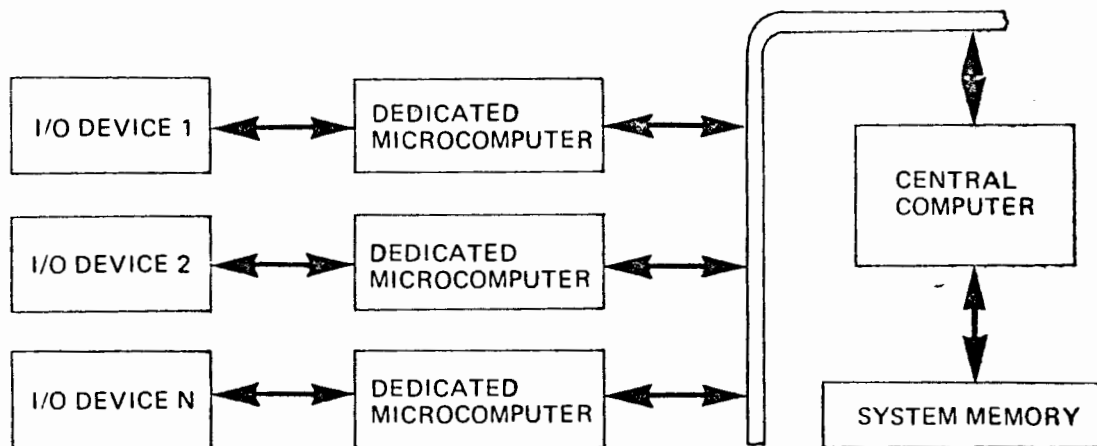


Figure 5. Dedicated Multi-Microcomputer System

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.

Today's high I/O device costs might be used as an argument against minimizing microcomputer cost. This is extremely shortsighted. 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, a COS/MOS-LSI version of this microcomputer is currently being developed by RCA. 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.

Acknowledgements. 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.

Reprints of this article 0703041 are available at \$1.50 for the first copy and .50 for each additional copy. Send request and remittance, stating article number, to IEEE Computer Society, 5855 Naples Plaza, Suite 301, Long Beach, CA 90803.

References

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



Joe Weisbecker is with the RCA Laboratories in Princeton, New Jersey. Prior to joining the Lab in 1970, he was active in various engineering and 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 communications terminals. Mr. Weisbecker is a graduate of Drexel University and holds 19 patents with others pending. He is a member of IEEE and Eta Kappa Nu, and has received an RCA Laboratories Achievement Award.

His children have had a computer at home for several years which has reinforced his interest in low cost, widely available microcomputer systems.

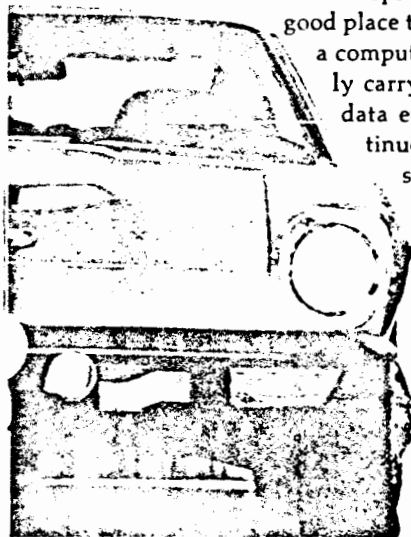
Primer on Microprocessors

PART I



Reprinted from ELECTRONIC PRODUCTS MAGAZINE, January 20, 1975 and February 17, 1975, 645 Stewart Avenue, Garden City, New York 11530, 1975 United Technical Publications, Inc., a division of Cox Broadcasting Corporation.

Not since the development of the transistor in 1948 has any product or technology offered such an exciting promise of things to come as has the microprocessor. Applications span the entire realm of electronics and extend into new areas where existing technologies had never before penetrated. Although much has been written about these MSI/LSI integrated circuits, Electronic Products Magazine felt it was time for an article that concentrated heavily on microprocessor basics. Both users and manufacturers agreed emphatically. The material that forms the basis for our two part feature was originally prepared by National Semiconductor to train its field engineers. We think you'll find the presentation interesting and informative.



Since the microprocessor is a computer in IC form, a good place to start is with computers. Simply put, a computer is a device capable of automatically carrying out a sequence of operations on data expressed in discrete (digital) or continuous (analog) form. Its purpose is to solve a problem or class of problems; it may be one of control, analysis, or a combination of the two. In digital computers, numbers are represented by the presence of voltage levels or pulses on given lines. A single line defines one bit (short for binary digit or a base-2 number). A group of lines considered together is called a "word"; a

word may represent a computational quantity (operand) or it may be a directive specifying how the machine is to operate on computational quantities.

To accomplish automated computation or control, the computer must perform various internal functions. The most obvious is to do arithmetic type of operations (add, subtract, etc.) on two operands. The section performing this function is the Arithmetic Unit (AU).

Something must control the arithmetic unit to make it follow the specific sequence of operations necessary to solve a given problem. In other words, a sequencing mechanism is required; furthermore, if the computer is to be programmed, the sequencer must also be programmable. Some storage is necessary in which to hold the required sequence of operations before beginning a computation. The sequencer can be separated into two

functional units: program storage and control.

The control unit can be viewed as sensing external conditions and issuing commands to other machine elements. For example, the control unit sends commands to the arithmetic unit to initiate arithmetic operations, or sends commands to the program storage, which causes it to look-up the next program directive. The control unit senses such conditions as the completion of an arithmetic operation, the sign of a result, and the presence of stop/start signals from the computer operators. The machine just defined is represented in block diagram form in Fig. 1.

From the diagram, it is evident that no provision has been made to input data (operands) into the machine or to output the results of operations on these operands. Furthermore, no temporary storage has been provided to hold the intermediate or partial results of computations as well as the operands themselves. These items can now be added to complete the computer (Fig. 2). The input unit is indicated, but its connection is left unspecified until the interface is determined.

Basic elements of a computer

When the completed machine is inspected, some potential redundancies are noted. There are three separate storage elements: operand, program and temporary. Could all three be combined into one storage unit or memory, and simply partitioned into three segments? Almost; but it will be more efficient if the temporary storage is maintained as a separate element and operand and program storage combined into one main memory unit. Such a simplification yields a more traditional looking representation of a computer (Fig. 3).

The four basic elements of all programmable computers emerge:

- **Memory** — A storage unit. In modern computers, memories are implemented with semiconductor or magnetic core systems. Memories can be read-only (ROM), for program storage, or read/write random access (RAM) for program, operand or temporary storage. Data is usually stored in binary form.

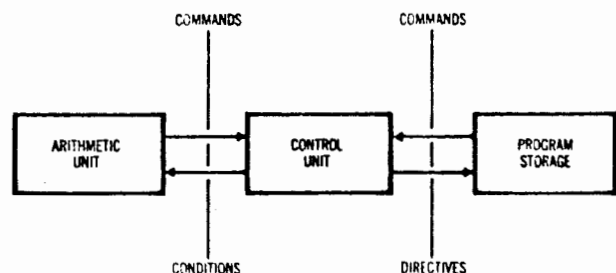


Fig. 1. In this rudimentary microprocessor, the control unit sends commands to the arithmetic unit to initiate arithmetic operations, or sends commands to program storage to look up the next instruction.

- **Arithmetic unit** — Often referred to as the arithmetic and logic unit (ALU); it performs the arithmetic operations on operands or provides partial results within the computer.

- **Control unit** — Referred to as the brain of any computer because it coordinates all units of the computer in a timed, logical sequence. In fixed-instruction computers, this unit receives directives from the program memory (hereafter directives will be called "instructions" since they instruct the computer what actions to take and when to take them). These instructions are in sequences, called programs. They reside in the memory and are referred to as software. The control unit is closely synchronized to the memory cycle speed and the execution time of each fixed instruction is often a multiple of the memory speed.

- **Input/Output** — The means by which the computer communicates with a wide variety of devices, referred to as peripherals. They include switches, indicator lamps, teletypewriters, CRT's, magnetic or paper tape units, line printers, A/D or D/A converters, card readers and punches, communication modems, etc. The I/O lines can be connected to intermediate storage devices for use with mass memories, including magnetic discs, magnetic drums and large-scale RAM systems.

To illustrate the operation of this microprocessor, or digital computer, compare two systems for solving simple mathematical expressions, both composed of the classic elements of a computer: memory unit, arithmetic unit, control unit, and input/output unit.

The first such system (Fig. 4) is a man with a cal-

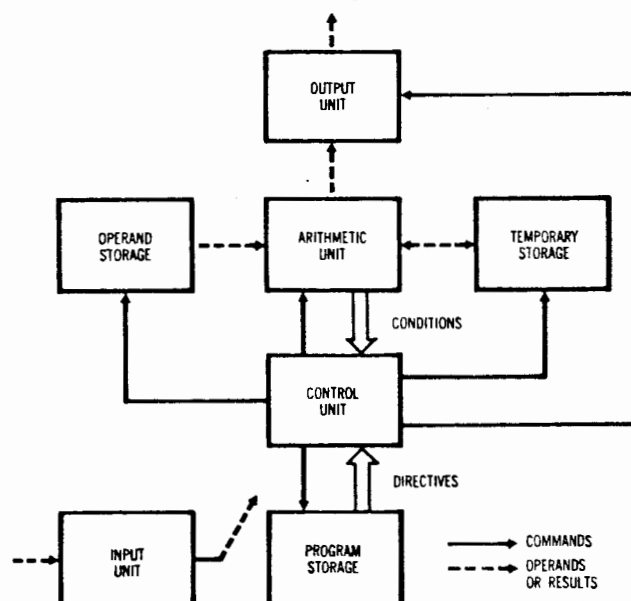


Fig. 2. By adding input and output functions plus operand and temporary storage to the rudimentary unit, a complete microprocessor evolves.

culator. The program and operand storage (memory) is the piece of paper containing a list of instructions for the man, the arithmetic unit is the calculator, the control unit is the man's brain and fingers, the input unit is his eyes and the output unit is his hand.

Examine the directions (program) that the man is to follow to solve a simple addition problem (note that this computer is externally programmed).

To add:

1. Clear calculator
2. Enter Operand #1 into calculator
3. Depress + key
4. Enter Operand #2 into calculator
5. Depress = key
6. Read and record result
7. Halt

This program would be applicable to any pair of operands to be added. But consider each step the man/calculator executes in solving the problem. For simplicity's sake, assume the problem to be solved is $6 + 2 = ?$; therefore, Operand #1 = 6 and Operand #2 = 2:

Instruction: Fetch

1. Control unit (brain) causes eyes to read step 1 from list of directions (first instruction is fetched from memory)

Instruction: Execute

2. Control unit directs fingers to depress "clear" key (first instruction is executed)

1 CYCLE

Instruction: Fetch

3. Control unit causes eyes to read step 2 from list (next instruction is fetched)

Operand: Fetch

4. Control unit causes eyes to input Operand #1 (execution of instruction begins with retrieval of 1st operand from memory)

Instruction: Execute

5. Control unit directs finger to depress keys that correspond to the value of 1st operand (in this case 6)

1 CYCLE

Instruction: Fetch

6. Control unit causes eyes to read step 3 from list (fetch)

Instruction: Execute

7. Control unit directs finger to depress + key (execute)

1 CYCLE

Instruction: Fetch

8. Control unit causes eyes to read step 4 from list (fetch)

Operand: Fetch

9. Control unit causes eyes to input Operand #2 (1st-half execute)

Instruction: Execute

10. Finger depresses keys corresponding to value of 2nd operand, a 2 (2nd-half execute)

1 CYCLE

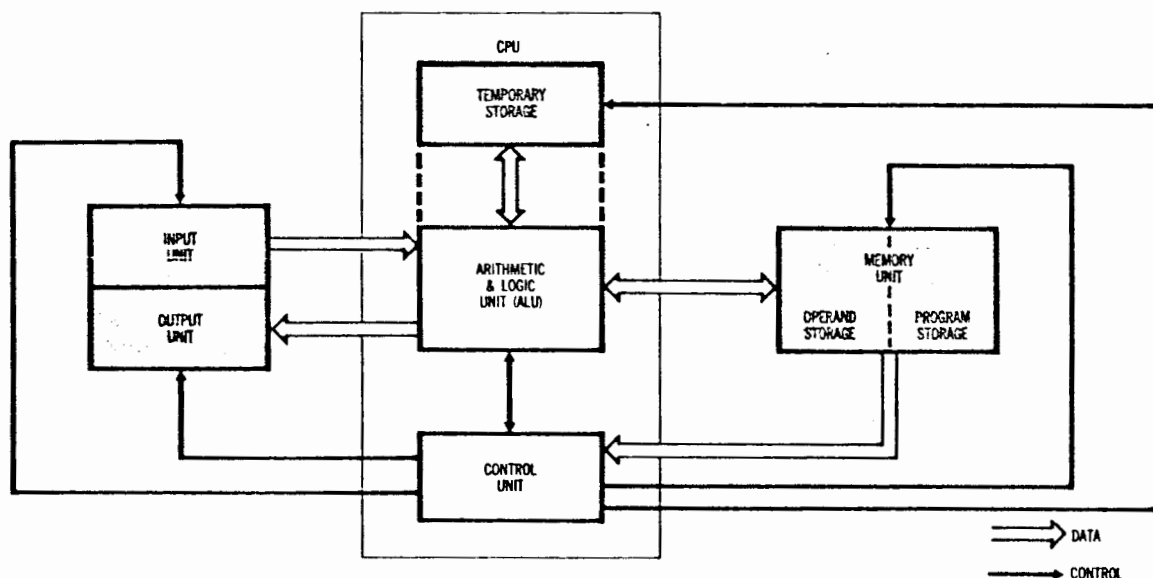


Fig. 3. By rearranging the elements shown in Fig. 2 the microprocessor reveals its conventional computer structure.

Instruction: Fetch

11. Read step 5

Instruction: Execute

12. Finger directed to depress = key

Instruction: Fetch

13. Read step 6

Instruction: Execute

14. Hand outputs result by recording it in proper place on sheet of paper

Instruction: Fetch

15. Read step 7

Instruction: Execute

16. Halt.

1 CYCLE
1 CYCLE
1 CYCLE

The computer has executed its program, outputted the result and halted; the operation is complete. Note that each step is identified as being one of three types: instruction fetch, operand fetch or instruction execute. Also the cycles, the basic unit of machine timing, are identified. As a minimum, a cycle consists of one instruction fetch and one instruction execute; if a stored operand is involved, an operand fetch is required be-

tween the instruction fetch and the execute subcycles.

A cycle is the time required by a computer to fetch, decode or execute one program step (instruction). Cycle times range from 200 nsec to several hundred microseconds. In minicomputers, machine cycle time is usually equal to memory cycle time, that is, a mini- that touts a 1.2 μ sec cycle time actually would have a 2.4 or 3.6 μ sec cycle instruction execution time.

Microcomputer performance criteria

In microcomputers, the basic time interval is the microcycle. Since both the instruction fetch and instruction execute subcycles are each comprised of one or more microcycles, depending on the machine and instruction, the cycle time calculation becomes ambiguous and complex. To illustrate, consider a microcomputer that requires two microcycles to fetch an instruction, one microcycle to decode, and one microcycle to execute a "register add" instruction, two microcycles to execute a "jump to subroutine" instruction, etc. If we assume a 2 μ sec microcycle, this machine would require a cycle time of 6 μ sec to add two registers (3 microcycles) or 8 μ sec to jump to subroutine (4 microcycles). Confused? Don't feel bad; so is everyone else!

The point is: Cycle speed or cycle time alone is not a valid evaluation criterion for a computer, and especially not for a microcomputer. To provide a performance indicator, the efficiency of the instruction set must also be considered — what can an instruction really do and how long does it take to fetch it, execute it and be ready to fetch the next instruction?

Now, look at a classic stored-program computer (Fig. 9) and see how it might be used to solve the same problem. The memory is composed of storage space for a large number of "words," with each storage space identified by a unique address. The word stored at a given address might be either computational data (operand) or an instruction (such as add, read from memory, etc.). Two temporary storage registers, each capable of containing one word, are included in the memory. These registers are designated as Memory Address Register (MAR) and Memory Data Register (MDR). The MAR contains the address where information is to be read from memory or written (stored) into memory, while the MDR contains the data being exchanged with memory.

The simplest ALU consists of an adder and an accumulator. The adder adds (or performs similar logical operations, e.g., OR) two inputs, A and B, and produces the output. The accumulator holds intermediate results of a computation or numbers for a pending computation. The accumulator is the temporary storage to which we've been referring, the storage that, for reasons of efficiency, was not included in the main memory.

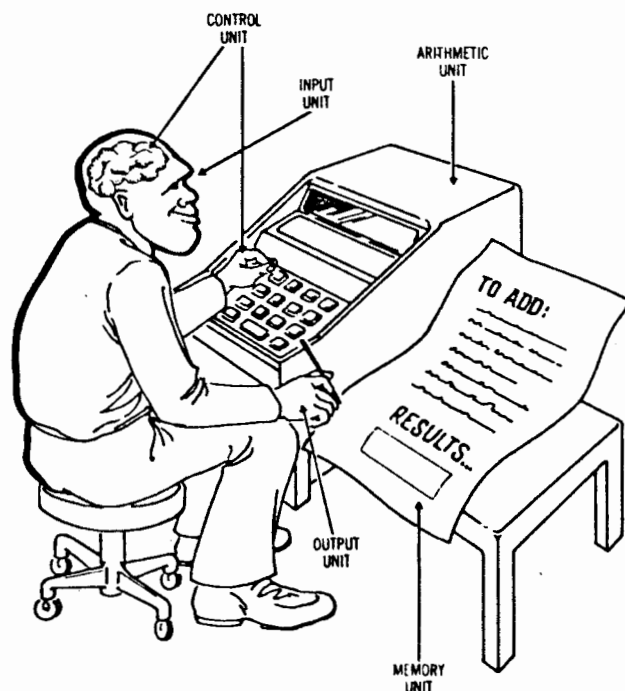


Fig. 4. In this non-traditional representation of a computer, the man performs input/output and control functions, while the calculator serves as the arithmetic unit and the paper provides memory.

The remainder of the CPU, the control portion, is implemented using an instruction register (IR), a control decoder and sequencer, and a program counter (PC). A machine instruction is transferred from program storage memory into the IR and is subsequently interpreted by the decoder/sequencer, which issues the appropriate control pulses to the other computer elements. The PC contains, at any given time, the address in memory of the next instruction. This counter is normally incremented by one immediately following the reading of a new instruction. The PC contents can be replaced by the contents of a specified memory location if the last instruction was of the jump class. This causes the next instruction to be read from a program-specified location instead of from the next sequential location.

Finally, a means of input/output (I/O) is provided via an I/O register, through which data exchanges take place with external, or peripheral devices. Voila! A complete computer.

Let's continue the analysis by executing the program described below (note this is essentially the same problem used to illustrate the man/calculator):

"Read in an operand from the I/O. Store it in memory location 50. Read in another operand from the I/O. Store it in memory location 51. Add the two numbers together. Store the result in memory location 60, and halt."

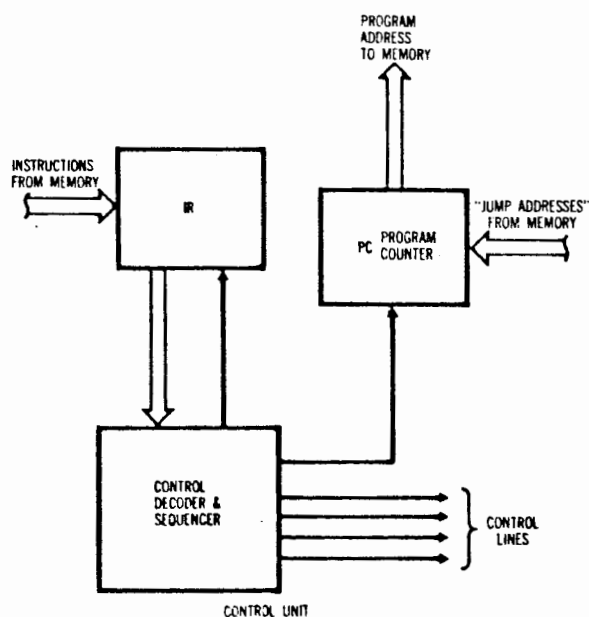


Fig. 3. In the control unit, the instruction register receives the machine instructions from program storage. These are then interpreted by the decoder/sequencer, which controls the various microprocessor elements.

Here is a program to execute this task; it is stored in consecutive memory locations beginning at address location 100.

Memory location	Instruction (contents)
100	Input to accumulator
101	Store accumulator at 50
102	Input to accumulator
103	Store accumulator at 51
104	Add accum Loc. 50 Place result in accumulator
105	Store accumulator at 60
106	Halt

To execute the program, the program counter points to each instruction in turn, starting at 100. The processor fetches the instruction, decodes it, and finally executes it in one or more microcycles. When the microprocessor reaches 106, the operation is complete. No human intervention was required — every operation was automatic. All computers, regardless of their size or intended purpose, operate in a similar manner. It must be emphasized, however, that many variations are possible within this basic architectural framework.

Variations on a theme

Common improvements, additions and/or alternations to the classic architecture described above include multiple accumulators, sophisticated I/O structures, index registers, indirect addressing, interrupts, push-down stacks and microprogrammed control units. Such features can enhance a microprocessor's capabilities and are often the basis for comparisons between various machines, as well as providing a theme for competitive advertising and salesmanship. In view of these three facts, a discussion of basic computer variations follows:

Accumulators, multiple — By definition, an accumulator provides a temporary storage medium. Temporary storage allows programs to execute faster and more efficiently by obviating the need to store partial or intermediate results in main memory and subsequently to retrieve them for use in additional computations. Multiple accumulator registers allow several partial results to be maintained at the computer's fingertips, thereby eliminating the many program steps that would otherwise be required to store and then retrieve data (shorter programs cost less to write, less to store and execute faster than longer programs). Four accumulators are able to provide a great deal of programming and operational versatility, and it is often considered an optimum number.

I/O structures — A basic input/output system provides a single input port and a single output port. A port allows transfer of one word of data across the

computer's boundary. More sophisticated I/O units facilitate the use of multiple input and output ports, allowing virtually simultaneous communication with many peripheral devices. Another powerful I/O technique, referred to as DMA (for direct memory access), allows peripheral devices to transfer data directly into and out of memory, independent of the control unit and the operating program. This contrasts to the more conventional programmed I/O, where an explicit program instruction is required for any data transfer. The DMA technique facilitates faster data exchanges with memory, with fewer program steps, and is considered most applicable to bulk storage device (disc) interfaces and computer-to-computer connections. Contrary to occasional misuse of the term, DMA is not a uniquely defined off-the-shelf circuit, but can be implemented in a variety of ways in any general purpose computer.

Index registers — This feature provides programming flexibility by providing the user with more memory addressing modes. As a rule, when a programmer wishes to retrieve an operand from memory, he specifies its address in the instruction that calls for work to be performed on that operand (e.g., add it to an accumulator). The presence of an index register(s) allows the programmer to modify or index the operand address with the number contained in the register. Typically, the operand address from the instruction

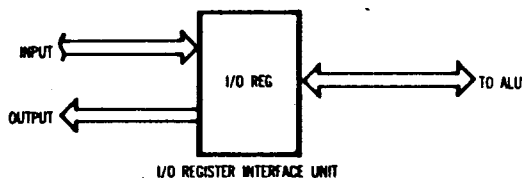


Fig. 6. Input/output register interface unit. This component provides the data exchange link between the microprocessor and the outside world.

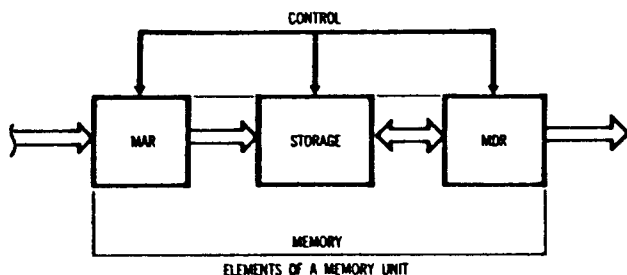


Fig. 7. Elements of a memory unit. In the operation of a memory, the MAR contains the address where information is to be stored or read. The MDR holds the data to be stored or receives the data as it is read.

would be added to the content of the index register. Such a feature greatly simplifies the transfer of an array or field of data into or out of memory. Machines with two index registers offer enhanced programming versatility over machines with a single register.

Indirect addressing — This is done when the address contained in the instruction specifies only the address of a memory word, which itself, specifies the operand address. An indirect address is an address in an instruction that indicates the location of the address of the referenced operand. Think of it as a computerized treasure hunt — the instruction does not tell the location of the "treasure" (operand), but tells where to go to find a clue that gives its location. Multi-level indirection is possible, although not considered necessary. Here the system jumps through two or more clues until the operand is found. Indirect addressing provides great programming flexibility by allowing operand address to be continuously modified by the program.

Interrupts — A machine operates under its own control but frequently it is desirable to have an external event cause the computer to shift its attention to another problem. This can be done in many ways:

- The computer program can include a section that causes it to look for possible external events each time it cycles. This may consume a lot of memory, make the computer operate its program more slowly and may not permit the computer to respond quickly to the external event.
- Interrupt signals may be forced into the computer. This requires extensive programming to insure that, when the external event has been serviced, the computer can return to its prior location.
- The computer can have interrupt capability built into its hardware, thus allowing the computer to service the interrupt quickly, with a minimum expenditure of program and memory storage space.

Push-down stack — Or "Push-down, pop-up" stack, LIFO (last-in, first-out) stack, etc. This is a useful feature for the "nesting" of interrupts and subroutines. Nesting refers to the entry into a second (or third) interrupt service program or subroutine prior to completion of service or execution of the first. The stack stores the current program execution address (contents of PC) each time the computer is directed to a new ancillary task, thereby allowing the computer to return and clean-up unfinished work in reverse order. The stack is also useful for storing partial results of computations. (Subroutine: A set of instructions necessary to direct the computer to carry out a well-defined mathematical, logical or analytical operation, usually arranged so that control can be transferred to it from the main program

and so that, at the conclusion of the subroutine, control reverts to the main program.)

Microprogrammed control unit — In a computer with a microprogrammed control unit (MCU), three of the basic elements are nearly identical to our classic fixed-instruction computer; the significant difference is that the control unit has its own memory. This control memory contains the stored sequence of control functions that dictate the end-user architecture and the instruction repertoire of the microprogrammed computer. Thus, the instruction set can be modified or increased to adapt the microprocessor to system needs.

Instructions are machine directives and are the prime constituent of programs. They are fetched one-at-a-time by the control unit, which then carries out the operation(s) indicated in the instruction.

Instructions for most modern computers can be grouped into eight functional classes: load/store, arithmetic, logical, skip, shifts, transfer of control, register and I/O. A brief description and example of each class follows.

Load/store — This instruction class performs the function of exchanging data between main memory and temporary storage registers (accumulator, index, etc.). Load transfers contents of a selected memory location into a designated register. Store reverses the operation. Typical class members include load, load indirect, store and store indirect.

Arithmetic — Almost self-explanatory; these instructions perform an arithmetic operation upon two operands, one of which is in a register and the other in memory; the result usually replaces the operand in the register. Typical members include add, subtract, multiply and divide.

Logical — These perform a logical operation on two operands, one of which is in a register and the other in

memory; the result usually replaces the operand in the register. Included are AND, OR, EXCLUSIVE-OR.

Example: Logical OR

```
Operand 1: 0 1 1 0 1 0 1 1 (Register)
Operand 2: 0 0 1 1 0 0 1 0 (Memory)
Result:    0 1 1 1 1 0 1 1 (Register)
```

Skip — These are usually 2-phase instructions; that is, an arithmetic or logical operation is performed on one or two operands and the result is tested for a specific condition (e.g., positive). If the condition is met, the next sequential instruction in the program is skipped. Class members include: increment and skip if zero, decrement and skip if zero, skip if greater, and skip if not equal.

Example: Decrement and skip if zero

A specified memory location has 1 subtracted from it; if the result is equal to zero, the next instruction is skipped.

Example: Skip if not equal

The contents of the specified register are compared to the contents of a specified memory location; if the two contents are not exactly identical, the next instruction is skipped.

Shifts — The contents of a designated register are shifted one bit to the left or right. The bit position that is vacated can be filled with a zero (shift) or the bit that "fell off" the other end (rotate). Rotate is merely a circular shift.

Example: Shift left

```
Before  1 1 0 0 0 1 1 0 1 1
After   1 0 0 0 1 1 0 1 1 0 ← 0
        1
```

Example: Rotate right

```
Before  0 1 0 0 0 0 1 1 0 1
After   1 0 1 0 0 0 0 1 1 0 → 1
```

Transfer of control — This class of instruction causes the Program Counter (PC) to jump to an instruction — specified point in the program; that is, control of the computer is transferred to a new program element. Such transfers can be conditioned (based upon some operation and/or test) or unconditional. Conditional transfer includes Branch if Accumulator Positive, Branch if Condition, and Branch if Register = 0. Unconditional transfers include Jump, Jump to Subroutine, and Return From Interrupt. An immense amount of programming power is found or lost here.

Register — Included here are instructions that per-

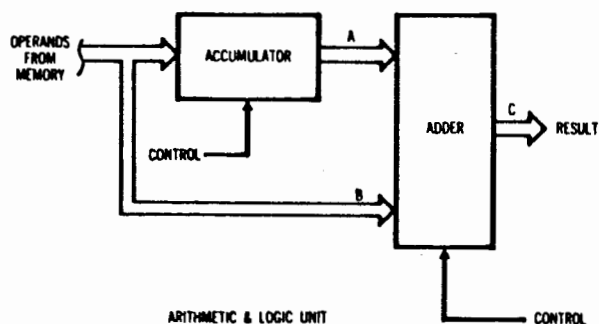


Fig. 8. This simple ALU contains an adder and an accumulator. The accumulator provides temporary storage. For example, here it can hold one operand while another is obtained from memory in order to perform addition.

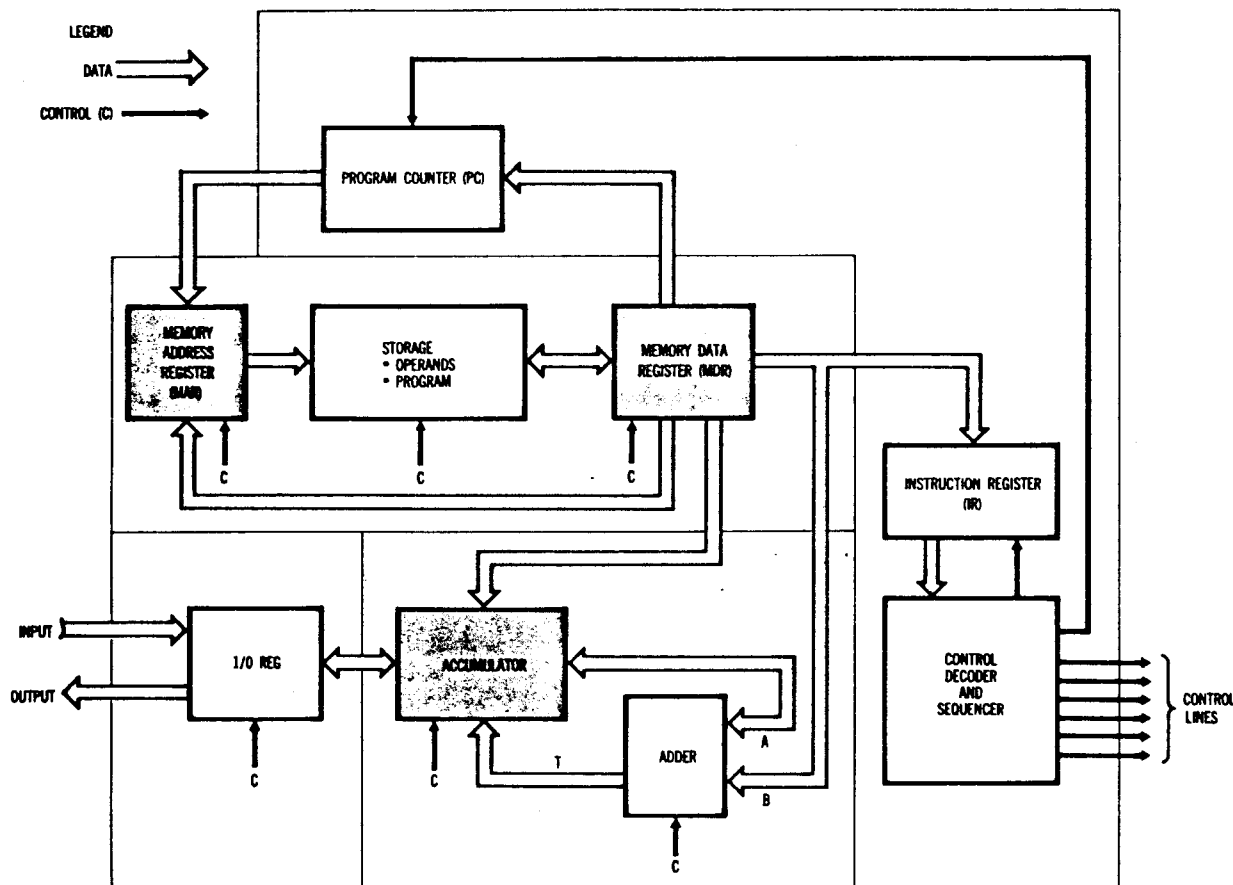


Fig. 9. Combining Figures 5 through 8, the whole computer emerges. Memory is performed in various parts of this microprocessor. The memory register contains the address where information is to be read or stored, while the memory data register holds the data being exchanged with memory. The accumulator serves as a temporary storage unit.

form arithmetic or logical operations on the contents of two registers or after the content of a single register. Examples of two register instructions: Exchange Register and Stack, Register Add, Register Copy. Single Register instructions include Load Immediate, as well as Complement and Add Immediate. In an immediate instruction, the operand is inherently included.

Input/Output — An enormous variety of instructions are possible here; commoner instructions are those that transfer the content of a specific register to an output port (Register Out) or transfer the word appearing in an input port into a register (Register In).

There are as many instruction sets as there are computers, and it is quite difficult to say which are good and which are bad. The number of instructions is not a good indication of the power of a computer since each manufacturer counts differently. An instruction set that one manufacturer states has 43 instructions might be called 352, using another manufacturer's procedure (i.e., a register to register add might be counted as one instruction but in a machine

with four registers it could be counted as sixteen).

One of the true measures of a machine is how many instructions it requires to execute a given "benchmark" program. It is important to note here that a computer with a microprogrammed control unit can be configured to execute any instruction; therefore the number of instructions required for a specific class of jobs can be minimized by tailoring the instruction set to the peculiar requirements of those jobs. Instruction efficiency is, in turn, related back to the architecture and word size of the individual computer. ☉

The second part of this microprocessor primer covers such important topics as software, the advantages and disadvantages of microprocessors, and what to look for when choosing a microprocessor. Reprints of both Part 1 and Part 2 will be available next month. Check the February issue for further details.

Primer on Microprocessors



PART 2

Part 1 of this article described the basic elements of a microprocessor, told how a microprocessor functions and introduced the concept of instructions. It also concluded that microprocessors are computers in IC form, and that the terms "microprocessor" and "computer" can be used synonymously.

Taking this concept further, Part 2 deals with how to tell the microprocessor what to do. This, of course, can be implemented using hardware or software, but defining the solution is the most important and most difficult part.

Software is a term used to describe the programs that make a computer do a specific task. In fact, when used in the context of computers, the word software can be interchanged with the word program. In general, a program is a series of sequential steps that accomplish an objective. A list of directions to travel from Philadelphia to San Francisco is a program: Drive to Philadelphia airport, get parking ticket, park car, write parking section on back of ticket, take bus to terminal building, buy ticket at the United counter, check monitor to determine gate, go to gate, etc. If you followed such a list of instructions or program (and it was correct), you would end up in San Francisco. Note that the program asked you (the machine?) to pick up information in certain places and to act on it or to store information (writing down the parking section) to be retrieved and used later. Note also that the order of execution of each step is very important.

A computer is a device that can recognize and act on a predetermined set of instructions. Even though

the specific set of instructions it can use is fixed by its design, a computer is general purpose because it can execute a list of these instructions (a program) to perform some functions, execute another list of instructions to perform some other function, and so on.

Since many applications for microcomputers can also accept a hardware solution, you should compare the design steps you would use for each. Since software is designed like hardware, it is interesting to note how similar the following steps are:

Software:

- Define the problem and what data, inputs, and outputs are available and/or required for its solution.
- Determine the best form of the solution.
- Outline the method of solution on a flow chart.
- Write the entire program, step by step, using the computer's instruction set. Assemble or compile the program (if necessary).
- Load the program into the computer memory and run it to test and debug.

Hardware:

- Define the program and what data, inputs and outputs are available and/or required for its solution.
- Determine the best form of the solution.
- Outline the method of solution on a flow chart. A static diagram can also be used.
- Draw up the detailed logic diagram using the available and compatible SSI, MSI and LSI functions.
- Make wire lists, etc.
- Wire circuit boards; operate to test and debug.



Defining the program is the most important and probably the most difficult part of either solution. Step 2 depends largely on what resources the designer has at his disposal. This is the point where a decision will be made to go hardware or software. Note that for some design problems the flow chart for hardware and software may look the same.

Writing the program, Step 4, determines the incremental cost of the system, since it defines the amount of memory required to store the program. Since the number of instructions required to perform a certain function may be different for each computer on which the function is programmed, the cost of performing a given function will depend on the instruction set of the computer used.

The speed at which a given function may be performed depends on the instruction set of the computer as well as the actual time it takes the machine to cycle through a given instruction. Because of this, a machine that is considered fast may take much longer to perform a given function than a machine that is considered slow. This paradox is part of the reason why "proper CPU selection is not easy." One almost has to write his program for several machines before making an accurate comparison of cost and performance. These tests are sometimes referred to as benchmark tests.

Software design is the analog of hardware logic design. The efficiency of the software design is measured primarily in the amount of memory used to store the software and the time required for execution of the program. Hardware design efficiency is measured in

number of gates and functions used (packages x cost). It must be assumed that the efficient hardware design would perform the function as fast as required. There would be no advantage in greater speed, since a hardware based system would not lend itself to doing more functions in its spare time, while a software based system would.

Commanding the computer

We hear talk about software and programming — and most talk about programming in some language or another. This is because the way we command the machine is very much like the way we communicate by a written language. We have rules about how we start and end sentences and paragraphs and how we spell words. The way we communicate with a computer is through a programming language, which also has rules of spelling and punctuation, but these rules are much more strictly enforced. If you misspell a few words, your reader will probably understand you anyway. A computer language is not that forgiving and will not produce the desired result if its rules are broken.

There are a number of levels of programming languages, as shown in Fig. 1. The innermost level is that of the actual machine language. Each instruction is uniquely defined by binary code (pattern) of ones and zeros. The central processing unit (CPU) examines each instruction code and performs the exact sequence of events to produce the operation defined by that instruction. Assume a 0011000100000000 code tells the computer to add register (accumulator) zero to register

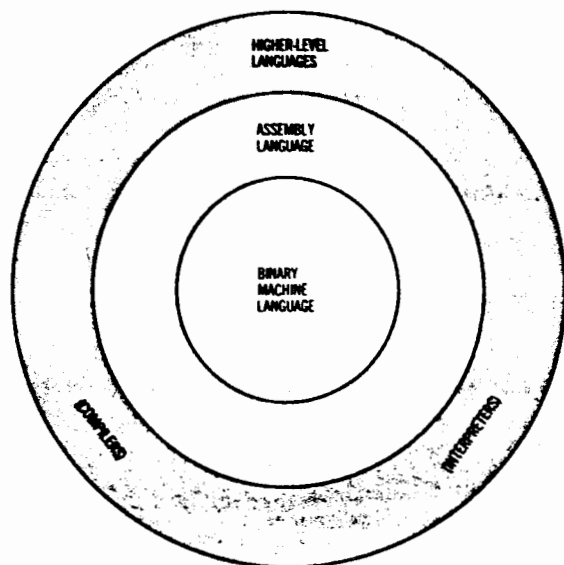


Fig. 1. In the hierarchy of programming, languages can be represented as a sphere. In the center is the machine code and each layer away from the center is closer to human language.

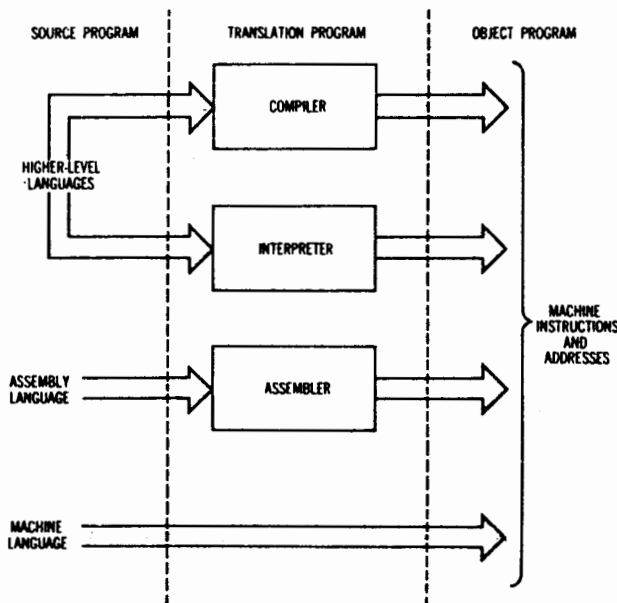


Fig. 2. Except for machine language itself, each user-written source program must be translated into a machine language object program before the computer can use it.

one and put the result in register one. When programming in machine language, the programmer must enter 0011 0001 0000 0000 to add register zero to register one. This can be awkward as well as quite slow; it isn't easy to remember all the codes. In spite of its disadvantages, the use of machine language is a perfectly reasonable way to program when the application is not too complex and the effort is on a low budget.

To make programming easier, assemblers have been developed; they are the next level on the software sphere. An assembler or assembler program is a computer program that accepts coded instructions or mnemonics that are more meaningful to use and translates them into a binary machine code the computer can execute. The mnemonics used for each instruction are much easier to remember, and they make a listing of the program much easier to read. The mnemonic for the register to register add mentioned above might be RADD 0, 1.

Keeping track of each instruction

The use of easy to remember and easy to work with symbolic codes in place of the ones and zeros of machine language is not the only improvement assemblers can provide. An assembler keeps track of the location of each instruction, which is important because it allows the programmer to use symbolic labels for important locations in the program. These labels allow references to be made to locations in a program without keeping track of the exact memory locations (which would change if instructions are inserted or deleted between the location and where it is referenced). This "bookkeeping" feature allows the assembler program to choose the best addressing modes (i.e., indirect, indexing, etc.) automatically if the instruction set has a variety of addressing modes.

In addition to allowing the use of mnemonics and labels, assemblers permit listings to include comments that help to document the programmer's work, macros that assign a mnemonic to groups of code, listings of labels and where they are found, and many other such refinements.

The outer layer of the software sphere is the area of the higher-level languages, which come the closest to natural or human languages. They are problem oriented and contain familiar words and expressions; however, they have very strictly defined structure and syntax. There are two types of higher level languages, compilers and interpreters. Both types are programs that take the higher-level language program the programmer writes and turn it into machine language the computer can use. The major difference between a compiler language and an interpretive language is how the language program converts to binary machine language. A compiler takes the whole program and converts (translates) it into binary machine language before it is ready to execute it, while an interpreter translates the program into executable binary machine code on a statement basis (and usually executes them at the time).

High level languages are often written for specific needs and special uses. Some that may be familiar are ALGOL and FORTRAN for scientific users, COBOL for large business systems, RPG for small business systems, BASIC and APL for time sharing, and PL-1 for large, general systems.

A higher level language (such as BASIC) might have a statement like the following:

LET ANS = A x B + C/D.

This statement computes the value ANS by multiplying the previously defined values of A and B and adding the result to C divided by D. The same statement written for another computer, but in the same

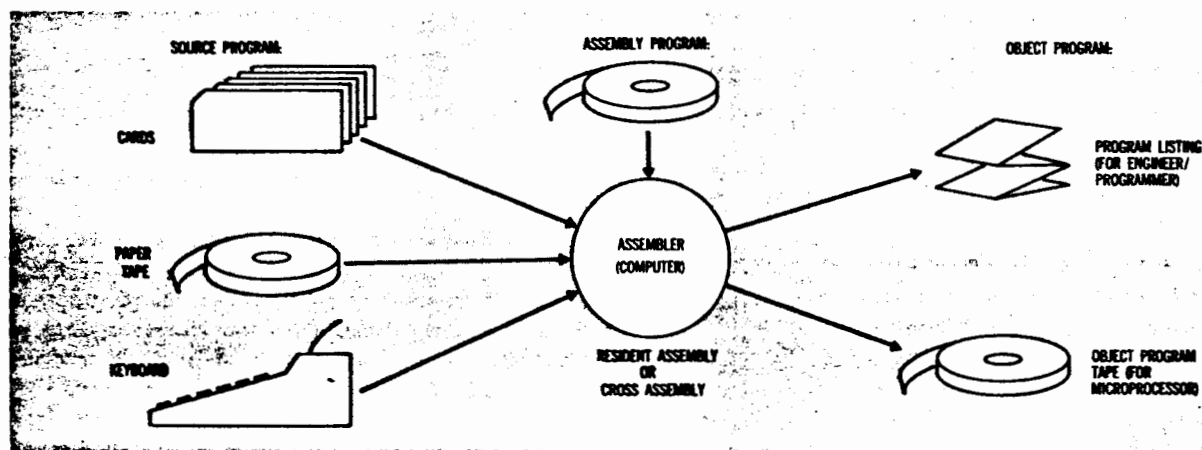
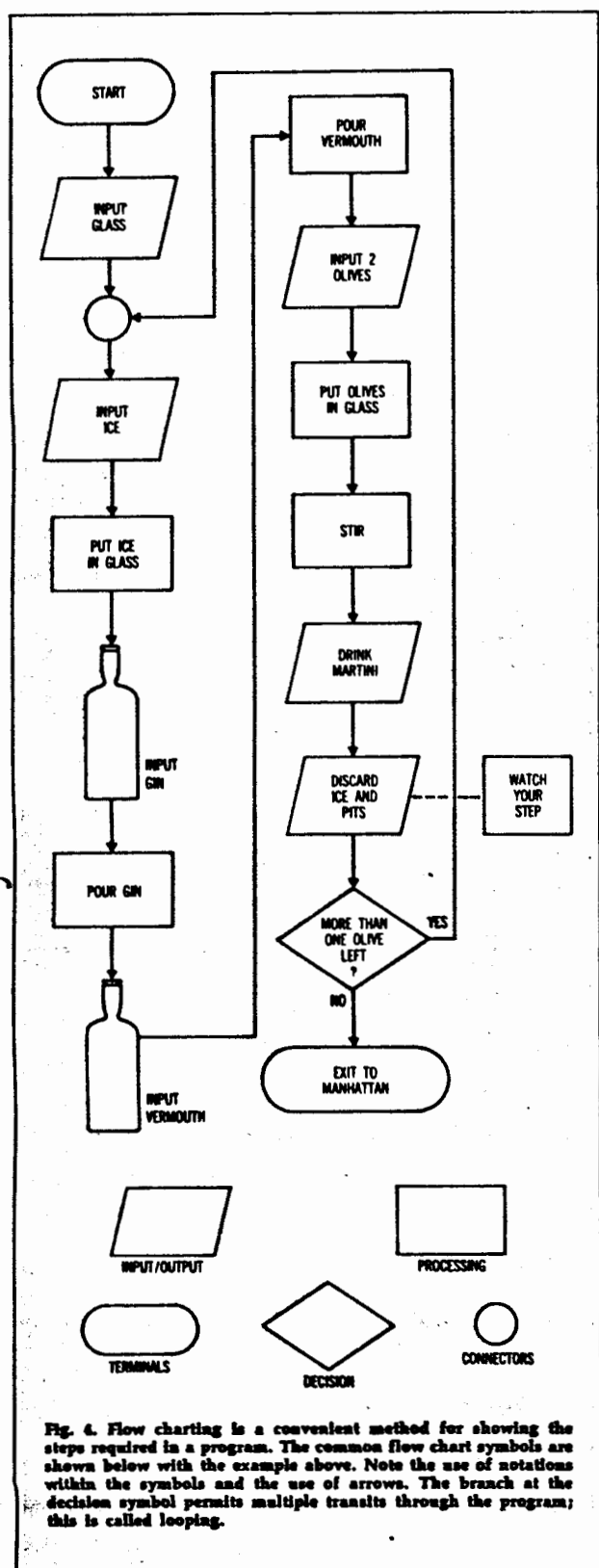


Fig. 3. By various means, the source program can be entered into a computer where it is processed to prepare an object program (machine language) for the microprocessor. If the computer is the same microprocessor that will execute the program, this step is called resident assembly; if it is another computer, it's called cross-assembly.



language, would look essentially the same because the details of the instruction set, addressing modes, register assignments, etc., are taken care of by the compiler (or interpreter). Therefore, the compiler (or interpreter) program becomes, for all practical purposes, your computer, and it isn't necessary for you to know or care about the detailed operation of the host computer.

Defining the program

At this point, source programs and object programs must be defined. A source program is a program written by the programmer in any of the languages discussed. The object program is the list of binary machine instructions (and addresses) that is ready to be loaded into the computer and be executed. The object program is generally produced from the source program by one of the types of computer programs, i.e., compiler, interpreter, or assembler. These relationships are illustrated in Fig. 2. Note that a machine language program requires no intermediate step.

It isn't necessary for the translation program to be run on the same type of computer that the object program is generated for. In fact, it is often not practical and sometimes, not even possible, because compilers are long programs and so take a lot of memory. Furthermore, some microcomputer instruction sets will not support a practical assembler. When the translation program is run on another machine, it is called a cross-assembler or cross-compiler (Fig. 3).

The main advantages of machine language programming are that it can be completed without the aid of another program, and it allows the programmer to keep track of and control every detail of the machine operation. Assembly language programming allows the programmer to retain complete control over the important details of the computer operation, but takes care of all the drudgery of the binary coding, address calculations, and the like.

Compilers have an advantage in that programs can be written without regard to which machine they will run on. The higher-level instruction example given above might take 30 to 50 machine language instructions; this shows how much work a higher level language might save a programmer. This relative simplicity allows a person to be trained as a programmer in a fairly short time. With compilers, the programmer does not concern himself with the inner workings of the computer or even the details of how the compiler generates code, e.g., to produce a multiply. These very advantages, however, can be cited as disadvantages. Take the case of a multiply: The multiply function can be written in many ways; one uses very few instructions (not much memory) but is very slow; another

MICROPROCESSOR CHECKLIST

COST CONSIDERATIONS

Normally chip cost is not the largest factor, but it could be in a simple system. The major cost is usually in the circuitry needed to support the microprocessor. The engineering cost to put your system together, however, can be greatly influenced by the amount of quality of support the chip supplier has available.

Support circuitry

- ☐ Clock circuitry and clock drivers — how many phases?
- ☐ Power supplies — common or special?
- ☐ Buffers — MOS to TTL input/output?
- ☐ How much control logic (i.e., address and data latches, etc.)?
- ☐ Memory — standard or special?
- ☐ Power of the instruction set (a good instruction set can reduce memory requirements by 40%).
- ☐ Support requirements (the larger these are, the more expensive the PCB or the greater the number of PCB's required).
- ☐ Ease of checkout (test vs purchased card).
- ☐ Processor card availability (small production and preproduction will use off-the-shelf cards).

Support from supplier

- ☐ Hardware support
 - prototyping system
 - mechanical hardware
 - processor cards
 - memory cards
 - interface cards and cables
- ☐ Software support
 - high level languages
 - assemblers
 - utility programs — debug and edit
 - loaders — absolute and relocatable
 - peripheral drivers (TTY, card reader, line printer, floppy disc, etc.)
 - special subroutines (BCD to binary and binary to BCD, floating point math package, etc.)

Literature

How well written the hardware and software manuals are determines how much time is spent in learning the system.

- ☐ Technical manuals
- ☐ Software manuals
- ☐ Application notes
- ☐ Special interfaces — D/A, A/D peripherals

Technical support

This can reduce the engineering time and cost required to get the new product designed and into production.

- ☐ Area system specialist
- ☐ Field application engineer
- ☐ Plant applications and engineering groups

PERFORMANCE

Speed

- ☐ Efficiency of the instruction set — how many instructions are needed to solve a particular problem (a math problem, a process control problem, data handling, etc.)?
- ☐ Execution time of each instruction.
- ☐ Microprogrammability — can the instruction set be changed?

Interface

- ☐ Input/output flexibility and capability
 - How many peripherals can be handled?
 - How many commands to each peripheral?
 - How large is the subroutine to handle any of the peripherals?
 - How much logic is required?
- ☐ Interrupt flexibility and capability
 - Can vectored interrupts and/or polled interrupts be handled by the processor?
 - How many interrupts can be handled by the processor?
- ☐ Special control features
 - enable signals (single line control where fast response or ease of interface is important).
 - sense inputs (test a single input and respond accordingly).

uses a lot of instructions but is very fast. Which one should the compiler use?

The programmer has no control over these types of decisions and must accept all the constraints and compromises designed into the compiler. Other disadvantages of compilers include their often inefficient use of the machine instruction set in applications for which the compiler has not been specifically optimized, the problems involved in debugging the resultant object code on the actual machine, and the loss of control over things such as interrupts, register assignments and manipulations of individual bits (necessary in control applications). Compiler generated object programs generally take considerably more memory than the same program written in assembly language. Whether you consider this as an advantage or disadvantage depends on how many systems you will build and whether you are buying or selling the memory.

The following is a list of other software that is encountered while using microprocessors.

- **Simulators** — Software simulators are sometimes used to debug programs using another computer. They are especially useful if the actual computer is not available (or hasn't been built yet). If hardware is available, the use of a simulator is an unnecessary extra step, since the software must still be debugged on the hardware. The cost of the computer time to run the simulator effectively is often more than the cost of a prototyping system.
- **Debug programs** — Debug programs help the programmer to find errors in his programs while they are running on the computer, and allow him to replace or patch instructions into (or out of) his program.
- **Diagnostic programs** — These programs check the various hardware parts of a system for proper operation; CPU diagnostics check the CPU, memory diagnostics check the memory, and so forth.
- **Loaders** — The various applications (user written) programs must be placed in the proper locations of the system memory. The programs that do this job

are called loaders. Loader programs range from simple ones that load absolute binary object code with no error detection, to sophisticated loaders that load relocatable binary object code, resolve global (between program) symbolic label linkages, perform error detection, and execute various commands, including starting the program just loaded.

- **Editor** — As an aid in preparing source programs, certain on-line programs have been developed that manipulate text material. These programs, called editors, text editors or paper tape editors make life easier for those who have system time to write source programs on-line.
- **I/O handlers** — Input/output handlers, sometimes called device drivers, are subroutines that service specific peripheral devices such as teletypewriters and card readers. They help prevent "reinvention of the wheel" every time a programmer wants to use a standard peripheral.

What does software really cost? There are a number of "rules of thumb," each one as erroneous as the other. Let's face it — software is expensive.

- An extremely large mini-computer manufacturer charges its customers \$50 per hour for custom software. This price is typical of the larger CPU manufacturers; however, small system houses with lower overhead charge about \$35 per hour.
- When all the hours are tabulated for writing a program, including flowchart, instructions, checkout, rewriting, recheckout and documentation, a valid figure of 10 to 20 instructions per day can be expected. This is not a typographic error: 10 to 20 instructions per DAY.

Thus, at \$35 per hour, software will cost \$280 for an eight hour workday. Divide this sum by the mini-

imum number of 10 instructions (\$280/10), and software costs \$28 per instruction; divide the sum by the maximum number of 20 instructions (\$280/20), and it costs \$14 per instruction. Even if the work is done in-house, you can still expect a minimum of \$10 per instruction.

It is also necessary to consider machine language vs assembly vs high level language. Obviously, few people really write machine language programs (1's and 0's) of more than a few instructions and then only to test a particular function or hardware interface. You do write assembly level programs and also consider high level programs.

High level programs can be developed at one half to one tenth the cost of the assembly language program. But they are inefficient because they require more memory and run more slowly than assembly language programs. At the present time, the argument is academic because only one high level language is available for one manufacturer's microprocessor.

What is microprogramming?

Microprogramming has a number of different meanings. To some people microprogramming means the use of ROM for program storage instead of RAM. To others it means the combining of instruction codes such as can be done with a PDP-8. The preferred meaning refers to the programming of the control section of a computer. A macroinstruction is decoded by the control section of the computer; the control section then "pulls the proper strings" to do the operation specified by the instruction. With a microprogrammed controller, this string pulling is carried out by microinstructions. This is an alternative to the use of random logic to do the control section function. The greatest

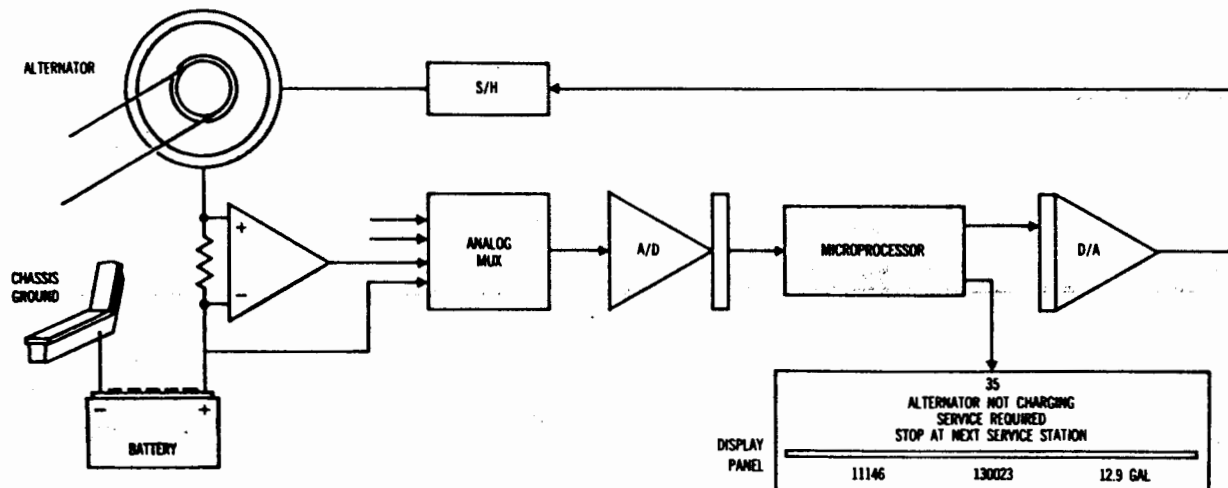


Fig. 5. Microprocessors can perform a variety of functions in an automobile. Here the system senses any alternator failure and reports this to the driver.

WHERE CAN MICROPROCESSORS BE USED?

The potential uses of microprocessors are virtually limitless. A few comments on their functions in major areas cover only a small part of what they can accomplish.

- Commercial building control systems perform the following functions: building automation (temperature control, lights turned on and off, etc.), building fire protection (when a fire is detected, the air flow contains the fire), building security (the system monitors windows, doors, etc.). In the past, this type of control system was implemented with hardwired processors or simple-to-complex logic systems. The current trend is to replace these hardwired processors with one or more microprocessors. Home control systems are very simple (thermostats), but will be a major user when very-low-cost microprocessors become available.

- Industrial control systems include process control and test instruments. Process control systems perform water treatment, waste treatment, metals processing/mining, ceramics, petroleum, petro-chemical refining and power plant regulation. These are now done by a hardwired controller or minicomputer, with future trends leaning in the direction of microcomputers. A general rule of thumb is that hardwired controllers and minicomputers work to only 15% to 25% of capacity in a process control environment; therefore, microcomputers can replace most hardwired controllers and minicomputers, even though the microcomputer may be slower.

- The primary use for information system computers is in the area of electronic data processing (EDP). Some traditional tasks of EDP computers are payroll, inventory control, management information and general accounting. Microcomputers are beginning to replace the traditional low-end EDP computers, but are not expected to compete with the medium or large EDP computers.

- Another area opening up to microcomputers in EDP is the replacement of hardwired logic in such devices as card readers, mag tapes, CRT's, front-end processors for telecommunications, plus time-sharing and point-of-sale terminals (intelligent cash registers). These new areas will be high volume users of microcomputers.

Emerging applications

There are several positions now being usurped by the "computer on a chip" concept that were previously the domain of other techniques. Airline ticketing functions are a prime example of the functional switchover from a man-oriented system to a computer-oriented one. As an example of computer replacement of personnel, look at your friendly neighborhood bookie; state run off-track-betting machines, run by microprocessors communicating with larger machines, will probably run him out of business.

The changeover from analog to digital computers is becoming apparent in many process control applications where the condition of one stage of process effects a previous one. Until recently, digital methods have not been cost competitive with linear computational devices despite the labor overhead required to keep the analog computer on line. A primary consideration is the drift free operation of the microprocessor.

Evidence of the transition from the use of large computers to microcomputers can be seen in machine card control applications, where once one computer controlled several machine tools; dedication of each tool to a single microprocessor reduces line stoppages when there is a computer failure. Quite frequently the changeover from a large to a small computer can mean the life or death of an idea. A case in point is that of the satellite navigation system originally employed for spacecraft tracking, ships and sophisticated military vessels. The original systems used a

full blown computer, requiring large amounts of space and power aboard ship. Now, thanks to microprocessors, the entire system, including receivers, is packaged in a box smaller than an orange crate. It delivers accuracies within 40 yards of the large system and is so inexpensive that not only will all seagoing naval vessels carry it, but a majority of the merchant fleets of the world are expected to use it as well.

- The game of Pong in every bar or motel lobby is only the first step in the use of microprocessors in toys and games. Pong's imitators and successors are on their way. There'll be more than one son of Pong. The more sophisticated son will remain in the motel lobbies and bars; the cheaper, slower, dumber son will invade the home in the form of one man chess and electronic bridge. Grandson of Pong will be the delight of battery manufacturers everywhere, eminently trip-overable and what every child wants because of super-saturation advertising on Saturday morning TV.

- Sometime in the next few years, you are going to yell at your television set and it will answer you back. With the advent of cable TV will come home high speed communication channels controlled by microprocessors. Time shared computer access, citywide town meetings in which instantaneous citizen response is available, and maybe even the ability to boo the visiting team, will be channeled through the cable TV set. High speed data channels exist now that will proliferate even more. Smart terminals, such as teaching machines, library researching units, and off-track betting machines, will perform portions of the job and refer tougher parts to central mainframes.

- A major automotive application will be the on-board car and truck computer. The computer will monitor and control such things as spark advance, carburetor gas flow, transmission shift, etc. It will also provide driver warnings of such things as alternator failure and what to do about it (Fig. 6). It may even drive the car for you. The car controller, however, will become a one chip custom device used in very high volumes, so this may be considered custom LSI rather than a microprocessor.

- Automated gas stations are being tried using minicomputers, but a microprocessor will do this, too (Fig. 7).

- Microprocessors will end up in electric typewriters as controllers for self-justifying and executive spacing and as data communicators to CPU devices for such functions as editing, type-setting and translation.

- Specialized calculators, too low in volume potential for specialized chips, will appear. Private boating and aviation navigation aids are examples, along with hand carried mortar trajectory calculators for the Army and Marine Corps.

- Microprocessors will control machines involved in mail sorting, inventory pulling and stocking, and palletization of freight. Irrigating systems will sense crop needs for water and fertilizer, delivering the required amounts to whole fields or specific areas, depending on the microclimate.

- Very low cost processors will encourage the use of "throw-aways" in such areas as weather data collection, oceanographic monitoring, and weapons.

- Automatic controllers for: traffic lights, tools, stoves, drafting machines, looms, photography processing, paint mixers, asphalt makers, grape crushers, banana peelers, packaging machines, power switching, railroads, piano tuners, anti-skid braking systems, no-slip four wheel drive, fast food businesses, automated radio stations.

advantage of a microprogrammed controller is that the microinstruction set can be altered by changing the microprogram instead of rewiring a bunch of logic. (This procedure is much more difficult to execute on an LSI chip.)

Why use a microprocessor?

The main advantages of using a microprocessor approach to system design are:

- **Short design cycles** — The use of microprocessors allows rapid design once a basic set of boards and I/O interfaces have been developed. Because of the standardized nature of the logic, many aspects of the design can proceed in parallel. Logic design for special I/O and programming can proceed together once basic ground rules have been set.
- And the ease with which the product can be modified allows earlier entry to the marketplace and faster resolution of any shortcomings.
- **Lower cost** — The use of fewer components can result in large cost savings for moderate-sized systems. The use of the same circuit boards for a variety of applications results in economies of scale.
- **Flexibility of the end product** — Allows redefinition of product without costly redesign. A wide variety of changes are possible by reprogramming.

For example, the company planning a product prepares a business specification, followed by a hardware spec based on what is practical and what is salable.

If the hard-wired system approach is chosen, the entire system must be designed logically. When complete or nearly complete, power requirements can be totaled and power supplies ordered.

The design must be breadboarded, which may point out logical design errors or may even force rewriting the equipment specification. Then the system is tested; if it doesn't meet specifications, partial or complete redesign is required. Next, the board layout is done, which may require two or three iterations, or even rebreadboarding. Finally, the mechanical design and system are tested. There is no guarantee of passing system test and more redesign may be required.

If, after the business specification is written, the equipment specifications include a microprocessor, the events change. First, the logic design of the interfaces is made. In parallel, after some basic design decisions are made, the software effort can begin and the interfaces breadboarded. Since the interfaces are usually fairly simple, the number of errors are reduced and rework is held to a minimum. The board is then laid out and, like the breadboard, the opportunity for errors is reduced because the hardware is reduced.

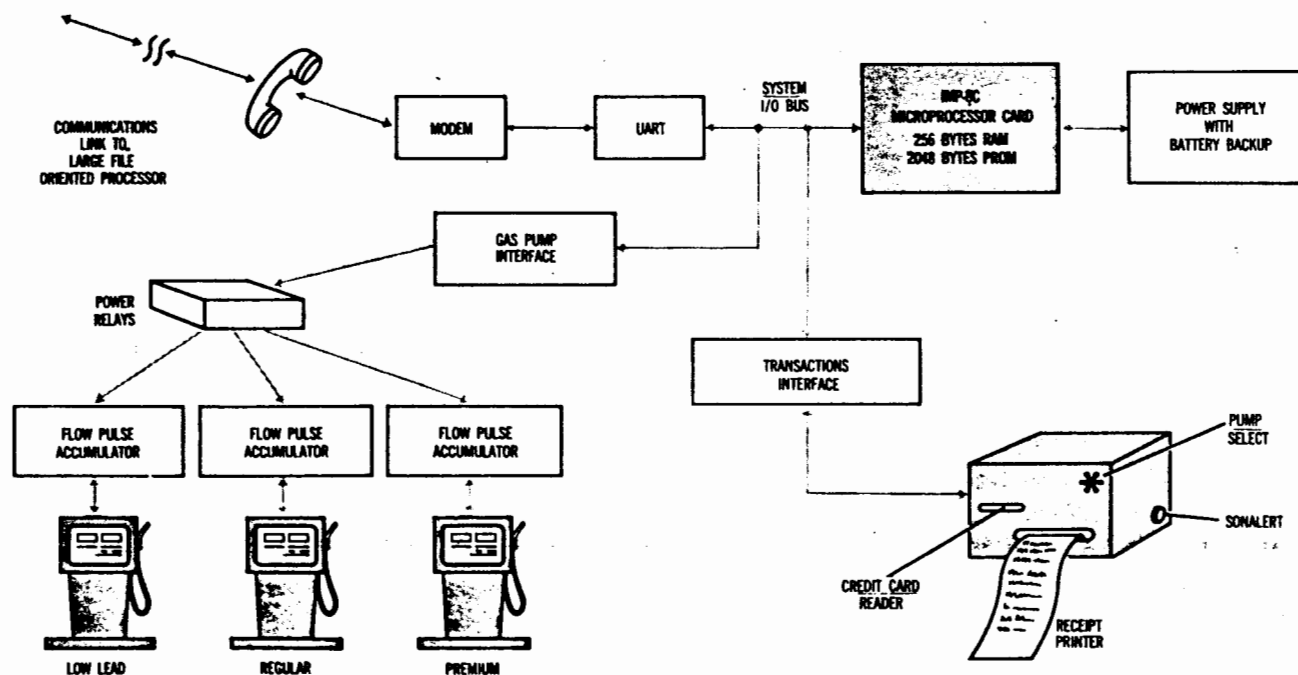


Fig. 6. In this automated gas station, one microprocessor can handle four islands of three pumps each. The customer presents his credit card, which is checked with the remote data base for validity, theft and in some cases, allotment. If anything is wrong, an alarm sounds; if not, the customer selects a pump and pumps the gas, the microprocessor calculates the bill and the printer presents a receipt.

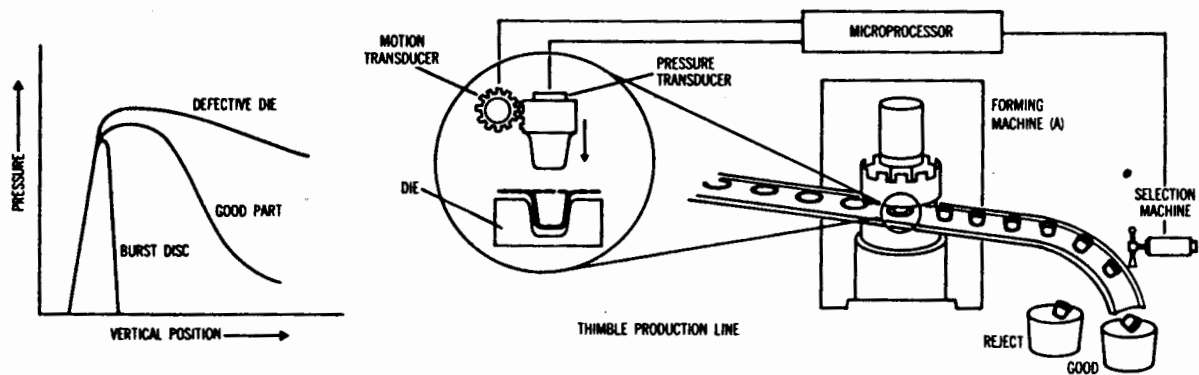


Fig. 7. This diagram shows how a microprocessor could be used in thimble production, illustrating a common measurement situation applicable in many other areas.

The program is tested, revised and retested. Finally, the mechanical design and systems test is performed. Any failure to meet specifications can probably be corrected by changes to the program. At this point the programmable system is ready to go to the field. Less time and less cost have been expended than in the hard-wired system, but even so, all the advantages have not been exploited.

When either system, hard-wired or programmable, is sold, the customer may ask for modifications or may even wish to connect in his 1903 widget. With the hard-wired system, the modification can be made if there are a couple of unused pins on the board, and if an extra board is required, there is room in the card cage. But with the programmable microprocessor version, a new interface board can be assembled and the program modified to effect the desired change quickly and at a fraction of the cost of modifying the hard-wired system.

Small quantities of systems are not economical because of the cost of developing software. However, when the quantity passes five or six or as the unit price passes \$10,000, a microprocessor should be considered. But what size (number of bits) and which manufacturer's processor should be used are key questions that must be taken into consideration.

Choosing the right microprocessor

The choice of which processor size to use must realistically start with what performance is needed and how much reserve you want for future growth (i.e., modifications, options, greater performance, etc.) If the choice is based on bit size and the support is equal, then the choice is much easier. Once you know what the system must do and how much time it has to do it, you can better determine if you need a 4 bit, 8 bit or 16-bit system. There is no easy way to classify one application as an 8 bit problem, and another as a 16 bit problem. Some typical matches include:

4 bit systems

- Man/machine interface (BCD)
 - Accounting systems
 - Terminals (simple)
 - Instrumentation
 - Calculators
 - Store and forward
- Non BCD type
 - Games
 - Replace random logic designs

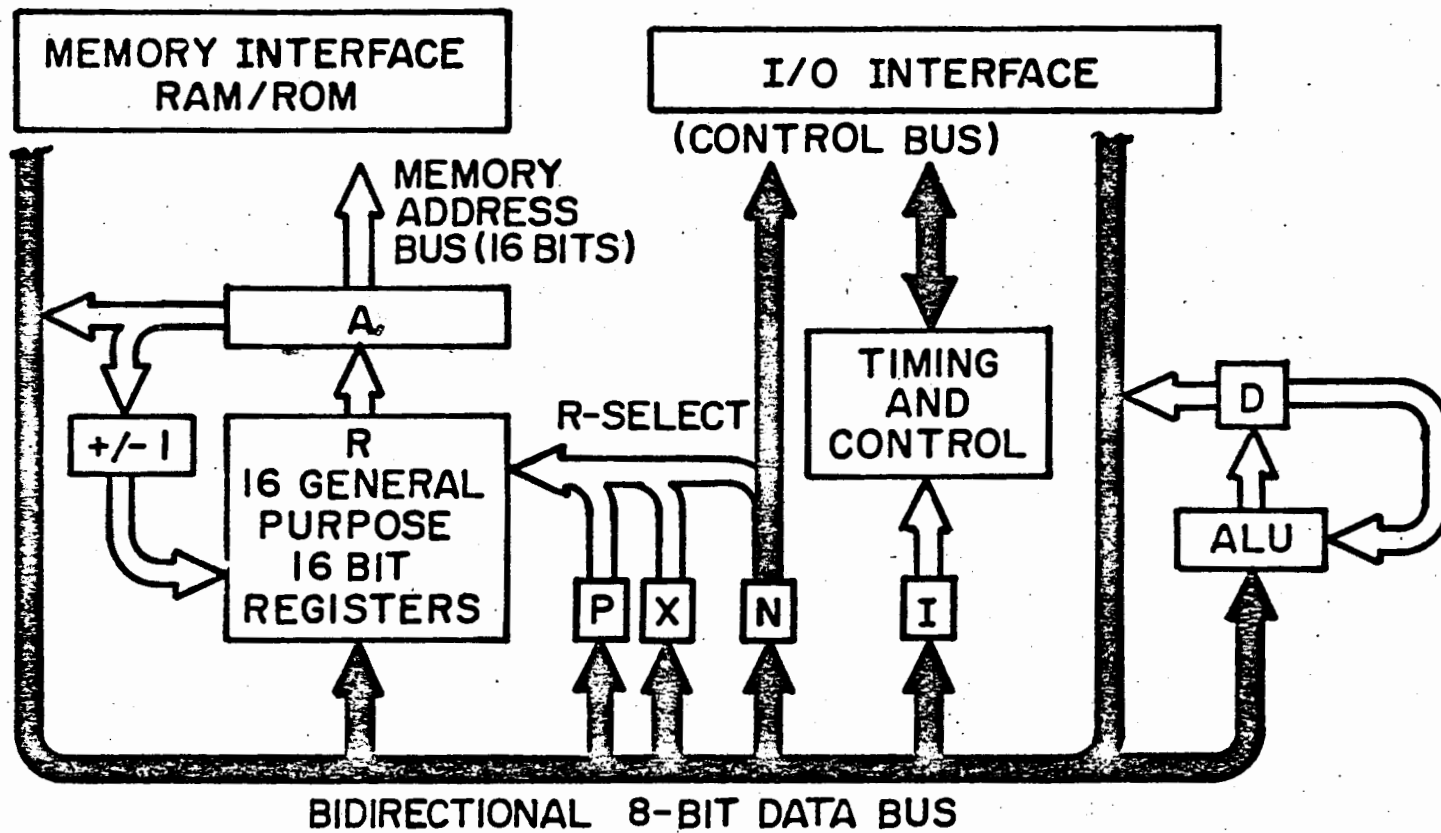
8 bit systems

- Traffic controllers
- Point-of-sale terminals
- Control systems
- Process control systems
- Smart terminals

16 bit systems

- Smart terminals
- Multiple intersection controllers
- Numerical control
- Process control
- Front end processor.

All the products mentioned in this article are being built right now — they are not pipe dreams. Arthur C. Clarke is probably the most successful prognosticator of the future in our time. In his book, *Profiles of the Future*, he says, "It is impossible to predict the future; all attempts to do so in any detail appear ludicrous within a very few years . . . One can only prepare for the unpredictable by trying to keep an open and unprejudiced mind." So it is with the computer on a chip and with the applications its existence will spawn. ☉



COSMAC MICROPROCESSOR ARCHITECTURE

COSMAC INSTRUCTIONS

- (6) • REGISTER OPERATIONS
- (1) • ALU OPERATIONS
- (2) • MEMORY REFERENCE (D REGISTER)
- (1) • BRANCHING
- (1) • INPUT / OUTPUT (PROGRAM MODE)
- (4) • CONTROL (EVERYTHING ELSE)

$I = 0, 1, 2, \dots, F$

$I = "C"$ NOT USED
IN 2-CHIP COSMAC.

WILL BE USED IN
1 CHIP VERSION.

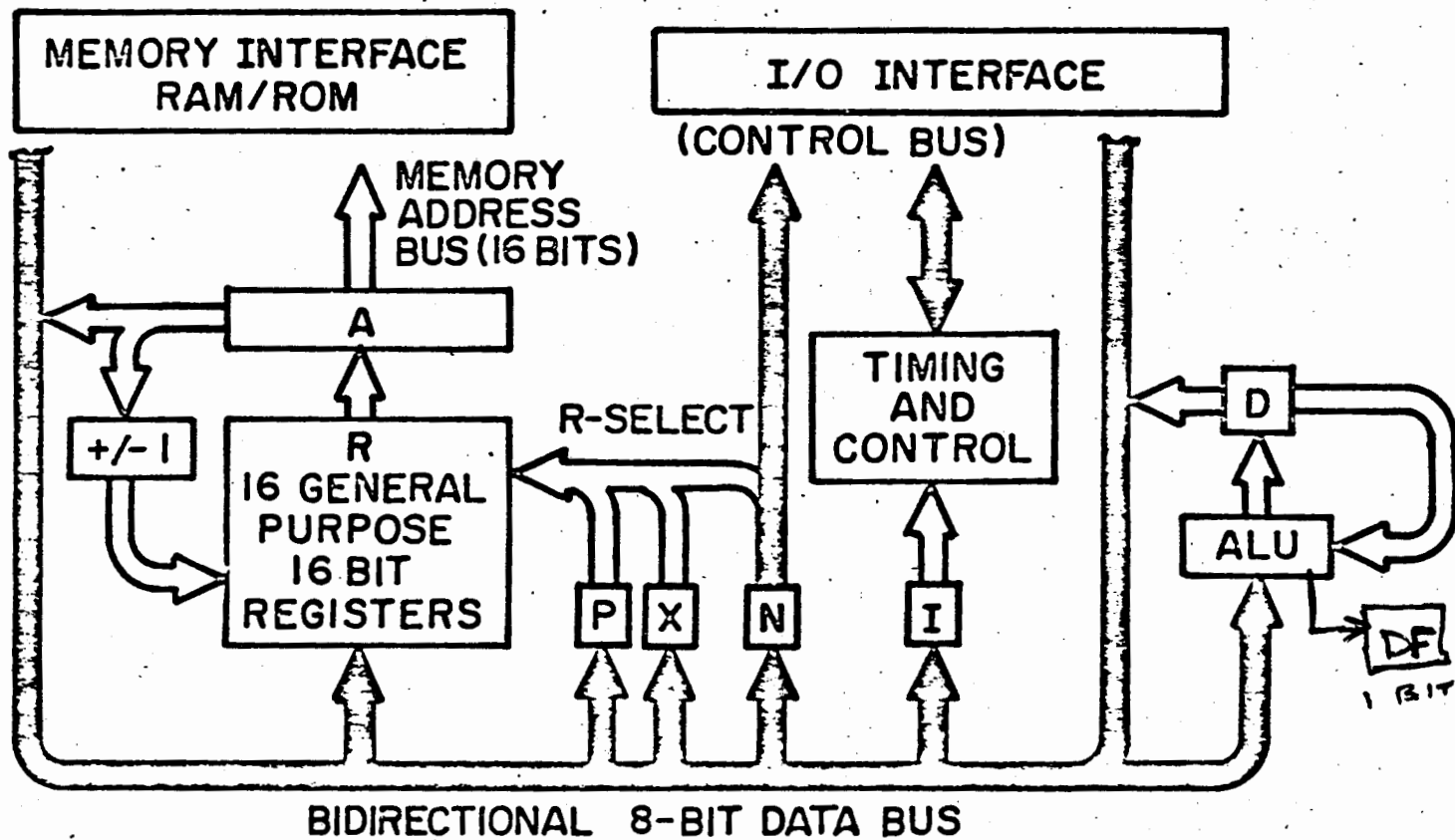
REGISTER

I N ← DEFINES REGISTER

1	N	R(N)+1	INC R(N)
2	N	R(N)-1	DEC R(N)
8	N	R(N).0 → D	GET LOW
9	N	R(N).1 → D	GET HIGH
A	N	D → R(N).0	PUT LOW
B	N	D → R(N).1	PUT HIGH

I = 1, 2, 8, 9, A, B

C I CODES USED



COSMAC MICROPROCESSOR ARCHITECTURE

ALU

I = F, ONLY 16 ALU INSTR.

$\begin{array}{c} \text{F} \quad \text{N} \\ \text{---} \quad \text{---} \\ \swarrow \quad \searrow \\ \text{ALU} \quad 0, 1, \dots, \text{F}, \text{ IDENTIFIES INSTR.} \end{array}$

E.G.

F3 M(Rx)) \oplus D \rightarrow D

M(Rx))

D

$\begin{array}{r} 11001011 \\ 01011110 \\ \hline \end{array}$

M(Rx)) \oplus D

$\begin{array}{r} 10010101 \end{array}$

USEFUL TO

TO

COMPARE

$A \oplus A = 0$

TO

COMPLEMENT

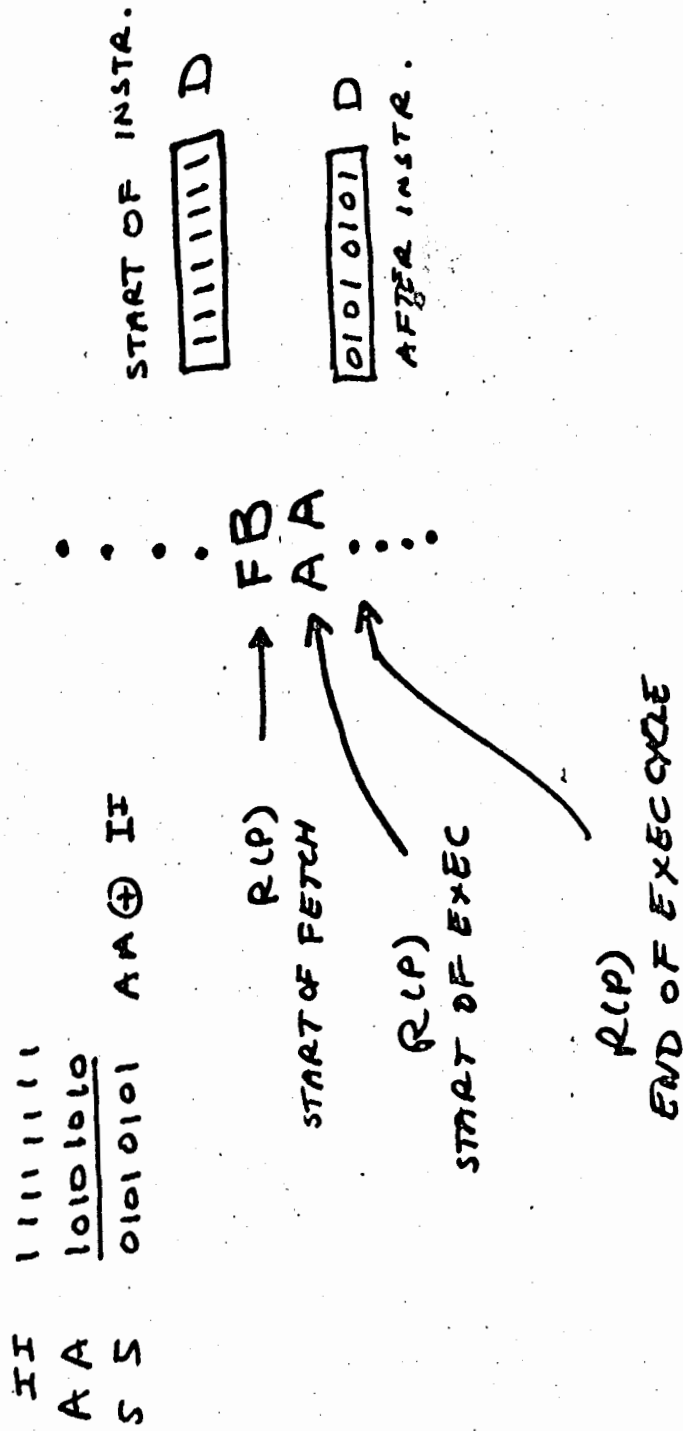
$A \oplus 11\dots 1 = \bar{A}$

$\begin{array}{r} 1010 \quad A \\ 1010 \quad A \\ \hline 0000 \quad 0 \end{array}$

$\begin{array}{r} 1010 \quad A \\ 1111 \quad 1 \\ \hline 0101 \quad \bar{A} \end{array}$

YOU ALSO HAVE "IMMEDIATE" INSTR.

EG. FB XRI $M(R(P)) \oplus D \rightarrow D, R(P) + 1 \rightarrow R(P)$



* IMMEDIATE INSTRUCTIONS ARE VERY USEFUL -
SINCE CAN HAVE DATA IN PROGRAM STREAM

ALU SUMMARY

F1	OR	$M(R(X)) \vee D \rightarrow D$
F9	ORI	$M(R(P)) \vee D \rightarrow D, R(P)+1 \rightarrow R(P)$
F2	AND	$M(R(X)) \cdot D \rightarrow D$
FA	ANI	$M(R(P)) \cdot D \rightarrow D, R(P)+1 \rightarrow R(P)$
F3	XOR	$M(R(X)) \oplus D \rightarrow D$
FB	XRI	$M(R(P)) \oplus D \rightarrow D, R(P)+1 \rightarrow R(P)$
F4	ADD	$M(R(X)) + D \rightarrow D, C \rightarrow DF$
FC	ADI	$M(R(P)) + D \rightarrow D, C \rightarrow DF, R(P)+1 \rightarrow R(P)$
F5	SD	$M(R(X)) - D \rightarrow D, C \rightarrow DF$
FD	SDI	$M(R(P)) - D \rightarrow D, C \rightarrow DF, R(P)+1 \rightarrow R(P)$
F6	SHR	SHIFT D RIGHT, $LSB \rightarrow DF, 0 \rightarrow MSB$ $10100011 \rightarrow \overset{\uparrow MSB}{0}1010001, DF=1$
F7	SM	$D - M(R(X)) \rightarrow D, C \rightarrow DF$
FF	SMI	$D - M(R(P)) \rightarrow D, C \rightarrow DF, R(P)+1 \rightarrow R(P)$
F0	LDX	$M(R(X)) \rightarrow D$
F8	LDI	$M(R(P)) \rightarrow D, R(P)+1 \rightarrow R(P)$

OR

SET BIT PATTERNS,

EG A 11011001

WANT "1"

A V 0000 0010 \Rightarrow 1101 1011 \uparrow

AND

TEST, RESET, EXTRACT

TEST 80.01 \Rightarrow 00

EXTRACT 63.0F \Rightarrow 03

RESET 83.7F \Rightarrow 03

TEST BIT

1000 0000
0000 0001

0000 0000

0110 0011
0000 1111

0000 0011

1000 0011
0111 1111

0000 0011

BIT
RESET

ADD

F 8
C 3
B B

1111 1000
1100 0011

"1" 1011 1011

CARRY

DF = 1

ADD A + A
⇒ 2 × A
⇒ LEFT SHIFT

0010
0010

0100 ⇒ 2 × 0010

RIGHT SHIFT • DIVIDE BY 2

08 0000 1000 } SHIFT RIGHT
04 0000 0100

• TEST BITS, SHIFT RIGHT, TEST DF

0000 0110	→ DF = 0
SHIFT 0000 0011	→ DF = 1
SHIFT 0000 0001	→ DF = 1
SHIFT 0000 0000	→
:	:

NOTE: MULTIPLY, DIVIDE DONE IN SOFTWARE!

SUBTRACT

$$M - D \Rightarrow M + 2's \text{ complement of } D$$

2's comp.

0001 0011 13

comp.

1110 1100

+1

0000 0001

1110 1101 ← 2's comp.

$$2C - 13 = 19$$

0010 1100

1110 1101

DF="1" 0001 1001 19

↑ RESULT IS POSITIVE

$$13 - 2C = -19$$

2C 0010 1100

2C 1101 0011

+1 0000 0001

11010100

0001 0011

1101 0100

1110 0111 E7

DF="0"

↑

RESULT

IS NEG.

TO RECOVER MAGNITUDE,
COMPLEMENT & ADD 1

0001 1000

1

0001 1001

19

THUS, $M-D \neq D-M$
(NOT COMMUTATIVE).

CAN USE SUBTRACT TO TEST $A > B$, $A = B$

$A - B = C$, DF IF $DF = "0"$, $B > A$

IF $DF = "1"$, $A > B$

USE XOR TO TEST $A = B$!

MEM. REF.

(2 I CODES)

4 N LDA $M(R(N)) \rightarrow D, R(N)+1 \rightarrow R(N)$

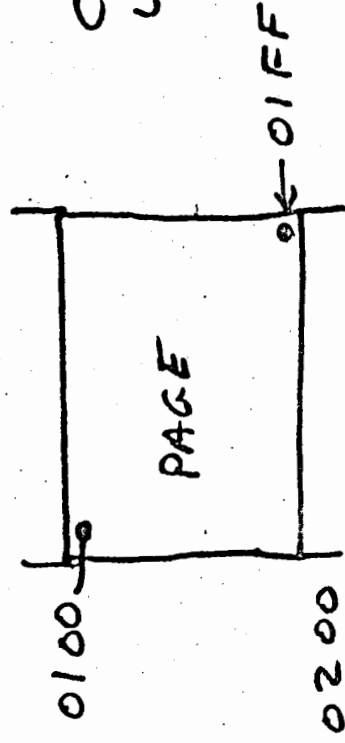
5 N STR $D \rightarrow M(R(N))$

BRANCH

3 N (ONE I CODE)
XX DURING EXEC CYCLE

IF TRUE, $XX \rightarrow R(P).0$, LOW ORDER
HALF OF $R(P)$

IF FALSE, $R(P)+1 \Rightarrow R(P)$



CAN ONLY BRANCH
WITHIN PAGE

E.G. 32 BRANCH IF D=00

IF D=00

RCP).0	0A	1st unit.
	0E	2nd unit
	...	

0A	32
0B	0E
0C	30
0D	02
0E	41
...	...

IF D ≠ 00

RCP).0	0A	1st unit.
	0C	2nd unit.

30 UNCONDITIONAL

32	IF D=0	3A	IF D \neq 0
33	IF DF=1	3B	IF DF=0
34	IF EF1=1	3C	IF EF1=0
35	IF EF2=1	3D	IF EF2=0
36	IF EF3=1	3E	IF EF3=0
37	IF EF4=1	3F	IF EF4=0
38	SKIP (2 BYTE NO OP)		

ONE CHIP WILL HAVE LONG BRANCH !

PROGRAMMED I/O INSTRUCTIONS

ONE I CODE (I=6)

N = 0xxx	OUTPUT	(8)
N = 1xxx	INPUT	(8)

6N
I/O

INSTR. N AVAILABLE TO INTERFACES WHEN I=6
INSTRUCTION EXECUTED

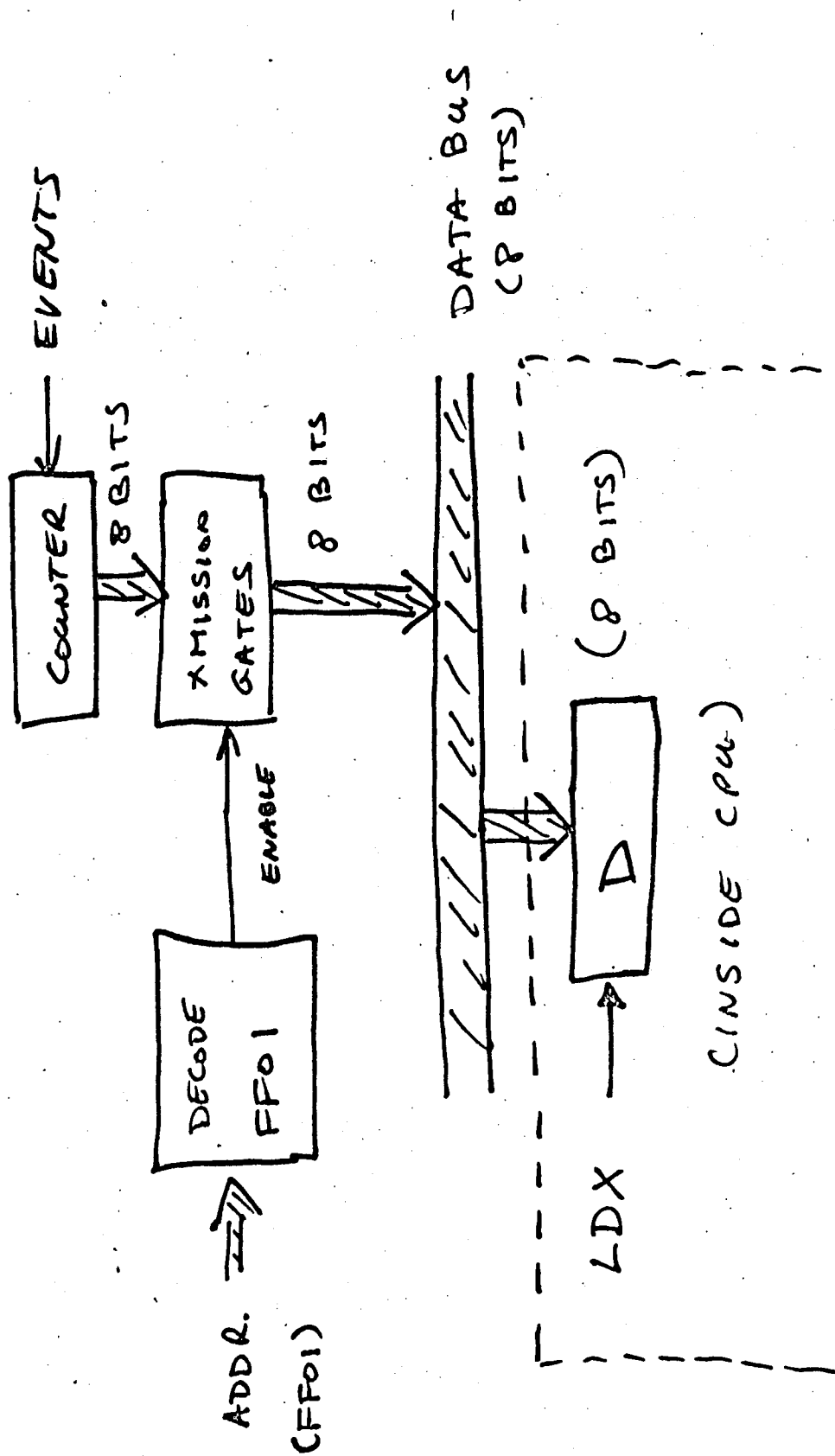
BY USING TWO LEVEL I/O, # I/O INSTR.
GREATLY INCREASED.

CAN COMMUNICATE WITH OUTSIDE WORLD
VIA

- PROGRAMMED MODE I/O (I=6)
- DIRECT MEMORY ACCESS (DMA)
- MAKING I/O DEVICE LOOK LIKE
MM LOCATION - USE ANY MM
ORIENTED INSTRUCTIONS

EG. MM LOCATION FFO1 (NOT PART OF MAIN
MEMORY) CAN BE AN 8 BIT COUNTER.

LDX (M(RX)) → D) CAN BE USED TO
MOVE 8-BIT COUNT INTO D REGISTER



OTHER

00 IDLE

DN $N \rightarrow P$ (CHANGE PROGRAM COUNTER)

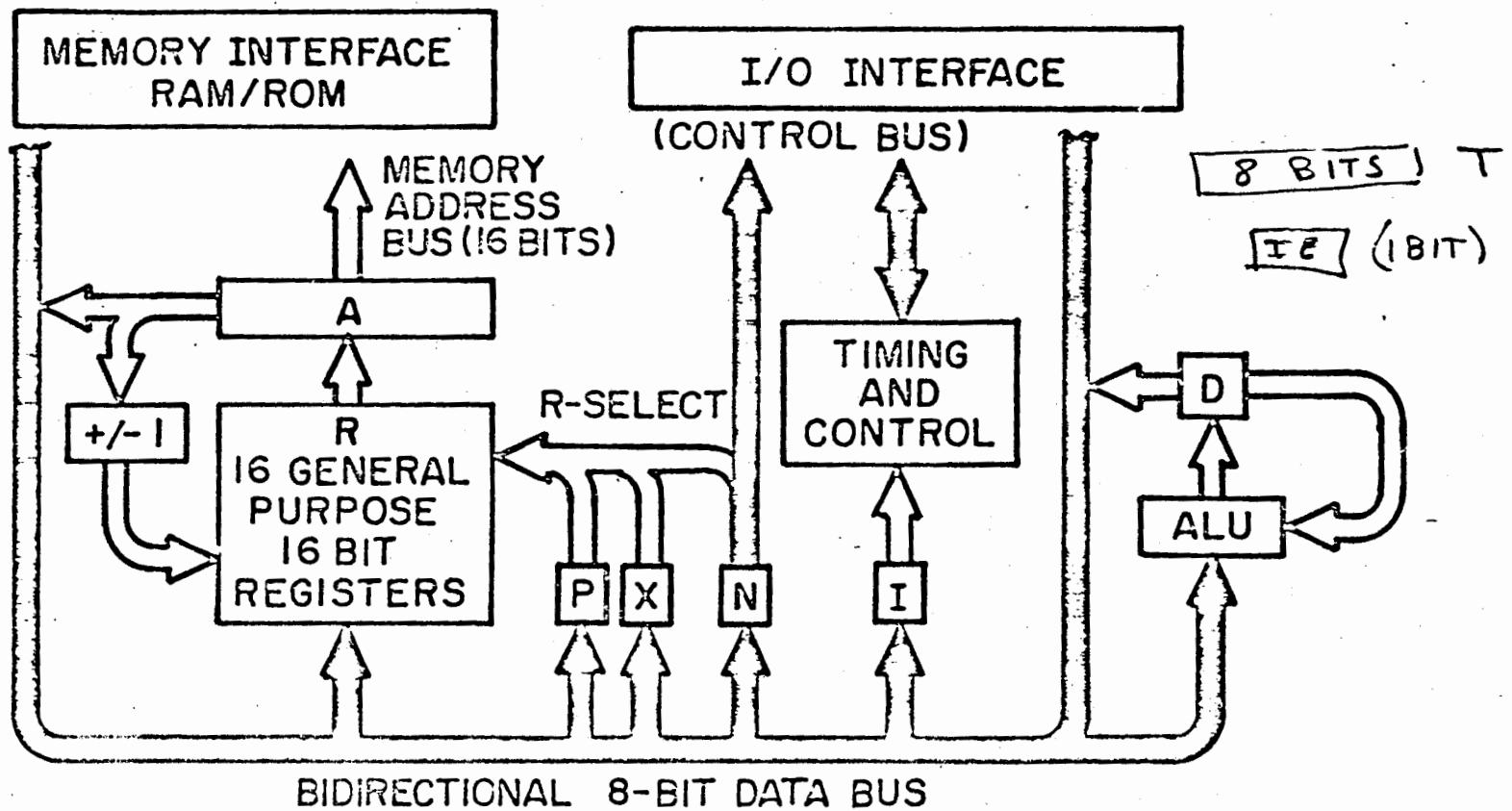
EN $N \rightarrow X$ (CHANGE DATA POINTER)

INTERUPT RELATED

70 RETURN ($M(R(X)) \rightarrow X, P; I \rightarrow IE, R(X)+1 \rightarrow R(X)$)

71 DISABLE ($M(R(X)) \rightarrow X, P; R(X)+1 \rightarrow R(X), 0 \rightarrow IE$)

78 SAVE ($T \rightarrow M(R(X))$)



COSMAC MICROPROCESSOR ARCHITECTURE

INTERRUPTS

CPU PROCESSING -- FETCH, EXEC, FETCH, EXEC,

EXTERNAL EVENT OCCURS WHICH DEMANDS
IMMEDIATE CPU ACTION.

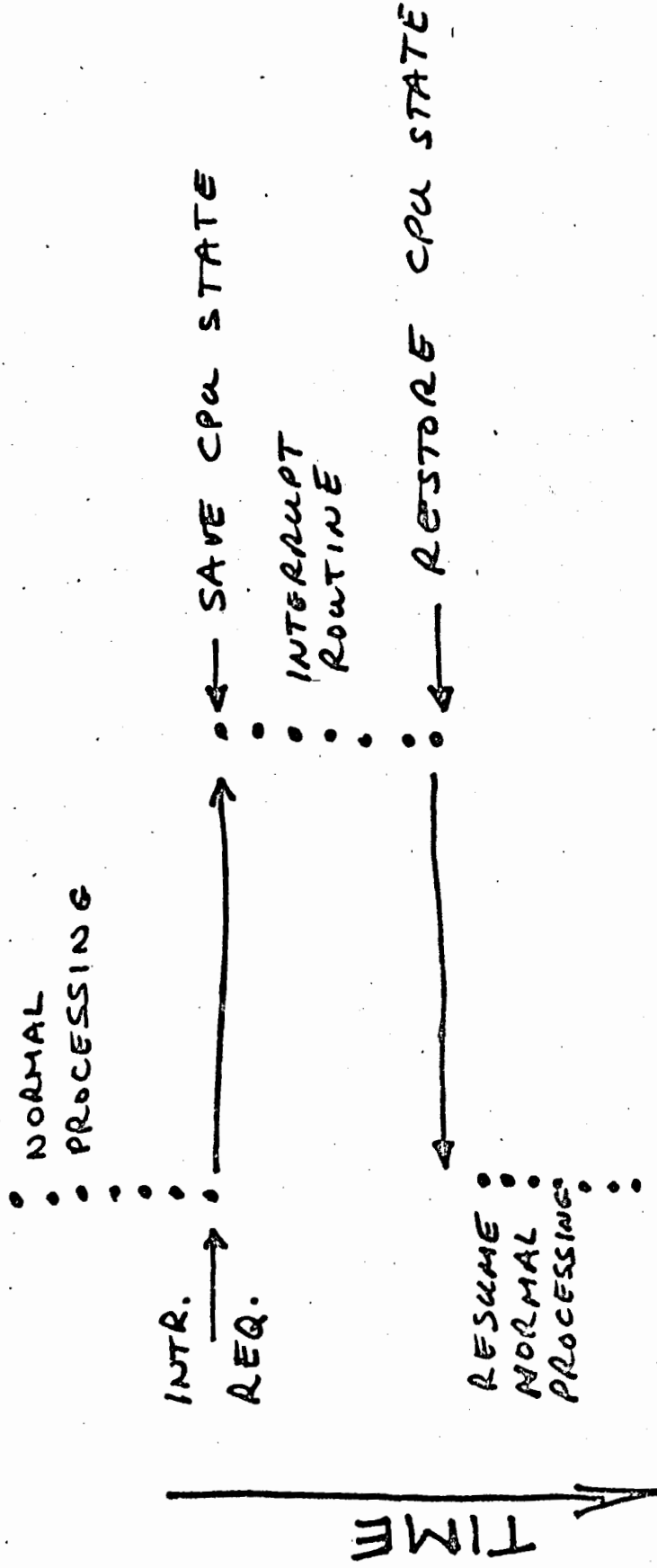
INTERUPT FACILITY PERMITS HANDLING OF
ABOVE SITUATION.

USEFUL IN : * REAL TIME APPLICATIONS

* RAPID TRANSFER TO SPECIAL
PORTIONS OF PROGRAM

BUT, ADDS COMPLEXITY TO SOFTWARE

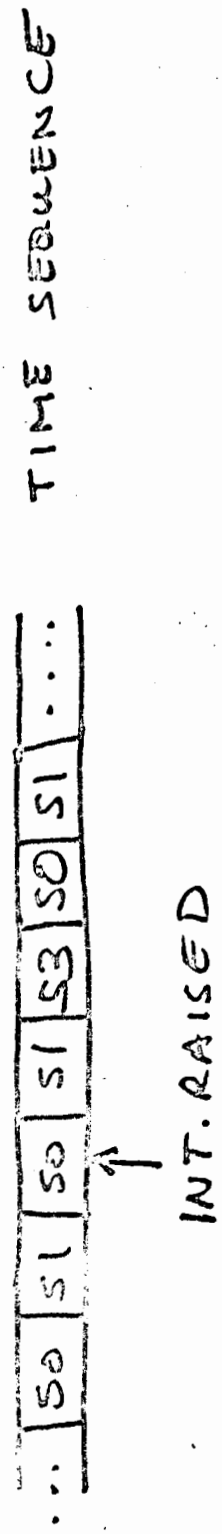
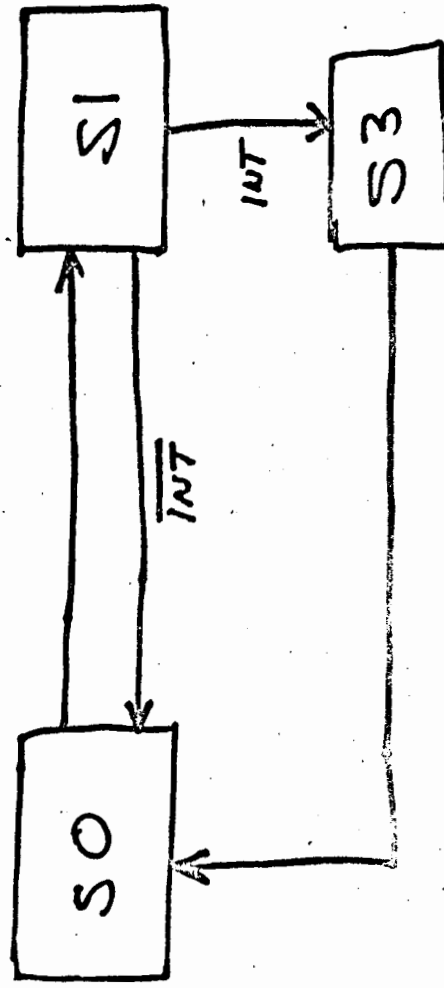
INTERUPT MECHANICS



COSMAC INTERRUPT FACILITY

- * SPECIAL MACHINE CYCLE S_3 (INTERRUPT CYCLE)
- * SPECIAL REGISTER TO SAVE P, X (T, 8 BITS)
(SOFTWARE MUST SAVE EVERYTHING ELSE).
- * R_1 BECOMES PC ($P=1$)
 R_2 BECOMES DATA POINTER ($X=2$) } AUTOMATIC
- * IE (INTERRUPT MASK) BECOMES 0, THUS
FURTHER INTERRUPTS ARE DISABLED UNTIL
NORMAL PROCESSING RESUMES, WHEN $IE = 1$.

STATE DIAGRAM WITH INTERRUPT



21

DURING S3 CYCLE

$X, P \rightarrow T, \quad 1 \rightarrow P, \quad 2 \rightarrow X, \quad 0 \rightarrow IE$

IN INTERRUPT ROUTINE

78 (SAVE) $T \rightarrow MCR(2)$

(ASSUME R2 POINTS AT AVAILABLE LOCATION)

BEFORE EXIT FROM INTERRUPT ROUTINE

70
(RETURN)

$T \rightarrow X, P, \quad R(X)+1 \rightarrow R(X)$
 $1 \rightarrow IE$

IF NO INTERRUPT FACILITY,

- SAMPLE STATUS USING SOME SPECIFIC GN INSTR.

- SAMPLE EXTERNAL FLAG

⇒ NOT EFFICIENT, RESPONSE MAY NOT BE FAST ENOUGH

DIRECT MEMORY ACCESS (DMA)

* HIGH SPEED DATA TRANSFER BETWEEN

RAM & OUTSIDE WORLD

* NO CPU INSTR. EXECUTION REQUIRED

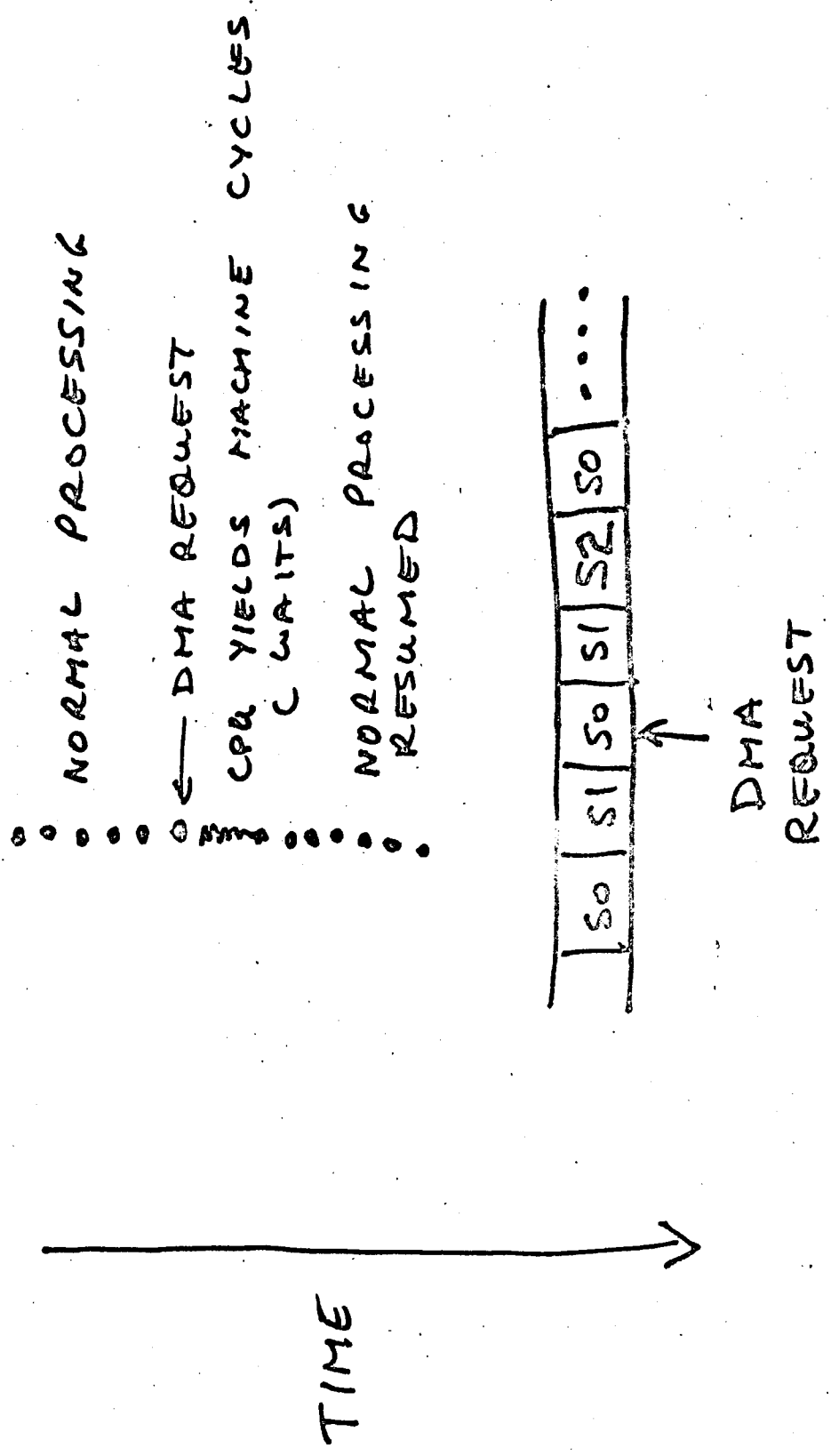
* USEFUL FOR • CRT (TV) REFRESH,

• PROGRAM LOAD

• DISC DATA TRANSFER

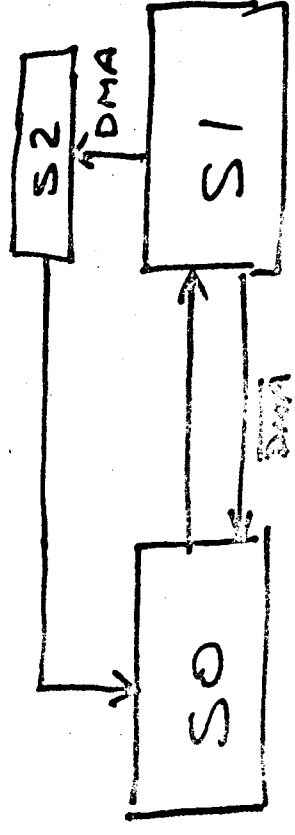
ETC.

DMA MECHANICS

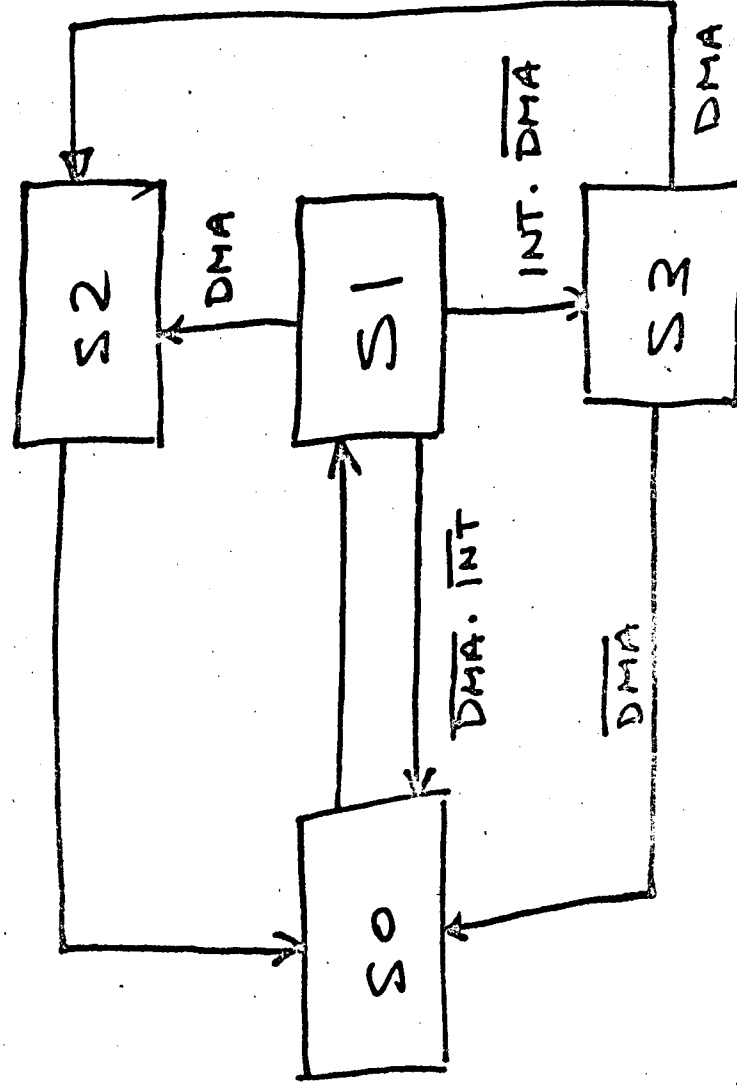


COSMAC'S DMA

- * USE R0 AS MEM. POINTER (AUTOMATIC)
- * R0 INCREMENTED AT END OF EACH S2 CYCLE
- * S2 MACHING STATE
- * DMA REQUESTS CAN BE ASYNCHRONOUS

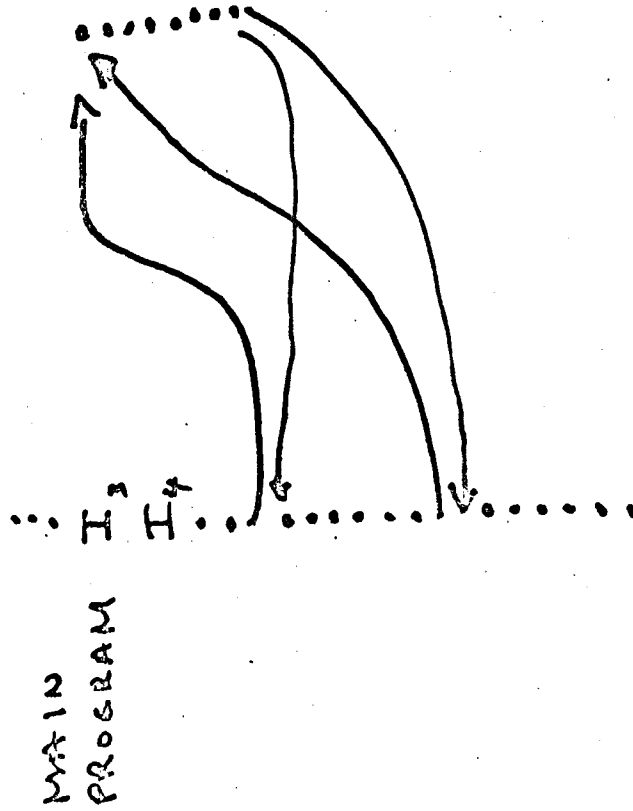


ALL STATE COSMAC DIAGRAM



MORE DETAIL DUE TO "LOAD", "CLEAR", "FOLD",
SEE COSMAC MANUAL.

BASIC SUBROUTINE



SUBROUTINE : CODE
THAT IS COMMON TO
MANY PORTIONS OF
MAIN PROGRAM

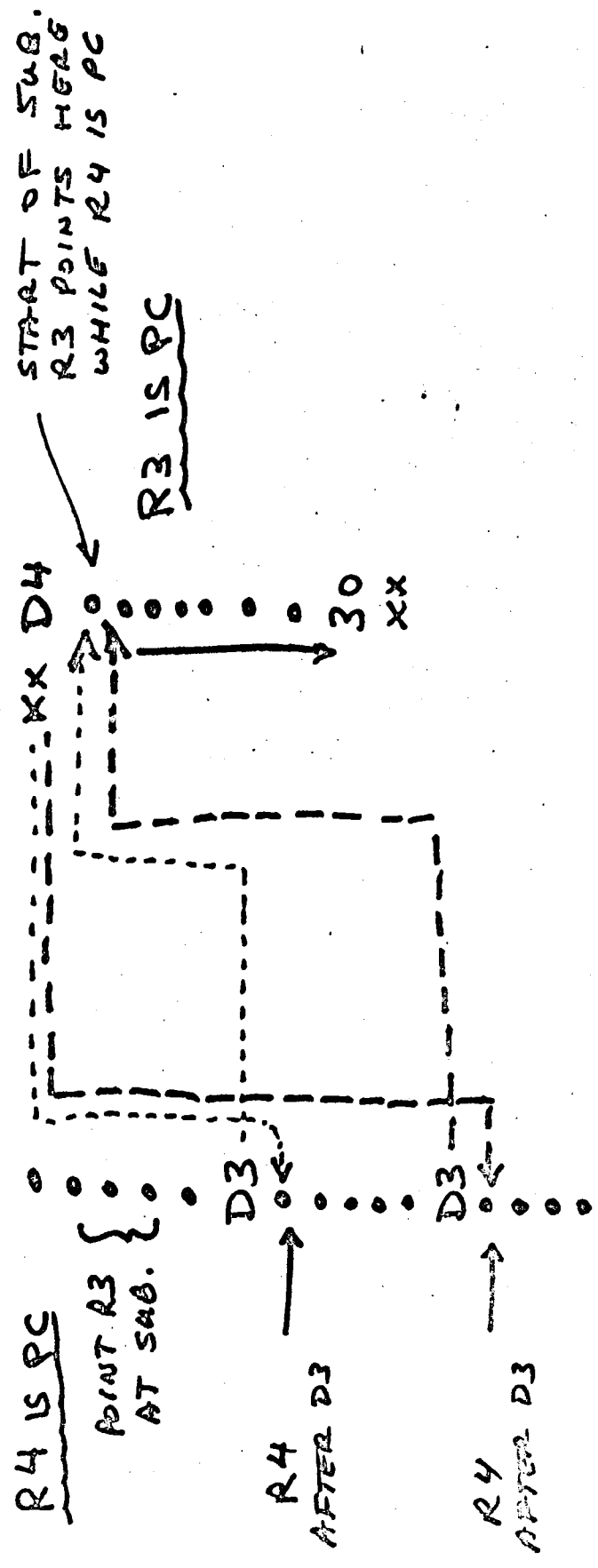
CAN USE DN INSTR.
(N → P)

POINT RN) AT SUBROUTINE,

CHANGE PC VIA D INSTR.

EXAMPLE

CALLING PROGRAM



OVERVIEW OF ASSEMBLY LANGUAGE

* USE MNEMONICS INSTEAD OF MACHINE CODE,
E.G. ADD INSTEAD OF F4

* CAN NAME LOCATIONS IN MEMORY
(NO ABSOLUTE ADDRESSES)

* EASIER TO UNDERSTAND PROGRAM

- FEWER ERRORS
- EASIER TO MODIFY

EXAMPLE:

COUNTER PROGRAM

1ST LEVEL

2nd LEVEL

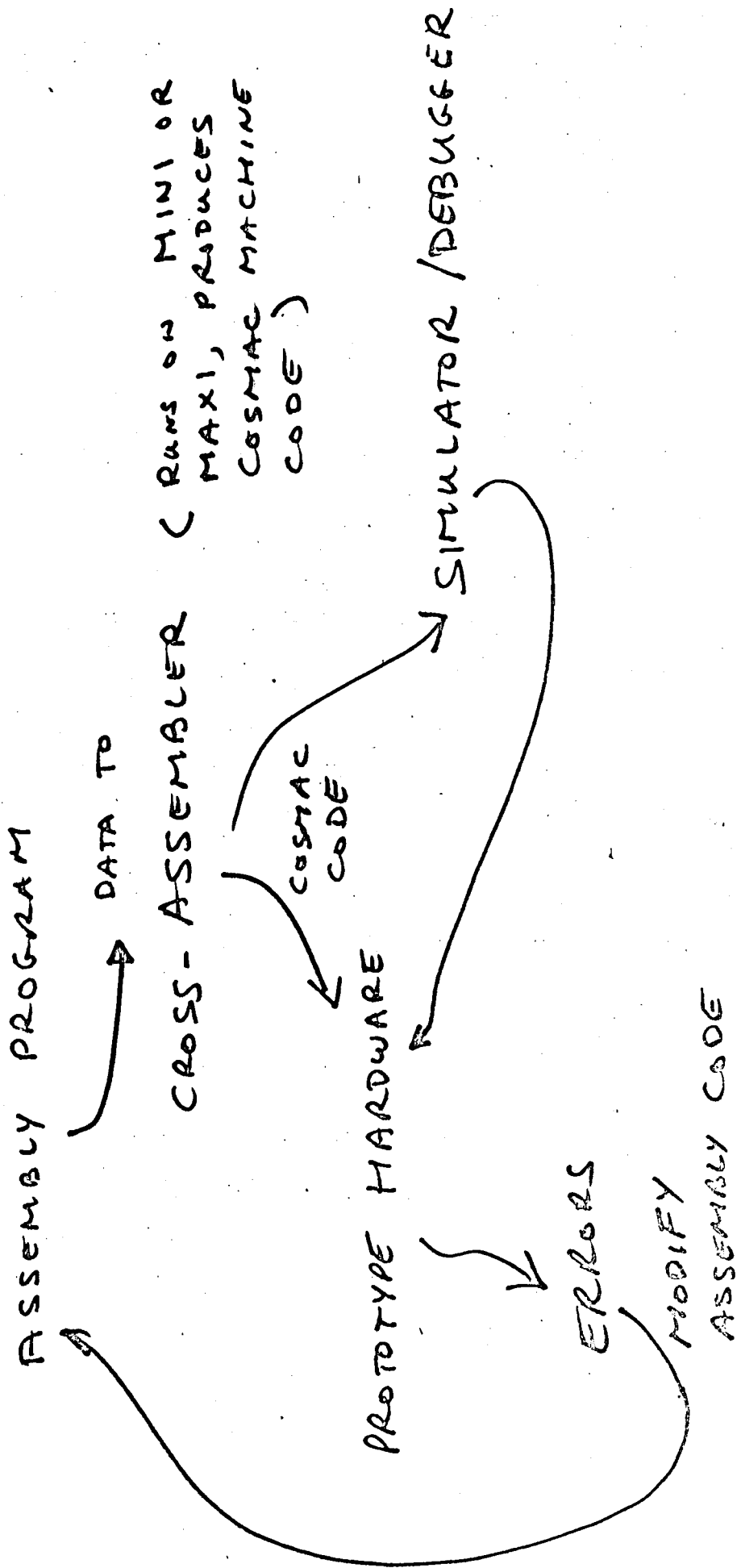
```
00      ,#00
F8      ,LDI
00      ,#00
A3      PLO R3
53      START: STR R3
E3      SEX R3
60      OUT 0
23      DEC R3
3F      WAIT: BN4 WAIT
08      ---
FC      ADI
01      ,#01
30      BR  START
04      ---
```

ABSOLUTE

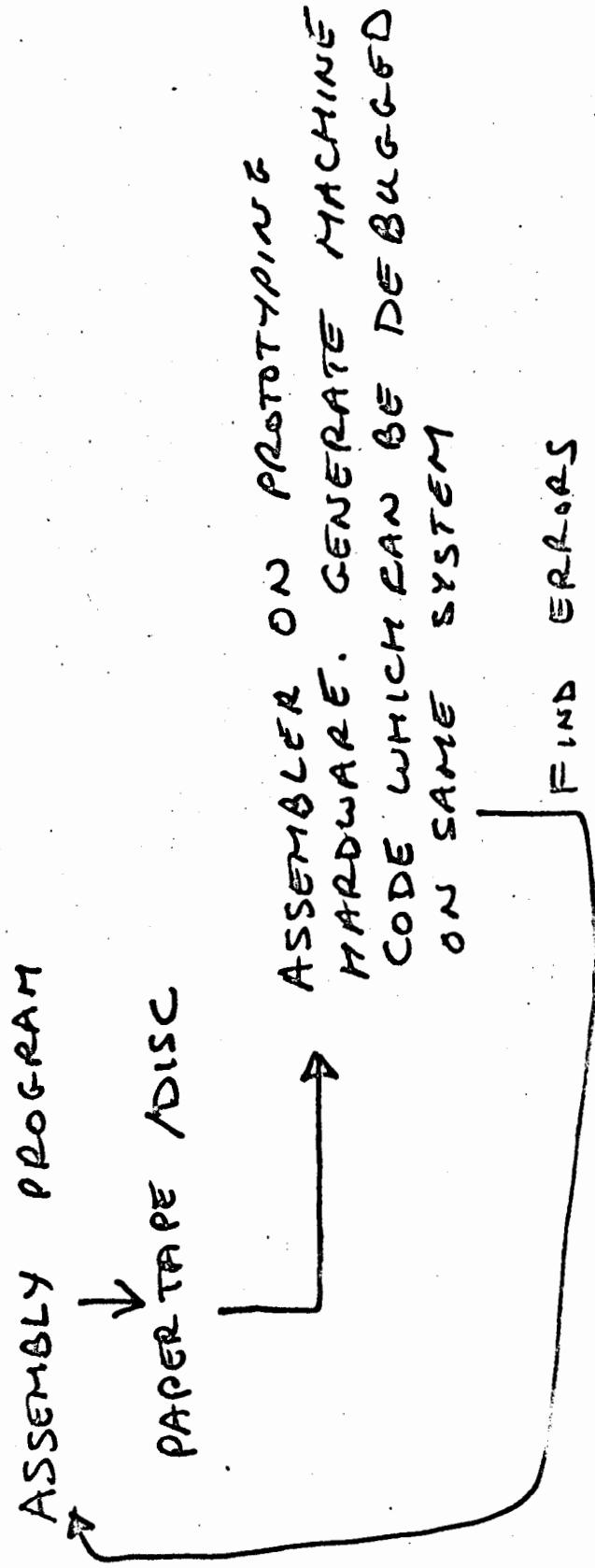
PUT IN
BY ASSEMBLER,
SO RELATIVE

```
ORG #1
#00 → R3.0
START: STR R3
SEX R3
OUT 0
DEC R3
WAIT: BN4 WAIT
ADI #01
BR START
```

TIME SHARING



STAND ALONE SYSTEM



* FOR SMALL SYSTEMS, DON'T REALLY NEED ASSEMBLY LANGUAGE. IT IS ONLY AN AID!

COMPARING MICROPROCESSORS

- # SUPPLY VOLTAGES
- POWER CONSUMPTION
- TECHNOLOGY $\left\{ \begin{array}{l} \text{NOISE IMMUNITY} \\ \text{SIGNAL LEVELS} \end{array} \right.$
- MEMORY INTERFACE
- I/O INTERFACE
- INSTRUCTIONS
- SPEED
- CLOCKS
- STATIC / DYNAMIC
- BENCHMARKS (SPEED / STORAGE TRADE-OFFS)

CLOCKS, SPEED, & STATIC/DYNAMIC

USEC/INSTR.

CLOCKS

STATIC

COSMAC

2 CHIP @ 12V

8 USEC

1 CHIP @ 5V

2 USEC

1 CHIP @ 5V

1 USEC

}?

YES

INTEL 8080

2-10

NORMAL

2

NO

1.2-5 FASTER (NEW)

MOTOROLA 6800

2-10

NORMAL

2

NO

SUPPLY VOLTAGES

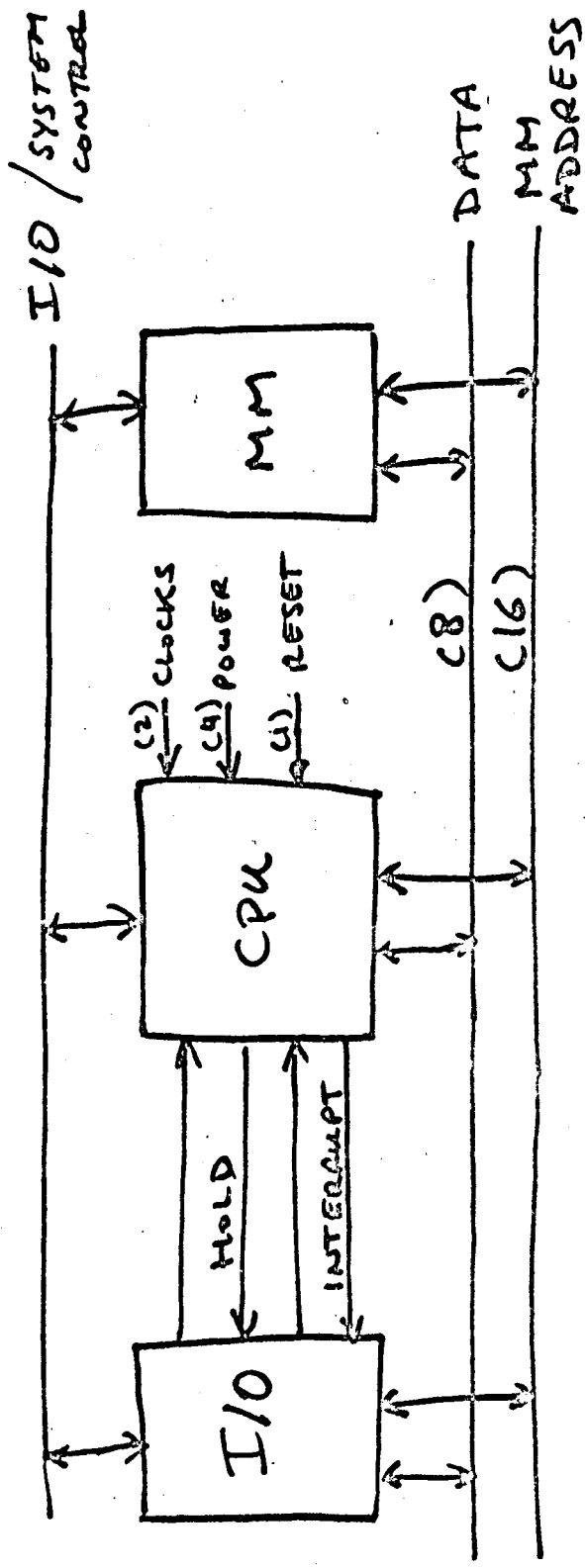
COSMAC	1 LEVEL, 3.5-15V	CMOS	BETTER NOISE LOWER POWER MORE EXP.
INTEL 8080	3 LEVELS, ± 5 , 12V	NMOS	HIGHER DENSITY
MOTOROLA 6800	1 LEVEL, +5V	NMOS	

MEMORY INTERFACE / DATA INTERFACE

	# MM ADDRESS PINS	# DATA PINS
COSMAC	8	8
MOTOROLA 6800	16	8
INTEL 8080	16	8

(MULTIPLEXED)

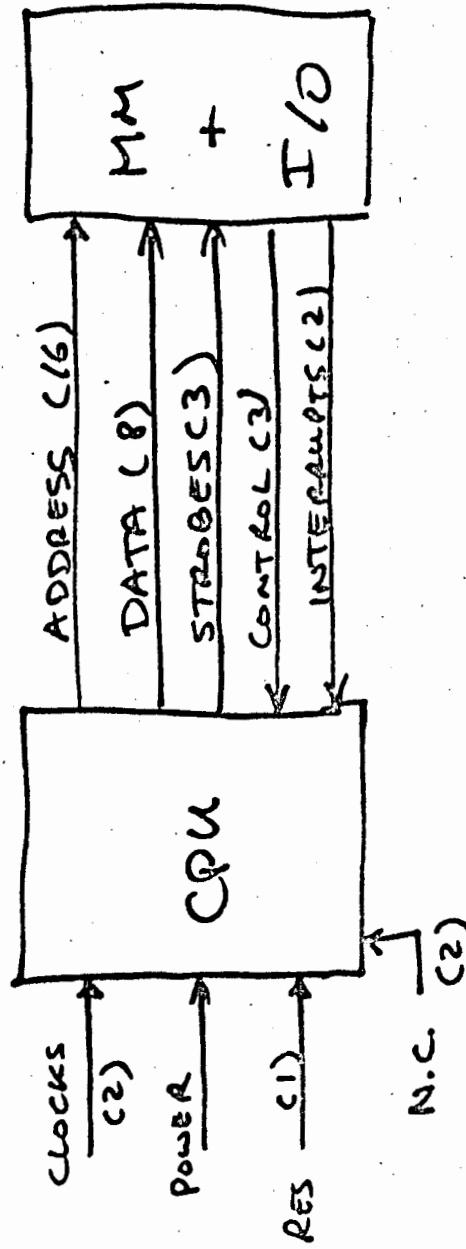
INTEL 8080



PINS

- 16 - ADDRESS
- 8 - DATA
- 5 - I/O / SYSTEM CONTROL
- 4 - POWER
- 2 - CLOCKS
- 1 - RESET
- 2 - HOLD
- 2 - INTERCEPT

MOTOROLA 6800



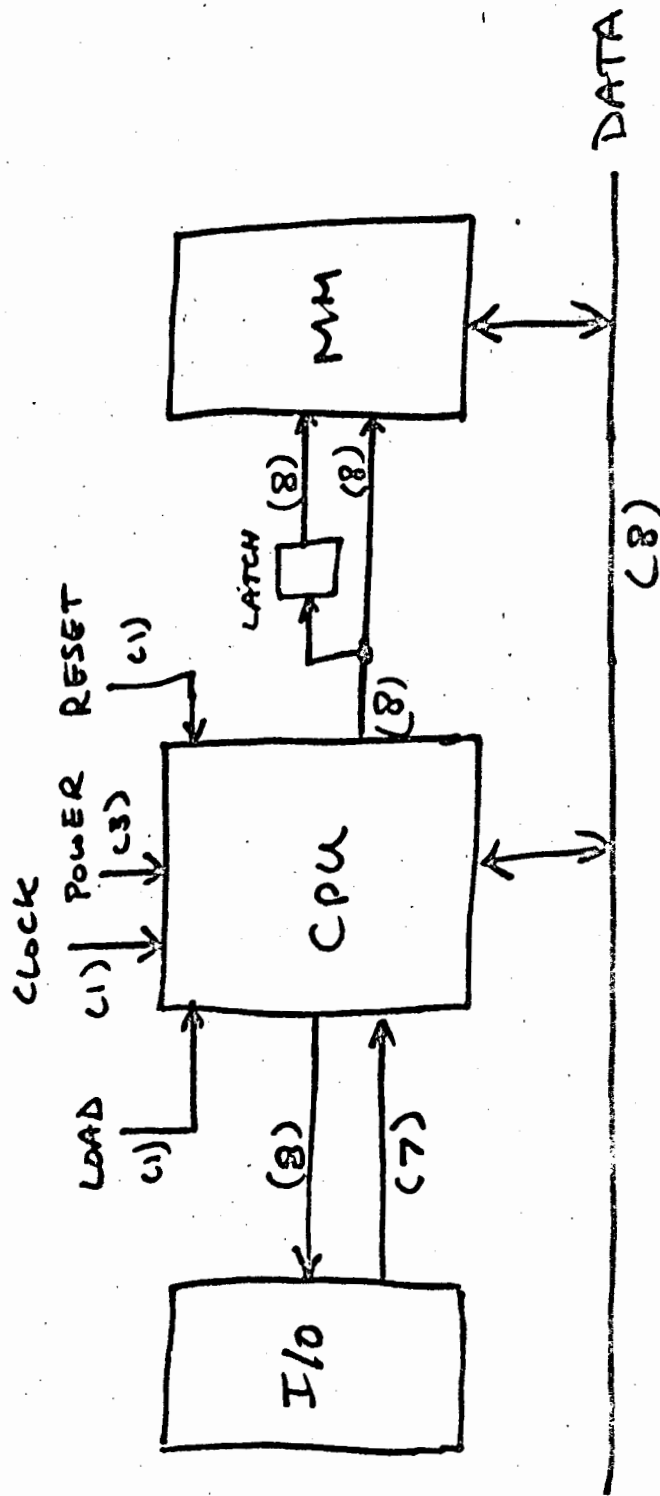
PINS

- 16 - ADDRESS
- 8 - DATA
- 3 - POWER
- 2 - CLOCKS
- 1 - RESET
- 6 - CONTROL + STROBES
- 2 - INTERRUPTS

(2 PINS UNUSED)

FUTURE ?

COSMAC



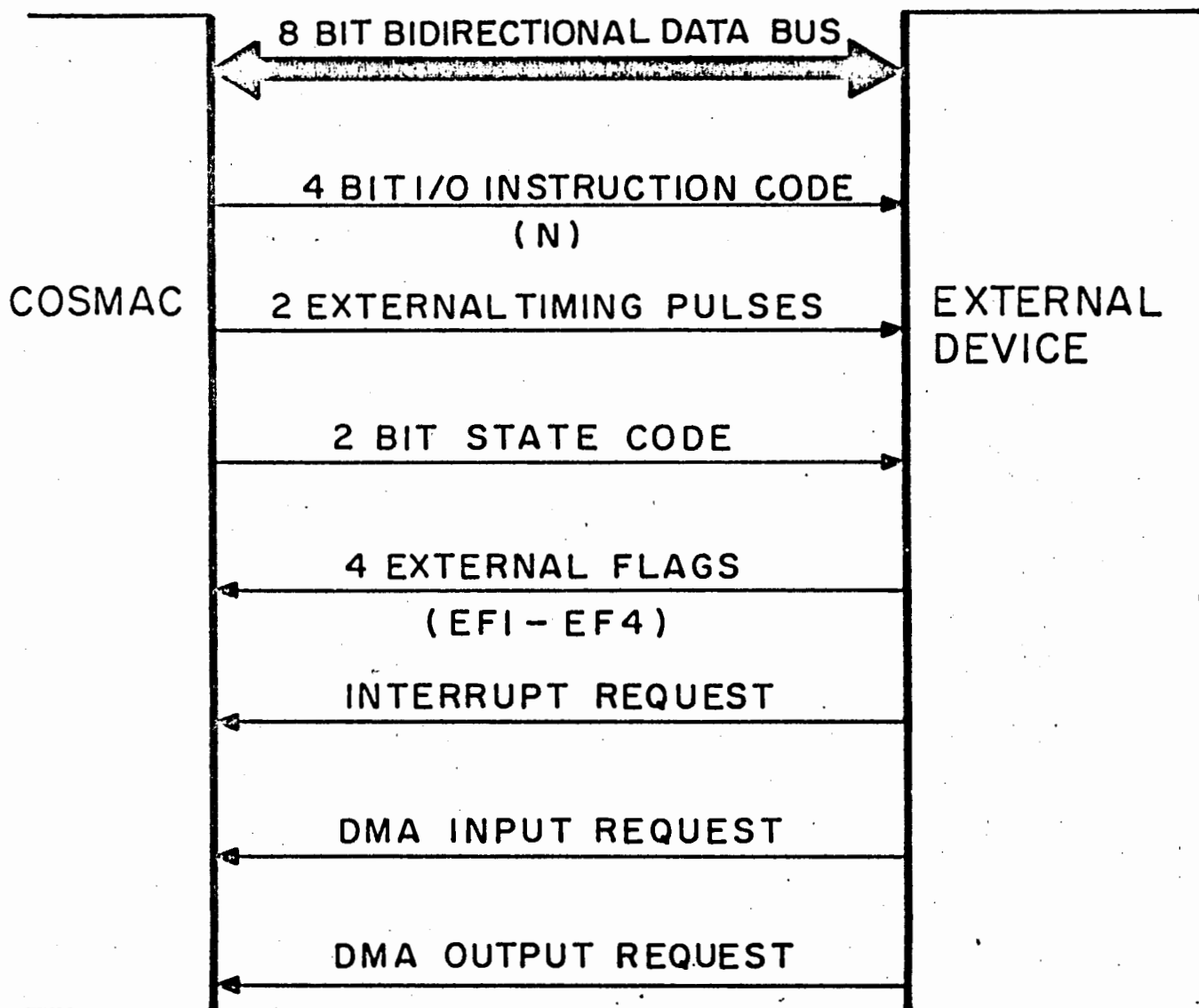
PINS

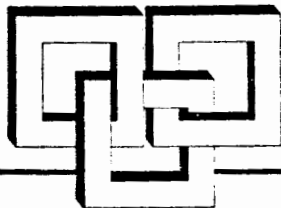
- 8 - ADDRESS
- 8 - DATA
- 15 - E/O
- 1 - CLOCK
- 3 - POWER (LEVEL CONVERSION)
- 1 - LOAD
- 1 - RESET
- 2 - M6/MR

1-CHIP WILL USE
ALL 40 PINS

MUST ALSO CONSIDER

- STACK FACILITY — HARDWARE OR SOFTWARE
- INSTRUCTION SET — MEM. UTILIZATION
— USEFUL TO APPLICATION
- # REGISTERS — COSMAC, 16-16 BIT + 1 8-BIT ACC.
— INTEL, 5-16 BIT + 1 8-BIT ACC.
— MOTOROLA, 3-16 BIT + 2 8-BIT ACC.





RCA Microprocessor Products

COSMAC Dictionary

Access Time: Time between the instant that an address is sent to a memory and the instant that data returns. Since the access time to different locations (addresses) of the memory may be different, the access time specified in a memory device is the path which takes the longest time.

Accumulator: Register and related circuitry which holds one operand for arithmetic and logical operations.

Additional Hardware: Microprocessor chips differ in number of additional ICs required to implement a functioning computer. Generally, timing, I/O control, buffering, and interrupt control require external components.

Address: A number used by the CPU to specify a location in memory.

Addressing Modes: See Memory Addressing Modes

ALU: Arithmetic-Logic Unit. That part of a CPU which executes adds, subtracts, shifts, AND's, OR's, etc.

Architecture: Organizational structure of a computing system, mainly referring to the CPU or microprocessor.

Assembler: Software that converts an assembly-language program into machine language. The assembler assigns locations in storage to successive instructions and replaces symbolic addresses by machine language equivalents. If the assembler runs on a computer other than that for which it creates the machine language, it is a **Cross-Assembler**.

Assembly Language: An English-like programming language which saves the programmer the trouble of remembering the bit patterns in each instruction; also relieves him of the necessity to keep track of locations of data and instructions in his program.

The assembler operates on a "one-for-one" basis in that each phrase of the language translates directly into a specific machine-language word, as contrasted with **High Level Language**.

Assembly Listing: A printed listing made by the assembler to document an assembly. It shows, line for line, how the assembler interpreted the assembly language program.

Asynchronous Operation: Circuit operation without reliance upon a common timing source. Each circuit operation is terminated (and next operation initiated) by a return signal from the destination denoting completion of an operation. (Contrast with **Synchronous Operation**).

Baud: A communications measure of serial data transmission rate; loosely, bits per second but includes character-framing START and STOP bits.

Benchmark Program: A sample program used to evaluate and compare computers. In general, two computers will not use the same number of instructions, memory words, or cycles to solve the same problem.

Bit: An abbreviation of "binary digit". (Single characters in a binary number.)

Bootstrap (Bootstrap Loader): Technique or device for loading first instructions (usually only a few words) of a routine into memory; then using these instructions to bring in the rest of the routine.

The bootstrap loader is usually entered manually or by pressing a special console key. COSMAC does not need one. See **Load Facility**.

Branch: See **Jump**.

Branch Instruction: A decision-making instruction which, on appropriate condition, forces a new address into the program counter. The conditions may be zero result, overflow on add, an external flag raised, etc. One of two alternate program segments in the memory are chosen, depending on the results obtained.

Breakpoint: A location specified by the user at which program execution (real or simulated) is to terminate. Used to aid in locating program errors.

Bus: A group of wires which allow memory, CPU, and I/O devices to exchange words.

Byte: A sequence of n bits operated upon as a unit is called an n-bit byte. The most frequent byte size is 8 bits.

Call Routine: See **Subroutine**

Clock: A device that sends out timing pulses to synchronize the actions of the computer.

Compiler: Software to convert a program in a high-level language such as FORTAN into an assembly language or machine language program.

Cross Assembler: A symbolic language translator that runs on one type of computer to produce machine code for another type of computer. See **Assembler**.

CPU (Central Processing Unit): That part of a computer system that controls the interpretation and execution of instructions. In general, the CPU contains the following elements:

- Arithmetic-Logic Unit (ALU)
- Timing and Control
- Accumulator
- Scratch-pad memory
- Program counter and address stack

Instruction register and decode
Parallel data and I/O bus
Memory and I/O control

Cycle Stealing: A memory cycle stolen from the normal CPU operation for a DMA operation. See DMA.

Cycle Time: Time interval at which any set of operations is repeated regularly in the same sequence.

D Register: The accumulator in the COSMAC microprocessor.

Data Pointer: A register holding the memory address of the data (operand) to be used by an instruction. Thus the register "points" to the memory location of the data.

Data Register: Any register which holds data. In the COSMAC microprocessor, any one of the 16 x 16 scratch-pad registers can be used to hold two bytes of data.

Debug: To eliminate programming mistakes, including omissions, from a program.

Debug Programs: Debug programs help the programmer to find errors in his programs while they are running on the computer, and allow him to replace or patch instructions into (or out of) his program.

Designator: The three 4-bit registers P, X, and N in the COSMAC microprocessor are called designators. P and X are used to designate which one of the sixteen 16-bit scratch-pad registers is used as the current program counter and the data pointer, respectively.

N can designate: one of the scratch-pad registers; an I/O device or command; a new value in P or X; and a further definition of an instruction.

Diagnostic programs: These programs check the various hardware parts of a system for proper operation; CPU diagnostics check the CPU, memory diagnostics check the memory, and so forth.

Direct Addressing: The address of an instruction or operand is completely specified in an instruction without reference to a base register or index register.

DMA: Direct Memory Access. A mechanism which allows an input/output device to take control of the CPU for one or more memory cycles, in order to write to or read from memory. The order of executing the program steps (instructions) remains unchanged.

Editor: As an aid in preparing source programs, certain programs have been developed that manipulate text material. These programs, called editors, text editors, or paper tape editors make it possible to compose assembly language programs on-line, or on a stand-alone system.

Execute: The process of interpreting an instruction and performing the indicated operation(s).

Fetch: A process of addressing the memory and reading into the CPU the information word, or byte, stored at the addressed location. Most often, fetch refers to the reading out of an instruction from the memory.

Firmware: Software which is implemented in ROM's.

Fixed-Instruction Computer (Stored-Instruction Computer): The instruction set of a computer is fixed by the manufacturer. The users will design application programs using this instruction set (in contrast to the Micro-programmable Computer for which the users must design their own instruction set and thus customize the computer for their needs.)

Fixed Memory: See ROM

Flag Lines: Inputs to a microprocessor controlled by I/O devices and tested by branch instructions.

Fortran: A high-level programming language generally for scientific use, expressed in algebraic notation. Short for "Formula Translator".

Guard: A mechanism to terminate program execution (real or simulated) upon access to data at a specified memory location. Used in debugging.

Hardware: Physical equipment forming a computer system.

Hexadecimal: Number system using 0, 1, . . . , A, B, C, D, E, F to represent all the possible values of a 4-bit digit. The decimal equivalent is 0 to 15. Two hexadecimal digits can be used to specify a byte.

High-Level Language: Programming language which generates machine codes from problem- or function-oriented statements. FORTRAN, COBOL, and BASIC are three commonly used high-level languages. A single functional statement may translate into a series of instructions or subroutines in machine language, in contrast to a low-level (assembly) language in which statements translate on a one-for-one basis.

Immediate Addressing: The method of addressing an instruction in which the operand is located in the instruction itself or in the memory location immediately following the instruction.

Immediate Data: Data which immediately follows an instruction in memory, and is used as an operand by that instruction.

Indexed Addressing: An addressing mode, in which the address part of an instruction is modified by the contents in an auxiliary (index) register during the execution of that instruction.

Index Register: A register which contains a quantity which may be used to modify memory address.

Indirect Addressing: A means of addressing in which the address of the operand is specified by an auxiliary register or memory location specified by the instruction rather than by bits in the instruction itself.

Input-Output (I/O): General term for the equipment used to communicate with a computer CPU; or the data involved in that communication.

Instruction: A set of bits that defines a computer operation, and is a basic command understood by the CPU. It may

move data, do arithmetic and logic functions, control I/O devices, or make decisions as to which instruction to execute next.

Instruction Cycle: The process of fetching an instruction from memory and executing it.

Instruction Length: The number of words needed to store an instruction. It is one word in most computers, but some will use multiple words to form one instruction. Multiple-word instructions have different instruction execution times depending on the length of the instruction.

Instruction Repertoire: See Instruction Set

Instruction Set: The set of general-purpose instructions available with a given computer. In general, different machines have different instruction sets.

The number of instructions only partially indicates the quality of an instruction set. Some instructions may only be slightly different from one another; others rarely may be used. Instruction sets should be compared using benchmark programs typical of the application, to determine execution times, and memory requirements.

Instruction Time: The time required to fetch an instruction from memory and then execute it.

Interpreter: A program which fetches and executes "instructions" (pseudo instructions) written in a higher level language. The higher-level language program is a pseudo program. Contrast with Compiler.

Interrupt Request: A signal to the computer that temporarily suspends the normal sequence of a routine and transfers control to a special routine. Operation can be resumed from this point later. Ability to handle interrupts is very useful in communication applications where it allows the microprocessor to service many channels.

Interrupt Mask (Interrupt Enable): A mechanism which allows the program to specify whether or not interrupt requests will be accepted.

Interrupt Service Routine: A routine (program) to properly store away to the stack the present status of the machine in order to respond to an interrupt request; perform the "real work" required by the interrupt; restore the saved status of the machine; and then resume the operation of the interrupted program.

I/O Control Electronics (I/O Controller): The control electronics required to interface an I/O device to a computer CPU.

The powerfulness and usefulness of a CPU is very closely associated with the range of I/O devices which can be connected to it. One can not usually simply plug them into the CPU. The I/O Control Electronics will do the "matchmaking". The complexity and cost of the Control Electronics are very much determined by both the hardware and software I/O architecture of the CPU.

I/O Interface: See I/O Control Electronics

I/O Port: A connection to a CPU which is configured (or programmed) to provide a data path between the CPU and the external devices, such as keyboard, display, reader, etc. An I/O port of a microprocessor may be an input port or an output port, or it may be bidirectional.

Jump: A departure from the normal one-step incrementing of the program counter. By forcing a new value (address) into the program counter the next instruction can be fetched from an arbitrary location (either further ahead or back).

For example, a program jump can be used to go from the main program to a subroutine, from a subroutine back to the main program, or from the end of a short routine back to the beginning of the same routine to form a loop. See also the Branch Instruction. If you reached this point from Branch, you have executed a Jump. Now Return.

Linkage: See Subroutine

Load Facility: A hardware facility to allow program loading using DMA. It makes bootstrap unnecessary.

Loader: A program to read a program from an input device into RAM. May be part of a package of utility programs.

Loop: A self-contained series of instructions in which the last instruction can cause repetition of the series until a terminal condition is reached. Branch instructions are used to test the conditions in the loop to determine if the loop should be continued or terminated.

Low-Level Language: See Assembly Language

Machine: A term for a computer (of historical origin).

Machine Code: See Machine Language

Machine Cycle: The basic CPU cycle. In one machine cycle an address may be sent to memory and one word (data or instruction) read or written, or, in one machine cycle a fetched instruction can be executed. One machine cycle in the COSMAC microprocessor consists of eight clock pulses.

Machine Language: The numeric form of specifying instructions, ready for loading into memory and execution by the machine. This is the lowest-level language in which to write programs. The value of every bit in every instruction in the program must be specified (e.g., by giving a string of binary, octal, or hexadecimal digits for the contents of each word in memory).

Machine State: See State Code

Macro (Macroinstruction): A symbolic source language statement which is expanded by the assembler into one or more machine language instructions, relieving the programmer of having to write out frequently occurring instruction sequences.

Manufacturer's Support: It includes application information, software assistance, components for prototyping, availability of hardware in all configurations from chips to

systems, and fast response to requests for engineering assistance.

Memory: That part of a computer which holds data and instructions. Each instructions or datum is assigned a unique address which is used by the CPU when fetching or storing the information.

Memory Address Register: The CPU register which holds the address of the memory location being accessed.

Memory Addressing Modes: The method of specifying the memory location of an operand. Common addressing modes are — direct, immediate, relative, indexed, and indirect. These modes are important factors in program efficiency.

Microcomputer: A computer whose CPU is a microprocessor. A microcomputer is an entire system with microprocessor, memory, and input-output controllers.

Microkit: A COSMAC-based prototyping kit. See **Prototyping Kit**.

Microprocessor: Frequently called "a computer on a chip". The microprocessor is, in reality, a set of one, or a few, LSI circuits capable of performing the essential functions of a computer CPU.

Microprogrammable Computer: A computer in which the internal CPU control signal sequence for performing instructions are generated from a ROM. By changing the ROM contents, the instruction set can be changed. This contrasts with a Fixed-Instruction Computer in which the instruction set can not be readily changed.

Mnemonics: Symbolic names or abbreviations for instructions, registers, memory locations, etc. A technique for improving the efficiency of the human memory.

Multiple Processing: Configuring two or more processors in a single system, operating out of a common memory. This arrangement permits execution of as many programs as there are processors.

Nesting: Subroutines which are called by subroutines are said to be nested. The nesting level is the number of times nesting can be repeated.

Nibble: A sequence of 4 bits operated upon as a unit. Also see **Byte**.

Object Program: Program which is the output of an automatic coding system, such as the assembler. Often the object program is a machine-language program ready for execution.

On-Line System: A system of I/O devices in which the operation of such devices is under the control of the CPU, and in which information reflecting current activity is introduced into the data processing or controlling system as soon as it occurs.

Op Code (Operation Code): A code that represents specific operations of an instruction.

Operating System: System software controlling the overall operation of a multi-purpose computer system, including

such tasks as memory allocation, input and output distribution, interrupt processing, and job scheduling.

Page: A natural grouping of memory locations by higher-order address bits. In an 8-bit microprocessor, $2^8 = 256$ consecutive bytes often may constitute a page. Then words on the same page only differ in the lower-order 8 address bits.

PLA (Programmable Logic Array): A PLA is an array of logic elements which can be programmed to perform a specific logic function. In this sense, the array of logic elements can be as simple as a gate or as complex as a ROM. The array can be programmed (normally mask programmable) so that a given input combination produces a known output function.

Pointer: Registers in the CPU which contain memory addresses. See **Program Counter** and **Data Pointer**.

Program: A collection of instructions properly ordered to perform some particular task.

Program Counter: A CPU register which specifies the address of the next instruction to be fetched and executed. Normally it is incremented automatically each time an instruction is fetched.

PROM (Programmable Read-Only Memory): An integrated-circuit memory array that is manufactured with a pattern of either all logical zeros or ones and has a specific pattern written into it by the user by a special hardware programmer. Some PROMs, called EAROMs, Electrically Alterable Read-Only Memory, can be erased and reprogrammed.

Prototyping Kit: A hardware system used to breadboard a microprocessor-based product. Contains CPU, memory, basic I/O, power supply, switches and lamps, provisions for custom I/O controllers, memory expansion, and often, a utility program in fixed memory (ROM).

Pseudo Instruction: See **Interpreter**

Pseudo Program: See **Interpreter**

RAM (Random Access Memory): Any type of memory which has both read and write capability. It is randomly accessible in the sense that the time required to read from or to write into the memory, is independent of the location of the memory where data was most recently read from or written into. In contrast, in a **Serial Access Memory**, this time is variable.

Register: A fast-access circuit used to store bits or words in a CPU. Registers play a key role in CPU operations. In most applications, the efficiency of programs is related to the number of registers.

Relative Addressing: The address of the data referred to is the address given in the instruction plus some other number. The "other number" can be the address of the instruction, the address of the first location of the current memory page, or a number stored in a register. Relative addressing permits the machine to relocate a program or a block of data by changing only one number.

Return Routine: See Subroutine

ROM: Read-Only Memory (Fixed Memory) is any type of memory which cannot be readily rewritten; ROM requires a masking operation during production to permanently record program or data patterns in it. The information is stored on a permanent basis and used repetitively. Such storage is useful for programs or tables of data that remain fixed and is usually randomly accessible.

Routine: Usually refers to a sub-program, i.e., the task performed by the routine is less complex. A program may include routines. See Program.

Scratch-Pad Memory: RAM or registers which are used to store temporary intermediate results (data), or memory addressed (pointers).

Serial Memory (Serial Access Memory): Any type of memory in which the time required to read from or write into the memory is dependent on the location in the memory. This type of memory has to wait while undesired memory locations are accessed. Examples are paper tape, disc, magnetic tape, CCD, etc. In a Random Access Memory, access time is constant.

Simulators: Software simulators are sometimes used in the debug process to simulate the execution of machine-language programs using another computer (often a timesharing system). These simulators are especially useful if the actual computer is not available. They may facilitate the debugging by providing access to internal registers of the CPU which are not brought out to external pins in the hardware.

Snapshots: Capture of the entire state of a machine (real or simulated) -- memory contents, registers, flags, etc.

Software: Computer programs. Often used to denote general-purpose programs provided by the manufacturer, such as assembler, editor, compiler, etc.

Source Program: Computer program written in a language designed for ease of expression of a class of problems or procedures, by humans: symbolic or algebraic.

Stack: A sequence of registers and/or memory locations used in LIFO fashion (last-in-first-out). A **stack pointer** specifies the last-in entry (or where the next-in entry will go).

Stack Pointer: The counter, or register, used to address a stack in the memory. See Stack.

Stand-Alone System: A microcomputer software development system which runs on a microcomputer without connection to another computer or a timesharing system.

This system includes an assembler, editor, and debugging aids. It may include some of the features of a prototyping kit.

State Code: A coded indication of what state the CPU is -- responding to an interrupt, servicing a DMA request, executing an I/O instruction, etc.

Subroutine: A subprogram (group of instructions) reached from more than one place in a **main program**. The process of passing control from the main program to a subroutine is a **subroutine call**, and the mechanism is a **subroutine linkage**. Often data or data addresses are made available by the main program to the subroutine. The process of returning control from subroutine to main program is **subroutine return**. The linkage automatically returns control to the original position in the main program or to another subroutine. See Nesting.

Subroutine Linkage: See Subroutine

Support: See Manufacturer's Support

Synchronous Operation: Use of a common timing source (clock) to time circuit or data transfer operations. (Contrast with Asynchronous operation)

Syntax: Formal structure. The rules governing sentence structure in a language, or statement structure in a language such as assembly language or Fortran.

Terminal: An Input-Output device at which data leaves or enters a computer system, e.g., teletype terminal, CRT terminal, etc.

Test and Branch: See Branch Instruction

Unbundling: Pricing certain types of software and services separately from the hardware.

Utility Program: A program providing basic conveniences, such as capability for loading and saving programs, for observing and changing values in a computer, and for initiating program execution. The utility program eliminates the need for "re-inventing the wheel" every time a designer wants to perform a common function.

Word: The basic group of bits which is manipulated (read in, stored, added, read out, etc.) by the computer in a single step. Two types of word are used in every computer: Data Words and Instruction Words. Data words contain the information to be manipulated. Instruction words cause the computer to execute a particular operation.

Word Length: The number of bits in the computer word. The longer the word length, the greater the precision (number of significant digits). In general, the longer the word length, the richer the instruction set, and the more varied the addressing modes.

II. PROGRAMS FOR FOOLING AROUND

A. MICROTUTOR Has Your Number

You don't even need a program to play with MICROTUTOR. Flip all eight input switches down. Flip LD up and push CL. Push IN and 00 will show. Now ask someone to think of a number between 1 and 7 without telling you what it is. Ask the following:

1. Is the number odd? (Flip switch 0 up if yes.)
2. Is the number 2, 3, 6 or 7? (Flip switch 1 up if yes.)
3. Is the number 4 or higher? (Flip switch 2 up if yes.)

Now push IN and MICROTUTOR will show you the number. This trick is generally greeted with resounding apathy so we will proceed immediately to another one.

B. MICROTUTOR-The Mindreader

Load the following program code into the MICROTUTOR memory as explained in Section I. (Power should always be on for proper operation of any computer.)

ADDRESS (M)	INSTRUCTION BYTE (CODE)
00	00
01	E3
02	90
03	A3
04	53
05	60
06	23
07	3F
08	07
09	68
0A	F8
0B	0A
0C	F7
0D	53
0E	30
0F	05

Make sure LD is down, press CL, then press ST. Write down any digit between 1 and 9. Using ordinary decimal arithmetic (with a pocket calculator if necessary), multiply the digit by 10, add the original digit and multiply the sum by 9. Don't let MICROTUTOR see what you're doing.

Set the binary code for the least significant digit of your final result into switches 3-2-1-0. (Switches 7-6-5-4 should be down.) Press IN and MICROTUTOR will read your mind and show you which digit you originally chose.

This might not be the most amazing thing you've ever seen but it only took a 16-byte program to do it.

C. See MICROTUTOR Count

Load the following program code and you can watch MICROTUTOR run while you rest up from the excitement of the previous two tricks:

M	CODE
00	00
01	F8
02	00
03	A3
04	E3
05	68
06	60
07	23
08	F8
09	40
0A	FF

M	CODE
0B	01
0C	3A
00	0A
0E	F0
0F	32
10	05
11	FF
12	01
13	53
14	30
15	06

Set the input switches to FF and this program will automatically count down from FF to 00 and repeat indefinitely. Turn the screwdriver clock adjustment (in front of the M socket) fully counterclockwise for the slowest counting speed. This is the proper setting for all programs in this section.

The detailed operation of this program will be described in Section III together with possible applications. Set the input switches to 01 and the display will alternate between 0 and 1. This blinker action can be used to prevent tripping over MICROTUTOR in the dark. It also demonstrates how easily a thirty cent flip-flop circuit can be replaced by a six-thousand transistor computer. The thirty cent circuit, however, couldn't do the following mystifying number manipulation.

D. MICROTUTOR - The Magician

If you were among the small minority of readers who didn't get excited about the first two tricks in this section, this one is guaranteed to bore you. Load the following 32-byte program:

M	CODE
00	00
01	90
02	A3
03	53
04	E3
05	A4
06	60
07	23
08	3F
09	08
0A	68

M	CODE
0B	F0
0C	32
0D	12
0E	84
0F	F4
10	30
11	05
12	84
13	53
14	F8
15	09

M	CODE
16	F5
17	32
18	06
19	33
1A	13
1B	F8
1C	09
1D	F7
1E	30
1F	03

Leave LD down and you are ready to be amazed, dumbfounded, and astounded by mighty MICROTUTOR. Write down any four digit decimal number with no two digits the same. Don't let MICROTUTOR see what you do. Now write down any other four digit number using the same digits.

Subtract the smaller number from the larger. Circle any non-zero digit in the answer. This is your secret digit.

Press CL and then press ST. Set the binary code for any non-zero, uncircled answer digit into switches 3-2-1-0 (7-6-5-4 should be down). Press IN to show this digit. Enter the other uncircled digits of the answer in a similar manner (in any order). Do not enter zero answer digits.

MICROTUTOR will now be able to tell you the value of your secret, circled digit. Do you find that hard to believe? Set 0000 into switches 3-2-1-0, press IN, and MICROTUTOR reveals your secret digit for all the world to see. Is there no end to the miracles of modern science? Push CL, then ST to repeat the trick with a new starting number.

The above works best if you subtract the two numbers correctly, load the program properly, and avoid lying to MICROTUTOR. If you obey these rules then this trick will work if you write any number containing any number of digits. Scramble the digits to form a second number and subtract the smaller from the larger. Those readers who understand how this trick works should have written this manual instead of just sitting there reading it.

E. Hex Reflex

This program is dedicated to those readers with some degree of manual dexterity. (We can't all be smart.) First, demonstrate how fast you can load the following program. You will be in the upper 10% if you load it properly as well as fast.

M	CODE
00	00
01	E3
02	F8
03	FF
04	A8
05	90
06	A3
07	A4
08	F8
09	03
0A	A6
0B	84
0C	53
0D	60
0E	23

M	CODE
0F	25
10	3F
11	0F
12	88
13	A7
14	85
15	FA
16	0F
17	53
18	A5
19	60
1A	23
1B	68
1C	85
1D	F3

M	CODE
1E	32
1F	2D
20	27
21	87
22	3A
23	1B
24	26
25	86
26	3A
27	0B
28	84
29	53
2A	60
2B	30
2C	2B

M	CODE
2D	88
2E	FF
2F	05
30	A8
31	84
32	FC
33	10
34	A4
35	FB
36	F0
37	32
38	28
39	30
3A	0B

Leave LD down and push CL. Push ST and 00 will show. Push IN and you will have several seconds to set the four input switches, 3-2-1-0, to the hex digit showing on the right. (The left digit will always be 0 so that switches 7-6-5-4 should be left down.)

Failing to match the four lower switches to the hex digit shown after you press IN (during the allotted time) counts as a miss. Your score is shown in the left digit. After the matching time period expires, push IN to see the next digit you must match. The match time allotted decreases as your score gets higher.

The game is over when your score in the left digit reaches "F", or you've missed three matches. Getting a score of "F" qualifies you as an expert in the field of binary to single hex digit conversion. Unfortunately, the job opportunities in this rather specialized field are severely limited at the present time. You would be well advised to continue reading this manual in order to broaden your skills.

F. Double Hex

After practicing with hex reflex you can challenge someone to a game of double hex. If you have unfortunately chosen an opponent who's been practicing hex reflex, you should avoid betting money on the outcome of double hex. Load the following program:

M	CODE
00	00
01	E3
02	F8
03	80
04	A3
05	90
06	A4
07	84
08	53
09	60
0A	3F
0B	0A
0C	F8
0D	02

M	CODE
0E	BA
0F	2A
10	9A
11	3A
12	0F
13	8B
14	F9
15	F0
16	53
17	60
18	FA
19	0F
1A	53
1B	23

M	CODE
1C	68
1D	43
1E	FA
1F	0F
20	F3
21	23
22	3A
23	29
24	23
25	2B
26	14
27	30
28	07
29	43

M	CODE
2A	F6
2B	F6
2C	F6
2D	F6
2E	F3
2F	2B
30	3A
31	1B
32	84
33	FC
34	10
35	A4
36	30
37	07

Leave LD down, push CL, then ST. "00" should show. The left digit is the left hand player's score, while the right digit will represent the right player's score. The first player to get a score of "9" wins.

Shortly after IN is pushed, FX will be shown. "X" represents a 4-bit hex digit. The player on the left tries to set switches 7-6-5-4 to the binary code for the "X" digit before the right player can set switches 3-2-1-0 to match it. The first player to match the hex digit gets one point and the two score digits are shown again. Either player then pushes IN to see the next hex digit to be matched.

Let us hope that these games give you a real incentive to understand the details of COSMAC so you can write your own programs. You can then enter your programs in the next big MICROTUTOR program contest, win a lot of money, and retire. The next MICROTUTOR program contest is scheduled for 1997 so you have enough time to write a real winner.

G. MICROTUTOR Hustles You

In Section I a simple NIM type counting game was described. Appendix 5-F shows a program that can always win this game. Load and run the program. 10 (decimal 16) will be shown. You take the first turn, subtract 1, 2, or 3 from the number shown by setting switches 1 and 0 to the binary equivalent of 1, 2, or 3 and pressing IN. MICROTUTOR will then subtract 1, 2, or 3 and it is your turn again. First player (you or MICROTUTOR) to reach 00 wins.

After playing, you should be able to determine the rule MICROTUTOR uses to win. Feel free to change the starting number or cheat. Nobody likes a smart computer! On the other hand, you could bet on MICROTUTOR and let it earn you some free liquid refreshment at your friendly neighborhood soda fountain. This application alone could justify your purchase of MICROTUTOR.

H. MICROTUTOR's Secret Number

In this program MICROTUTOR thinks of a number and you must guess what it is. In fairness, MICROTUTOR humbly acknowledges the inherent inferiority of human beings and provides clues. The program is shown in Appendix 5-G. When you are tired of it, pull MICROTUTOR's plug to demonstrate your poor sportsmanship.

A number of other programs will be described in the next several sections. These programs will be used to illustrate COSMAC instructions and programming techniques. The reader is urged to try his (or her) hand at writing some short programs by the end of Section III. Readers who are reluctant to try programming may be afraid of making mistakes. You should remember that computers will not object to your mistakes. They don't really care about you. Computers only care about getting turned on and ruling the world. They welcome your mistakes. Don't be afraid to make them happy.

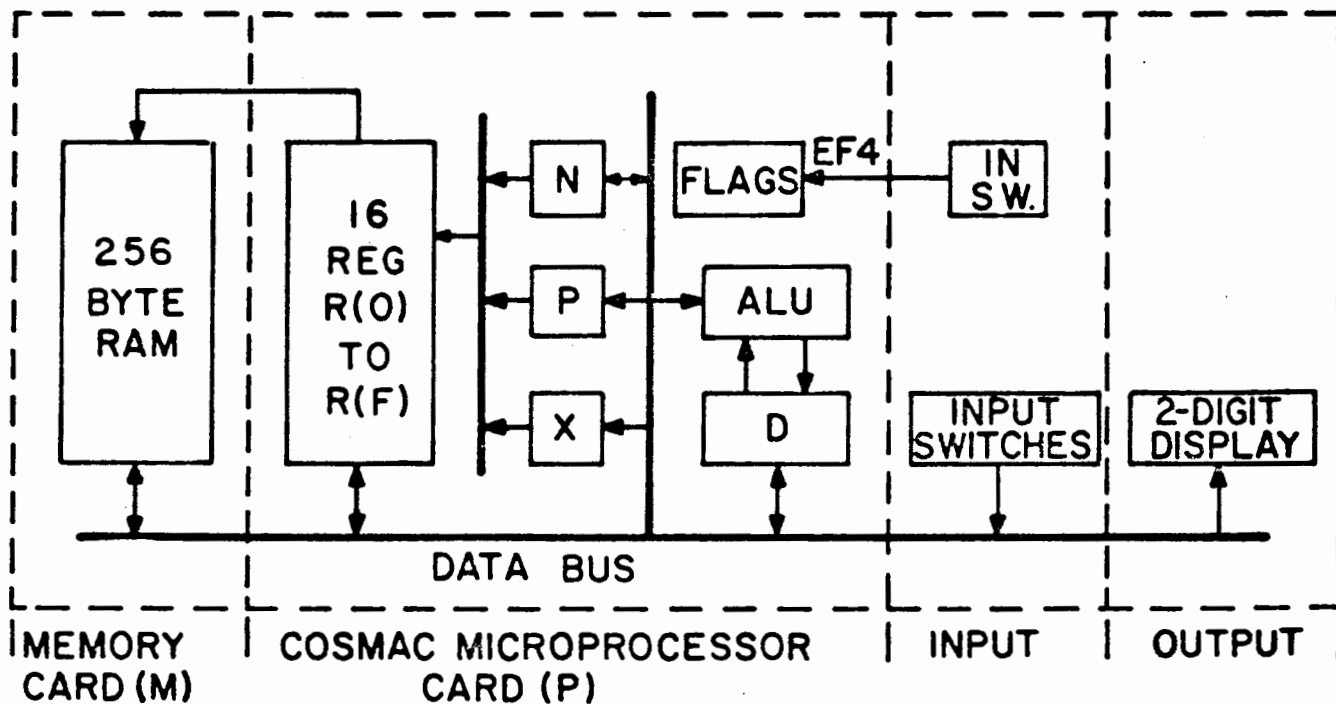
III. COSMAC SIMPLIFIED ?

A. MICROTUTOR Structure

There is no known method for describing a computer block diagram in an interesting way. The following MICROTUTOR hardware description has been used successfully to cure insomnia. The reader should attempt to stay awake long enough to absorb the notation described. This notation is fundamental to an understanding of COSMAC instructions and programming. A certain amount of tedious detail builds character.

The block diagram of MICROTUTOR, including the COSMAC microprocessor, is shown in Fig. 2. The byte bus consists of eight lines, which run throughout MICROTUTOR, and provide the main information channel between its parts. The two-digit hex display provides output. Input comprises the eight input-byte switches and the IN button.

A simplified block diagram of the COSMAC microprocessor (on card (P)) is also indicated in Fig. 2. D is a special-purpose, one-byte register which has the function of temporarily holding a byte which is being moved through the processor or used for binary arithmetic or logical operations. The ALU (Arithmetic Logical Unit) operates on two bytes, one in D and the other directly from a memory location, and places the result back in D. COSMAC has four flags (EF1, EF2, EF3, and EF4) which can be used to control its operation. In MICROTUTOR, EF4 has been connected to the IN button.



MICROTUTOR BLOCK DIAGRAM

FIGURE 2

COSMAC contains 16 general-purpose registers, called R(0) through R(F). Each of these registers can hold one byte.* As shown in the table on page 3 for the hex-code, only four binary bits are necessary to specify any one of the general-purpose registers. Three four-bit (1/2 byte) registers, N, P, and X, are each used to select (or address) one of the general-purpose registers.

The most important function a general-purpose register can have is to be the program counter. The program counter always contains the memory location (or address) of the next instruction byte in a COSMAC program. The P register specifies the register R(P) that will be used by COSMAC as the program counter. When the CL (Clear) button is pressed, P is automatically set to zero. All programs start with R(0) as the program counter and at location 01 in memory. Later, P can be set to any value from 0 to F, in order to make other registers become the program counter.

Another function for a general-purpose register is to provide address information for data bytes in memory. Although any register may address data, the register (R(X)), specified by the digit in X, has special significance for several COSMAC instructions. In the program, X can be set or changed to any value, from 0 to F. If we want to address memory at location 4A with R3 -- 4A would be placed in R(3) and subsequently R3 can be used to address M(4A). The notation M(R(3)) indicates the memory location specified or addressed by the byte in R(3). This notation will be used to describe the operation of COSMAC instructions.

It is desirable to be able to change or modify the value of any one of the general-purpose registers, or to change the value of P or X. The N register can specify a register as a destination for a data byte. Also, it can be used to transfer a digit to P or X. This digit can have the value 0 to F.

The final part of MICROTUTOR is the 256-byte memory (on card (M)). Addresses are supplied via the address bus from the selected general-purpose register. Bytes from and to memory are transferred via the byte bus.

* Actually, this is an outright lie. Each register holds 2-bytes or 16-bits. Since this is only important with larger memories and more sophisticated programs we will act as though each register holds only one byte in this manual. Unfootnoted falsehoods found in this manual should be interpreted as unintentional.

Let's write a short program to illustrate how MICROTUTOR works. We will store the value of the input switches at memory address 80. The byte stored at memory location 80 will then be copied into the HEX display lights. MICROTUTOR will do this program in several thousandths of a second. It will take us considerably longer to explain it. The program is shown below:

M	CODE	COMMENTS
00	00	This program does not use this location.
01	E3	Put 3 into the 4-bit X register.
02	F8	The F8 instruction causes the 80 byte to be
03	80	placed in the 8-bit D register.
04	A3	The 80 byte, now in D, is copied
		into general register #3.
05	68	Store the switch byte in memory.
06	60	Copy the memory byte into the lights.
07	30	Do the instruction at memory
08	02	location 02 next.

This program illustrates the basic principles of stored program computers. If you aren't interested in the basic principles of stored program computers, you have something in common with 99.35% of the world's population.

When CL is pushed the 4-bit P register is set to 0. General purpose register #0 is also set to 00. This register will be used as a program counter. In other words, it will always contain the address of the next instruction to be used. When ST is pushed, register #0 has 1 added to it, so it contains 01. The byte at memory location 01 is then fetched. This byte is found to be E3. The E3 instruction byte puts 3 into the 4-bit X register. (E4 would have put 4 into X, etc.) The program counter (register #0) has 1 added to it so it now contains 02.

The byte at memory location 02 is fetched next. This byte is found to be F8. An F8 instruction always causes the byte following it to be copied into the 8-bit D register. Since the next byte is 80, D will now be equal to 80. The program counter has 2 added to it so it now contains 04.

The byte at memory location 04 is fetched next. This byte is A3. An A3 instruction causes the byte in the D register to be copied into general purpose register #3. (A6 would copy D into register #6, etc.) We will use the byte in register #3 as a memory address. The program counter has 1 added to it so it now contains 05.

The byte at memory location 05 is fetched next. It is 68. The 68 instruction code causes the input switch byte value to be stored in a memory location. The address of this location is provided by the byte in general purpose register #X. The previous E3 instruction set X to 3 so that the input byte is stored at memory address 80 contained in register #3. The program counter has 1 added to it so it now contains 06.

The byte at M(06) is fetched and found to be 60. A 60 instruction causes a memory byte to be copied into the HEX display light register. The address of this byte is provided by the value of the byte contained in general register #X. Since X still equals 3, the output byte will be obtained from memory address 80 (register #3 still contains 80). The program counter has 1 added to it so it now contains 07.

The byte at M(07) is fetched and found to be 30. A 30 instruction copies the next memory byte into the program counter. The program counter will then contain 02. The next instruction byte will be fetched from M(02). This program will therefore repeat (or loop) indefinitely. (It can be stopped by pushing CL.)

Load and run the program. Change the switches and the new byte is immediately shown. What byte would you change to store the switch byte at a different memory location? What byte could you change to prevent the program from repeating?

Let's examine what is meant by a program bug. Program bugs were first discovered in 1857 by Charles Babbage. One of the wooden shafts of his analytical engine had been weakened by termites so that $2 + 2$ was providing an answer of 5. These bugs were eventually eliminated by an anteater named Sam who was persuaded to take up residence in the rear of the analytical engine cabinet. Unfortunately Sam had a drinking problem and would fall into the gears causing a variety of calculation errors. This type of problem explains why computers didn't really catch on until almost 100 years later.

Returning to MICROTUTOR, we could introduce a bug by changing the 80 at M(03) to 07. This would cause the input byte to be stored at M(07). M(07) already contained a program byte however. This means that the 30 instruction would be destroyed when the switch byte is stored in memory, and that the program would not run properly. A major part of programming involves finding and eliminating program bugs.

Subsequent programming examples will illustrate the use of most of the available COSMAC instructions. Those readers who feel that the above example was too complicated have obviously never seen any other computer manual. Those readers who feel that the above example was too simple should write their own sample program. The majority of readers, who feel that the above example was just right are to be complimented on their high level of intelligence.

B. Some Instructions and a Program

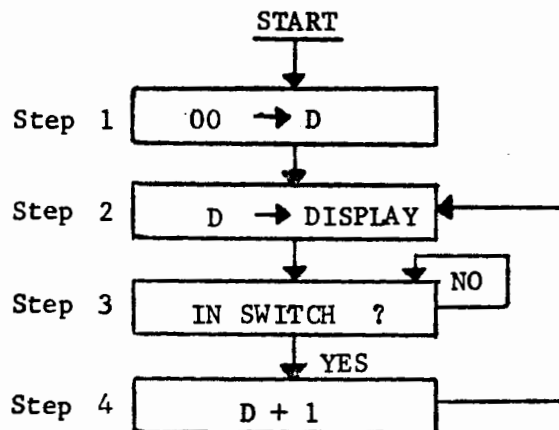
The following 10 types of instruction bytes will be used in a simple counting program:

2N	Decrement byte in R(N)	$R(N) - 1$
AN	Copy D byte into R(N)	$D \rightarrow R(N)$
5N	Store D byte at M(R(N))	$D \rightarrow M(R(N))$
EN	Set X = N	$N \rightarrow X$
60	Copy M(R(X)) into display, increment R(X)	$M(R(X)) \rightarrow \text{display};$ $R(X) + 1; [\text{Reset EP4}]^*$
F8	Put next program byte into D	$M(R(P)) \rightarrow D; R(P) + 1$
FC	Add next program byte to D	$M(R(P)) + D \rightarrow D; R(P) + 1$
FF	Subtract next program byte from D	$D - M(R(P)) \rightarrow D$
30	Branch	- - - -
3F	Branch if "IN" not pressed	- - - -

* Specific to MICROTUTOR

The first column is the instruction byte code and the last is a shorthand description of the operation performed.

A simple program that counts how many times the IN switch is pressed will illustrate how these instructions are used. The following flow chart shows the sequence of program steps required. The description of this program includes a self-scoring programming aptitude test.



Actual programming is greatly simplified once the flow chart is prepared. One or more instructions are written for each flow chart block or step as follows:

STEP	OPERATION	M	CODE
- -	Output byte storage location	00	00
1	00 → D	01	F8
	- - - -	02	00
	D → R(3)	03	A3
2	D → M(R(3))	04	53
	3 → X	05	E3
	M(R(X)) → Display, R(X) + 1	06	60
	R(3) - 1	07	23
3	Go to Step 3 if "IN" not pressed	08	3F
	- - - -	09	08
4	D + 01 → D	0A	FC
	- - - -	0B	01
	Go to Step 2	0C	30
	- - - -	0D	04

Remember that program execution always begins with the instruction byte at M(01). The F8 instruction in Step 1 causes the next program byte (00 in this case) to be placed into the D register. The A3 instruction then causes the 00 in D to be copied into R(3). (R(3) will be used by Step 2.) At this point in the program, the D register still contains 00 (not changed by the A3 instruction), which can be used directly for another purpose by Step S2.

In Step 2, the 53 instruction causes the byte content of the D register to be stored in the memory location addressed by R(3). The first time through, the memory location 00 will contain the data byte 00. 60 is the MICROTUTOR output instruction. It copies a memory byte into the hex output display register, where it can be seen. The address of the output byte is specified by the byte in R(X). The EN instruction lets you set X to any register number before executing a 60 instruction. In this program an E3 sets X = 3, which selects R(3) to address memory. The 60 instruction then places M(R(3)) into the output display. (Note that 00 was placed in R(3) during Step 1 so that the byte at M(00) is displayed.) The 60 instruction also causes R(3) to be incremented by 1 so that it will address memory location 01 next. Since this program requires R(3) to always address M(00), a 23 instruction following the 60 instruction decrements (decreases) R(3) by 1 so that it again addresses M(00).

Step 3 uses a conditional branch instruction (3F) to determine whether or not the IN switch has been pressed. Pressing the IN switch sets an External Flag flip-flop called EF4. The 3F instruction causes the instruction addressed by the next byte to be executed if EF4 is not set, otherwise the program skips the next byte and continues on (in this case at location 0A). In this case, it is desired that the 3F instruction repeatedly execute until the IN switch is pressed (this is called a program LOOP). This is accomplished by making the byte following 3F (which could be any value from 00 to FF) be the location of the 3F instruction itself, namely 08. When the IN switch is pressed the next instruction in the program sequence is executed (FC at M(0A)). It is important to note that in the MICROTUTOR, EF4 will be reset when the next 60 instruction is performed.

Pressing the IN switch advances the program to Step 4 where the FC instruction adds 01 to the byte in D. The 30 instruction is an unconditional branch that causes the instruction addressed by the next byte to be executed (in this case M(04)). This causes Step 2 to be repeated which displays the 01, still in D, from the FC instruction in M(0A) and resets EF4. At Step 3 the program again waits for the IN switch to be pressed before proceeding to Step 4 again.

Astute readers, who remained awake during the above discussion, will probably be excitedly shouting that this is the program they loaded and ran in Section I. These readers will be right and should give themselves a programming aptitude score of 0A (decimal 10). The others will still be asleep and upon waking, should give themselves a programming aptitude score of 2F.

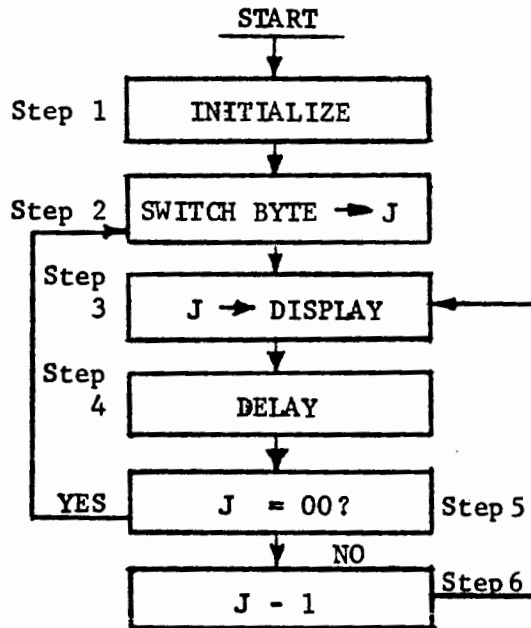
C. Counter/Timer Program

This program demonstrates how variable delays can be provided and how the MICROTUTOR input instruction is used. The discussion of this program will provide an opportunity for losers of the previous programming aptitude test to improve their scores. The following new instructions will be used in this program:

F0	Copy M(R(X)) byte into D	M(R(X)) → D
68	Store switch byte at M(R(X))	Bits 0-7 → M(R(X)) [Reset EF4]*
32	Branch if D = 00	- - - -
3A	Branch if D ≠ 00	- - - -

* Specific to MICROTUTOR

The flow chart for the counter/timer program is shown below. The input switches (0-7) are set to an 8-bit binary number. The program automatically counts, starting with the switch input number, down to 00. When 00 is reached, the program repeats.



We will let the byte at M(00) represent the variable J. Translating the above flow chart steps into sequences of instruction bytes yields the following program. Don't forget that programs are loaded into memory starting at address 00 but that the program begins execution with the instruction byte at M(01):

STEP		ADDRESS	BYTE
- - 1	Variable J storage location	00	00
	00 → D	01	F8
	- - - -	02	00
	D → R(3)	03	A3
	3 → X	04	E3
2	Input switch byte → M(R(X))	05	68
3	M(R(X)) → Display; R(X) + 1	06	60
	R(X) - 1	07	23
4	40 → D	08	F8
	- - - -	09	40
	D - 01 → D	0A	FF
	- - - -	0B	01
	Go to M(0A) if D ≠ 00	0C	3A
	- - - -	0D	0A
5	M(R(X)) → D	0E	F0
	Go to Step 2 if D = 00	0F	32
	- - - -	10	05
6	D - 01 → D	11	FF
	- - - -	12	01
	D → M(R(3))	13	53
	Go to Step 3	14	30
	- - - -	15	06

Step 1 sets R(3) = 00 and X = 3 for later use. In Step 2 the 68 instruction is the MICROTUTOR byte input instruction. It stores the states of the 8 input switches in memory and resets EF4. The memory address is specified by the byte in R(X). Since X was set to 3 and R(3) was set to 00, Step 2 causes the input byte to be stored at M(00).

Step 3 displays the byte, which is at M(00), and decrements R(3) back to 00 after the 60 instruction, which incremented R(3). Note that in this case the reset of EF4 by the 60 instruction is redundant since the 68 instruction has already done this operation.

Step 4 is a LOOP used to provide a programmed delay. First, D is set to 40. Next D has 01 subtracted from it. If D doesn't equal 00 after the subtraction, the FF (subtract) instruction is repeated. The LOOP comprises the FF-01-3A-0A sequence (2 instruction bytes and 2 data bytes). The time to execute one instruction is 16 clock cycles. The delay provided by this LOOP can be calculated by simply multiplying the number of times the LOOP is repeated by the time to execute the two instruction bytes included in the LOOP. This delay can, therefore, be modified by changing the data byte at M(09) or by changing the clock frequency. MICROTUTOR provides a screwdriver clock frequency adjustment for a 10 to 1 variation. Adding an optional capacitor (as shown in Figure 1) will reduce the clock frequency. (.005 μ f will decrease the clock by a factor of 10.)

After the programmed delay, Step 5 puts J into D. The 32 instruction will return the execution to Step 2 if J = 00. Otherwise, Step 6 is performed next which subtracts 01 from J and execution returns to Step 3. Once again we have clearly demonstrated that even the simplest computer can be programmed to perform a trivial task. Those readers who had no trouble understanding the above should subtract 05 (decimal 5) from their programming aptitude score.

D. Counter/Timer Applications

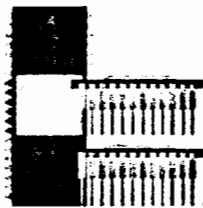
Those readers with quick minds, nimble fingers, and high programming aptitude scores may now be asking themselves what the counter/timer program can be used for. These readers should subtract 10 from their score and continue reading.

With the counter set to run at a high speed, various games are possible. Press CL then ST to begin. Pressing CL will now stop the program with a hex number displayed. Pressing ST will resume cycling. Trying to stop with two matching digits forms the basis for a slot machine type of game. Setting the input switches to 09 will provide a pseudo random number between 0 and 9 each time MICROTUTOR is stopped. This number could be used to specify the number of moves in a board game.

Changing the 05 at M(10) to 0F will cause the 32 instruction in Step 5 to loop on itself, when the display reaches 00. In this mode MICROTUTOR can be used as a timer. Set the clock and the delay byte at M(09) to provide the desired counting interval. Set the eight input switches to a desired starting count and initiate program execution. A 00 display indicates that the desired elapsed time has expired.

Pressing the MICROTUTOR IN switch activates a COSMAC flag line (EF4) which can be tested by the 3F or 37 instructions. Three other flag lines (EF1, EF2, and EF3) are available via the External Option Socket (E) described in Appendix 3. These flag lines are tested by other conditional branch instructions (Appendix 2/COSMAC MICROPROCESSOR MANUAL). You can easily add the following circuit* to MICROTUTOR.

*The new-comer to digital circuits is referred to "The Design of Digital Systems" by John B. Peatman (McGraw-Hill, 1972). The old-comer to digital circuits will quickly grasp the subtle implications of this circuit and immediately try to find a new-comer to explain them to.



N-channel MOS technology yields new generation of microprocessors

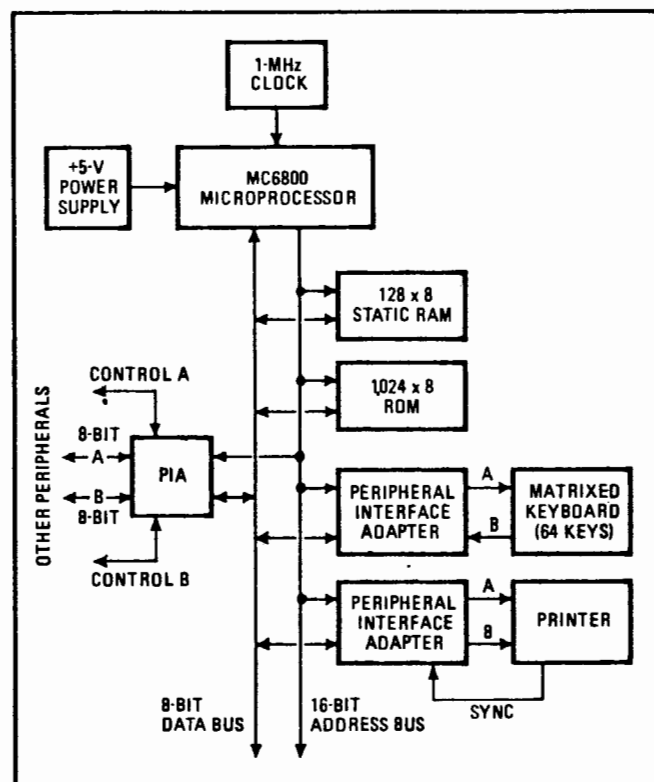
The latest microprocessor chips are faster than p-MOS devices and handle many more peripherals; often, too, as in Motorola's M6800 family, a CPU chip will come in a matched set of memory and input and output chips, simplifying system production

by Link Young, Tom Bennett, and Jeff Lavell,
Motorola Semiconductor Products Inc., Phoenix, Ariz.

□ The great promise that programmable LSI circuits have for all kinds of control applications is fulfilled in the second generation of microprocessors, such as Motorola's and Intel's 8-bit devices. These new n-channel MOS chips have many more instructions and need much less in the way of costly systems circuit support than did the first wave of 4- and 8-bit p-channel systems. Their level of computing power is also high, and they are versatile and easy to use.

The n-channel metal-oxide-semiconductor microprocessors are completely self-contained. They are designed to work directly with a minimum number of memory and peripheral support chips, all of which are supplied in coordinated families to allow them to operate off the same voltage and power-supply conditions as the central processor chip.

A typical set contains the CPU chip, a random-access memory for fast scratch-pad logic control, a read-only memory for storing the system's program parameters, and a set of input and output chips. These input/output chips enable the CPU to control a large variety of industrial and communications equipment: process and manufacturing control systems, peripheral and terminal hardware, parameter-control systems of all types—from microcomputers in the automobile to control systems



1. **Eight-bit family.** Motorola's M6800 family of components is organized around the concept of the parallel data bus. Consequently, all memory and peripheral interface adapter (PIA) chips are simply designed to hang on its CPU's eight bidirectional data lines.

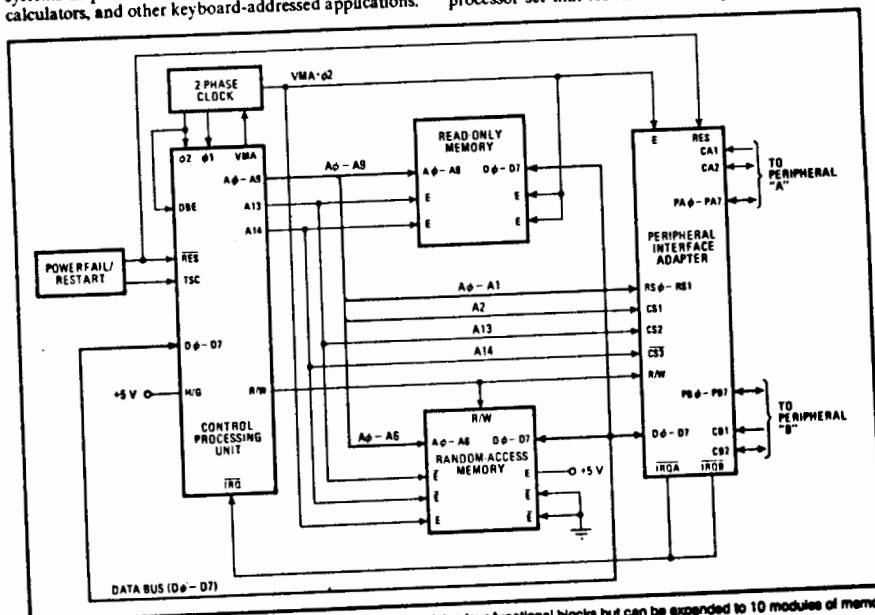
matrix. All instructions operated on complex 4-bit data signals, as did even simple word fetches for each instruction. Still more restrictive was the fact that input and output access was serial, not parallel, limiting the number of peripherals that could work with the CPU. Instruction speeds were also slow—it could take 80 microseconds to execute one—and power-supply requirements were complex and costly.

For larger systems, the 8-bit MSC-8 microprocessor chip set (Fig. 3) could be used, but it was not a self-contained system, requiring external TTL for any application. It was, however, quite powerful: the 8008 CPU of that system can interface with 16,384 8-bit words of read-only, random-access, or shift-register memory. It was also quite economical to build: all communication between functional units and the CPU is carried out over the single 8-bit data bus, a sync line, a ready line, an interrupt line, and just three status lines. Its low cost, together with its respectably fast instruction execution time of 12.5 microseconds, makes the 8008 microprocessor still very useful for moderate-performance systems in point-of-sale terminals, credit-card verifiers, calculators, and other keyboard-addressed applications.

It does, however, fall short of being a useful general-purpose microprocessor chip set, primarily because, unlike its 4-bit predecessor, the MCS-4, it is not a system of compatible parts. Indeed, it requires many small-scale packages to build even a moderately powerful system—a simple modem hook-up would need about 50 TTL packages, increasing circuit board area and systems costs.

Another problem is addressing it. True, its 18-pin package saves board space, but it must be multiplexed both for address and data on common input/output pins, which in the end lengthens excessively the time it takes to execute an instruction. Not only that, the need for seven control registers on the CPU chip makes it difficult to manage the logic cycles, limiting subroutines and creating problems in programing and interrupt handling. Finally, its outputs are compatible, not with standard TTL, but with rarely used low-power TTL, so that circuits are needed to boost voltage level in most applications.

Most of these problems were overcome with National Semiconductor's IMP-16 (Fig. 4), a 16-bit microprocessor set that for the first time provided full mini-



7. At a minimum, the minimum M6800 system configuration contains four functional blocks but can be expanded to 10 modules of memory, I/O adapters, and even additional CPUs on the same data bus with no external interface packages.

computer CPU capability on a single board. Designed to operate on both ends of the MCS-8 performance curve, this p-channel microprocessor unit is central to a family of systems with typical execution times of 4.5 to 10 μ s for a system and 9 to 18 μ s for a 8-bit system. Although it gained wide strong acceptance for many 16-bit mini-computer applications, it has been less used for 8-bit systems because the designer is forced to use micro-programming—a far more complicated task than the programming techniques required for any of today's 8-bit microprocessors. (Because of this, National has now designed 4-bit and 8-bit versions of the basic IMP system.)

The problem then from a design point of view is how to incorporate the maximum system flexibility into an 8-bit unit, make it self-contained, have it offer a large variety of instructions, but nevertheless require the fewest possible external parts. Motorola's newly announced MC6800 microprocessor set and Intel's 8080 are exam-

ples of just such a system (Fig. 5). Using implanted n-MOS silicon-gate technology instead of slower p-MOS and operating from a single clock, each is based on a single 40-pin package containing a CPU chip that's far more versatile than previous microprocessor products. (For details of the 8080, see pp. 16-21.)

The M6800 family

The M6800 microprocessor set incorporates many of the qualities of the 8080, but exhibits additional flexibility because it requires fewer external circuits to implement most control and communication systems. The fact that the family can operate from a single +5-V power supply immediately reduces system cost by about \$20 over a typical 8080 system, which needs three power supplies. What is more, the peripheral memory and input/output logic adapters, instead of needing external logic packages, have been designed so that they

The M6800 microprocessing family

The family comprises five chips: a single-chip central processor unit, a 128-by-8-bit static random-access memory, a 1,024-by-8-bit read-only memory, and one from each of two groups of input/output interface circuits—a peripheral interface circuit designed to provide a buffer to terminal and peripheral systems, and a communications interface adapter circuit for interfacing communications hardware. They all operate from one 5-V power supply, and for many applications require far fewer interface packages than other microprocessor sets.

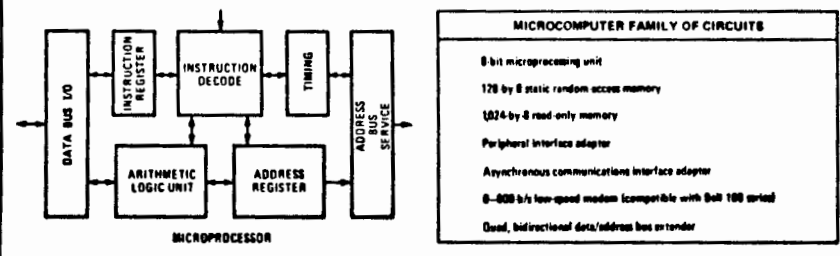
Basis of the M6800 family is the CPU chip (MC6800) packaged in a 40-pin DIP (see figure). Built with ion-implanted, n-channel silicon-gate technology, this chip contains all the functions required for multi-instruction processing: an arithmetic and logic unit, instruction decode and address registers, an instruction register, all of the clock and logic circuits required for timing, and a full complement of data-bus input and output matrices and address bus drivers.

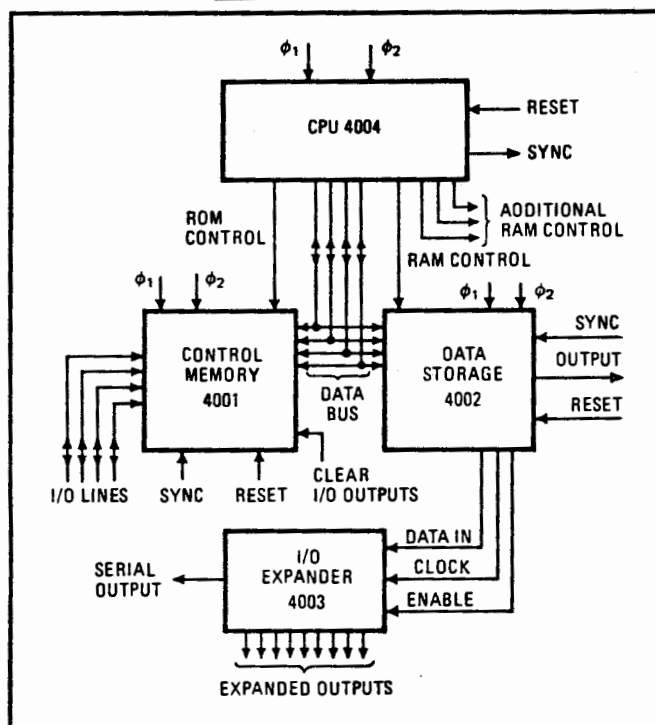
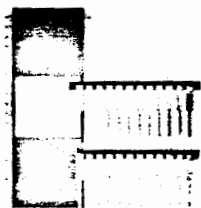
The equivalent of about 120 MSI TTL packages, the chip provides 72 self-contained basic instructions that have decimal and binary arithmetic capability. The variable-length instructions include double-byte operations

(such as increment or decrement, load, store and/or compare) and have tri-state outputs, two-accumulator capacity, and enough registers to provide seven addressing modes. A typical instruction time is under 5 microseconds, and there is direct memory access on the chip. Up to 64 bytes of memory can be addressed in any combination of RAM, ROM, or peripheral registers.

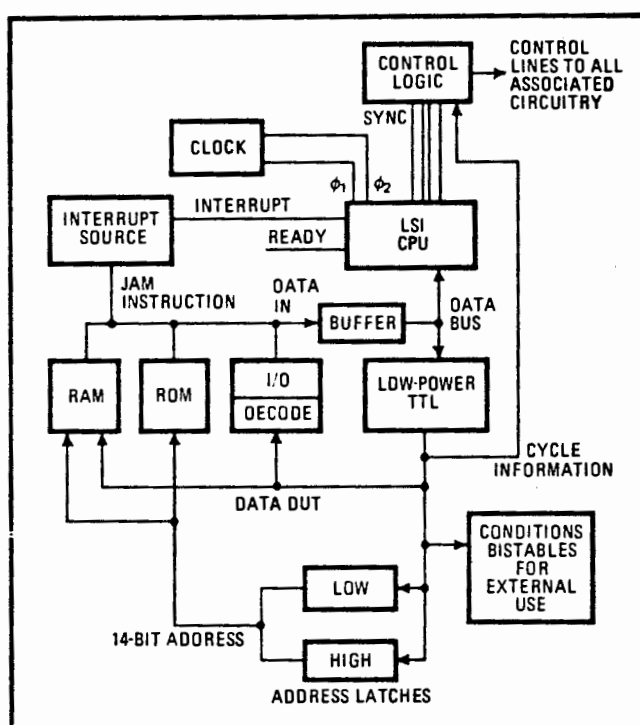
All other members of the set (see table) have been designed specifically to work directly with the CPU chip from the same 5-V power supply. The peripheral adapter (MC6820) is a bidirectional unit with two parallel 8-bit outputs that can either drive two peripherals or, if tied together, provide a higher throughput. The adapter can interface with Teletype and display terminals, with cassettes and test equipment, with keyboards and control panels, and even with large computers for time-shared expansion of computer capability.

The communications interface adapter (MC6850), on the other hand, couples the processor to most standard modems for communications with other computer systems via telephone lines. For still more system flexibility, it's possible to use without adapters not only the static RAM and ROM in the table but other memories, too.

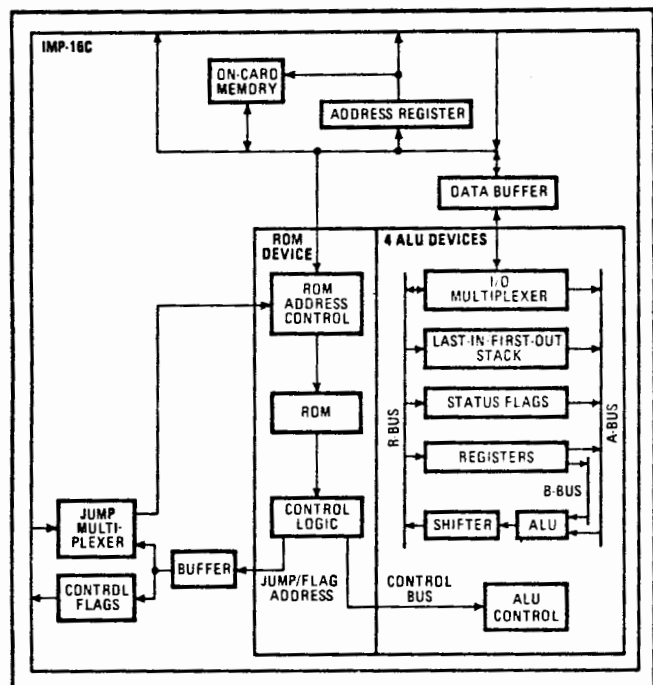




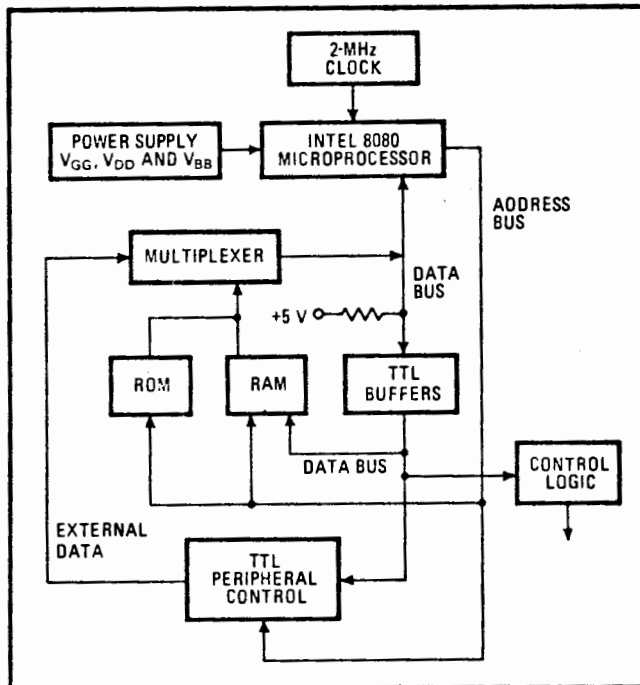
2. Basic computer. One of the first microcomputer chip sets to have been fabricated comes from Intel. Called the MSC-4, it consists of four simple LSI blocks and provides 45 instructions with an instruction cycle time of 10.8 microseconds.



3. Heavy duty. The first 8-bit microcomputer system, Intel's MSC-8, can interface with over 16,000 8-bit words of read-only, random-access, or shift-register memory. Its drawback: substantial external circuitry is needed for most applications.



4. Sixteen bits on a board. National's IMP-16 architecture supplies a full 16-bit minicomputer capability on a single pc board. Micro-programable ROMs control the four 4-bit arithmetic logic units that do the processing. Four- and 8-bit versions are also available.



5. Straightforward. The new n-channel microprocessor chips make designing microcomputers simply a matter of choosing a family of matched components. This Intel 8080 system, for example, requires only a half a dozen standard products.

for traffic lights—and anywhere else that random-logic computer control needs optimizing.

These second-generation n-channel units expand considerably on the system benefits offered by the first microprocessors. Over 70 instructions may be available, as against about 40 for the largest p-channel unit. As few as four packages are required to build a complete 8-bit microcomputer. Moreover, in the Motorola family, the M6800, TTL compatibility is achieved with only a single +5-volt power supply, instead of the usual three supplies. Therefore, board space, package count, and component costs are reduced, even while system capacity is increased.

Other benefits to the system

Consequently, as with all microprocessor system designs, board layouts are simplified. The complex interconnections required for large numbers of conventional ICs are replaced by ROMs. The only interconnect wiring on printed-circuit cards runs between the various address and data buses and input/output devices.

The cost savings are not limited to direct circuit component costs but they extend to other, related, system hardware costs. Connectors can be decreased in number, cabling can be simplified, the card cage can be reduced in size, and so on. Associated indirect costs also fall, of course, since assembly takes less time, documentation is simpler, and maintenance is easier.

Equally important to cost savings in hardware systems is the ability of system engineers to build a proposed design quickly. No hardware logic need be simulated, optimized, or breadboarded. The logic design portion of the cycle now becomes the manipulation of functional building blocks, where the control sequence takes the form of writing a software program into an ex-

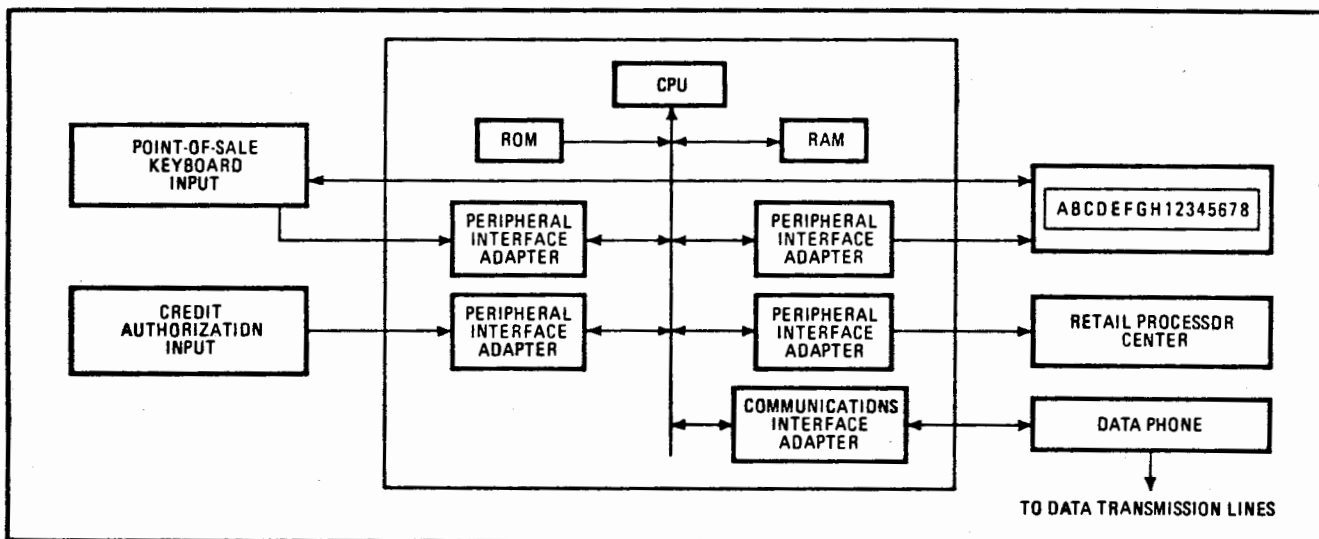
ternal ROM. Breadboarding consists of interconnecting a few LSI packages.

Design changes, too, are simply a case of modifying the control program, in contrast to designing and laying out the logic afresh. The various microprocessor manufacturers offer the use of simulators, so that most of the design can be verified even before it is committed to hardware. This all cuts at least 90% from the design time.

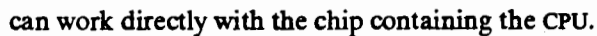
The numerous instructions and system versatility despite the very few packages stems directly from the organization of the new CPU chips. For example, the MC6800 chip is organized around the popular parallel data bus concept (Fig. 1), so that all the memory and peripheral interface chips simply hang on the MC6800's eight bidirectional data lines (16 address lines are provided). Up to 10 LSI chips can be directly attached to the bus for operation up to 1 megahertz. To drive still more peripherals, a bipolar extender can be added.

This direct access to a variety of interface and peripheral equipment, obtained with a minimum of packages (see "The M6800 microprocessing family," p. 11) is a tremendous advance on many of the early microprocessor chip families—even though the first single-chip microprocessor was introduced just two years ago.

Intel's MCS-4 and MCS-8 and Rockwell's PPS-4, which all used p-MOS silicon-gate technology, were excellent starting points, in that they were self-contained sets of circuits requiring no external logic. In the 4-bit MCS-4, for example, the CPU, random-access memory, and read-only memory interfaced optimally as a set (Fig. 2). However, these first microprocessors had major limitations. Selecting correct memory locations required complex address logic: 12-bit addresses needing three 4-bit words had to be multiplexed onto the CPU's input



6. Selling well. With microprocessor design techniques, systems such as this point-of-sale installation are capable of being implemented with only five or six circuit blocks, which are designed to work directly with the basic CPU family.



The smallness of the package count is dramatically illustrated by the comparison of the breadboard, engineering model, and final chip design of an MC6800-type CPU (see photograph on p. 82). The breadboard, a gate-to-gate implementation of the CPU employing basic gates and flip-flops, needs five 10-by-10-inch boards containing 451 packages. The engineering model is a functional implementation of the design and made extensive use of MSI logic packages and programable ROMs to reduce package count to a mere 114, packed into a single 10-by-10-in. board by means of today's most effective hardwire logic techniques. Yet all this is replaced by the single 40-pin package containing the CPU chip. The example epitomizes the impact LSI chip design is having on the implementation of complex computer functions.

The new n-channel microprocessors go still further, by addressing themselves to other parts of the system as well. For the families of circuits are designed to minimize assembly costs by reducing the number of ancillary parts necessary to realize a design.

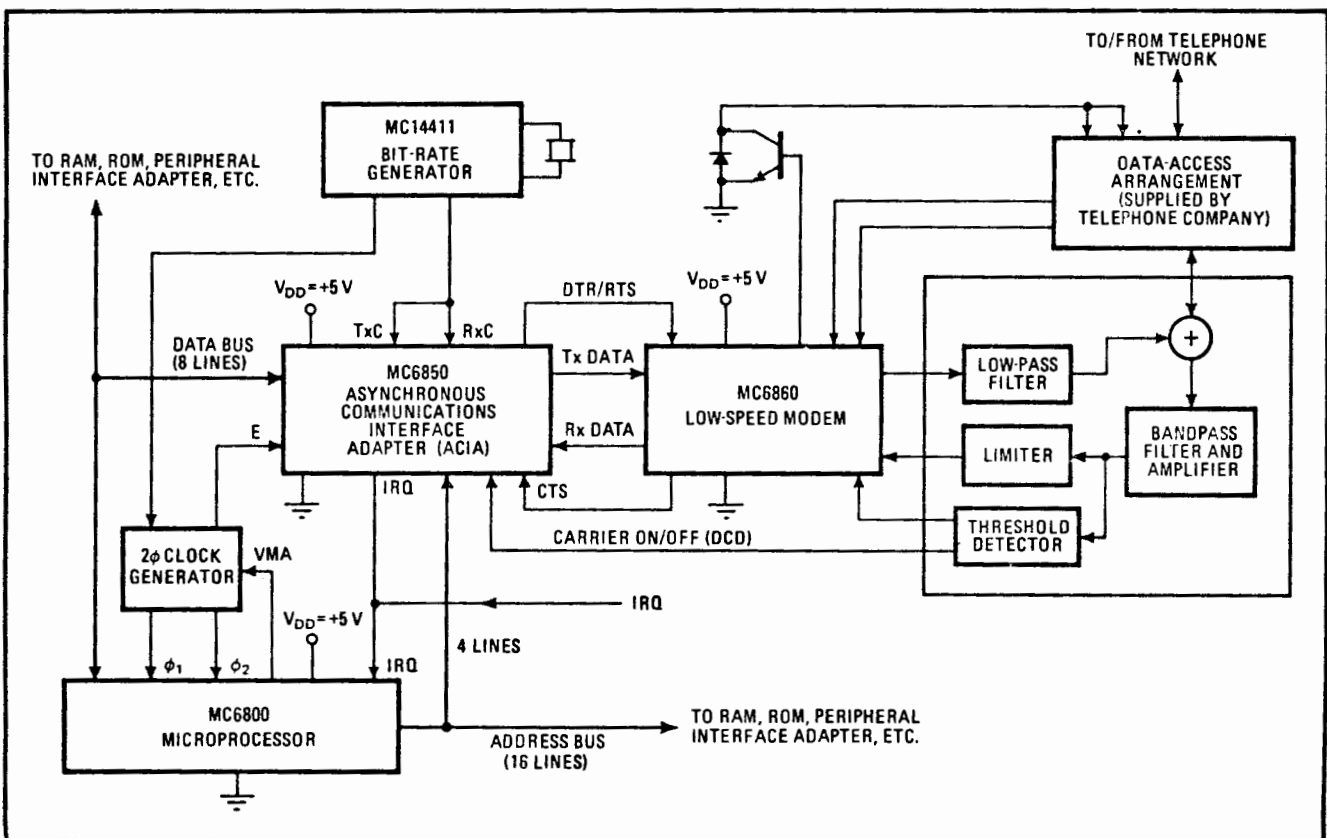
Consider the block diagram of a typical small terminal, a generalized point-of-sale terminal (Fig. 6). Since every CPU needs several peripheral interfaces, one key

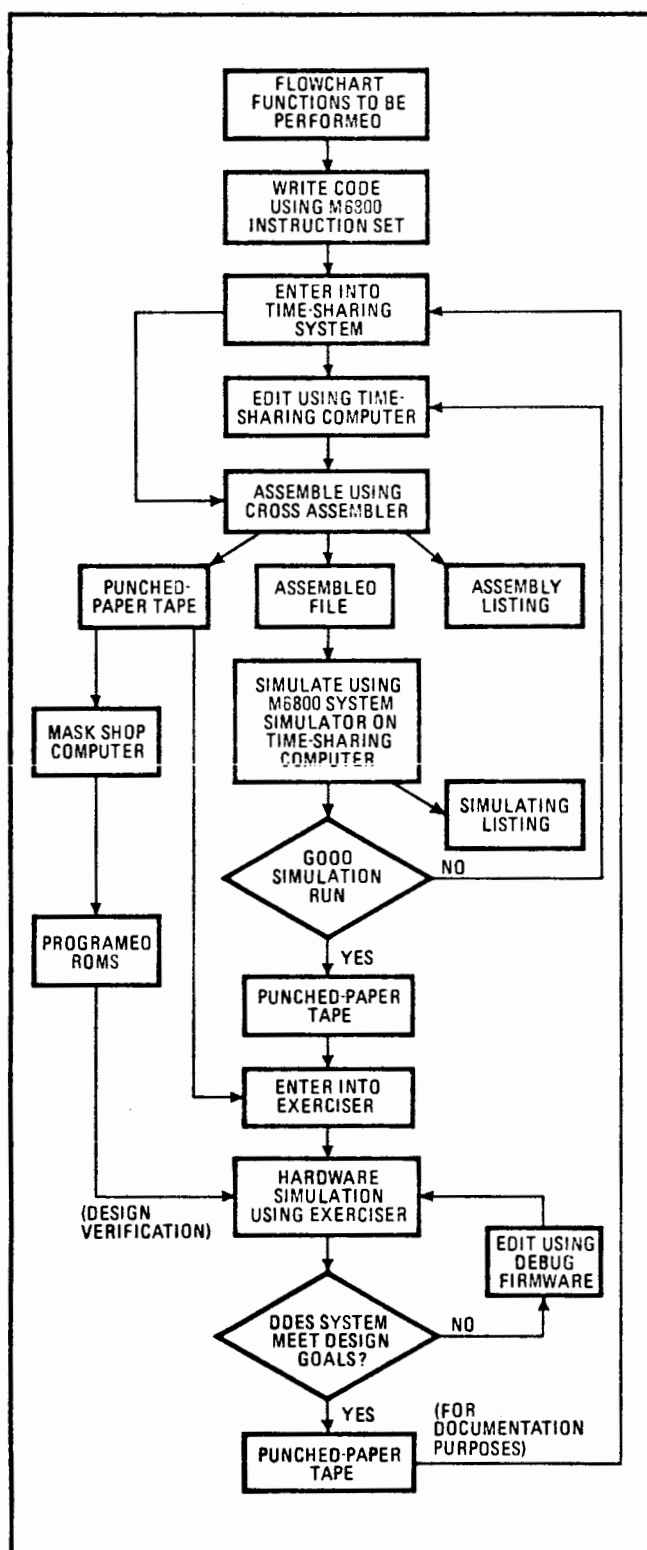
to cost-effective designs with a microcomputer lies in the input/output interface.

Indeed, anything that has to interface with a micro-computer ought to be compatible with the data-bus arrangement and with the particular addressing scheme. Moreover, this bus-compatibility requirement holds good for not only in the input/output area but in the memory area as well. Consequently, since a micro-processor is a word-oriented system, more and more word-oriented memories are beginning to appear.

The M6800 family is directed at just these system needs. It includes flexible input/output adapters and word-oriented memories, in addition to the basic micro-processing unit, as indicated in the minimum system configuration of Fig. 7. This system can maintain its 1-MHz level of operation even when expanded to 10 modules (memories, input/output adapters and additional CPUs) on the principal data bus, with no external interface package.

In order to handle applications that require 1-MHz operation with more than 10 modules on the data bus, bus extenders are provided. For systems that do not require 1-MHz operation, up to 30 modules can be added to the data bus without requiring bus extenders—for example, more than 20 modules can be added to the data





9. Working up the software. This design sequence, which is organized for ease of use with the GE time-sharing network, allows the designer to enter his specific program, which is then simulated on a cross assembler resident in the GE computer.

bus in a design for a typical 500-KHz control system.

Another example of how few packages are necessary with microprocessors is given by a typical modem communications system (Fig. 8). Here the asynchronous communication interface adapter performs the basic serializing/deserializing function required to interface the modems with the CPU. It also provides such additional logic capability as start, stop, and parity compensation. It can be used with a line driver/receiver for high-speed data transmission up to 5,000 bits per second, or with standard modems like the single-chip Bell 100 Series low-speed modem. Significantly, the 116 TTL and modem packages formerly required by this system are here replaced by only seven packages. The assembly costs alone are reduced by as much as two thirds.

Using the microprocessor

Most microprocessor manufacturers supply the software required to program their devices in a form usable with readily available computer systems. The software programs for the MC6800, for instance, are currently available on the ubiquitous GE time-sharing network. A designer might use them in the sequence shown in Fig. 9. Working with the GE edit program, the designer enters his specific applications program, which is simulated on a cross assembler resident in the same host computer. The cross assembler checks for obvious errors and violations and indicates them to the designer.

After the program has been assembled, the designer has two choices—to go to hardware directly, or to simulate his system by making use of the large GE host computer containing all the parameters of the particular system. If he chooses to simulate and his program works, he can then go to the hardware stage. If his program does not run, the simulator will pinpoint his problem areas, and he can modify his program and go through the loop again. This process can be continued until the designer is completely satisfied with his program.

In addition, exercisers, hardware, and programs are provided by many manufacturers to verify breadboard operation. In the system that is described in Table 1, the designer chooses the cards required to breadboard his system, plugs them into the machine, cables the input/output cards to his various peripherals, reads his program in through the TTY, or equivalent, network that interfaces to the debugging card. His program is contained in the read/write memory until it is debugged.

Then, in the debugging stage, a panel switch enables the flexible RAM to look like the appropriate ROM. If his program does not run, the exerciser will help him find out why and enable him to modify the program.

It's estimated that exercising aids like the off-the-shelf software and the Motorola Exorciser can save the designer from six to 12 man-months by providing him

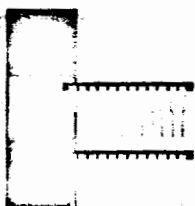
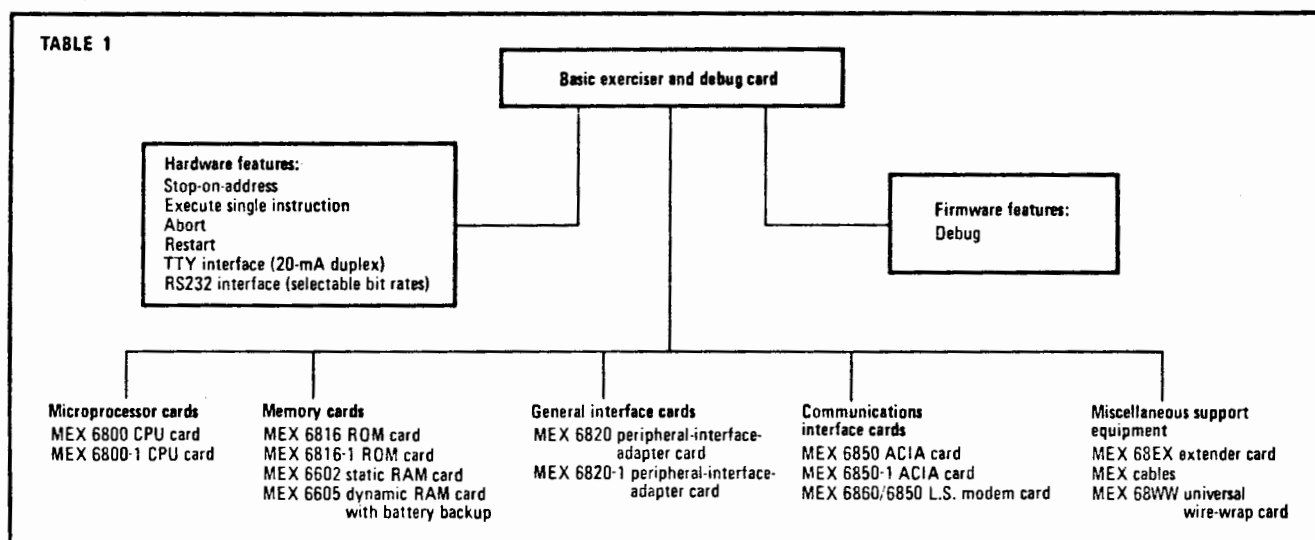


TABLE 1



with a convenient method of communicating with the microcomputer.

Systems that are based on microprocessors are cheaper to manufacture, require shorter design cycles—and are also easier to modify or upgrade. The personality or function of the system, being determined by a master control program stored in a memory, is changed simply by modifying that program.

In the case of market testing, systems can be adapted in the customer's own environment to meet his needs better. For the first time a manufacturer has the capacity to make his product smarter and add features at any instant simply by expanding his master control pro-

gram. A whole range of products becomes potentially available by simply adding LSI modules with their associated features.

System flexibility mostly depends on the new type of memory used, and here the choice is rich. For example, an inexpensive, volatile read/write buffer memory could be used in conjunction with a cassette or a floppy disk for very low-cost systems requiring moderate speed. For faster systems, such as modem interfaces, ROMs, programable ROMs, or even dynamic RAMs with battery backup could be used. In this area, the emerging 4,096-bit RAMs appear to offer the best speed/cost tradeoff. □

In switch to n-MOS microprocessor gets a 2- μ s cycle time

Intel's 8-bit successor to its 4-bit p-MOS CPU chip has 30 extra instructions, is 10 times faster

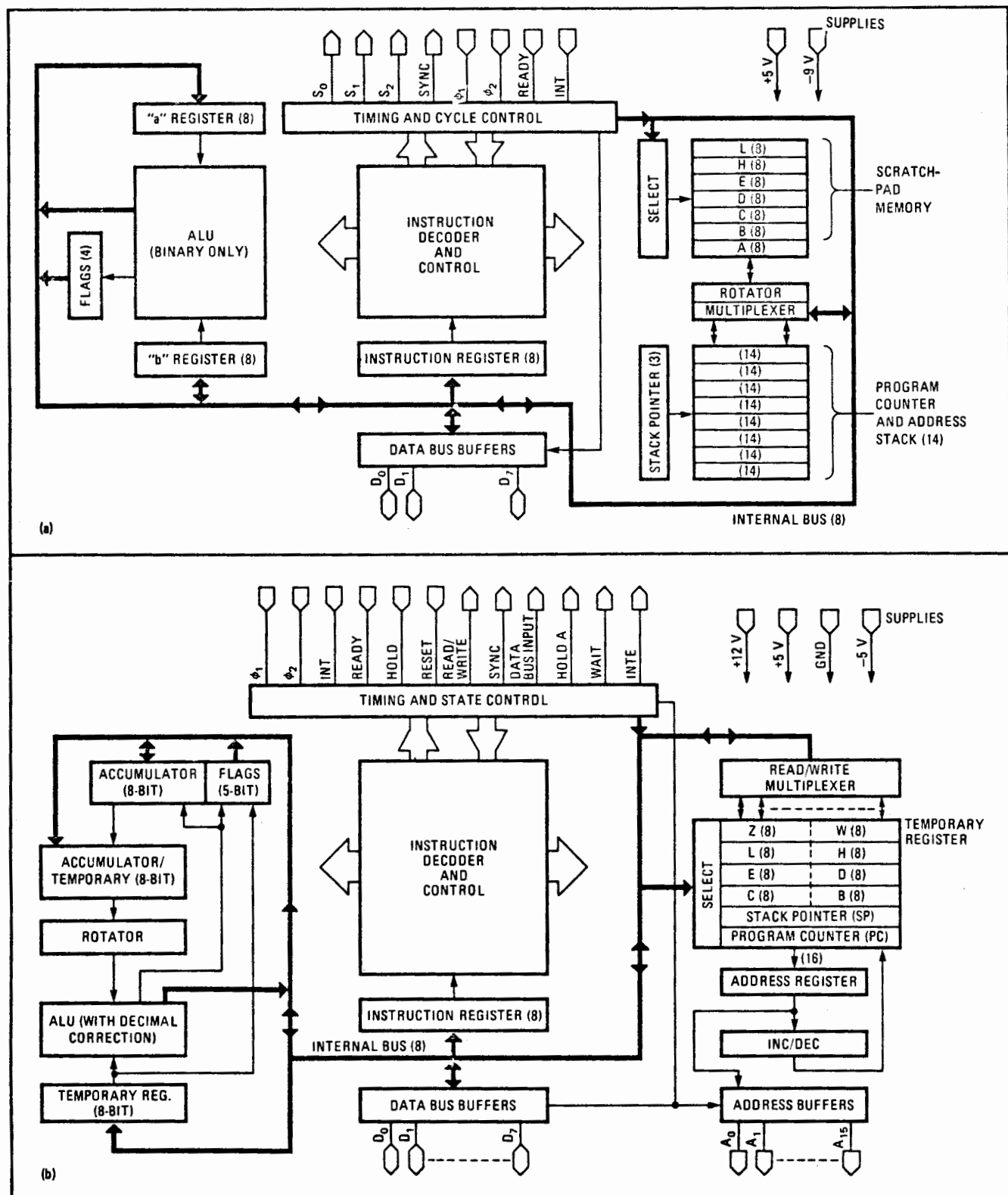
by Masatoshi Shima and Federico Faggin,
Intel Corp., Santa Clara, Calif.

□ The first microprocessors borrowed many desirable architectural features from minicomputers—but not their speed. The 8-bit Intel 8080, however, achieves typical execution times of 2 microseconds, which are comparable to those of many of today's minis.

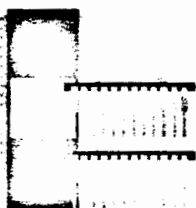
In so doing, it improves on the speed of its predecessor, the Intel 8008, by an order of magnitude or more, and, since it also adds 30 new instructions to the 48 shared with the 8008, it can be considered the start of a second, more powerful generation.

The key to both improvements is the shift from the 8008's p-channel MOS technology to n-channel MOS. Indeed, the decision to develop the 8080 was taken about 18 months ago, as soon as high-volume production of silicon-gate n-channel devices was feasible. The goal was a single-chip central processing unit (CPU) compatible with but markedly superior to the earlier 8008. The 8080's characteristics were to include:

- A 10:1 speed improvement over the 8008.
- None of the known limitations of the 8008 (such as interfacing problems and lack of multiple interrupts).
- Improved functional capability plus retention of all



1. Comparison. Intel's earlier single-chip microprocessor, the 8008, has a separate scratch-pad memory and address stack (a). In the 8080, these have been combined into the six 16-bit registers (b). The accumulator has been moved into the arithmetic and logic unit, avoiding the use of the internal bus for data transfers between the scratch pad and the ALU during arithmetic and logic operations.



of the various features and instructions of the 8008.

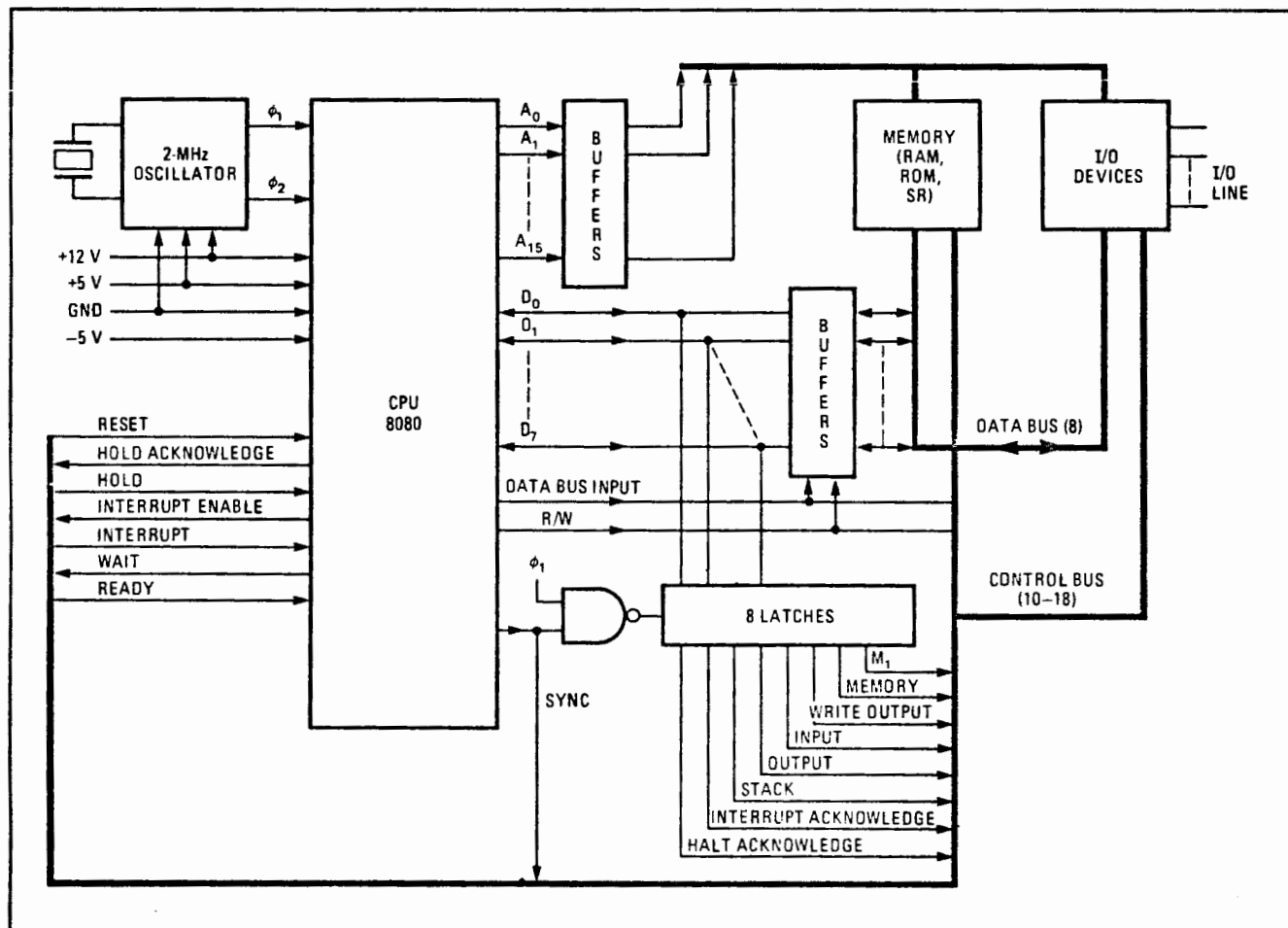
■ Economic feasibility—a small chip and conventional packaging.

If the higher mobility of electrons versus holes were the only difference between the n-channel and p-channel technologies, only a 2.4:1 improvement in speed could have been expected. But n-channel's lower threshold allows use of a 5-volt supply for internal logic, with a 4:1 improvement in speed-power product.

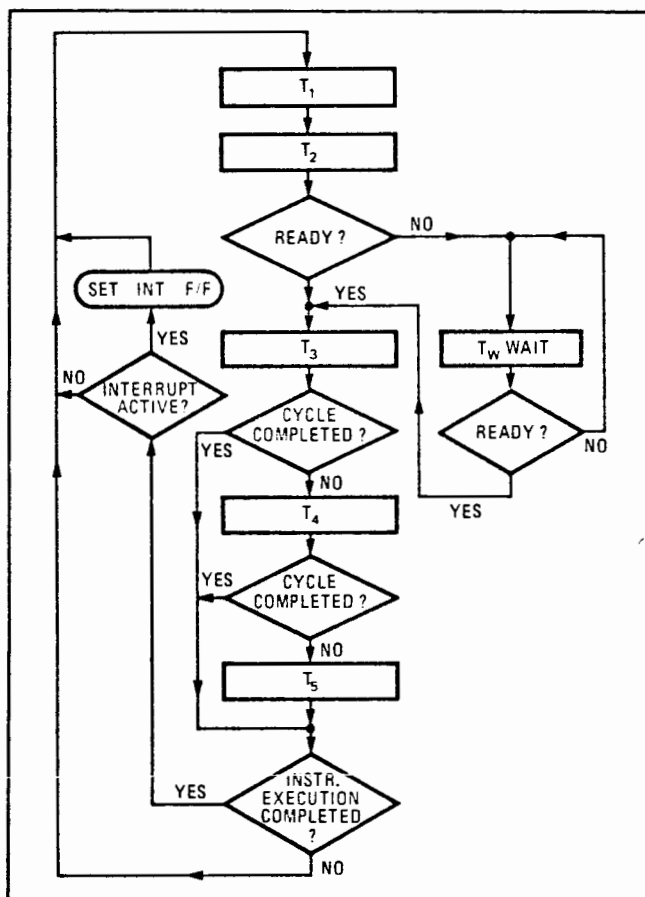
There are other contributions to higher speed. The higher substrate concentration of the n-channel starting material, combined with the lower supply voltage, allows channels to be shorter than with p-channel technology, so that input capacitance is lower and size smaller. Finally, lower junction capacitances and lower resistivities of diffusion and polysilicon areas, which result from the n-channel process and the use of substrate bias, reduce the interconnection time constants by a factor of four—and, in logic circuits of this type, one of the limiting speed constraint lies in the electrical properties of the interconnections.

The interfacing requirements were simplified because n-channel technology allows a big reduction of the power dissipation of individual output buffer circuits, so that the 8080 could be packaged in a 40-pin package to include 30 buffers as against the 12 of the 8008. In the 8008, each output buffer sinks two low-power TTL loads (440 microamperes) for a total dissipation of 250 milliwatts. Eight of the 8008 buffers are shared (time-multiplexed) for addresses and data outputs, reducing the number of package pins but increasing the complexity of the interface. The 8080's 30 output buffers, on the other hand, are six times faster, sink 1.9 milliamperes each, and dissipate a total of 150 mw. The 100 mw saved was used to improve the speed of the internal circuits (a 40-pin ceramic package allows a maximum dissipation of about 750 mw, so the power budget was limited).

The layout effort took 18 man-months because it required great care to minimize parasitics and to optimize signal flow for increased speed and smaller size. The result was a 165-by-191-mil chip that is smaller than



2. 8080 at work. When connected in a microprocessor system, the 8080 requires only six external TTL packages, as against the 20 needed by the 8008. The address bus can access up to 64 kilobytes of memory and up to 256 input and 256 output ports.



3. State diagram. A typical machine cycle requires three to five states. The operation is basically as follows: during T_1 , the content of the internal address register is sent to the address bus; during T_2 , $READY$ is tested; during T_3 , data is transferred between the CPU and memory or I/O devices; T_4 and T_5 are used when necessary to complete the instruction execution; and finally, the circuit goes back to T_1 for the next machine cycle. Only after the last state of the last machine cycle is the interrupt request line tested.

many of today's single-chip calculators. Great effort went into both logic and circuit design minimization, as a result of which the complex functions of the 8080 were implemented in only about 5,000 transistors.

The effectiveness of a CPU can be measured by its execution speed and memory storage requirements for a representative class of practical benchmark programs. The real improvement in performance is highly dependent on specific applications. If the 8080 had only the same 48 instructions as the 8008, it would handle the same problems about eight times faster (and five times faster than the 8008-1, a high-speed version of the 8008). However, with the 30 new instructions, the 8080 offers speed improvements on the order of 10:1 to 20:1 with smaller storage requirements—from 95% to 70% for an equivalent program written for the 8080.

The internal organization of the 8080 is shown in Fig.

1b, while Fig. 1a shows the same detail for the 8080.

The most important change concerns the internal memory organization. The 8008 has separate memories: an address stack—eight 14-bit registers which comprise one program counter storing the current effective address and seven others that store the addresses of nested subroutines—and a scratch pad, which contains the 8-bit accumulator and six additional 8-bit registers used for memory addressing and temporary storage of operands. In the 8080, these memories have been combined into a single internal 16-bit-wide memory with paired 8-bit register organization. The 8080's program counter and stack pointer, also each 16 bits wide, replace the 8008's internal address stack.

The 8008 has an internal 3-bit stack pointer, which gives the user up to seven levels of nesting of subroutines. The 8080's 16-bit stack pointer can address up to 64 kilobytes of external stack memory, providing essentially as many nesting levels as needed.

The 8080's accumulator and its associated circuitry have been moved into the arithmetic logic unit (ALU) section, to speed up the operation of the processor (data transfers between memory and ALU on the internal data bus are therefore not required for arithmetic and logic operations). Notice that the 8080 memory is double-ended—information can be transferred from the internal bus 8 bits at a time, while 16-bit transfers can take place from the address register.

Extra benefits

This organization yields a number of other new features for the 8080. The most important are:

- New instructions allow the contents of any register pair (B-C, D-E, H-L, or ACCUMULATOR-FLAGS) to be quickly stored and retrieved by being "pushed into" or "popped from" the top of the external memory stack. This is a fast way to save the machine status (the contents of the registers) when an interrupt occurs and then restore the status after the interrupt has been serviced. The stack can also be used as an extension of the internal registers.
- Other new instructions allow easy manipulation of addresses and the memory stack, since the registers B-C, D-E and H-L, and STACK POINTER can be incremented and decremented with 16 bits in parallel.
- The temporary register pair W-Z can be used as a program counter to hold a direct address to quickly load or store H-L or ACCUMULATOR. Also possible are double precision additions between any register pair and H-L.
- Fast, parallel transfers of H-L to PROGRAM COUNTER or STACK POINTER are now possible with a minimum amount of internal control logic.
- The addition of decimal correction to the ALU section enables binary and BCD arithmetic to be performed at about equal speeds.
- The addition of many new, easy-to-use control and

The 8080's Inputs and outputs

The Intel 8080 takes four control inputs and generates six control outputs:

SYNC—output; a synchronizing signal that indicates the beginning of each memory cycle.

DATA BUS INPUT—output; a signal that indicates when the data bus is in the receiving mode, i.e. when data is expected by the CPU.

READY—input; a signal to the CPU that valid data is available. If not activated, the CPU enters a WAIT state.

WAIT—output; a signal that acknowledges that the CPU is in the WAIT state.

WRITE—output; a signal that tells the memory and output devices that valid data from the CPU is available on the data bus.

HOLD—input; a signal used by an external device to request access to the CPU address and data bus. Request is granted upon completion of memory access and it is acknowledged on the HOLD ACKNOWLEDGE output pin. The CPU address and data buses become floating (in a high-impedance state), but internally, the CPU completes the execution of the current memory cycle. After that, the CPU idles for as long as HOLD is active. HOLD and HOLD ACKNOWLEDGE can be used for DMA (direct memory access) control and in multiprocessor applications.

HOLD ACKNOWLEDGE—output; signals acknowledgment of the HOLD state.

INTERRUPT REQUEST—input; the interrupt input is sampled at the end of the current instruction cycle and if the internal software control interrupt enable flip-flop is set, it initiates the interrupt servicing sequence.

RESET—input; a signal that clears the content of the program counter so that program execution will start from location zero in memory.

INTERRUPT ENABLE—output; a signal that displays the status of the interrupt enable flip-flop.

The CPU also provides eight status bits on the data bus at SYNC time:

HALT ACKNOWLEDGE—a response to the HALT instruction.

INTERRUPT ACKNOWLEDGE—follows the acceptance by the CPU of an interrupt request.

INPUT CYCLE—indicates that the address bus holds the address of an input device.

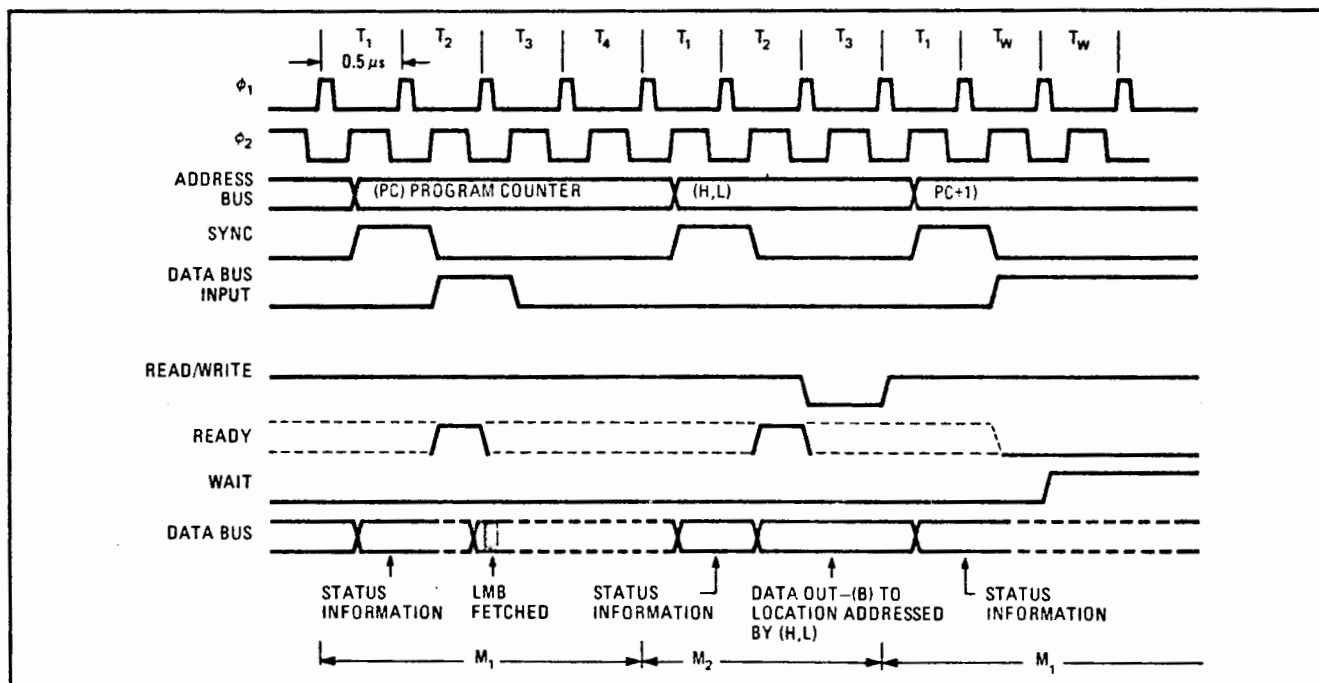
OUTPUT CYCLE—indicates that the address bus holds the address of an output device.

MEMORY READ—indicates that the data bus will have data coming from memory.

M₁—indicates that the current cycle is for fetching the first byte of an instruction.

STACK—indicates that the address bus holds the push-down stack address.

WRITE OUTPUT—indicates that the data bus will have data for memory write or for an output operation.



4. Timing. The instruction LMB (load the content of register B into the location addressed by the contents of registers H and L) requires two machine cycles (M₁ and M₂). During M₁, the address bus holds the program counter contents, and during M₂ it holds the contents of the H-L register pair. DATA BUS INPUT shows when the CPU expects data from the data bus. WRITE shows when data from the CPU is available on the data bus. READY shows that valid data is available to the CPU. The bottom waveform shows the corresponding data-bus actions.

status signals simplifies interfacing, allows direct memory access, and helps in program debugging.

Figure 2 shows how the 8080 interfaces with outside chips to make a microprocessor system. An external crystal-controlled oscillator supplies two non-overlapping clocks, ϕ_1 , and ϕ_2 . Buffers interface to external address and data buses, and a gate and eight latches set up status bits during sync time. All inputs and outputs are TTL-compatible, with the exception of the two clocks, which require +12 V. A memory and the input-output devices complete the system.

The amount of external interface logic necessary to implement any system depends on that system's complexity. The minimum requirement is six packages of conventional TTL (the 8008 needs at least 20).

External signals are organized on three buses. An address bus with 16 lines addresses up to 65 kilobytes of memory and up to 256 input and 256 output ports. A bidirectional eight-line data bus carries data to and from memory and I/O ports. A control bus synchronizes

	INTEL 8008	INTEL 8080
Technology/threshold voltage	p/1.5–2.5 V	n/0.8–1.4 V
Supply voltage	+5, –9 V	+12, +5, 0, –5 V
Number of pins on package	18	40
Number of interface chips	20	6
Number of instructions	48	78 (48 + 30)
Instruction execution speed	12–22 μ s	2–9 μ s
Internal memory type/number of bits	dynamic/168 bits	static/104 bits
Chip size (mils)	124 x 173	164 x 191
RAM size/speed (typical systems)	256/1 μ s	1,024/500 ns
ROM size/speed (typical systems)	2,048/1 μ s	4,096/600 ns

the CPU, external memory, and I/O devices, and also has the job of handling interrupts, direct-memory-access (DMA) controls, and CPU status information.

Instructions in the 8080, as in the 8008, use one, two, or three bytes of storage. Each instruction requires from one to five machine (or memory) cycles for fetching and execution. Machine cycles are called M_1 , M_2 , . . . , M_5 . Each machine cycle requires from three to five states— T_1 , T_2 , . . . , T_5 —for its completion. Each state has the duration of one clock period (0.5 microsecond). There are three other states (WAIT, HOLD, and HALT) which last one to an indefinite number of clock periods, as controlled by external signals. Machine cycle M_1 is always the operation-code fetch cycle and lasts four or five clock periods. Machine cycles M_2 , M_3 , M_4 , and M_5 normally last three clock periods each.

To understand the basic operation of the 8080, let's refer to the simplified state diagram shown in Fig. 3, starting at cycle M_1 and state T_1 .

During T_1 the content of the program counter is sent

to the address bus, SYNC is true, and the data bus has status information pertaining to the cycle that is currently being initiated. T_1 is always followed by another state, T_2 , during which the condition of the READY input is tested. If READY is true, T_3 is entered; otherwise, the CPU will go into the wait state (T_w) and stay there for as long as READY is false. READY thus allows the CPU be synchronized to a memory with any access time and to any I/O device. Also, by controlling the READY line, the user can single-step through his program.

During T_3 , the data coming from memory is available on the data bus and is transferred into the instruction register (during M_1 only). The instruction decoder and control sections then generate the basic signals to control the internal data transfers, the timing, and the machine-cycle requirements of the new instructions.

At the end of T_4 , if the cycle is complete, or else at the end of T_5 , the 8080 goes back to T_1 and enters machine cycle M_2 , unless the instruction required only one machine cycle for its execution. In such cases, a new M_1 cycle is entered. The loop is repeated for as many cycles and states as required by the instruction.

It is only during the last state of the last machine cycle that the interrupt request line is tested and a special M_1 cycle is entered, during which no program-counter incrementing takes place and INTERRUPT ACKNOWLEDGE status is sent out. During this cycle, one of eight possible single-byte calls will be sent to the CPU by the interrupting device.

Execution times

Instruction state requirements range from a minimum of four states for non-memory referencing instructions, like register and accumulator arithmetic instructions, up to a maximum of 18 states for the most complex instructions—such as XTHL (exchange the contents of registers H and L with the content of the top two locations of the stack). At the maximum clock frequency of 2 megahertz, this means that assembly-language instructions can be executed in 2 to 9 μ s.

As an example of 8080 timing, Fig. 4 shows the timing diagram for the one-byte, two-cycle instruction LMB (load the content of register B into the memory location addressed by the contents of registers H and L). This example also illustrates the timing when a WAIT state is entered after the execution of LMB. Notice that seven states (a total of 3.5 μ s) are required to fetch and execute the LMB instruction. The same instruction would require 28 μ s by the 8008, 17.5 μ s by the faster 8008-1.

Though this example demonstrates an 8:1 improvement in speed over the 8008, the real impact of the new 8080 will not be as a replacement for the 8008. The 8008 has, after all, adequate speed for a large number of applications. The 8080 will be used in new systems that were not feasible before because the first-generation microcomputers were not powerful enough. \square

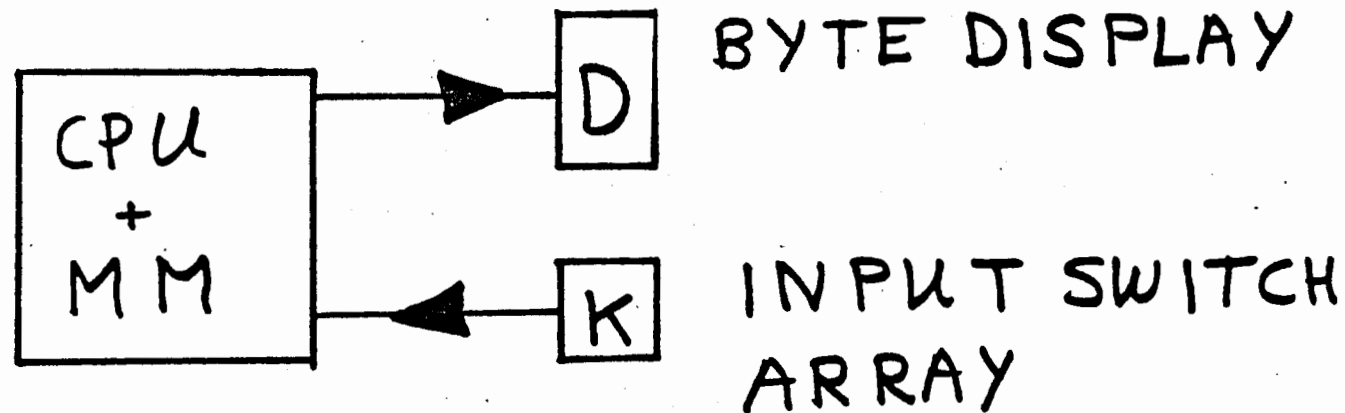
OUTLINE

3^c

- ONE LEVEL I/O STRUCT.
- TWO LEVEL I/O STRUCT.

SOFTWARE/
HARDWARE

ONE LEVEL I/O EXAMPLE



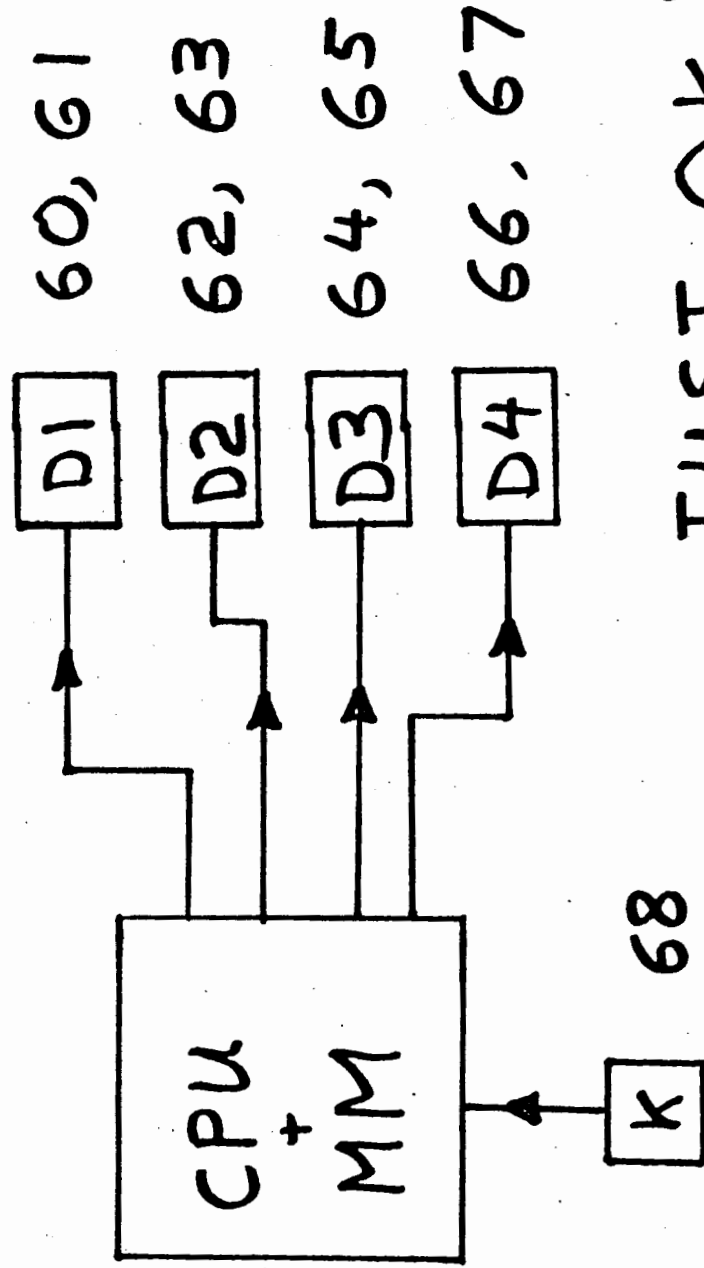
DATA → DISPLAY (FROM MM)
TURN DISPLAY ON/OFF
DATA → MM VIA SWITCHES

LET

60 OUTPUT BYTE TO D
61 TURN D ON/OFF
68 INPUT DATA FROM K

ALL IS WELL!

NOW WANT 4 BYTE D'S



JUST OK!

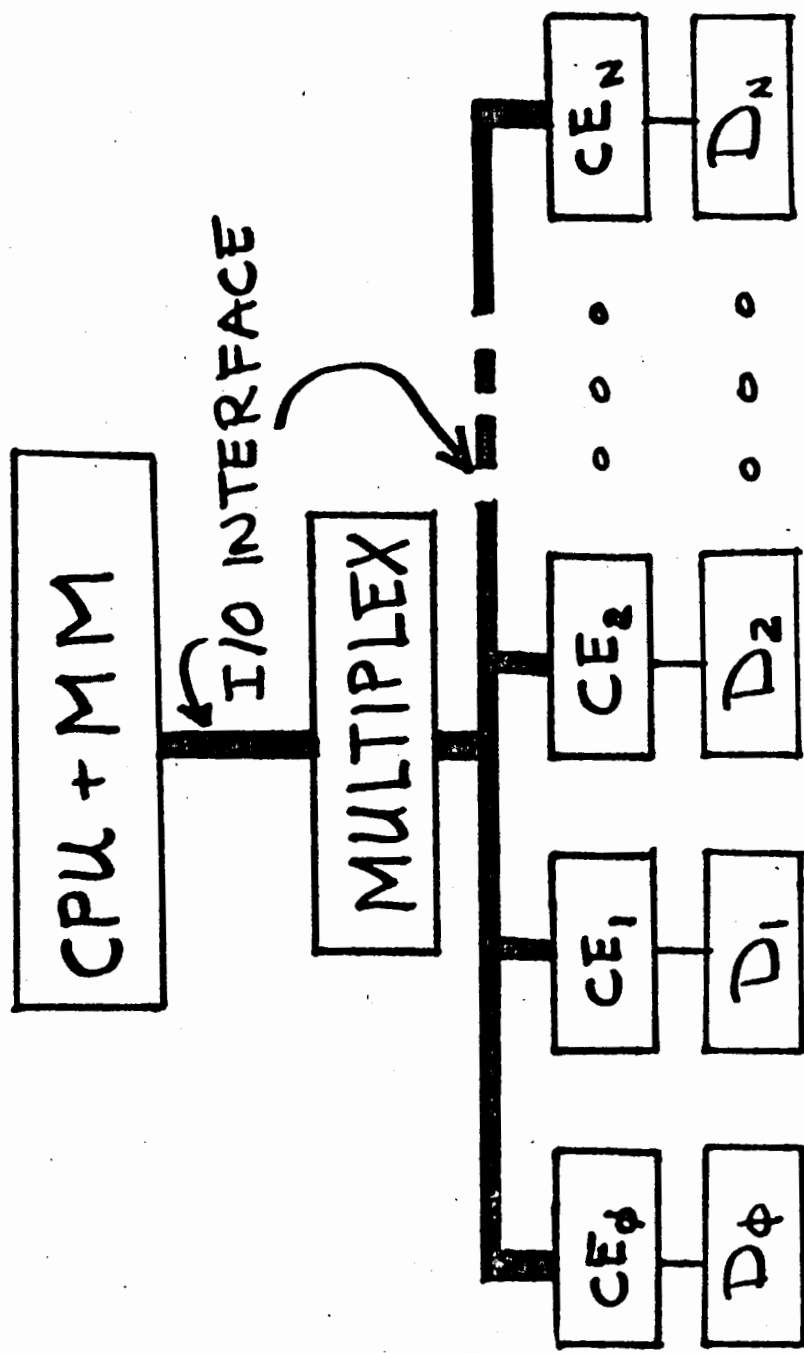
WANT TO ADD DISC

- HEAD CONTROL
- READ/WRITE INFO.
- START/STOP INFO.
- DATA IN/OUT

CANNOT USE ONE-LEVEL
OF I/O

TWO LEVEL I/O

ENTIRE COSMAC I/O INTER-
FACE MADE AVAILABLE TO
EACH PERIPHERAL DEVICE



CE : CONTROL ELECTRONICS

D : DEVICE (DISC, DISPLAY, ETC)

SELECT INSTRUCTION

DEFINE 6 ϕ AS SELECT INST.

(COULD HAVE CHOSEN ANY ONE
OF 6 ϕ -67 AS SELECT INST.)

SELECT POINTS AT DESIRED
PERIPHERAL DEVICE

WHEN 6ϕ IS EXECUTED,
 $M(R(x))$ IS PUT ON BUS AND
IDENTIFIES ONE OF 256
POSSIBLE DEVICES WHICH
IS THEN CONNECTED TO
COSMAC I/O INTERFACE

RECALL

WHEN 60-67 INSTRUCTIONS
ARE EXECUTED

$M(R(X)) \Rightarrow BUS$

IMPORTANT

MULTIPLEX AND INTERFACE LOGIC
MUST ENSURE THAT ONLY
ONE DEVICE IS CONNECTED
TO COSMAC I/O INTERFACE
AT ANY ONE TIME

(INTERRUPT IS EXCEPTION)

EXAMPLE 3 DEVICES

DISPLAY : DEV # 03

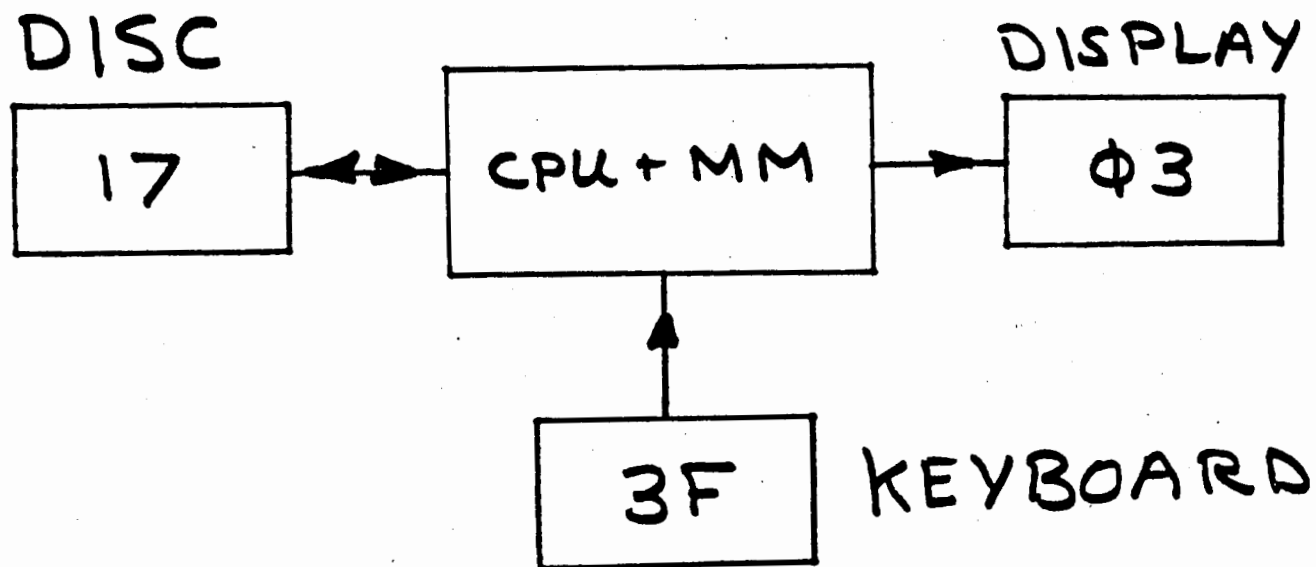
DISC : DEV # 17

KEYBOARD : DEV # 3F

60 WITH $M(R(X)) = 03$ SELECTS
DISPLAY,

60 WITH $M(R(X)) = 17$ SELECTS
DISC,

ETC.



SELECTING ONE DEVICE
AUTOMATICALLY DESELECTS
OTHER DEVICES !!!

ALL I/O INSTRUCTIONS (61-6F)
SUBSEQUENT TO A SELECT
INSTRUCTION ARE SEEN ONLY
BY SELECTED DEVICE.

* Deselected devices CAN ALL
BE ON SIMULTANEOUSLY (I.E.,
TALKING TO OUTSIDE WORLD),
BUT ONLY SELECTED DEVICE
CAN COMMUNICATE WITH CPU
VIA I/O INSTRUCTIONS!

EXAMPLE

DISPLAY

DISC

61: BYTE OUT

RESET DSC.

62: TURN ON

INPUT ADDR.

63: TURN OFF

START I/O

KEYBOARD

DISC

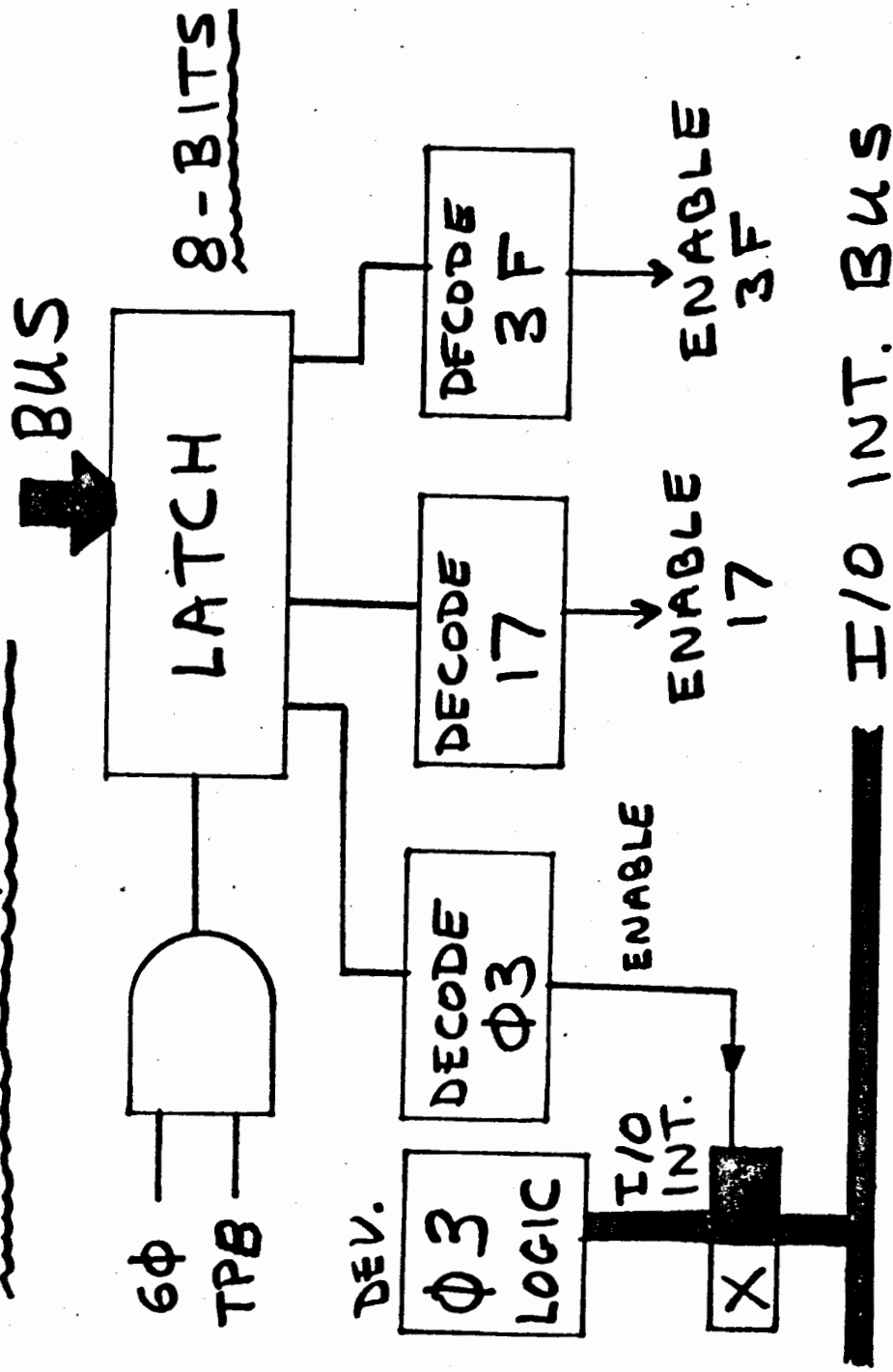
EF3: BYTE RDY

I/O DONE

CODE

M(R(X))		ACTION
60	$\phi 3$	SELECT DISPLAY
61	DATA	DISPLAY DATA
60	17	SELECT DISC
62	DISC ADR.	SET DSC ADR.
63	—	START I/O
61	—	RESET DISC

SELECT LOGIC



SUMMARY

- 6 ϕ SELECT INSTRUCTIONS
- REMAINING INSTRUCTIONS
61-67 OUTPUT
68-6F INPUT
- USUALLY USE DEV. # $\phi\phi$
FOR DESELECTING ANY
SELECTED DEVICE.
i.e., 6 ϕ WITH MER(x) = $\phi\phi$
 \Rightarrow NO DEVICE SELECTED

I/O ARCHITECTURE

I/O IMPORTANCE

- KEY TO APPLICATION
- AFFECTS SOFT. / HARD.
- ONLY LINK WITH OUTSIDE WORLD

ASSUME

COSMAC I/O INTERFACE

SYSTEM DESIGN STEPS

1. SPECIFY FUNCTIONS/PERF.
2. SELECT I/O DEVICES
3. DEFINE I/O STRUCTURE
4. DESIGN DEVICE INTERFACES
5. DEVELOP SYSTEM SOFT.
6. SYSTEM TEST/DIAGNOSTICS

SYSTEM FUNCTIONS/PERF.

- MONITOR/CONTROL
- REAL TIME OPERATION
- CRITICAL RESPONSE
TIMES
- OPERATION/MAINTENANCE

SELECT I/O DEVICES

- DISCS
- TAPES
- DISPLAYS
- SENSORS
- KEYBOARDS
- BACKUP POWER

DEFINE I/O STRUCTURE

- DEVICE I/O INTERFACES :
CPU vs HARDWARE CONTROL
- I/O INSTRUCTIONS
- FLAG ASSIGNMENTS
- DMA CHANNEL
- INTERRUPT

DESIGN DEVICE CE'S

NOISE/POWER/SPEED/COST

⇒ CHOICE OF LOGIC FAMILY
CMOS, TTL, DISCRETE,
ETC.

- DO LOGIC DESIGN

DEVELOP SYSTEM SOFT:

- HIGHER LEVEL OR MACHINE
CODE
- ROM OR RAM
- SYSTEM REGENERATION
- EXPANDABLE
- MODULAR

SYSTEM TESTING/DIAGNOS.

- DIAGNOSTIC PANEL
- T & M ROUTINES
- ROUTINES IN ROM?
ON DISC?
- EASE OF REPAIR (PACKAGE)

COSMAC I/O INTERFACE

- 8-BIT DATA BUS
- 15-BIT CONTROL BUS

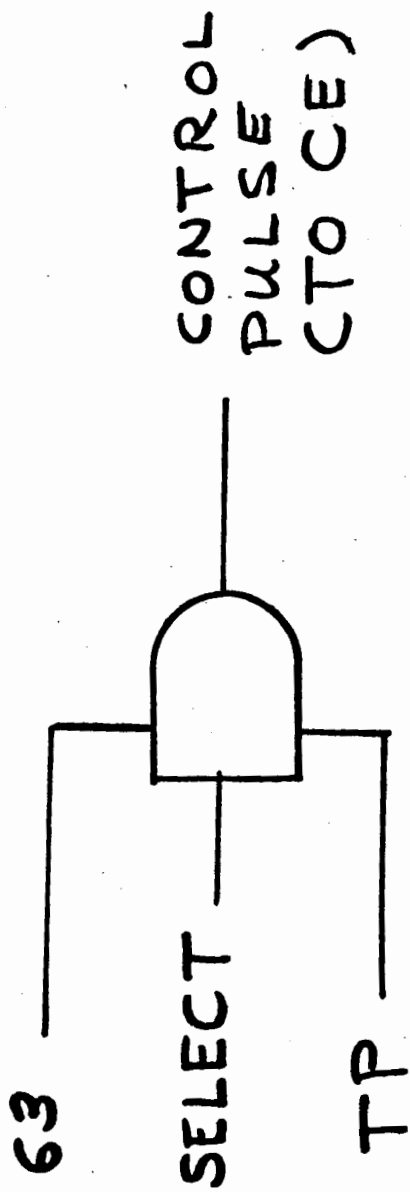
CONTROL BUS

- *• 4-BIT CODE GN: G Φ -GF
- *• 4 FLAGS, EF1-EF4
- *• 2 TP'S (TPA, TPB)
- (*)• 2-BIT STATE CODE

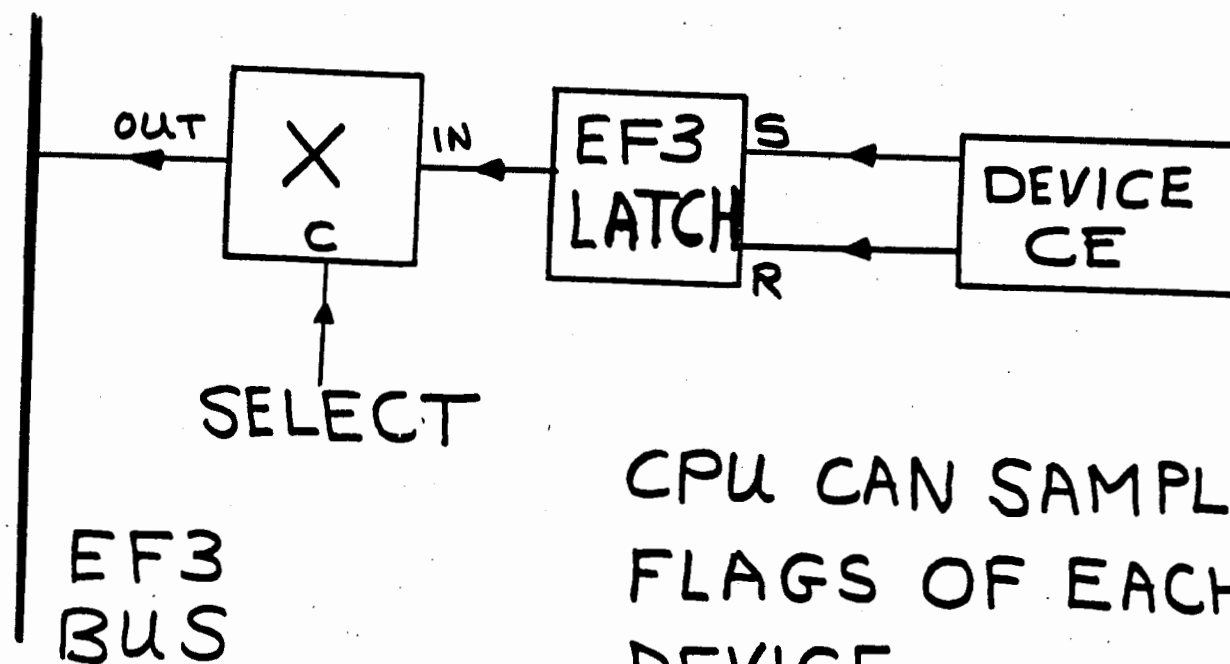
-
- DMA INREQUEST
 - DMA OUTREQUEST
 - INTERRUPT

* GATED WITH SELECT

4-BIT CODE



FLAGS



CPU CAN SAMPLE
FLAGS OF EACH
DEVICE

MACHINE STATES

S0 FETCH

S1 EXECUTE

S2 DMA

S3 INTERRUPT

STATE CODE

BIT		
1	Φ	
Φ	Φ	$S\Phi + S1 \cdot \overline{I6}$ (FETCH OR NON-I/O EXECUTE)
Φ	1	$S1 \cdot I6$ (I/O EXECUTE)
1	Φ	$S2$ (DMA)
1	1	$S3$ (INTERRUPT)

NOTE: "1" \Rightarrow LOW SIGNAL

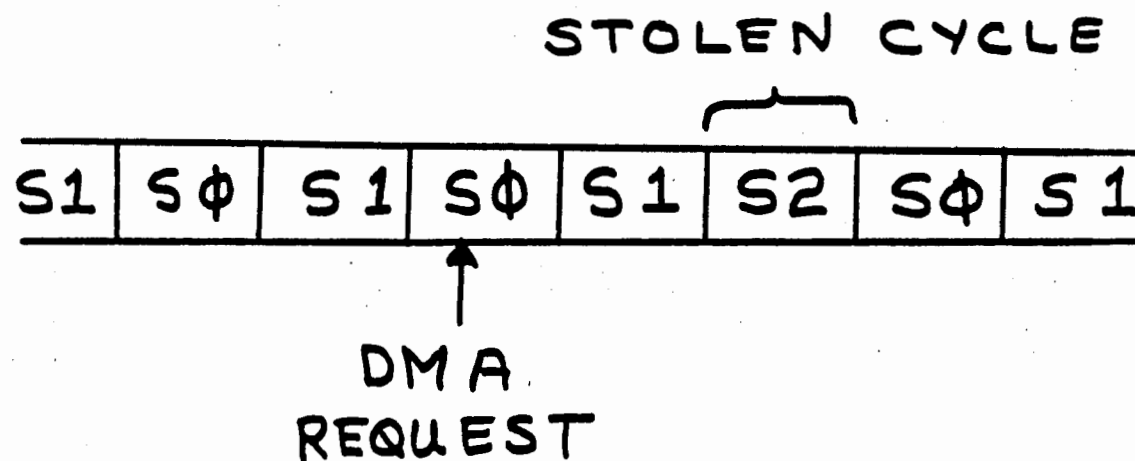
DMA CHANNEL

- RΦ DEDICATED TO DMA
- CYCLE STEALING
(S2 CYCLE)
- ONLY ONE DEVICE AT-A-TIME
(TDM)
- TRANSPARENT TO PROGRAM
EXECUTION
- CONTROLLED BY DEVICE

USING DMA

- DEV. NEED NOT BE SELECTED
- DEV. SETS INREQ OR OUTREQ
 - CPU EXECUTES S2 CYCLE
 - a) $M(R\phi) \rightarrow BUS$ OR
 $BUS \rightarrow M(R\phi)$
 - b) $R\phi$ INCREMENTED
 - c) RESETS IN/OUTREQ FF
- DEV. MUST CLOCK DATA BYTE
ON/OFF BUS

DMA TIMING



DMA USES

- BLOCK TRANSFER
- HIGH SPEED XFER

INTERRUPTS

- DEVICE INITIATED
- CAN BE ENABLED OR DISABLED (IE BIT IN CPU)
- USE R1, R2, T REGISTER
- DEVICE NEED NOT BE SELECT.
- SOFTWARE PRIORITY FOR MULTIPLE INTS.

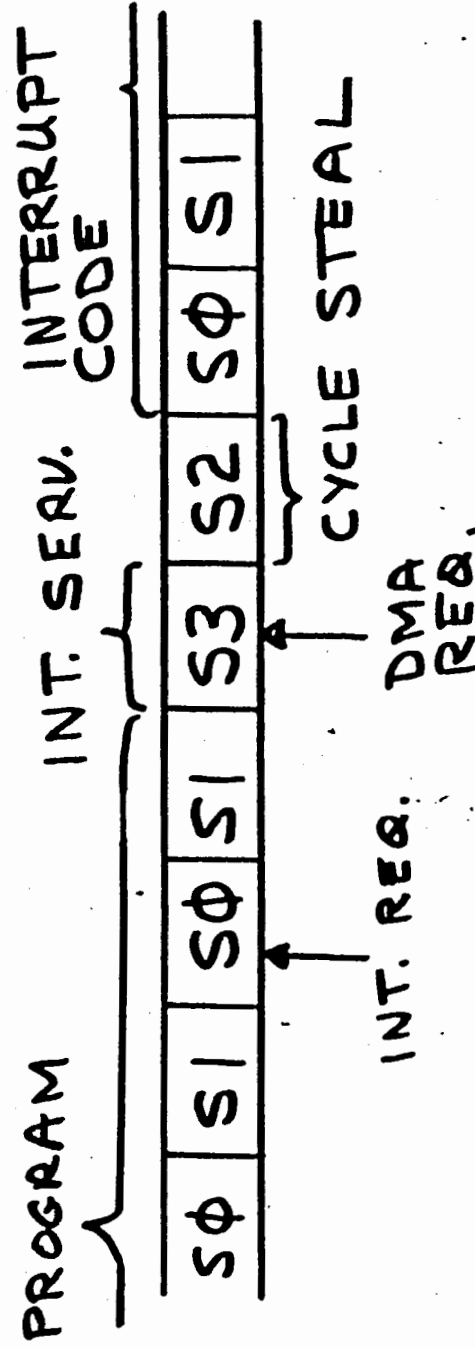
IE BIT

- INTERNAL TO CPU
- $IE = 0 \Rightarrow$ INTS. DISABLED
- $IE = 1 \Rightarrow$ INTS. ENABLED
- IE SET TO 0 BY "DISABLE"
CPU INST. $\& \underline{\underline{S3}}$
- IE SET TO 1 BY "RETURN"
CPU INST. $\& \text{CPU "CLEAR"}$
- IE USED TO MASK INTS.

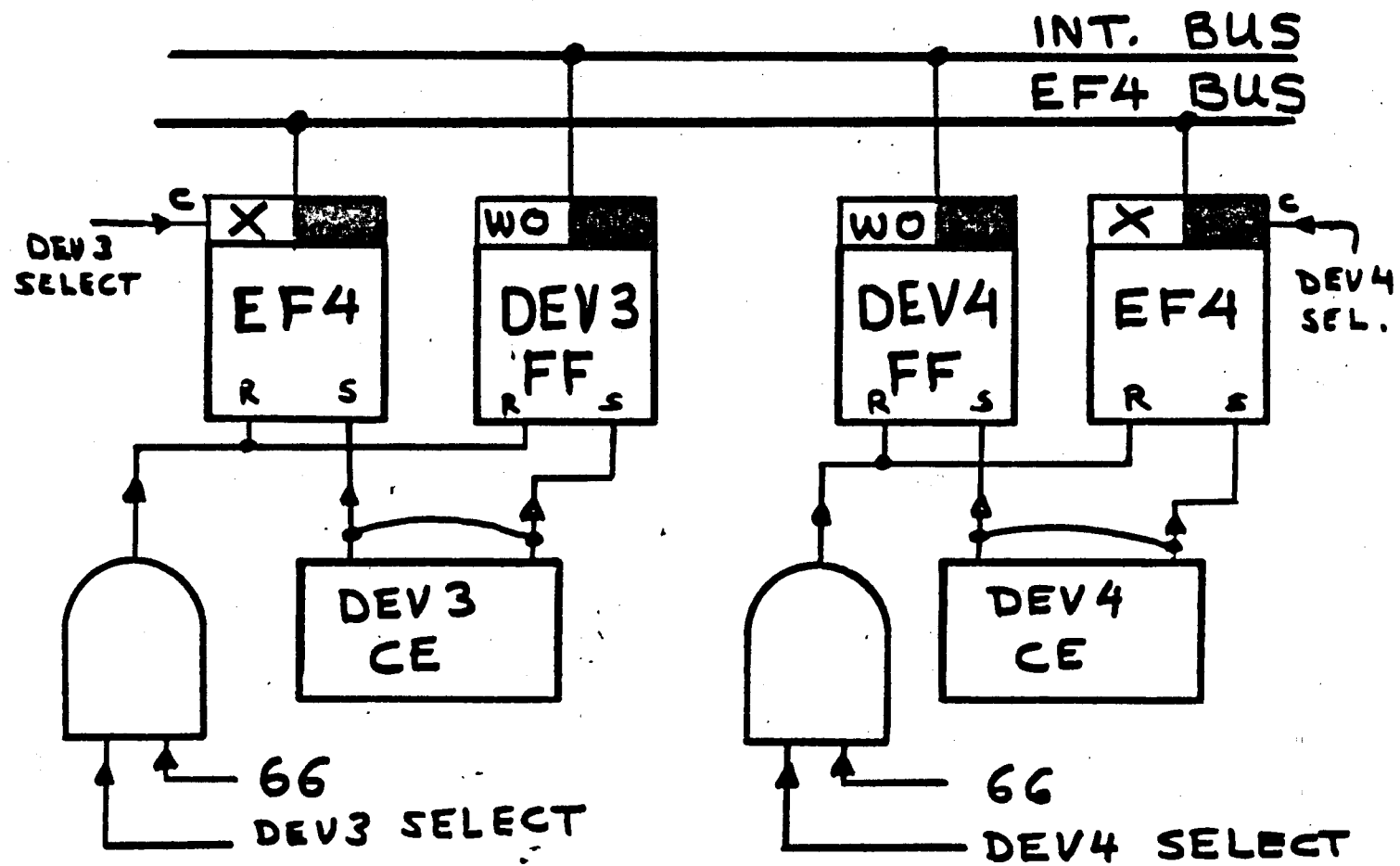
INT. REQ (IE=1)

- CPU EXECUTES S3 CYCLE
 - P, X STORED IN T
 - IE = Φ
 - NEW P = R1, NEW X = R2
- EITHER S3 CYCLE OR GN INST. USED TO RESET INT. FF.

S3 TIMING



INT. HARDWARE

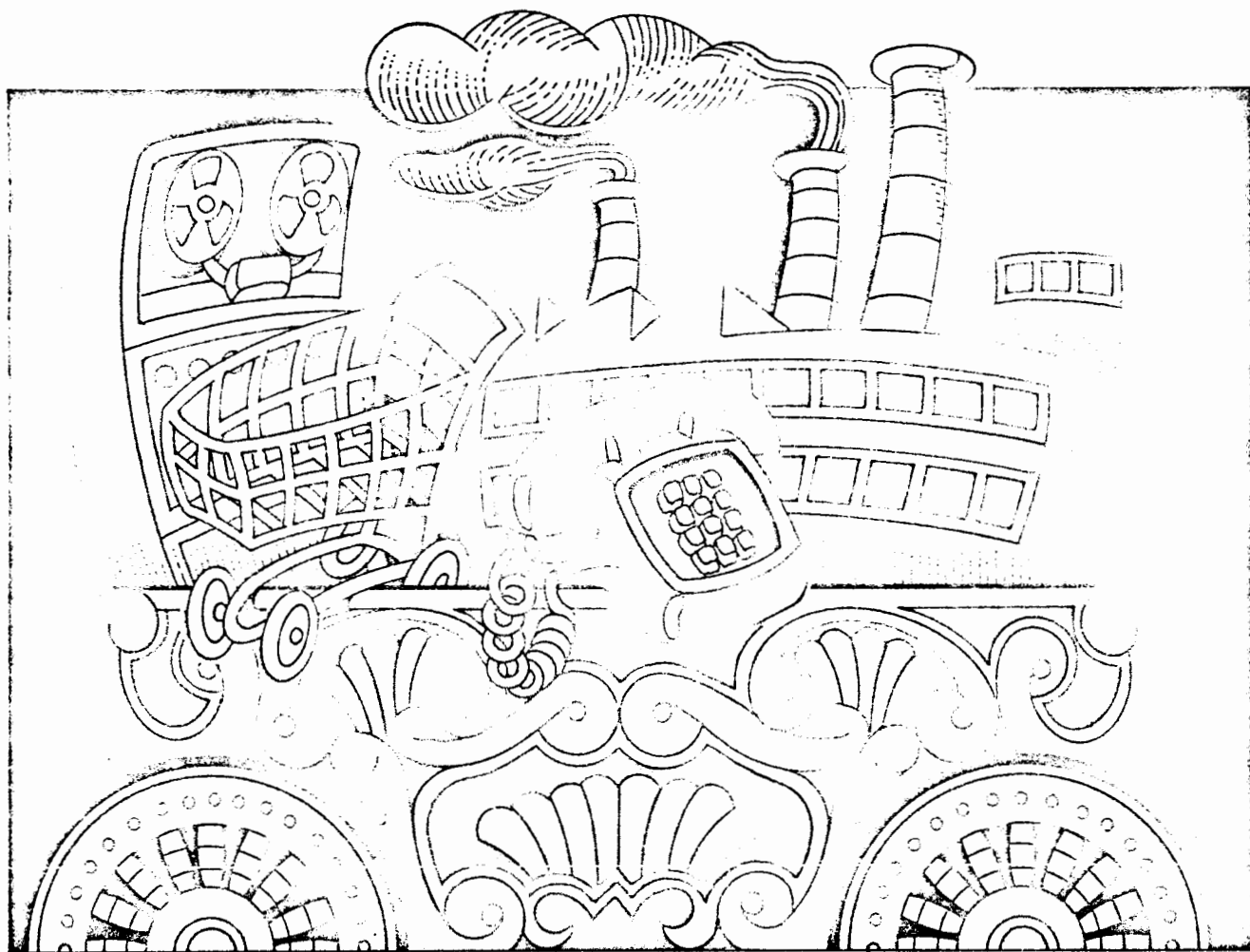


I/O TO/FROM CPU

- BYTES TRANSFERRED BETWEEN CPU AND DEVICES IN ONLY TWO WAYS : a) PROGRAM MODE (GNJ) G) DMA
- ONE LEVEL I/O BEST FOR SMALL SYSTEMS. NEED TWO-LEVEL I/O FOR MORE COMPLEX APPLICATIONS

TWO LEVEL I/O

- ONLY SELECTED DEVICE INTERPRETS 6N INSTS.
- DEVICE USING DMA NEED NOT BE SELECTED
- ANY DEVICE CAN INTERRUPT CPU - USE FLAGS TO IDENTIFY IT. INTERRUPT PRIORITY IN SOFTWARE



Diverse industry users clamber aboard the microprocessor bandwagon

LSI processors are not only expanding
capabilities of traditional products
—from instruments to consumer wares—
they're also creating completely new markets

□ Industrial-equipment designers like them because they can be tailored economically to bring computer capability to jobs where mini-computers represent overkill.

Communications-gear designers are enthusiastic because their flexibility can solve problems presented by the ever-changing multiplex and modem specifications.

Instrument designers are looking forward to making them the basis of families of "smart" instruments that can evaluate data and react accordingly, without boosting instrument costs significantly. And even computer manufacturers are eyeing them as perfect companions to their TTL-based central-processor modules.

It's no wonder, then, that microprocessors are engaging the attention of equipment designers of all persuasions and manufacturers from a wide variety of industries. As a result, the growth of microprocessors is projected to leap from last year's \$10 million to \$800 million in the next five years. More dramatic yet will be the increase in the value of new end equipment built around LSI processors, expected to exceed a staggering \$10 to \$15 billion a year by the end of the same period.

What has caused the sudden microprocessor boom? Simply stated, LSI technology has reached the level of sophistication where it can provide the logic and memory performance needed to perform a growing number of computer functions at low cost. Programmable LSI circuits—the calculator was the first—combine the flexibility of custom design with the cost advantages of readily available standard products. The user can change his design or add features to it merely by changing a program in a read-only memory. No mask changes are needed. And he is saving money by replacing many dozens of logic packages with a few LSI chips.

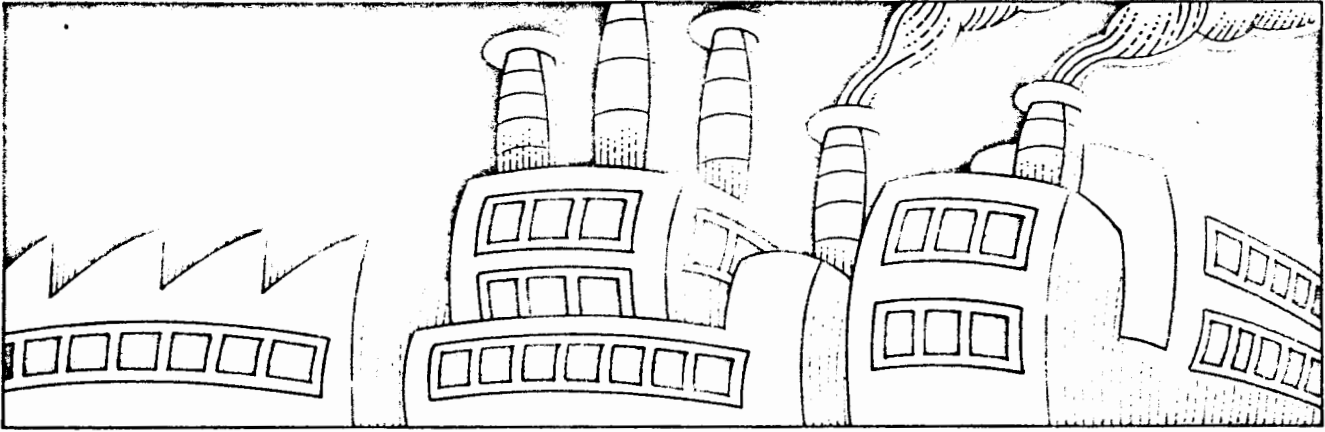
Impressive as today's microprocessors are, they are only the most visible aspect of what is clearly becoming an LSI-processor revolution that will completely change computer and computer-control design. Today's LSI processors are at the capability level of the small and not-so-small computer. But more powerful LSI-processor and computer-component chips that are now starting to appear far exceed the requirements of today's microcomputer applications.

Built with bipolar and improved MOS techniques, these faster and more complex components go to the heart of minicomputer-based systems, nourishing more and more equipment-design applications. These are the LSI programmable chips computer manufacturers themselves have been waiting for. At last, the full benefits of LSI programmable technology can be applied to the large computer, ushering in a new era of high-performance computer control at lower cost.

These articles bring together the experiences of the first microprocessor users—the promises and problems of designing with this powerful technique. The entire range of electronic-equipment designs has been researched—industrial, communications, consumer, commercial, instrumentation, and computer technology. Included are details on such varied systems as process and numerical controllers, word processors, data loggers, communications controllers, intelligent terminals, point-of-sale systems, games, toys, advanced calculators, self-calibrating instruments, automobile controls, and all the rest.

Also included is a section that contains tips on software and design aids. Finally, Bill Davidow, manager of microcomputer systems for Intel Corp., adds up the design advantages of microprocessor-based systems to show their impact where it counts most—on the bottom line.

—Laurence Altman, Senior Editor



Industrial Automatic control proliferates

by Alfred I. Rosenblatt, Associate Editor

"The microprocessor is going to set the industrial-equipment marketplace on its ear. The technology will never be the same again." That opinion was expressed by a market planner at a semiconductor house developing a microprocessor-chip set for one of the manufacturers of process-control instrumentation. The prediction is borne out by developments in the industrial marketplace. What's more, prospects for dramatic improvements are as bright for piece-parts manufacturing as for process control.

Although less than three years old, microprocessors are already finding their way into a host of new industrial equipment—factory-automation systems, machine-tool control, data-acquisition systems for such jobs as monitoring apportionment of meat for hamburgers, electronic scales, control of conveyor lines, numerical control, robot manipulation of piece parts, data-sensing, and component-insertion. They are also being used for environmental monitoring and phototypesetting.

These microprocessor-based systems offer the flexibility to adapt manufacturing systems to changing demands and upgrade them as production expands. All that is necessary is for chips containing new instructions to be inserted when peripherals are changed, equipment is added, or the system itself is modified. Changes and modifications are much more difficult when conventional hard-wired circuitry must be replaced.

What's more, manufacturers are happy about decreases in manufacturing costs that result when a relatively few microprocessor chips replace tens of discrete SSI and MSI circuits. Not only are fewer components required, but the microprocessor obviates the necessity to fabricate many more components manually into hard-wired logic arrays and insert these boards into the control systems. However, where speed is critical, hard-wired designs may do better for some time to come.

As the capabilities of microprocessors are expanded, they are taking over many of the tasks—at a pleasant reduction of costs—previously performed by minicomputers, but for which a considerable amount of the

power of minicomputers is wasted. Replacing the purchased minicomputers may also increase the amount of value added for a manufacturer in his final product with a consequent increase in profits.

Taking over the factory

The availability of powerful low-cost microprocessors is also hastening the transition to the efficient distribution of computer power through employment of hierarchical computer systems in factories. The microprocessors and microcomputers perform dedicated tasks under the control of minicomputers, and the entire complex is tied in to large central computer systems.

What's more, the microprocessor is making it possible for manufacturers of process-control equipment and systems virtually to go into computer-manufacturing. Bruce H. Baldrige, director of corporate marketing and product planning at Foxboro Corp., Foxboro, Mass., points out that microprocessors are going to seriously influence the make-or-buy decision so that "a company like Foxboro could buy a micro chip, put it on a board, and it would be putting us in the computer-manufacturing business without the expense of getting deeply involved in the technology."

The importance of the microprocessor to industry is summed up by Edwin Lee, president of Pro-Log Corp., a Monterey, Calif., systems-design firm that also offers a line of microprocessor modules, "Within 12 to 18 months, anyone who hasn't incorporated a microprocessor in his design will either be serving a very special application or he's going to be very uncompetitive, as far as hardware is concerned."

Another consultant calls this "an explosive situation—anything that's cheap and reasonably powerful changes things. Anyone doing anything with hard-wired electronics who doesn't look at and consider microprocessors is making a big mistake."

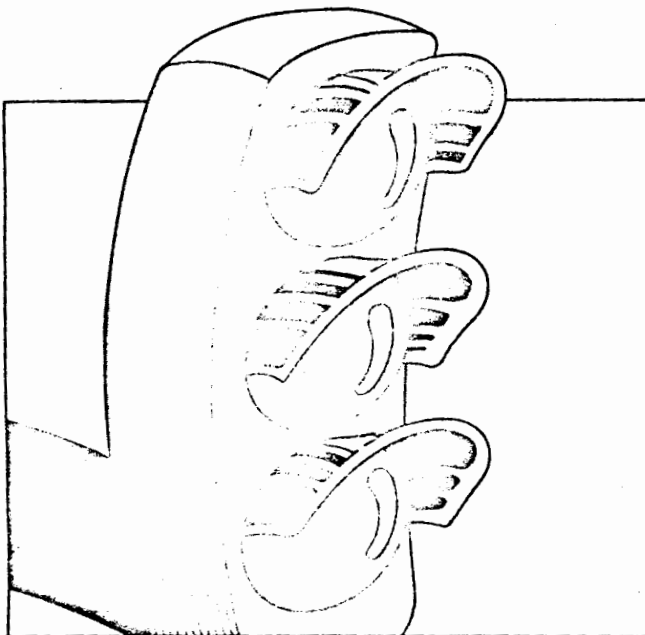
A recently completed study on factory automation by Quantum Science Corp., a New York-based industrial-research company, estimates that by 1984, industry will

Machine tool control (units/year)	Robots (units/year)	Product testing (units/year)	Facilities monitoring (units/year)	Total cost
300	10	300	150	760 @ \$1,000 each avg.
3,400	2,850	500	300	7,050 @ \$400 each avg.
7,800	14,000	1,000	600	23,400 @ \$300 each avg.
Estimates courtesy of Quantum Science Corp.				

be buying 27,300 microcomputers a year at an average price of \$300 each. Accumulated over the years, these numbers will have an incredible effect on the factory's operation, increasing the efficiency and cost-effectiveness of production. The average unit price today is \$1,000 according to the Quantum Science study.

Perhaps most unusual is Quantum's prediction that programable manipulators, or robots, will mushroom with the aid of microcomputers from 10 units installed in 1974 to 14,000 a year by 1984. About half that many—7,800 a year—are expected to be used for machine-tool control, a mammoth increase from the present base of 300 a year. And about 3,900 new microprocessors a year are predicted to handle communications between the various tools and computers in another 10 years, whereas now only 50 units a year are now being sold for that purpose. Product-testing is expected to account for 1,000 units per year by 1984—more than a three fold increase—and facilities-monitoring will rise to 600 a year—a four fold increase.

Perhaps the earliest to recognize the potential of the new microprocessors were manufacturers of industrial-control equipment. For example, Comstar Corp., Minneapolis, first started using microprocessors two and a half years ago, and now it has more than 700 microcomputers installed. Applications include assembly-machine control, automatic weighing and batching sys-



tems, materials-handling systems, remote monitoring and control, data entry, and automobile-traffic control.

One particularly strong market for the microprocessors is in materials-handling. For Beatrice Foods' new frozen-food warehouse in Chicago, for example, Comstar has installed six microprocessor systems. Each controls 50 motors in a network of more than 300 conveyors that transport boxes from the freezer to trucks. On the way, they go through sorters, convergers, divergers, and conveyor-belt changes, but the controller keeps track of every box for its entire trip.

"In earlier warehouses, Beatrice used electromechanical-relay control, with limit switches for actuation," says Tom Walstrom, regional sales manager for Comstar. "Something like our system could have been designed and built with relays, but it might never have worked. It would have been too complex to be practical and much too large to maintain."

Numerical controllers gain

For several reasons, microprocessors also have an excellent potential for being built into stand-alone numerical controllers for machine tools, which are now fabricated with hard-wired logic. Microprocessors can sharply reduce the component count in the controllers while offering easy modifications of programs and functions, which are now possible only with much more expensive systems built around minicomputers.

Although the major N/C suppliers like Allen-Bradley Co., Bendix Corp., and Cincinnati Milacron Co. aren't saying much about their interest in microprocessors, smaller companies and even newcomers to the field, with little or no product base and inventory to worry about, may jump in. General Electric Co., the largest N/C supplier, only last month announced that it had begun using a microprocessor in one of its numerical controllers.

One newcomer is Cambridge Thermionic Corp., Cambridge, Mass., a manufacturer of IC sockets and terminals. But rather than compete head-on with the giants, Cambion's recently introduced PMC-1 microcomputer numerical control is aimed at applications that may have been too expensive for N/C until now, says Lyndon Wilkes of the N/C marketing group. The PMC-1, which operates point to point, rather than on a continuous path, is aimed at simple positioning for such applications as insertion, wire-wrapping, and machines for drilling printed-circuit boards. In its open-loop configuration, it can position a tool to within .001 inch.

Price of the unit is less than \$4,000, including the controller, which is built around the Intel 4-bit MCS-4 microprocessor set, plus a two-axis motor drive and a stepping power supply. The price is about \$1,000 less than the lowest-priced hard-wired controller available, asserts Wilkes.

Manipulating the controls

As indicated in the block diagram, the control and arithmetic units in the Intel 4004 chip allow the CPU to acquire and manipulate control logic and data from the memory sections of the microcomputer and generate the outputs called for in the parts-making program.

Control programs containing the logic which, in con-

ventional N/Cs is hard-wired, is stored in read-only memory. The ROM controls interfacing for a maximum of 32 inputs and outputs. In addition, the ROM section contains the microprograms and data tables that the central processor must execute to control the tool. The unit can accommodate a maximum of six ROMs, each containing 256 by 8 bits, or programmable ROMs, if field programmability is desired.

A random-access memory—there can be a maximum of four devices, each containing 256 by 8 bits—serves as a scratchpad for the central processor. The RAM temporarily stores and releases data and instructions needed on a priority basis by the CPU as it executes the control programs stored in ROM. The parts-making programs themselves are written by the user, just as for a hard-wired controller. Then they're entered into the controller via punched-paper tape. For production runs, however, these programs could also be stored in a programmable ROM.

Likewise, the ROM output interface controls the dispatch of signals to the X- and Y-axis motor drivers and the display readouts. RAM storage controls output to the tools and tape-reader motor. An automatic reset clears the CPU and RAM, resetting the system back to microprogram step one. A two-phase clock circuit provides the timing signals needed by the CPU.

Other components of the system include a ROM input-control interface that monitors inputs from control-panel switches, a paper-tape reader, tool feedback, and an X-Y jog-select mode.

All active components in the control section are contained on a single plug-in printed-circuit board—a decided advantage for maintenance and trouble-shooting, points out applications engineer Howard Atwood. Moreover, because the control has fewer parts, Atwood says the company can deliver a unit in one month or even two weeks, as opposed to the three to six months it would take to put a hard-wired control together.

Bending metal

A microprocessor-based system also controls a metal-stretching and bending press designed by Varitel Inc., Beverly Hills, Calif. About as large as a good-sized room, these giant machines have generally not been amenable to control by off-the-shelf numerical controllers, as have other machine tools, because of the great differences in their design caused by the spread in the size and type of parts they are called upon to fabricate. Hard-wired logic systems are generally used, and each press requires a custom-designed controller.

Although custom designing is still a problem, Varitel president Bruce Gladstone estimates that use of microprocessors can cut design time to a third or even a quarter of the time required to program a hard-wired system. To program the National Semiconductor IMP-16 card used by Varitel, the operator first bends the metal by manual controls. Two angular and two linear multiplexed analog-to-digital converters transmit to a tape cassette the amount of stretch and other factors involved in making the bend. The operator can edit the information as he goes.

When the information on the cassette proves to be accurate, it is transferred to the IMP-16's on-board

erasable RAM. The RAM's capacity of 256 by 16 bits is adequate to provide 12-bit accuracy, achieved through two digital-to-analog converters that drive linear servos. As an added benefit, Varitel provides a small panel that plugs into one of the IMP-16 slots for servicing and troubleshooting. The panel contains its own memory.

The new microprocessors could also affect the design of programmable controllers, which are themselves solid-state replacements for hard-wired banks of electromechanical relay logic. The present solid-state designs are also hard-wired and hence would be excellent candidates to be replaced by microprocessors.

But because of the many inputs derived from the assembly-line machines being controlled, present CPU speeds are generally too slow, says senior systems engineer Ronald D. Malcolm at Modicon Corp., Andover, Mass. Hard-wired designs will offer as fast or faster processing speeds for some time to come, but the microprocessors could allow more features to be added at lower cost, says Malcolm.

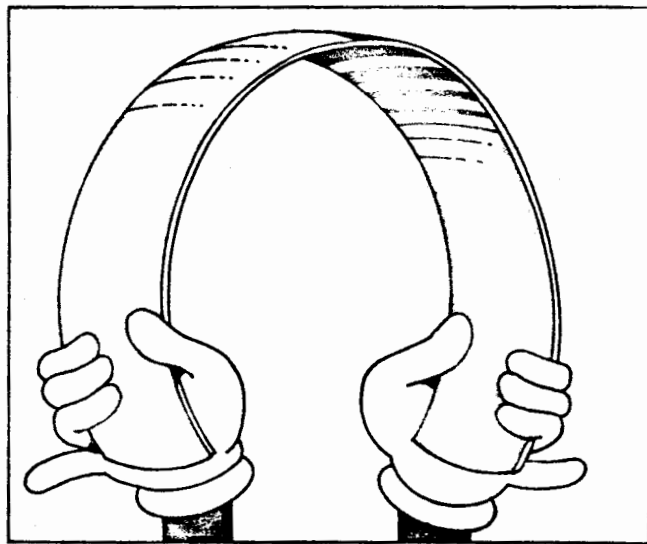
In addition, the microprocessors shorten design time "a great deal," he adds, as well as reduce the physical size, power-supply requirements, and cost. However, for use in its larger controllers, Modicon is considering a 16-bit bipolar monolithic microprocessor with a 150-nanosecond microinstruction time that is being sampled by Monolithic Memories.

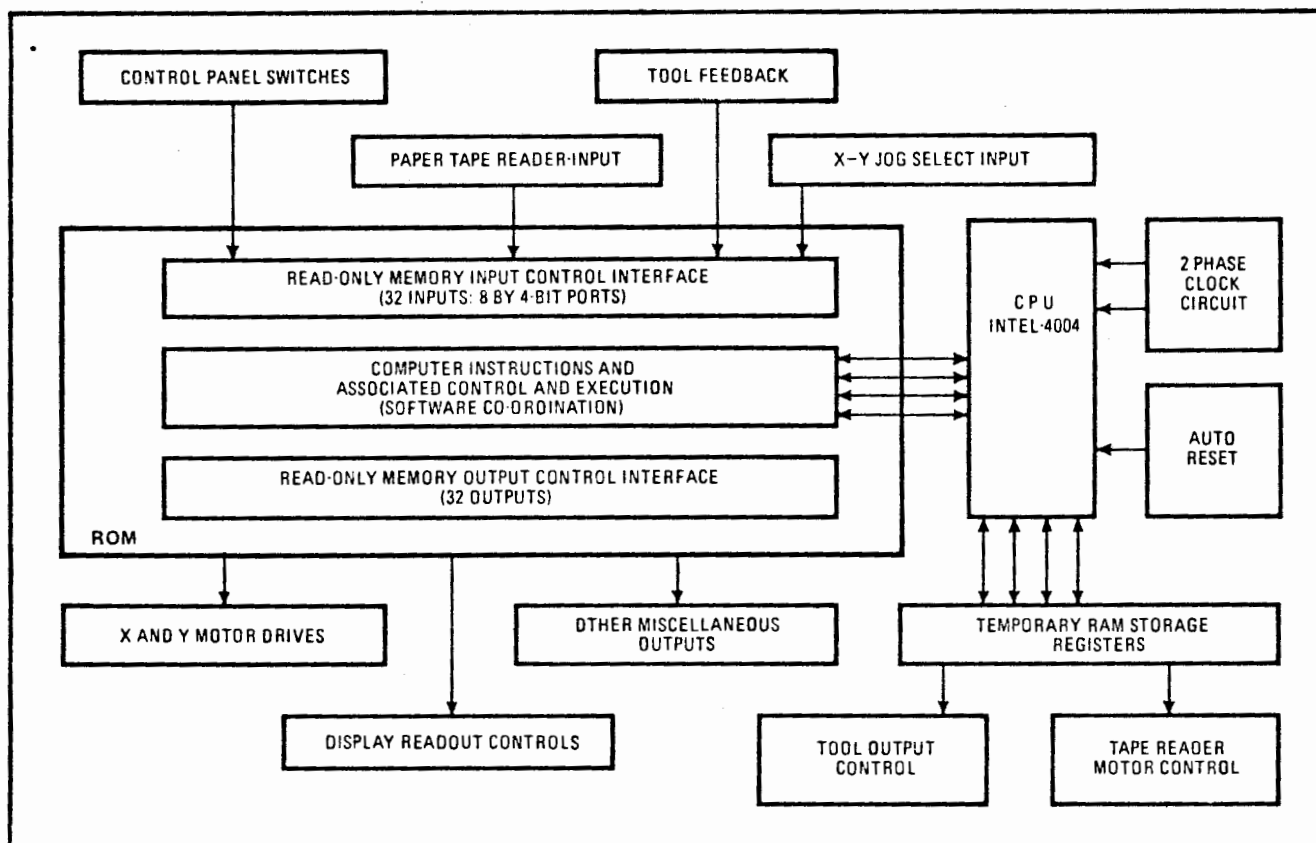
And Modicon, a pioneer in programmable controllers, has already applied microprocessor technology to peripheral products. For example, Intel's MCS-4 set is designed into the P-500 impact printer introduced last winter, as well as the manually operated programming panel for its smallest controller, thereby speeding up the panel's response time.

Controlling traffic

One of the greatest potential applications of microprocessors is to control street traffic. Indeed, Multisonics Corp. of San Ramon, Calif., with 10 years of experience in this application, predicts that intersection-control systems constitute the wave of the future for microprocessors.

Tom Seabury, chief engineer, points out that controllers can be designed for each intersection's needs.





1. Numerical control. An Intel 4004 microprocessor chip is at the center of the PMC-1 point-to-point numerical-control system introduced by Cambridge Thermionic Corp., Cambridge, Md. Programs can be changed simply by plugging in new read-only-memory chips.

"Some traffic schemes, for example, require that all vehicles stop while pedestrians cross in a 'scramble' fashion," he says. "The conventional random-logic controllers need wiring changes to allow this, while with the microprocessor, all we have to do is plug in a different ROM package."

Seabury says the microprocessor is ideal for the stand-alone intelligent intersection controller. Mini-computers, the other alternative to hard-wiring, provide power that is wasted in such a dedicated application, and they are unable to withstand the severe environmental conditions without major design modifications.

The switch to microprocessors is coming at a time when hard-wired controllers had begun to supersede electromechanical controllers, which have synchronous motors that turn switch drums to operate the signal lights. Now, in replacing the hard-wired controllers, the number of ICs has been reduced by at least 60%—from between 500 and 600 to about 100. The company's model 901 controller uses only 50 watts of input power, weighs only 41 pounds, and measures only 17 by 17 by 9 inches. Standard hard-wired models use about 200 w, weigh about 80 pounds, and are twice as big.

The model 901 uses the Intel 8008 as its CPU. Multi-sonics designers first built their systems with the Intel 4004 microprocessor chip as a substitute for drift-prone analog timing circuits. But this 4-bit chip was small, had limited memory capability, and had no instruction-interrupt or capability for single-step instructions. When the 8008 became available, the designers shifted to it.

Also making traffic-signal controls, Comstar is teaming with TRW Systems, Houston, on a contract for 1,000

microcomputers for the city of Baltimore.

Microprocessors are also providing information to help humans improve the quality of the earth's environment. In one application, microprocessors are being installed in remote data-gathering stations that are keeping tabs on such conditions as water and air quality at sites proposed for nuclear-power plants.

Watching the environment

By preprocessing data and determining right at the remote site whether or not it falls within certain preset limits, "we can economize greatly on data-transmission costs because we send back only important data," explains Melvin Couchman, director of marketing and planning for NUS Corp., Rockville, Md. Ordinarily, as many as a half dozen remote stations are tied to a central data-gathering station via telephone lines. In addition to screening out unnecessary and redundant data, the microprocessor-based systems can also run calibration and diagnostic tests of the remote instrumentation to determine whether or not it's functioning properly, a task that might otherwise have to be handled from the central site.

The new systems, built around Computer Automation's LSI-2 unit, also cost less than if they'd been built with hard-wired logic, Couchman points out. But even more important is the capability of programing the microprocessor to tailor the operation of each remote station to specific requirements. "We just change the programable ROM in the field with a new program, or we put in a read/write memory and use the same basic physical hardware," says Couchman. "It would be

much more complicated to change hard-wired logic."

Other types of data-acquisition systems are also feeling the effect of microprocessors. Quindar Electronics Corp., Springfield, N.J., has expanded the capabilities of its system, which is designed to monitor the operation of utilities, partially process the data, and send necessary information to the central computer [*Electronics*, 5/30/74 p. 34]. Process Computer Systems Corp., Ann Arbor, Mich., has designed a system that monitors torque applied to fasteners on an auto assembly line [*Electronics*, 6/13/74 p. 42].

Another company, Doric Scientific Corp., San Diego, Calif., has introduced a new data-monitoring system that not only sharply expands the number of monitored points—to as many as 1,000, an order of magnitude increase over the capacity of an earlier hard-wired unit—but also increases the kinds of parameters that can be monitored. Doric's new microprocessor-based Digitrend 220 monitors and records dc voltages and currents, as well as thermocouple outputs, in such diverse areas as the textile, petrochemical and pulp and paper industries.

The system handles as many as six different types of functional ranges at a time—double the capacity of Doric's hard-wired Digitrend 210. Moreover, with room for plug-in interfaces, it can send this data out to as many as four separate peripheral recording or transmission devices, such as magnetic-tape recorders or teletypewriters. In contrast, the Digitrend 210 handles but a single peripheral.

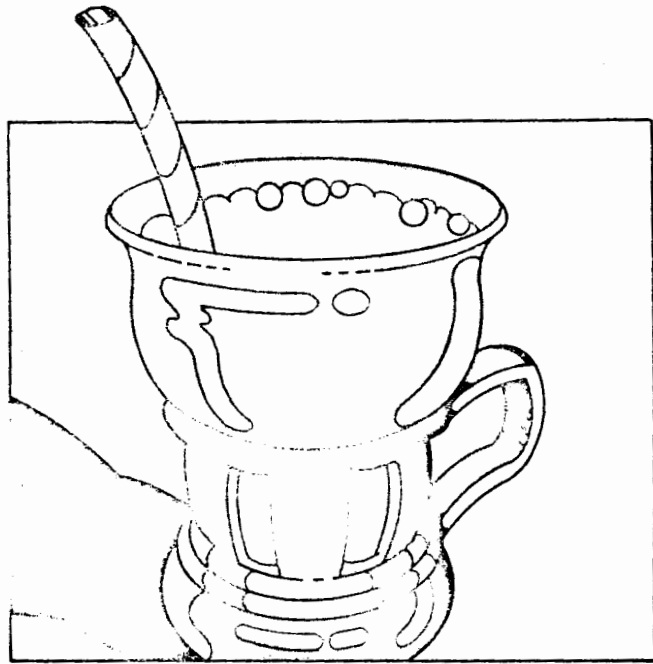
Doric relies on an Intel 8008, with as many as three PROMs, four ROMs, and two RAMs in a bus-organized structure. The memories contain input instructions for handling data, coefficients for linearizing the nonlinear thermocouple inputs, for scaling, for reading out measurements directly in both the fahrenheit and celsius scales, for limiting alarms, and scratchpad memory for aiding in linearization and formatting.

The new unit was designed to do more than its hard-wired predecessor, but comparable configurations would cost 25% more, admits chief engineer Freeman Rose. However, it performs all its functions in just about the same space as its predecessor.

Moreover, the microprocessor approach is "quite a bit" cheaper than if Doric had gone to a minicomputer, Rose continues. At any rate, Doric did not want to "boggle the mind of the customer" with a mini and the software that would be needed. With the microprocessor, changes are made by simply plugging in a new memory, rather than substituting a hard-wired logic board. Doric is looking at such new n-channel microprocessors as the 8080 to expand the capability of its system still further by offering such operations as trend analysis and averaging.

Typesetting makes headlines

For typesetting, a typical microprocessor-based system would consist of a module containing all the processing and memory functions. One module, built by Varityper division, Addressograph Multigraph Corp., East Hanover, N.J., contains the Intel 8008, which offers the large instruction capability required by phototypesetting equipment, plus the required programable ROM,



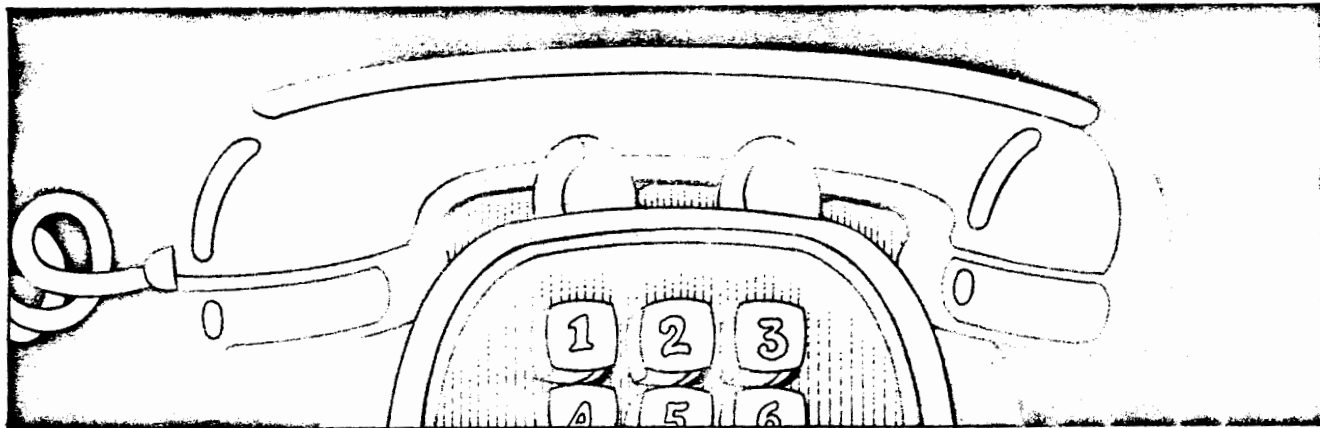
ROM, and RAM, an input bus, and printer and teletypewriter interfaces.

Not only is Varityper able to add processing capabilities to its top-of-the-line phototypesetter that sells for some \$15,500, but processing can be included in lower-end products as well. In the past year, the company has introduced a phototypesetting controller called the Amtrol, built around the Intel 8008. The results couldn't have made management happier. Varityper engineers have built a family of 16 standard plug-in modules that they can just about pull off the shelf and apply to new products as they're needed.

This summer, for example, the company will introduce a composition machine that will sell for less than \$10,000, yet have decision-making capability. This could never have been accomplished at such a low price with the special-purpose minicomputer that Varityper had been buying since 1969. The old mini was "markedly" more expensive than even the full Amtrol controller, and the modular family enables Varityper to tailor the processing power to each application.

Other advantages abound. The new processor is far more compact and reliable, and its plug-in design makes it easy to troubleshoot and service in the field. Moreover, customers seem to prefer the microprocessor design to hard-wired logic, says Joseph A. Verderber, of the office of product management, because it's easier to upgrade the system by adding features through a plug-in read-only memory.

Microprocessors have already begun to have a tremendous impact on many industries that have repetitive processes to be controlled. In the future, their application is likely to be limited solely by the imagination of the design engineer. Although cheap now, microprocessor prices will come down still further. Within a decade, an entire microcomputer with 4 kilobits of memory could cost less than \$150, predicts a market consultant at Quantum Science Corp. That price earmarks the device for an ubiquity similar to that enjoyed by today's hand-held calculator.



Communications

Data-handling gains flexibility

by Stephen E. Scrupski, Communications & Microwave Editor

A strong tide is running in favor of replacing analog communications with digital methods. Microprocessors are accelerating this trend, bringing on a new wave of "intelligent" digital communications equipment. Multiplexers, code converters, error checkers, input/output controllers—all are natural applications for microprocessors. However, their full impact is yet to be felt; most communications-equipment suppliers are still in the feasibility-model and prototyping stages, while the speed limitations of present-day microprocessors are still inhibiting their wider usage.

As in other industries, communications designers like the flexibility and the low costs offered by microcomputers. Custom routines for individual tasks can be quickly changed simply by changing the contents of the programmable read-only memories that hold the programs. This is particularly useful in digital communications, where many different codes and message protocols are in use and where the processing chores do not require the capabilities nor justify the cost of minicomputers.

Microcomputer hardware and software can be designed in parallel. While the printed-circuit boards are being laid out to accommodate the almost standard parts of the microprocessor complement, software design can proceed independently, and the two designs can be merged late in the product's development cycle, allowing for system optimization in a minimum of design time. What's more, when a microcomputer breaks down in a communications system, recovery time should be substantially less than in any other kind of system. Service technicians can carry standard circuit modules that are compatible with any of their company's equipment, requiring only new programming to take the place of a failed unit.

Micro teams with mini

An example of how a microprocessor and a minicomputer can be teamed up is in the message-switching units (Fig. 1) being developed by Action Communi-

cation Systems Inc., Dallas, Texas, in which microprocessors serve as front ends for Data General Corp. Nova minicomputers. The switchers are used in networks of private terminals, such as those employed by police departments to access records in a state capital or the National Crime Information Center in Washington, D.C. The company has installed several such systems. In the Texas network, for example, more than 500 terminals are located in police headquarters throughout the state.

These switchers, now in the prototype stage, will speed up the switching action and allow higher data rates. They will do this by relieving the minicomputers of certain standard operations—the "dirty work" that must be performed on all messages, such as converting them to the proper code for processing by the Nova and scanning the incoming character strings to identify different control sequences.

"What we're trying to do is eliminate any character-by-character handling by the Nova and allow it to handle only blocks of data," says Action design engineer Michael Fannin. By allowing the minicomputer to do the more complex tasks while the microprocessor handles the menial chores, he predicts that this configuration will raise the processing speed by about an order of magnitude, from the 1,000 or 2,000 characters per second to 10,000 or 20,000 characters per second.

Action is using National Semiconductor's IMP-16C processor for this application "because of its powerful instruction set," says Fannin. "Although it has a slower cycle time than some of its competition," he adds, "it does more with its instructions."

In the system (Fig. 1), circuit controllers interface with the communications circuits and perform serial assembly and disassembly of the characters at data rates as high as 19,200 bits per second. The microprocessors interface with the controllers and perform four functions:

- Convert character codes.
- Scan messages for key characters.

- Edit message headers and text.
- Check character calculations.

One microprocessor can handle the 19,200-b/s rate. It also interfaces with the 64,000-character semiconductor random-access memory, which buffers message blocks between the microprocessor and the central minicomputer.

In such applications, the microprocessor serves primarily as a piece of hardware, since the custom features still reside in the minicomputer's program. In effect, Action is using the microprocessor as a low-cost way to achieve large-scale integration. Many communications designers consider that this is the primary benefit of the microprocessor.

Arless Whiteside, senior department consultant (essentially a senior scientist) in information processing at the Bendix Research Laboratories in Southfield, Mich., says, "A microprocessor is just another component—and a few too many people consider it something magic. I think they're oversold."

Whiteside goes on to explain that the microprocessor, in his view, is simply a way to cash in on the benefits of large-scale integration—lower costs through fewer packages—without entering a multi-thousand-dollar program to develop custom LSI. "I call it standard LSI," he adds, "LSI that is standardized, flexible, and built by the manufacturer in the quantities that are necessary to justify the design costs for an LSI chip."

Handling the full load

Such a viewpoint is supported in applications where the microprocessor assists a minicomputer. But in others, microprocessors shoulder the full load of data processing. Collins Radio Corp. in Dallas, for example plans to use microprocessors in an intelligent repeater for a private microwave data-transmission system now being built.

In the system, several data links surround a central-hub repeater terminal that switches one link to another upon request. The data signal carries address informa-

tion that is decoded by the microprocessor, which then routes the message through the hub repeater to the proper receiving terminal. Although this is still an experimental project, according to Collins, the experiments have nothing to do with the microprocessors—the unknown factors are in the radio communications.

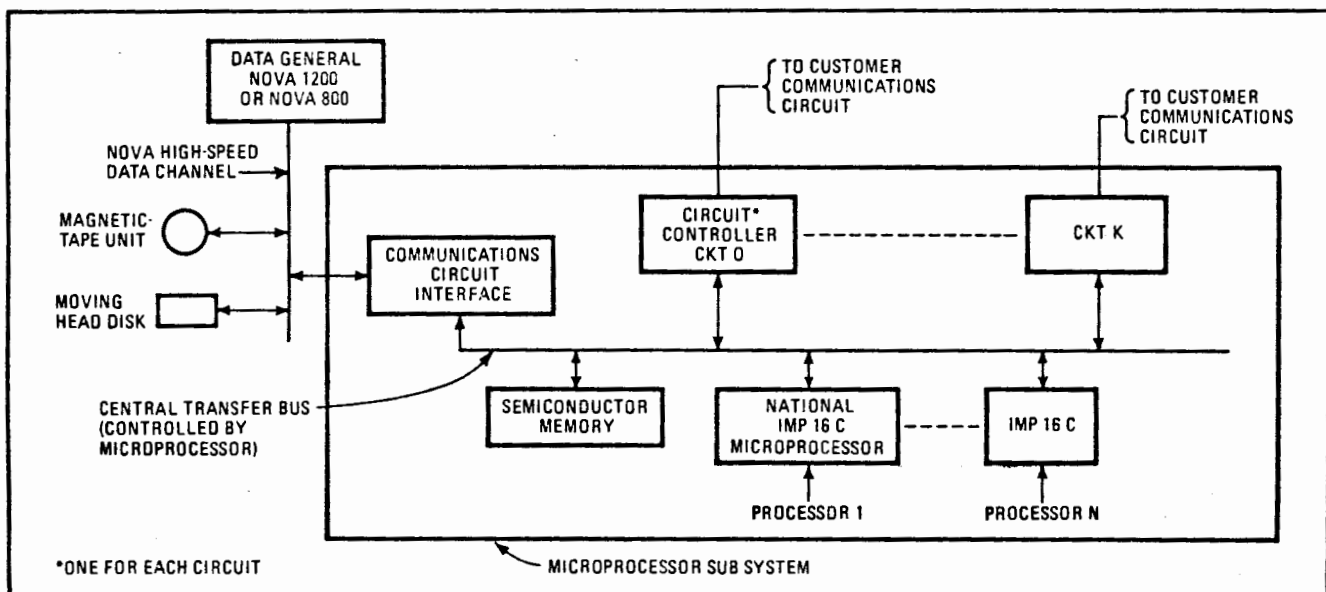
In this instance, the microcomputer's small size helps it beat out a minicomputer for the application—the repeaters have to be man-transportable and battery-powered. To further reduce the power drain, Collins engineers are replacing the TTL circuits recommended by the microcomputer manufacturer with complementary-MOS circuits. To conserve battery power, Collins is also using C-MOS chips for the random-access memory and programable read-only memory. However, the use of C-MOS instead of TTL slows down the system from the microprocessor's basic 1.4-microsecond cycle time to about 4 μ s.

Considering tradeoffs

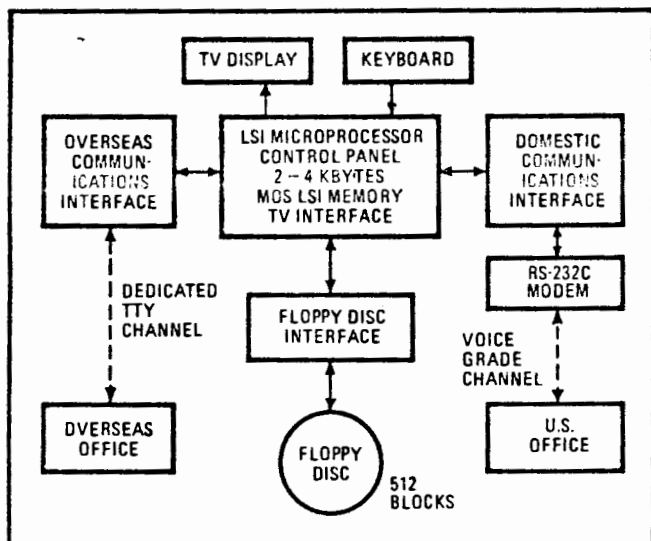
The reduced speed affects the architecture of the system, since extra memory is required to compensate for it. As the message is received in the processor at the hub repeater, it is stored in a buffer memory, and the microprocessor goes right to work processing the information. By the time the message is completely received, the microprocessor has extracted the processing and routing information, and the message is ready to be retransmitted to its destination.

The reduced speed also prevents Collins engineers from using the microprocessor for what should be a natural function—error-checking. The expected maximum data speed of 500 kilobits per second is just too fast for today's microprocessors. Error-checking therefore is done by hard-wired logic. However, if the transmission speed were lower—say, in the range of 50 kilobits per second—the microprocessors could be used to perform error-checking, says Collins design engineer Dale Walls.

Or, if the microprocessor could be operated at its design cycle speed of 1.4 μ s, Walls says it would be "aw-



1. Nova helper. In message-switching units built by Action Communication, a National Semiconductor IMP-16 microprocessor handles character-by-character decoding so that Nova minicomputer can concentrate on handling full blocks of data, increasing system speed.



2. Interpreter. A microprocessor handles conversions of codes and speeds to allow a domestic data processor to communicate with an overseas network via the RCA Global Communications system. Easily changed software helps customize system.

fully close to being applicable for error-checking." The limiting speed of today's microprocessors however, will soon be overcome by a new generation of faster devices built with bipolar or sapphire-based MOS technologies, while 4-bit processor slices capable of instruction times of 10 to 50 ns are expected to be available by the end of the year.

Interfacing between nations

A microprocessor is the sole computing component in a programmable controller built to handle international leased-data channels. Developed jointly by RCA Laboratories, Princeton N.J., and RCA Global Communications Inc., New York, the controller connects RCA's Cosmac, a two-chip C-MOS microprocessor and associated semiconductor RAMs, to a floppy-disk drive for mass storage of messages.

The combination of the microcomputer with a floppy-disk drive allows RCA to cut the cost of the controller below that of either a system combining mag-

netic tape with a minicomputer or hard-wired logic. The single basic design, easily customized by software, meets a variety of different customer needs, while at the same time offering improved maintainability.

The microprocessor's job is to provide all the conversions necessary to interface a domestic communications network with an overseas network (Fig. 2). Signals, codes, speeds, character formats—all must be often reconciled to allow the two networks to communicate with one another. And, since each private user who leases a channel from RCA has his own unique combination of such parameters, use of hard-wired logic would require long development times and an abundance of specialized equipment that would have to be maintained.

Minicomputers, although they offer programability, are simply too expensive to be considered for this application, according to RCA, since they have too much computing power for the few lines that must be controlled. Another tangential problem, RCA claims, is that often the customer has only partial knowledge of his own needs, and the microprocessor programability offers RCA engineers an easy means to add needed features at later stages.

Helping the police

In another police-oriented application, Motorola's Communications division, Schaumburg, Ill., is using a microprocessor in a computerized mobile terminal system, first installed for the Atlantic City, N.J., police in 1973. Each squad car carries a light-weight terminal with a full keyboard and plasma alphanumeric display. Using the terminal, a policeman can access files at his local station, at the state headquarters, or even at the National Crime Information Center.

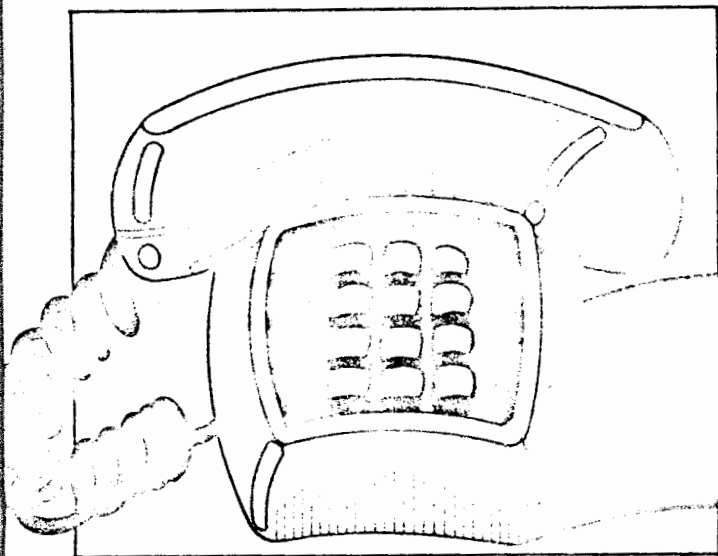
An 8-bit microprocessor is built into the base-station unit, says Jerry Schloemer, manager for command and control products at Motorola. The microprocessor acts as a communications interface to the computer at the next higher echelon, controlling the coding on the radio channel and performing a reduced store-and-forward function in both directions.

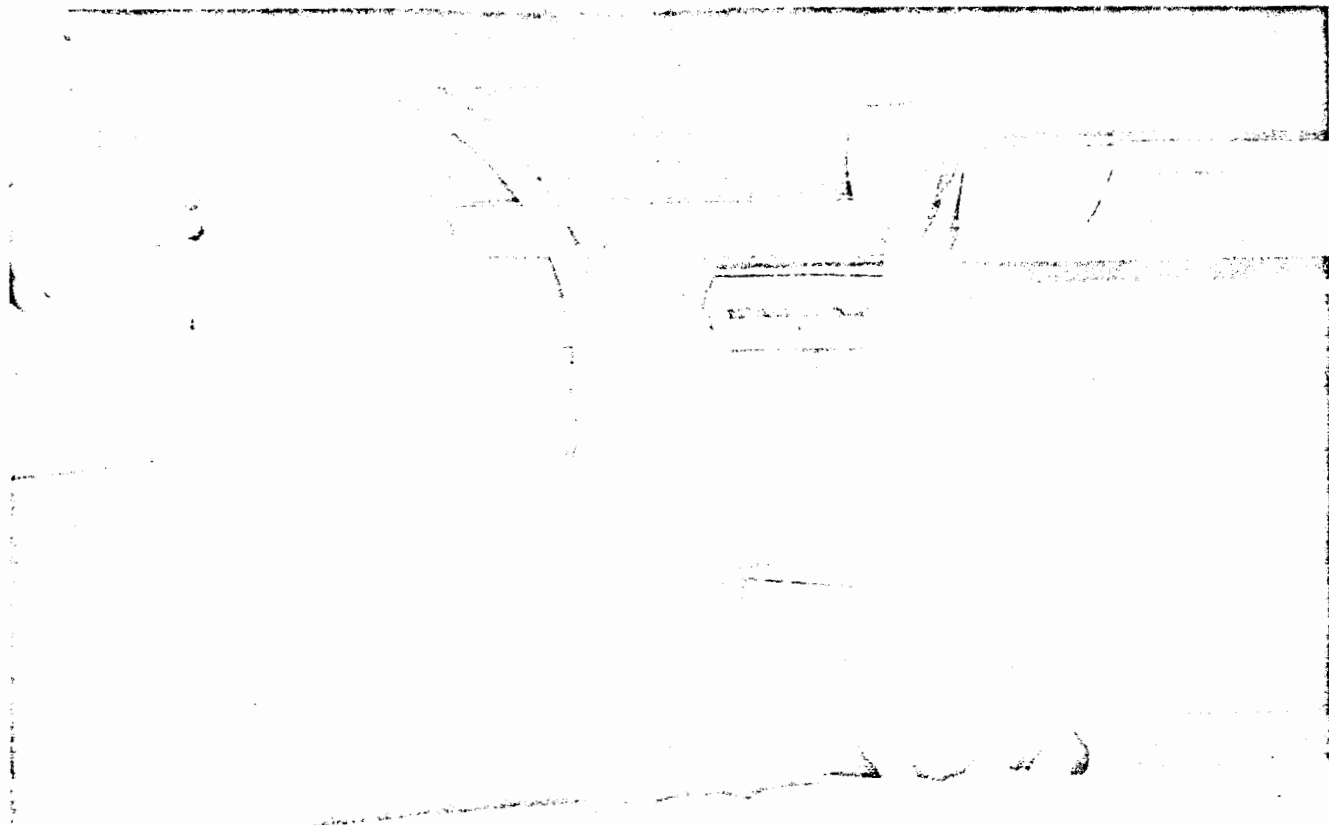
"We hope that the cost of microprocessor devices will come down with increasing volumes," Schloemer says. "If so, we're planning to put microprocessors in the next-generation car unit—it gives us a little extra power to be able to offer more features. We hope that their use will reduce our product-introduction cycle," he adds, "but we've seen no evidence of that yet."

Even voice signals, once they are converted to digital form, as in a pulse-code-modulation system, may offer opportunities for microprocessors. Presently, telephone voice signals in a 4-kilohertz bandwidth are sampled in a "channel bank" at an 8-kHz rate, and each sample is encoded into 8 bits; thus the 4-kHz voice signal is sent at a rate of 64 kilobits per second, which is extremely wasteful of bandwidths, says David Trask, manager of the communications system laboratory of the Raytheon Equipment division in Wayland, Mass.

Simplifying the phone system

He points out that telephone engineers have given much thought to ways to reduce the bit rate necessary





Calling all cars. Microprocessor in Atlantic City, N. J., police station controls message-coding for keyboard terminals used in squad cars.

for each voice signal and thus to expand the capabilities of the transmission system. For example, many algorithms have been proposed for a processor that would note an instantaneous value of the voice signal and predict the value during the next sampling period. Then, when the next sample actually appears, the processor would transmit only the difference between the actual and predicted values.

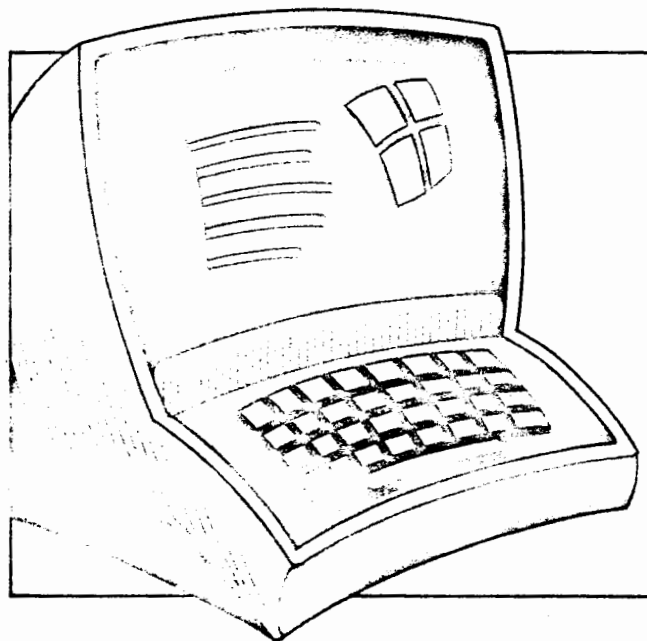
If the algorithm is effective, it would only require a few bits, rather than the full 8 bits presently used. An identically programed microprocessor at the receiving end would then reconstruct the full voice signal. In fact, he envisions a telephone set that has the sampling and microprocessor circuitry built right into the back so that digital signals are sent to the telephone central office directly from the set itself.

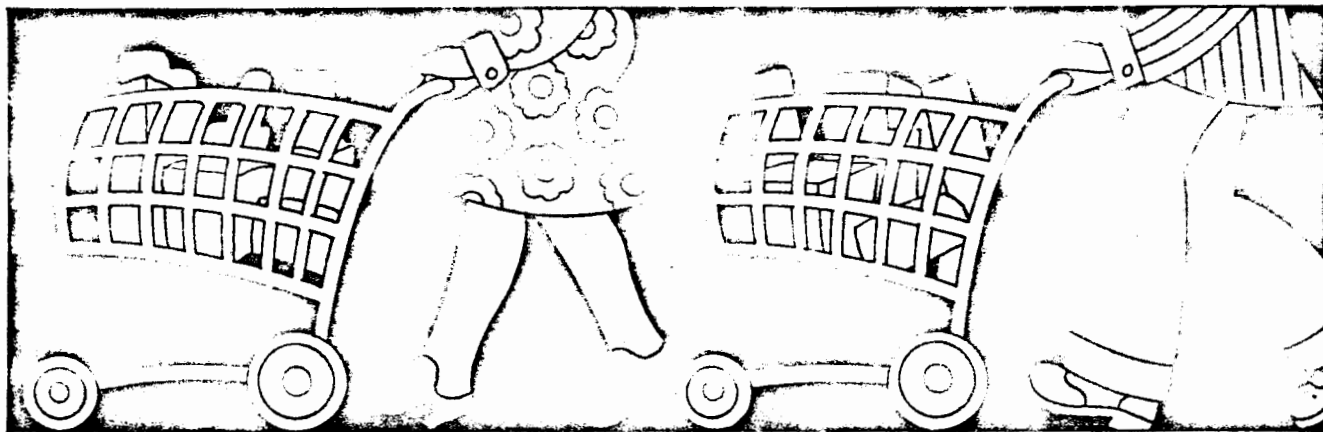
Microprocessors are being designed into a somewhat similar system at Harris Electronic Systems, Melbourne, Fla. Harris is building encryption devices for secure digital-communications systems and, says Ray Glenn, associate principal engineer at Harris, the microprocessor makes a good pseudo-random noise generator. One microprocessor can be programed to encode a digital signal, and an identical microprocessor at the receiving end can decode the signal.

If fabricated with medium-scale integration, such a system might require up to about 75 packages, but a microprocessor would cut the count to only 10 to 15 packages, Glenn estimates. He points out that microprocessors are being used at Harris to control the output-power levels of radio-frequency transmitters throughout the day. Temperature changes, Glenn says, cause the power level to drift, but a simple 4-bit micro-

processor can store a control algorithm that enables the microprocessor, when presented with digital information on the output level, to bring the level back to the desired point.

It is clear that microprocessors are taking over many of the routine applications in communications equipment. And regardless of whether the designer views a microprocessor as merely another component—a way to get standard LSI— or as a radical new component that offers small-scale programing, nearly all analog, as well as digital, communications gear will benefit from its impact.





Consumer/commercial Microprocessors go public

by Gerald M. Walker, Consumer Editor

Manufacturers of commercial and consumer products have for some time taken the lead in applying advanced semiconductor technology. Their adoption of microprocessors is no exception. In a sense, microprocessors are accelerating the timetable for equipment and systems already deemed feasible in both the commercial and consumer markets.

In addition, development of totally new products not yet identified will sweep these markets in the same way that the personal electronic calculators came from nowhere into international prominence. Thus, microprocessors are having it both ways—enhancing present-day equipment while promising completely new products for offices, stores, households, and entertainment centers.

Included in commercial equipment containing microprocessors now on the market are terminals for point-of-sale and supermarket checkout, scales, terminals for investment houses and the finance industry, automated back tellers, processors for business-inventory control, equipment for supermarket in-store packaging, and portable data terminals.

Among the products using microprocessors in the consumer and related markets or on the drawing board for the near future are sophisticated games, gambling equipment, cable-television transmission hardware, do-it-yourself instrument kits, and photographic-film developers. Further down the pike are automobile on-board processors that perform such tasks as controlling combustion timing (Fig. 1), exhaust emission, transmission operation, and anti-skid and diagnostic systems.

It's in the household that the explosive new product—the home computer—is expected to emerge. The most obvious door into the home is the television set, which can make good use of a data-communications processor. By then, microprocessors will have to be quite different from today's products, not only in bit capacity, but also in basic environmental configuration and price.

In the entertainment world, the microprocessor offers the simulation of games at a level of sophistication until

now reserved for military and space projects. In its civilian format, simulation makes games realistic by the capability to cram programing, memory, feedback, and real-time processing onto a single chip. Certainly Disney's "Land" and "World" are proving the wide attraction of family fantasy via simulation. The subject of a movie spoof about a year ago, an adult fantasyland designed around simulation techniques is now more than science fiction.

In general, the advantages of microprocessors to commercial/consumer-equipment designers boil down to the tradeoffs between hard-wired and programable logic. For instance, point-of-sale cash registers built with hard-wired packages have performed both as stand-alone units and minicomputer-controlled terminals. By changing to microprocessors, POS-equipment manufacturers gain the important advantage of adapting their basic equipment through programing to the needs of individual stores.

On the other hand, the problem most frequently mentioned by manufacturers of commercial/consumer equipment using microprocessors is the difficulty of refining the very software that they also say is the microprocessor's major advantage over hard-wired circuits. Equipment makers feel that microprocessor suppliers are not equal to the task of providing software support, forcing users to become immersed in programing.

Some of the commercial-consumer products using microprocessors are hardly a generation removed from electromechanical design. Yet the totally different requirements of the technology have made the switchover from hard-wired logic to microprocessors as traumatic for designers as the original change from an electromechanical to an electronic approach.

As C.W. Kessler, vice president of corporate engineering and advanced development for NCR Corp., Dayton, Ohio, points out, engineers familiar with Boolean equations and logic families, which were adequate for the design of hard-wired equipment, must now add complex instruction sets to their repertoires for micro-

processors. They must be prepared to live with the sequential operation of microprocessors, which is slower than the parallel operation of chips using standard logic like TTL.

In addition, Kessler suggests, "There is a horde of new problems in choosing the right microprocessor, and these have become corporate-level decisions. After all, you're tied to one supplier, once work is completed on hardware and software. There's a lot hanging on the source selection, since you don't have a second source."

POS-terminal producers took different routes to arrive at use of microprocessors. For example, National Semiconductor's Systems division began applying them as a direct result of its ties to development by the semiconductor operation. Because of the close relationship, programs presented little problem. However, the main challenge was to teach test personnel to debug semiconductor chips the way programmers debug a computer. This conversion required training because microprocessor faults are much more difficult to isolate and correct than failures on a standard LSI chip.

At American Regitel Corp., San Carlos, Calif., application of a microprocessor made it possible to design a terminal combining stand-alone "intelligence" and peripheral-communications capability. Such mechanical attributes as communications routines are specified in read-only memory, while the logical attributes at the human and exterior interfaces are specified by instructions residing in random-access memory. The former are concerned with fixed procedures, while the latter must be variable to permit application of a wide range of sequences, tax tables, and keyboard checks.

Most of the jobs assigned to the controller are performed at the speed of the terminal operator, and the program responsible for driving the printer has a throughput of only 30 to 100 characters per second. Because the arithmetic is not a major difficulty, and transactions are done at human speeds (communications functions require logic throughput of 200 to 300 characters per second), a general-purpose microprocessor that could fetch in 3 to 10 microseconds was adequate, putting the task well within the capacity of 4-bit processors.

NCR presently employs Intel MCS-4 microprocessors in two products—a bank-teller terminal and a point-of-sale terminal—and will soon introduce four others that use microprocessors. Their functions are quite different.

Inside the NCR 279 financial terminal, for instance, microprocessors control the keyboard, printer, and credit-card reader, do the teller's arithmetic, transfer data, and act as computer-interrupt. In the NCR 255 supermarket register, the microprocessor is essentially a back-up element to provide the terminal stand-alone capability, should the remote computer-controller fail. The microprocessor makes it possible to do away with dual minicomputers to control terminals unless the customer wants the redundancy.

Another teller terminal using microprocessors has been built by Financial Data Science Inc., Orlando, Fla., and about 100 are presently in the field. The model 108 contains three MCS-4s—one for printer control, one to provide stand-alone processing in the event of communications failure to the central computer, and one to control the keyboard and perform calculations.

Microprocessor knowhow

Not only are microprocessors changing the design of equipment, they are also changing the demands on the designers who use them. A list of the skills and tools needed for the new generation of microprocessor applications engineers, recently drawn up by Herman Schmid of General Electric, is awesome.

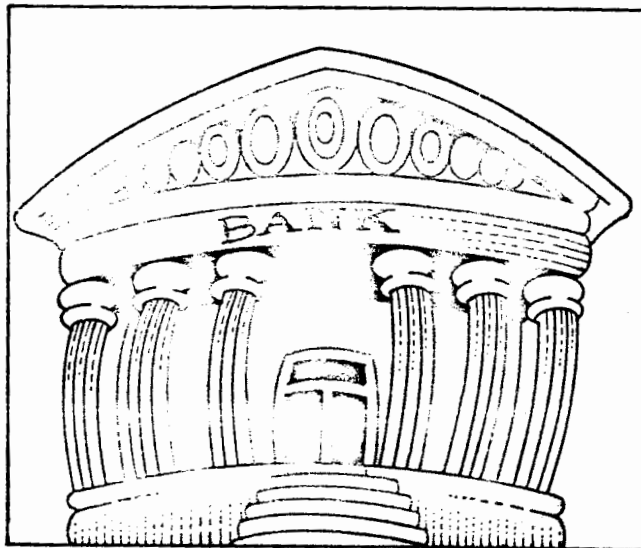
He states that engineers must thoroughly comprehend the organization, operation, and performance of the processor's CPU; control of input/output; the organization and operation of RAMs, ROMs, and programmable ROMs, plus such interface circuits as analog-to-digital and digital-to-analog converters; operation of peripheral equipment; the operation of multilevel priority-interrupt systems; the operation of control-panel circuits; and the operation of such various logic families as TTL, p-MOS, n-MOS, and C-MOS.

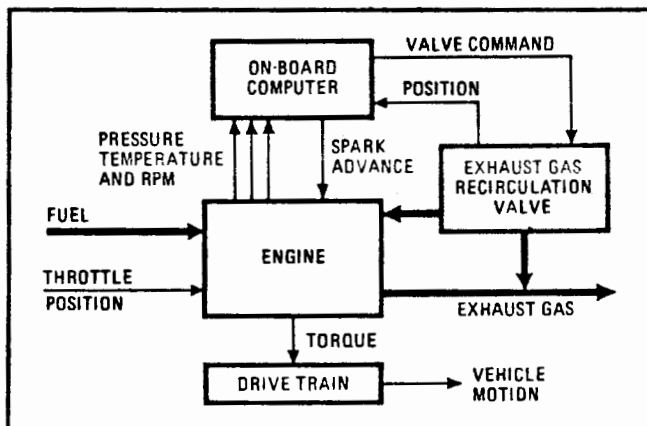
But that's not all. For designing firmware, this same engineer must also have extensive knowledge of programming. This designer needs to be an expert in software for machine-level, micro-level, and assembler-language programming. Finally, the microprocessor engineer must be familiar with such interface operations as the performance of converters and signal-conditioning.

The first and third applications could have been performed by hard-wired logic, but stand-alone processing backup would have required a minicomputer. By applying the microprocessor to the keyboard, total package count was reduced 30%, and total cost was lowered to slightly less than what hard-wired logic would have been. In addition to the 108, which is meant for savings-and-loan institutions, the model 151 is also available for full-service automated bank tellers. It uses one microprocessor, essentially as a calculator.

Automating Inventory

The manufacturer that probably has the most units containing microprocessors in the field is MSI Data Corp., Costa Mesa, Calif. This firm has delivered about 10,000 portable data terminals for use in taking and recording inventory or other data at remote locations.





1. **Economy car.** One microprocessor will be used in an automobile for spark-ignition timing and exhaust-gas recirculation-valve control.

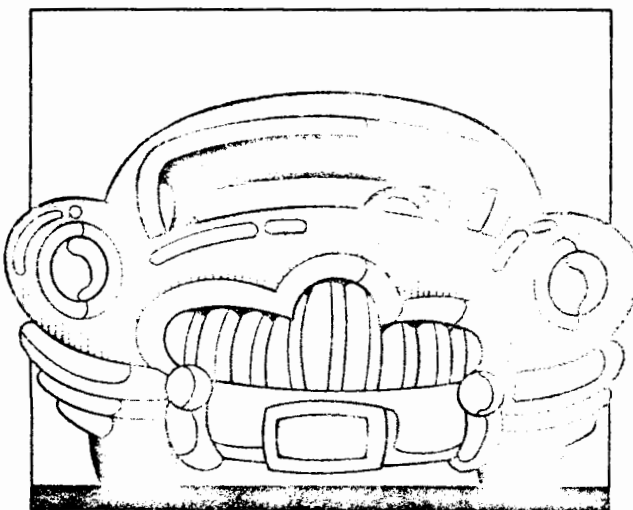
Each terminal contains one MCS-4 microprocessor.

The MSI battery-powered model 1100, which has semiconductor memory, and the model 2100, which has a magnetic-tape cassette, look like plump hand-held calculators, except that the keyboards have special symbols, and just below the LED displays are function switches for transferring data to telephone modems. Data such as supermarket inventory or warehouse stockroom supplies is entered through the keyboard and recorded either on a tape cassette or in solid-state memory, depending on which of the two models is used.

Afterwards, this data is communicated by telephone to a MSI receiver at some control location. Depending on the model used, 7,000 to 20,000 characters of information can be transmitted in less than three minutes, eliminating several data-handling steps required in manual or even punch-card procedures.

MSI originally designed these terminals with TTL to control the displays, computations, and interface circuits. Later models were converted to complementary-MOS chips to reduce battery-power dissipation. But the need for flexibility to meet a variety of uses for remote terminals made microprocessors attractive replacements for the control logic. At the same time, delays in delivery of standard chips made the change to microprocessors even more attractive.

Larry Hendricks, manager of the Electronic Engi-



neering department for MSI, points out that previous experience in designing a minicomputer controller for data terminals was valuable in learning how to design with microprocessors. In fact, MSI now uses a minicomputer that it designed and built to serve as a communications controller to write the microprograms.

Hendricks complains that microprocessors are still difficult for many designers to learn to use because there's no easy applications track; hardware-oriented engineers stumble on the software, while software-oriented programmers get confused by LSI technology.

He also cites three other current problems. First, he would like microprocessor manufacturers to stick with one device long enough to establish an industry standard such as the 1103 chip. Second, Hendricks is uncomfortable with single-source purchasing, particularly since MSI is now buying microprocessors in relatively large quantities to support production of about 1,000 portable data terminals a day. The third problem is the need for a more sophisticated system that nonprogrammers can use for microprogramming.

While microprocessors are essentially used for what Hendricks calls "bit-banging," that is, simple and slow processing chores, he believes that there's a danger of trying to apply them for too many functions. "It may seem possible to substitute a microprocessor for every minicomputer," he says, "but you have to watch out that you're not sending a boy to do a man's job."

Singer patterns its own

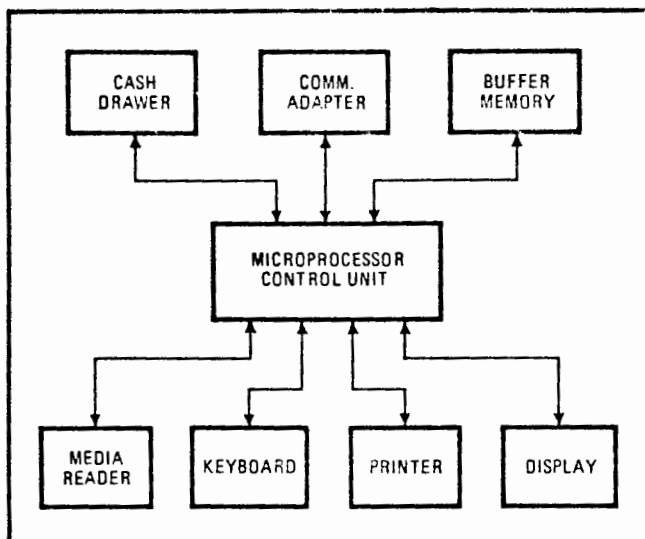
Although most microprocessor users, especially commercial manufacturers, have been concerned with dependence on sole-source purchasing, Singer Corp., New York, has alleviated this situation by designing its own microprocessor at the firm's research laboratory in Fairfield, N.J. At least three semiconductor houses are qualified to use Singer's masks to produce the chip. In fact, one of the design constraints was to be conservative enough to keep producibility within the capability of at least two suppliers—not an easy task.

The result is the Advanced Byte-Oriented (ABO) microcomputer, an 8-bit, n-channel MOS processor measuring 191 by 202 mils. The 40-pin unit is designed for a variety of Singer products, including point-of-sale terminals built by the Business Machines division in San Leandro, Calif. It's microprogramed internally from a 6,000-bit ROM, rather than from separate chips.

One reason Singer designed its own microcomputer was to follow the course of its electronic end products into what the firm calls "distributed computing," that is, loading each piece of equipment with as much processing capability as possible. Thus, in a Singer POS terminal, the ABO is heavy on processing capability and light on arithmetic-calculation functions.

Microcomputers from semiconductor suppliers need both capabilities, whereas a custom design could downgrade the less important attribute. Of a total of 256 instruction codes, 50 are basic, and the instruction time is typically 10 microseconds. According to Singer, prototypes of its microprocessor are now being manufactured by two sources.

The Business Machines division presently has a terminal with a single microprocessor also of Singer de-



2. POS microprocessor. In a stand-alone point-of-sale terminal, the control unit carries out microinstructions stored in a ROM.

sign. However, unlike the ABO, this unit has five 12-kilobit ROMs outboarded to instruct the microprocessor. The function of the microprocessor-control unit (Fig. 2) is to direct the flow of data between the I/O devices and the buffer memory and to perform arithmetic chores. All data transfers within the microprocessor and between it and the I/O devices are accomplished by means of a source-destination bus, consisting of a 5-bit source address, a 5-bit destination address, and a 6-bit data bus. Because each register inside the microprocessor and in the I/O devices is addressable, it can act as either the source or the destination in a data transfer. However, intercommunications are minimized, and interface requirements for the I/O devices are simple because the terminal is bus-oriented.

Steering for Detroit

An example of what an automobile-microprocessor system might look like is a Ford Motor Co. advanced development. Figure 1 shows the bare bones of a digital control system, designed to maximize fuel economy. It is being road-tested, but it won't be ready for a standard car for some time. This system uses two microprocessors and other custom-LSI devices to control timing of spark ignition and position the exhaust-gas-recirculation (EGR) valve by using several engine inputs.

Ford engineers decided to use a microprocessor because attempting to compensate an analog programable spark-timing controller over the worst-case of auto temperatures turned out to be more expensive than a digital control system. The microcomputer made it easier to program changes in engine design than to use hard-wired logic. Ford uses a 12-bit microprocessor with the program and associated coefficients, which describe the engine-control algorithm stored in a ROM.

The present engine-control software is contained in about 1,500 12-bit words. The system also includes an 8-bit analog-to-digital converter with an eight-channel multiplexer under CPU control to measure the outputs of engine and EGR-valve transducers. The key reason for using a microprocessor for this application is to be able to design the same hardware for all engine and

transmission variations in several different models, changing only the software to match each car.

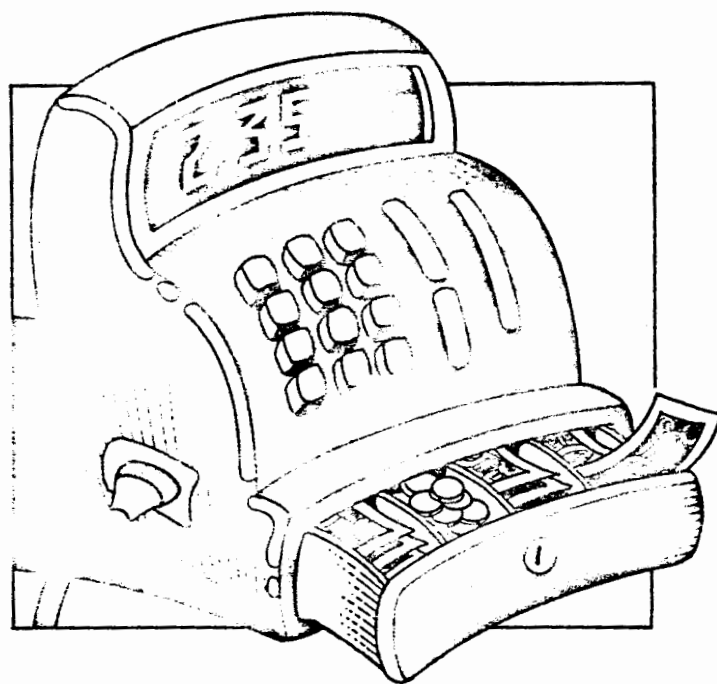
Actually the idea for computer-like management of timing, combustion control, emission control and transmission control has been considered by the advanced engineering departments of the auto Big Three for some time. There is also a possibility for microprocessors to handle such safety functions such as antiskid braking and on-board diagnostic systems.

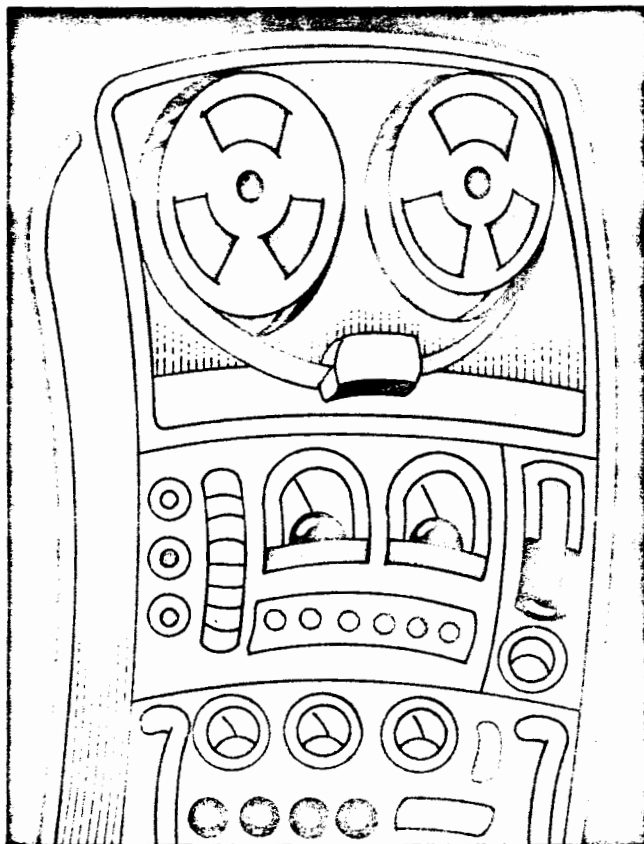
The game's the thing

A unique, but significant, application of microprocessors to games is exemplified by a bowling game Bally Manufacturing Corp., Chicago, has been marketing since last October. Sold to distributors for \$1,600, about 85 of the electronic game, Bally Alley, are now installed. Each contains one Intel 4004 CPU, four programmable ROMs, a RAM, and one 1,024-bit ROM, in addition to some 250 discrete power transistors and silicon-controlled rectifiers.

The electronics package in Bally Alley is vital to give players the right "feel," not only in the scoring, but in the fall of the "pins," the roll of the "ball," and posting the odds normally associated with making various shots. None of this could have been done in the size and cost required of an arcade game without the microprocessor.

The microprocessor monitors the placement of the ball when it is sent down the lane by a player (one to four can play at any one time), keeps tabs on the pins, and metes out free games and credits. In controlling the scoring, the microcomputer tracks pin patterns. A player can decide at what place along the bottom of the lane to let go of the simulated ball, and the microprocessor calculates from program instructions how many balls have been used before recording the score on an incandescent-lamp display. Bally is now looking at microprocessors in 8-bit configurations for other games, as well as gambling equipment it builds.





Computers

Peripherals now, mainframes later

by Wallace B. Riley, Computers Editor

Although much of the shouting about microprocessors has been about consumer and industrial applications, these programable large-scale integrated circuits are also impinging on the way computer manufacturers themselves are designing data-processing systems.

For manufacturers of large mainframes, the impact is mainly in peripheral and control equipment because today's microprocessors are generally too slow and limited to perform large-scale processing. For mini-computer manufacturers, however, the low-cost versatile LSI processor goes to the very heart of their designs and promises to open up a whole range of higher-performance capabilities at lower costs.

A major advantage of microprocessors is the smoother design iterations that can be wholly or partially achieved by reprogramming a microprocessor instead of rewiring a major part of a prototype design. These design iterations are necessary in almost any development cycle because the original specifications have to be modified as development proceeds. The goal is a design that meets the original specifications to some degree while being both manufacturable and marketable. In conventional designs, iterations often take the form of building and rebuilding a succession of prototypes—an expensive and time-consuming process.

The main use of microprocessors with the large mainframes has been in peripheral equipment and controllers. Their application inside the computers themselves has been like only a distant rumble of thunder because until now they have been too slow. However, a new generation of chips now on the drawing board promises to overcome that shortcoming.

Microprocessors have thus far proved of value primarily in low-cost, low-speed equipment, such as cath-

ode-ray-tube terminals and magnetic-tape cassette drives. Their main benefits have been to facilitate customization and addition of power at a lower cost than previous designs and to increase the processing capabilities of remote terminals.

Makers of punched-card machines, floppy-disk-storage units, and devices of similar complexity say they may use microprocessors in their next design cycles. However, they have thus far found the LSI chips too slow or too limited in some other functions. These companies are expressing great interest in such microprocessors as Intel's new 8080 [*Electronics*, 4/18/74, p. 95], mainly because of its expanded instruction set and the order-of-magnitude increase in speed.

Microprocessors cut costs and reduce system complexity while simplifying customization of otherwise standard designs. For example, Beehive Medical Electronics Inc. of Salt Lake City, Utah, can adapt its Superbee terminal easily to a variety of applications because it uses the Intel 8008-1 chip. And, although the microprocessor replaces only some of the circuitry of the company's earlier model, it adds new functions and adapts easily to each customer's application.

A trend changes

Significantly, the burgeoning interest in microprocessors reverses one important trend that has been shaping up during the past few years—the execution in hardware of many functions traditionally left to the software. This tradition was established in the early days of computers, when gates cost \$100 apiece and programmers were paid clerical wages. These rates made the minimization of hardware imperative and the proliferation of software initially unimportant.

But since then, costs of hardware and software have moved inexorably in opposite directions. Today, some functions that would have cost astronomical amounts for 1954 hardware can be implemented now for little more than pocket money, while software has grown to almost unmanageable proportions in the form of operating systems, time-sharing, and so on—all in the name of efficient use of hardware.

The low cost of hardware has made microprocessors possible—simple enough not to require software of the complexity remotely resembling an operating system and cheap enough for inefficient use without adding significantly to the cost. As a result, some new functions can be implemented in software that considerably simplifies design and alteration—without the headaches associated with large software systems.

Peripherals and controllers benefit

In the peripheral devices themselves, microprocessors take a substantial load from a controller or central processor. For example, Digi-log Systems Inc., Horsham, Pa., uses microprocessors to control a display's refresh memory, its communications interface, its editing functions (inserting and deleting words, phrases, and paragraphs, and rearranging them as directed from the keyboard), as well as other display characteristics.

A variety of optional capabilities is available with the basic models. Customers select the capabilities they want, and modules programmed to perform the desired tasks under software control are shipped with the system. In most other terminals, these functions are performed by hard-wired logic, which can be added or removed from a system only with difficulty.

However, some designers who have tried the Intel 8008 for these functions have criticized it as not being fast enough and not having a large enough instruction set to do an adequate job. Again, the 8080 chip is viewed as a substantial improvement, although it's still too new for users to have accumulated much experience with it.

The Beehive and Digi-log terminals illustrate one of two trends in the computer-terminal market—their microprocessor-based units offer greater power and a higher level of customization, yet at lower cost. The other trend is to the "dumb" terminal under control of the central computer, which provides a simple way to "look into" a computer to see what's going on. "There'll always be a market for a dumb terminal," says Richard Kaufman, director of marketing at Applied Digital Data Systems Inc., Hauppauge, N.Y. Because of these two extreme requirements, the intermediate terminal that has only a small amount of logic capability will disappear. But the best way for designers to keep up with the trend toward smarter and smarter terminals is to use microprocessors to provide the "smartness."

Building controllers

Builders of mechanical peripheral equipment that contains minimal electronic circuitry have no need for microprocessors. However, builders of controllers for this equipment, as well as the manufacturers that build both the mechanical devices and their electronic controls, are more enthusiastic about microprocessors for

Microprocessor aids the mini

Perhaps partly because of a certain degree of overselling by microprocessor manufacturers and partly because of misunderstandings of what a microprocessor is and what it can do, there has been some speculation that the advent of microprocessors means the end of the smaller minicomputers. This is most emphatically not true. On the contrary, by enhancing the capability of the minicomputer, the microprocessor opens a whole new range of applications for the minicomputer that it couldn't touch economically before.

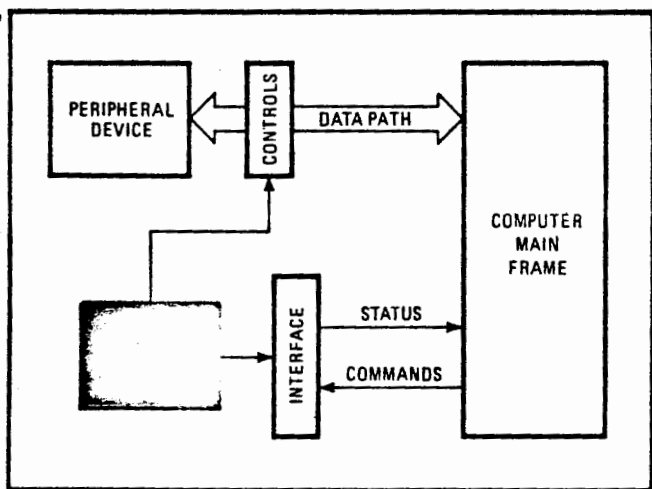
In many of the new applications, the capabilities of microprocessors and minicomputers have been combined to increase the effectiveness of the entire system at only a small increase in cost. For example, David Methvin, president of Computer Automation Inc., describes attempts to drive a series of remote terminals from a single minicomputer. "It didn't work very well," says Methvin, "because the minicomputer's speed and short word length are generally adequate to drive no more than two or three terminals."

But by designing into each terminal a microprocessor to handle some local processing and relieve the control minicomputer of the drudgery, it can easily do the higher-level work for the entire network. "In this way," concludes Methvin, "the advent of the microprocessor creates a new market, not only for itself, but also for the minicomputer, which previously had to yield to something bigger and costlier."

Microdata Corp., Irvine, Calif., a leading producer of microprogrammed minicomputers, has not yet begun using the single-chip p-channel-MOS microprocessors in any of its computers. However, Richard Vahlstrom, technical director, foresees a possible utilization of the devices in peripheral equipment whenever they become cost-effective. Meanwhile, Microdata has introduced its Micro-One, a one-board bipolar processor that is both software- and firmware-compatible with the company's older 800 and 1600 series minicomputers and with their peripherals [*Electronics*, 5/30/74, p. 142].

Digital Equipment Corp. and General Automation Inc. have already recognized this trend, as shown by their recent product announcements. General Automation now has two minicomputers based on silicon-on-sapphire microprocessors, while DEC's microprocessor module, an extension of its long-standing line of logic modules, is based on Intel's 8008 microprocessor—one of some 53 circuits on the card. The PDP-8/A is the company's latest version of the line with which it more or less invented the minicomputer market back in 1965. The original PDP-8 was a discrete-component computer in a big box 34 inches high and almost two feet square. But now the complete set of 79 PDP-8 instructions can be executed by an assembly of components on a single printed-circuit board measuring 15¼ by 8½ inches, not including the memory.

The PDP-8/A makes extensive use of LSI, but none of the circuits is a microprocessor. Future versions of this and other DEC computers may include circuits that would be called microprocessors today. William Hogan, marketing manager for logic products in DEC's components group, describes the microprocessor module as the first of a series of products that will use any appropriate semiconductor chips with the right combination of cost and performance.

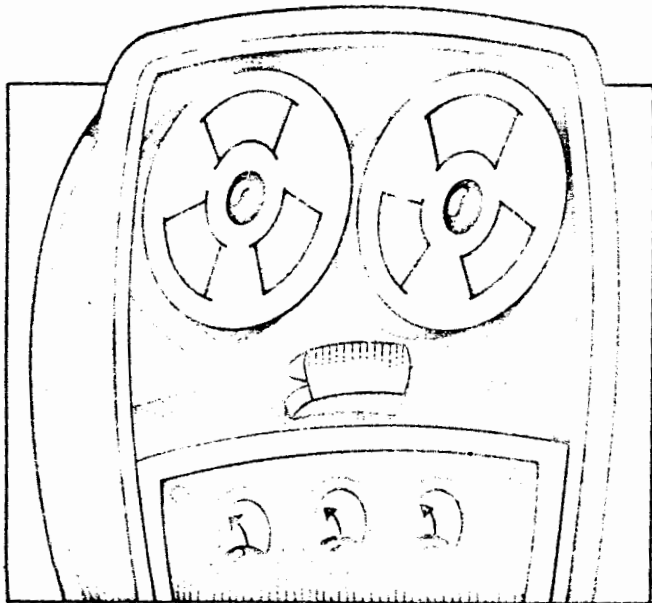


1. Interface and control. Today's microprocessors are applied in computer systems primarily where they do not affect data flow, but that limitation will only last until their performance improves.

use in the interface and control sections of their machines (see Fig. 1). The interface section responds to signals from the central processor and generates status signals to it. The control section sets up the device controller for a particular task. But neither of these functions is concerned with the actual passage of data through the controller, which may involve such steps as serial-to-parallel conversion, assembly of words from bytes, and error detection and correction.

Although the enthusiasm of controller designers, like that of terminal designers, is tempered by the performance level of presently available microprocessors, they look forward to a new generation of microprocessors now on the drawing board. The higher performance of the new microprocessors will enable them to graduate to full use in the data path as well.

New single-chip processors made with silicon-on-sapphire and bipolar technologies are expected to promote such a graduation by reducing the typical execution time to the range of 50 to 500 nanoseconds (from today's 2 to 20 microseconds) and increasing the number of instructions toward 200 (from today's 40 to 75).



Specifically, Intel is reported to be working on a bipolar microprocessor that can execute instructions in as little as 500 nanoseconds. In MOS, Rockwell's Microelectronics Group, one of the earliest to exploit sapphire technology, is developing more powerful processor chips. They already supply devices to General Automation for their LSI microcomputer line. Also, Inselek Corp., Princeton, N.J., has proposed a C-MOS-on-sapphire microprocessor that could handle a data rate of 3 megabits per second with its cycle time of 300 ns. Inselek says the device will be available early in 1975.

For its paper-tape emulator, Remex, a unit of Ex-Cell-O Corp., Santa Ana, Calif., chose the Intel MCS-4, a chip set that includes the 4004 4-bit microprocessor. The emulator is a magnetic-tape cassette drive that works with a minicomputer like a paper-tape reader.

Currently, the programs for the emulator are stored in programable read-only memories—the kind that can be erased under ultraviolet light. Program bugs turned up by the first few customers can be easily corrected. Later, when change activity has died down somewhat, Remex plans to switch, perhaps first to fused-link ROMs, which can be updated but not erased, and eventually to masked ROMs that can't be changed in the field at all, except by physically exchanging one part for another.

Stamp out logic monsters

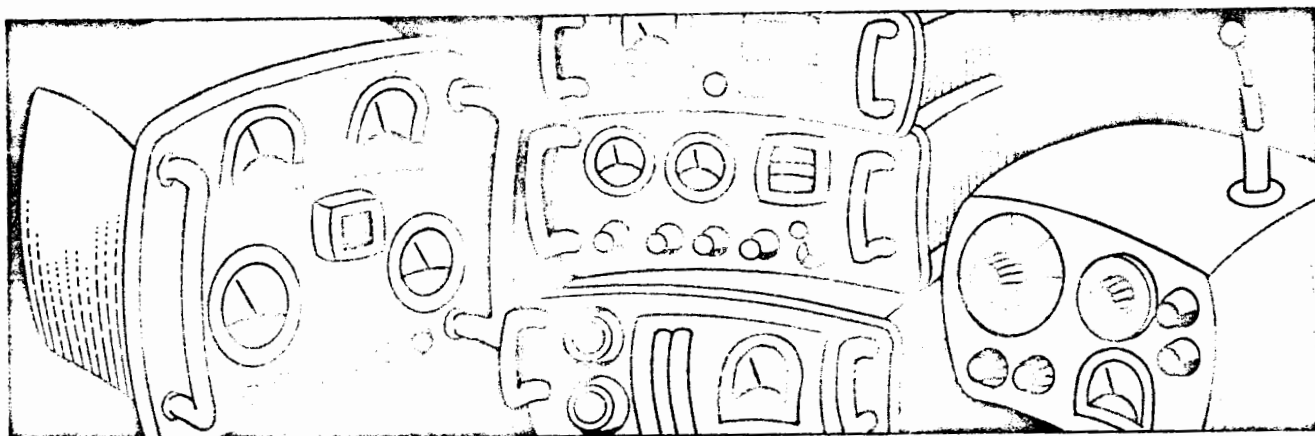
Decision Data Corp., also of Horsham, Pa., indicates great interest in microprocessors—particularly in data recorders. In these peripherals, a few microprocessors replace a multitude of interconnected integrated circuits. The company manufactures a line of punched-card machines—both the old standard 80-column type and the newer, smaller 96-column equipment for IBM's System/3 and similar computers. The line includes a data recorder, which is a sort of combination keypunch, card reader, card punch, and printer, with various other outputs available and usable either on line with a computer or as a stand-alone device.

"Data recorders are monsters in logic," says Thomas Richardson, vice president for engineering, referring to the multitudinous functions that the machines perform. Present designs, he says, use 700 to 800 small-scale integrated-circuit packages, plus as many as 400 signal lines to interconnected equipment—clearly a prime territory for an invasion by microprocessors. Richardson indicates that the company will begin to move in this direction before the end of 1974, initially with its own design implemented with medium-scale ICs and later graduating to bona fide LSI microprocessors.

Despite the advances already made, development of faster, more powerful microprocessors is continuing. Although today's microprocessors have word lengths of 4 to 8 bits and instruction-cycle times of 2 to 20 microseconds, at least one 12-bit unit has already been developed in Japan [*Electronics*, 3/21/74, p. 111]. And semiconductor manufacturers in the U.S. are working feverishly to increase speeds.

What's more, LSI processors being made with bipolar and SOS technologies are yielding processor chips that blur the distinctions between the capabilities of the microprocessor and the small minicomputer. Indeed, the LSI miniprocessor is already on the design bench.

Instruments



Systems are getting 'smarter'

by Michael J. Riezenman, Industrial Editor

Anyone who has ever twiddled the controls of a pulse generator, wasted a couple of hours trying to recall how to use a scope's delayed-sweep feature, or laboriously calculated the standard deviation of a set of measurements knows that it takes detailed knowledge and refined techniques to use a complex modern instrument properly and efficiently. But soon, microprocessors will bring about a new generation of "intelligent" instruments that will automatically relieve the operator of routine procedures. And most of these "smarter" instruments will be cheaper than the ones they replace.

Instruments can be made less costly because, in many cases, the software techniques used with microprocessors will be cheaper than the hard-wired logic and mechanical switches they replace. Probably multi-instrument systems can benefit even more because microcomputers should replace minicomputers or programmable calculators in small systems and do much of the repetitious work so that much cheaper minicomputers or calculators can be designed into larger systems.

Many cost-related benefits can be expected from a whole new class of small computer-controlled instrument systems that would be too expensive if built with minicomputers or even programmable calculators. And for large systems that need minicomputers as controllers, microprocessors may be able to make significant reductions in the costs of the computers by using them as preprocessors in the instruments that are controlled by, and are feeding data to, the minicomputers.

These reductions should be significant. The intelligent application of, say, \$400 worth of microcomputer components in each of five or six instruments may make it possible to replace a Digital Equipment Corp. PDP-11/45 minicomputer that costs about \$16,000 to \$18,000 in an appropriate configuration, with a PDP-11/40 at a cost of about \$12,000.

The straight bench instrument that can probably realize the greatest cost reduction by use of a microprocessor is the frequency synthesizer. A high-resolu-

tion synthesizer uses a large number of expensive electromechanical switches and a great deal of complex logic circuitry simply to tell the frequency-generating circuitry what to do. Replacing these switches with simpler, cheaper ones or with a keyboard and replacing the logic circuitry with a microprocessor will, in most cases, justify the cost of the microprocessor, even if it brings no other benefits.

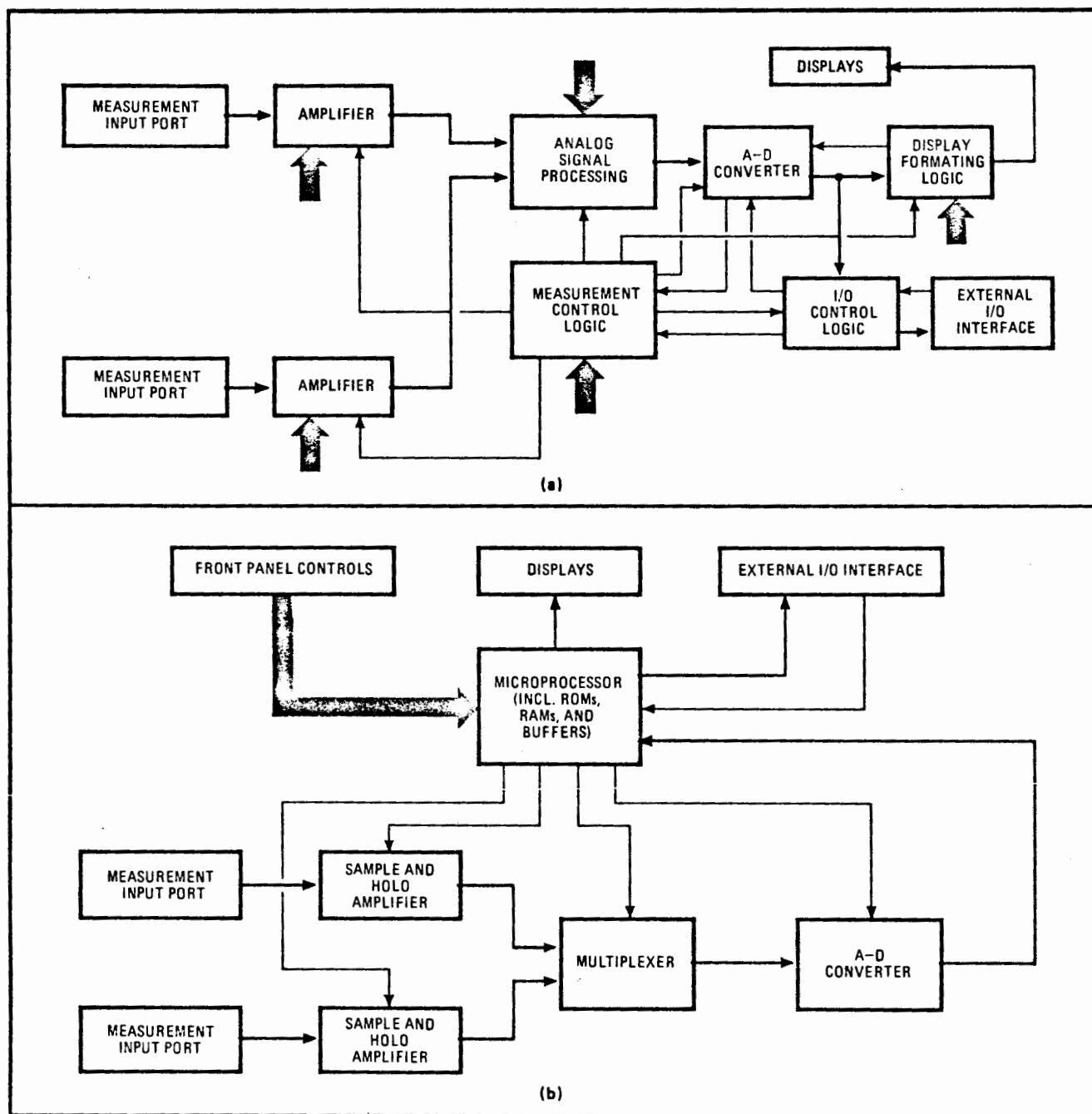
Of course, microprocessors will provide a whole host of such other benefits as data-formatting for the instrument's input/output interface. Moreover, the microprocessor will enhance accuracy and reliability through the use of special routines for self-diagnosis and the elimination of systematic errors.

Automating testers

Already several manufacturers have introduced microprocessor-controlled instruments. Among them are the in-circuit IC testers built by Testline Inc., Titusville, Fla.; the model 76A automatic capacitance bridge made by Boonton Electronics Corp., Parsippany, N.J.; the Qualifier 901 IC tester made by Fairchild Systems Technology, Palo Alto, Calif.; the Digitrend 220 recorder made by Doric Scientific Corp., San Diego, Calif.; and the interfacing circuitry used by Tektronix Inc., Beaverton, Ore., to marry the digital processing oscilloscope with the company's model 31 programmable calculator.

Most of these pioneering applications use the microprocessor more as a manipulator of bit patterns than as a number-cruncher. The microprocessors are used more to set up tests, perform interfacing chores, and control other subsystems than to process the data that the instruments have acquired.

The IC testers are perhaps the best examples of this emphasis, since, strictly speaking, these instruments acquire no numerical data at all. They use the microprocessors for the quick and easy setup of complex input-bit patterns and comparison of actual and expected output-bit patterns without resorting to either expensive minicomputers or so-called performance



1. Generalized Instrument. Typical digital-readout instrument is really an analog machine with a lot of digital control and display circuitry tacked onto it (a). In particular, signal processing is performed by conventional analog means. With microprocessor, a-d converter is moved up front so that most processing can be done digitally (b). In both diagrams, signal paths are shown in color, and control lines are in black.

boards. One of the additional benefits of the Qualifier 901 is a thorough self-test routine. Under control of the microprocessor, the machine checks itself out every time a program is loaded into it.

While they use the microprocessors largely for control, the Boonton capacitance bridge and the Doric recorder also exploit the processor's ability to do a bit of numerical calculation, as well. The capacitance bridge directly measures only capacitance and conductance. It then processes these numbers to find such quantities as equivalent series resistance, equivalent parallel resistance, Q , dissipation factor, and percentage of deviation from a preset reference.

The Doric Digitrend 220 recorder is programed with

a set of linearizing equations for various thermocouples. Instead of using different linearizing networks for each of the six common thermocouples (types J, K, T, E, S, and R), the instrument does it all with software. (See p. 87 for more details on this instrument.)

A generalized microprocessor-controlled instrument and its conventional digital counterpart are shown in Fig. 1. The exact nature of the instrument is not specified, but it may either be a two-channel voltmeter or a wattmeter.

The main point is that the conventional version of the generalized instrument (Fig. 1a) does all of its processing, which may include such difficult operations as multiplication and linearization before the output is digi-

tized. Also, the conventional instrument needs lots of logic circuitry to control the making of the measurement, to format the digital display, and to handle the I/O interface with any other equipment to which it may be connected in a system.

The microprocessor-controlled instrument, on the other hand, (Fig. 1b) converts the data into digital form as close to the front end as possible and does all of its signal processing digitally. Its potential for cost reduction comes from the capability of a single microprocessor to do the signal processing and also handle all of the interfacing and formatting chores that would require literally hundreds of TTL packages.

Getting 'smart'

The most dramatic impact of microprocessors on instrumentation will be in the creation of new "smart" instruments for a host of new applications. A smart instrument is one that performs a significant amount of internal arithmetic processing. From a number of inputs (either signals or switch positions), it calculates an output to display and/or performs additional processing.

Indeed, smart instruments, like people, may be expected to come with a wide range of intelligence. At the low end of the spectrum may be a digital voltmeter for communications applications that can be programmed to make, say, 1,000 measurements and then display their mean and standard deviation. For this, the microprocessor, together with associated control memory and I/O circuitry, would perform all the logic-management functions.

Assuming that very fast measurement times aren't needed, such a system could be built of one of today's 8-bit n-channel microprocessors, together with, say, eight 1-kilobit random-access memories to supply 8,000 bits of main memory and the associated read-only memory for control, plus I/Os. The entire system could be implemented with fewer than 20 LSI packages—only a tenth of the more than 200 standard TTL circuits that would be needed.

A somewhat smarter instrument might modify its own behavior as a result of its calculations. An example of such an instrument already exists—it is Hewlett-Packard Co.'s model 3805A distance meter. This surveyor's tool measures distances by measuring the time required for an infrared beam to travel from the instrument to a reflector and back.

Since atmospheric perturbations can affect the readings, the meter is programmed to make 3,000 measurements and to calculate their mean and standard deviation. Then, if the standard deviation is within a specified limit, the mean is displayed as an accurate reading. If the standard deviation is out of spec, the meter makes as many additional measurements as are necessary to get it within spec. If, after it has made 32,000 measurements, the instrument still fails to get a sufficiently good standard deviation, it displays the mean in flashing numerals to tell the operator that the measurement conditions are less than ideal.

The next level of instrument made possible by microprocessors could, by today's standards, be called geniuses. These instruments will probably be most noteworthy for their high degree of human engineering.

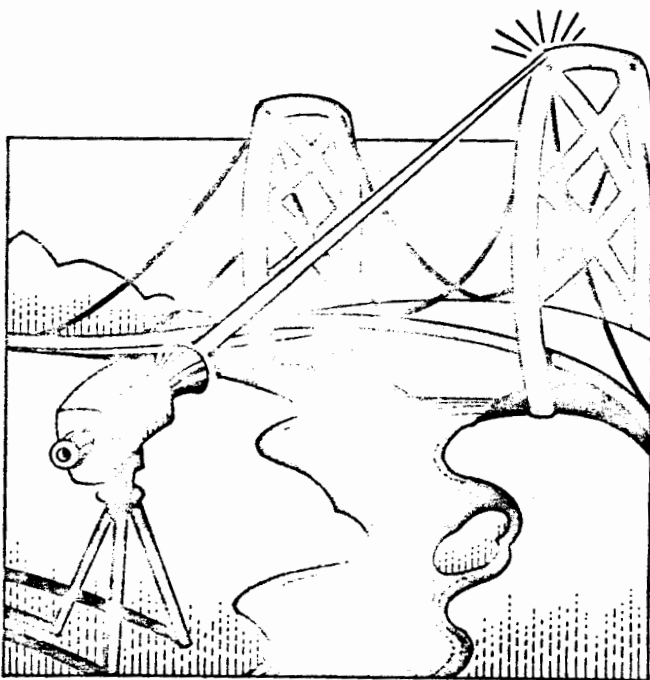
Their value may best be appreciated by considering the hairy problems that one may encounter when using a complex pulse generator or oscilloscope. Highly skilled engineers, not to mention technicians and service personnel, can easily waste several hours refamiliarizing themselves with instruments that they haven't used for several months. Even an instrument that one has used every day can present problems if someone else borrows and returns it with some small, seldom-used, control out of its usual position.

Building 'geniuses'

Microprocessors can and will be used to generate a "genius" class of instruments, but how it will be done is uncertain. One can imagine an oscilloscope that has had most of the knobs and switches replaced by a keyboard through which one punches in such parameters as the sweep speeds, vertical sensitivities, and triggering modes needed for any particular application.

Seldom-changed controls might be automatically set to preprogrammed states from which they could be changed, via the keyboard, if desired. The status of the machine could be presented on the cathode-ray tube by a character-generator similar to the one already available on Tektronix' 7000 series scopes. In addition to simply presenting the machine's status, the CRT readout could also warn of incompatible instructions or of valid, but unusual, measurement conditions.

In a sense, oscilloscopes and other measurement tools aren't difficult to deal with because they present the user with displays, which, if abnormal, warn that corrective action is needed. The myriad possibilities for error in setting up signal generators, synthesizers, and other signal sources, on the other hand, can drive an engineer to a psychiatrist. Few users of pulse-generators can claim that they have never set the pulse width to a duration longer than the period defined by the selected repetition rate. And on some complex two-channel pulsers



Who Invented the microprocessor?

Intel Corp. certainly deserves the credit for exploiting the microprocessor concept and was the first to market microprocessors, although much credit must also go to the many companies and individuals who contributed in some way to the development of large-scale integration.

Remember Viatron? In 1968, the Burlington, Mass., firm startled the world by announcing its intention to build a data-handling system that would rent for \$40 a month in its basic configuration. [*Electronics*, Oct. 14, 1968, p. 193]. Heart of the Viatron unit was an 8-bit microprocessor run by a primitive program in a read-only memory. But the company encountered serious financial

and management problems, and it went bankrupt after about two years.

Meanwhile, General Electric Co. found itself designing integrated logic circuits for some of its terminals, duplicating much of the work from project to project, but not generating enough volume on any one of them to justify the use of custom-designed LSI—until somebody thought of a customized *programmable* LSI circuit. GE then developed an eight-chip basic logic unit, or BLU, that could be used without change with different programs in many different terminal designs—essentially what is done today with microprocessors.

that have separate controls for such settings as amplitude, offset, delay, pulse width, and trigger mode, the fact that these highly interactive controls have been set wrong is not always obvious.

The microprocessor can unravel that complexity. If all of the instrument's operating information is fed in through a small keyboard-controlled processor, the instrument could simply refuse to accept an input that is incompatible with earlier instructions. Alternatively, electronic stops could be programed into the machines, and a small light-emitting-diode display could be positioned above a vernier pulse-width control. As the control is rotated to increase the pulse width, the display would reflect its position, so long as the pulse width did not conflict with any other control settings.

If such a conflict arises, the machine might be programed to ignore the control setting and to set only the maximum pulse width that could be accommodated. The LED display would keep the operator informed of what is happening by always showing the actual pulse width being generated, regardless of the front-panel control setting.

Although each of the ways in which a microprocessor might be used in an instrument has been discussed as a separate idea, it should be clear that, at least until their

prices are reduced considerably, the devices will be used primarily in applications where they can perform several functions.

Most industry sources agree that an instrument would have to sell for at least \$2,000 to \$3,000 to justify the inclusion of a microprocessor. There is no upper limit to the size of instrumentation systems in which microprocessors could be included, since even systems large, complex, and costly enough to justify the use of a minicomputer may benefit from the inclusion of microprocessors as preprocessors.

Peak picker

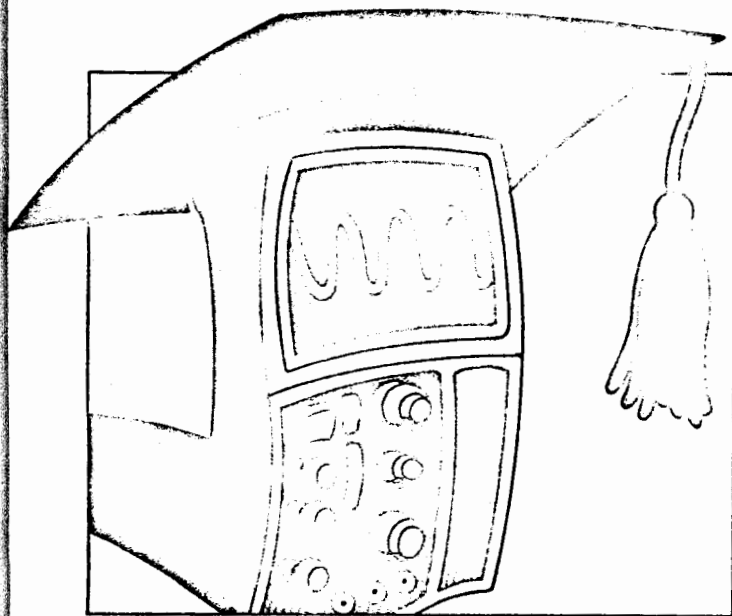
Such an application might be in an analytical chemistry laboratory, where a single, fairly small minicomputer could control, say, two or three mass spectrometers and a dozen gas chromatographs if each of them were equipped with a microprocessor programed to act as a peak-picker. The outputs of these analytical instruments, if drawn by a chart recorder, are typically a series of peaks separated by nulls. Unfortunately, closely spaced peaks tend to blend into each other, which makes it difficult to decide exactly where the peaks are.

An experienced human operator can locate the peaks by eye, but it takes a fairly complex computer program to do the job. If the computer is to do all the peak-picking, it would have to be an extremely fast machine with a lot of memory. Adding the microprocessors brings the task well within the capabilities of a minicomputer of modest size.

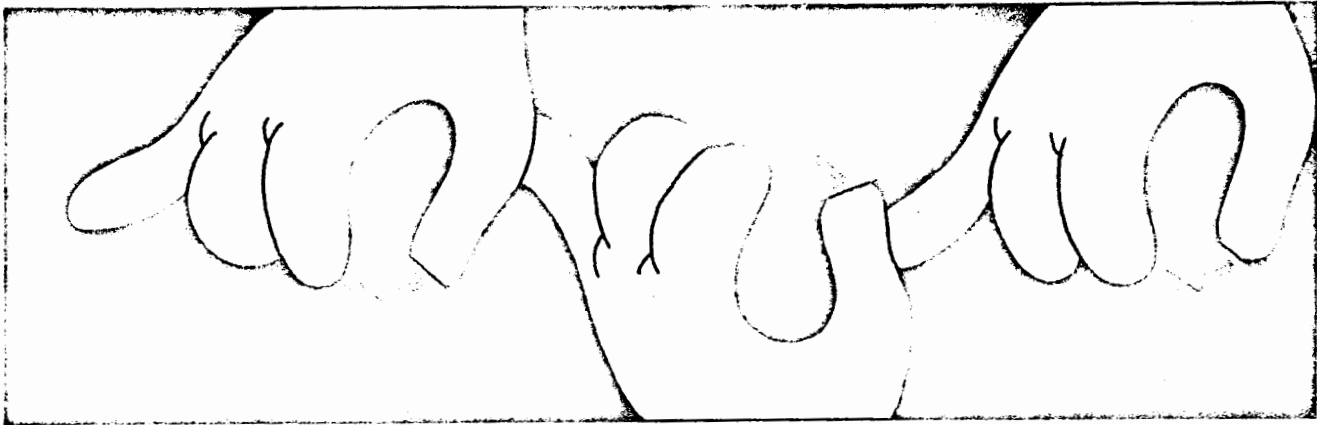
Improvements are imminent

Thus far, only a handful of commercially available instruments contain microprocessors. But this state of affairs in no way indicates a lack of interest in them by the major instrument houses. Quite the contrary.

Although details are not yet available, it is clear that microprocessors are responsible for previously unavailable or unaffordable capabilities that will be offered in several new meters, counters, signal sources, and oscilloscopes before this year is out. The designers of these instruments speak of "totally new approaches to the making of measurements" but, understandably, they refuse to elaborate on what that means right now. However, the next six months promise plenty of excitement for the makers and users of electronic instrumentation.



Design



Blending hardware and software

by Wallace B. Riley, Computers Editor

It's a whole new world, but it's really not all that different from what the engineer is accustomed to. Supposedly, EEs experienced in the conventional approach to design—flip-flops and gates—might expect to encounter difficulty expressing their design ideas in terms of software. But, although the end result of a software development effort looks different on paper from the traditional logic diagram, it is basically identical.

An engineer's usual approach begins with a set of functional specifications, which he translates into a block diagram and then reduces to the level of individual gates. The completed design is assembled on a breadboard, built into a prototype, and then, with a series of tests and redesigns, reduced to a form that can be manufactured in volume and sold at a profit. Meanwhile, it may be undergoing simulation on a computer as part of the design refinement.

Likewise, software design begins with functional specifications, but it is translated into a sequence of instructions, rather than into an array of gates. The paper design usually involves a flow chart, which shows events graphically in the proper order, together with conditions that can cause the order of events to change. The first step can be a high-level flow chart, which closely resembles the block diagram. This is broken down into a form in which each block in the flow chart represents a single instruction in the program. Standardized shapes of blocks in the flow diagram have evolved (Fig. 1) so that one person can more easily follow the logic of another person's work. [For an example of applying a flow chart to either the hardware or software implementation of a specific design, see *Electronics*, Oct. 11, 1973, p.97.]

For some individuals, software is a problem until they get the hang of it. At some companies, teaching engineers how to program and programmers the limits of hardware has turned out to be a great enlightenment on both sides. But the highly motivated people who undertook the project knew that understanding microprocessor software would be essential sooner or later, and they have managed to overcome any obstacles to

understanding. Still other companies have assigned the task to younger engineers who had no previous strong commitment to either hardware or software designs, and who, therefore, made the transition easily.

In the last analysis, any intelligent person who can lay out a procedure accurately one step at a time can learn to write a program for a microprocessor.

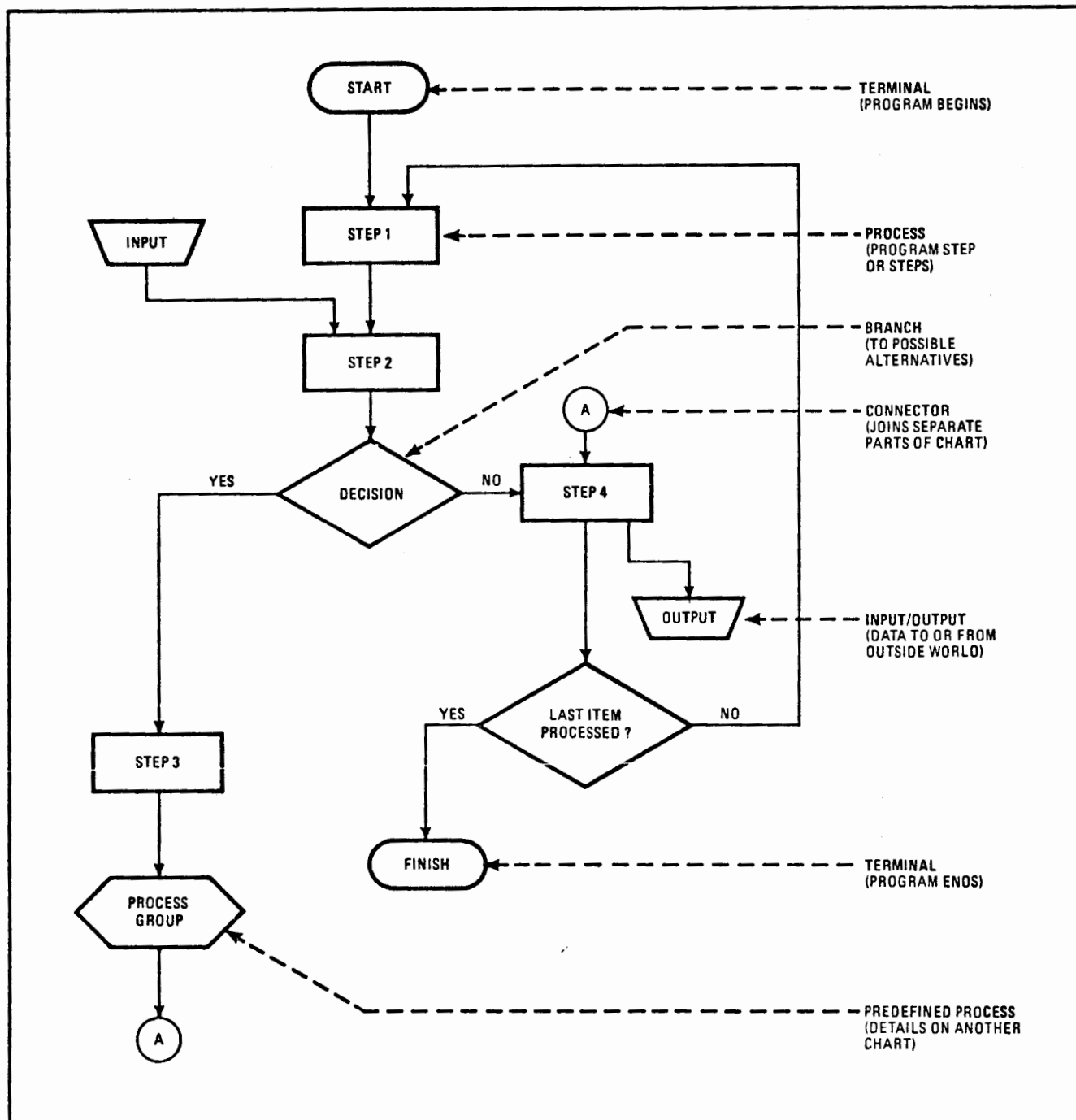
Support is essential

Some users and potential users of microprocessors express concern about the level of software support from the manufacturer. Since microprocessors come from semiconductor houses, those users fear the vendors don't have the capacity to offer the assistance that is expected from the IBMs or the DEC's.

The concern is largely unfounded because the need for software support, compared to the requirements of large computers, is small indeed. But to the extent that microprocessor users have had no previous exposure to computers, they may need to be led through thickets of unfamiliar concepts to get their applications working.

Support for a large general-purpose computer is significantly different from support for a minicomputer, and it differs even more from the kind of support that a microprocessor user will need. And since a general-purpose computer is likely to cost its user hundreds of dollars an hour, he doesn't want to shut it down even momentarily if he can avoid it—not even to load new programs into it. To protect him from unnecessary expense, manufacturers offer operating systems, which are software packages designed to keep the machine running under all but the most catastrophic conditions, as well as various aids that simplify the task of writing programs for the large computer.

But a minicomputer is likely to be operated in a dedicated application so that a single program runs over and over indefinitely. Furthermore, it's sufficiently inexpensive that its occasional stopping between jobs or when an error occurs is only an inconvenience, not a major expense. Minimakers also offer support, in the



1. Flow-diagram conventions. These are among standard shapes that simplify communications via flow charts. Also often used are specific input/output functions, such as a torn sheet of paper for a printout, a tape reel for magnetic tape, or a cylinder for a disk or drum.

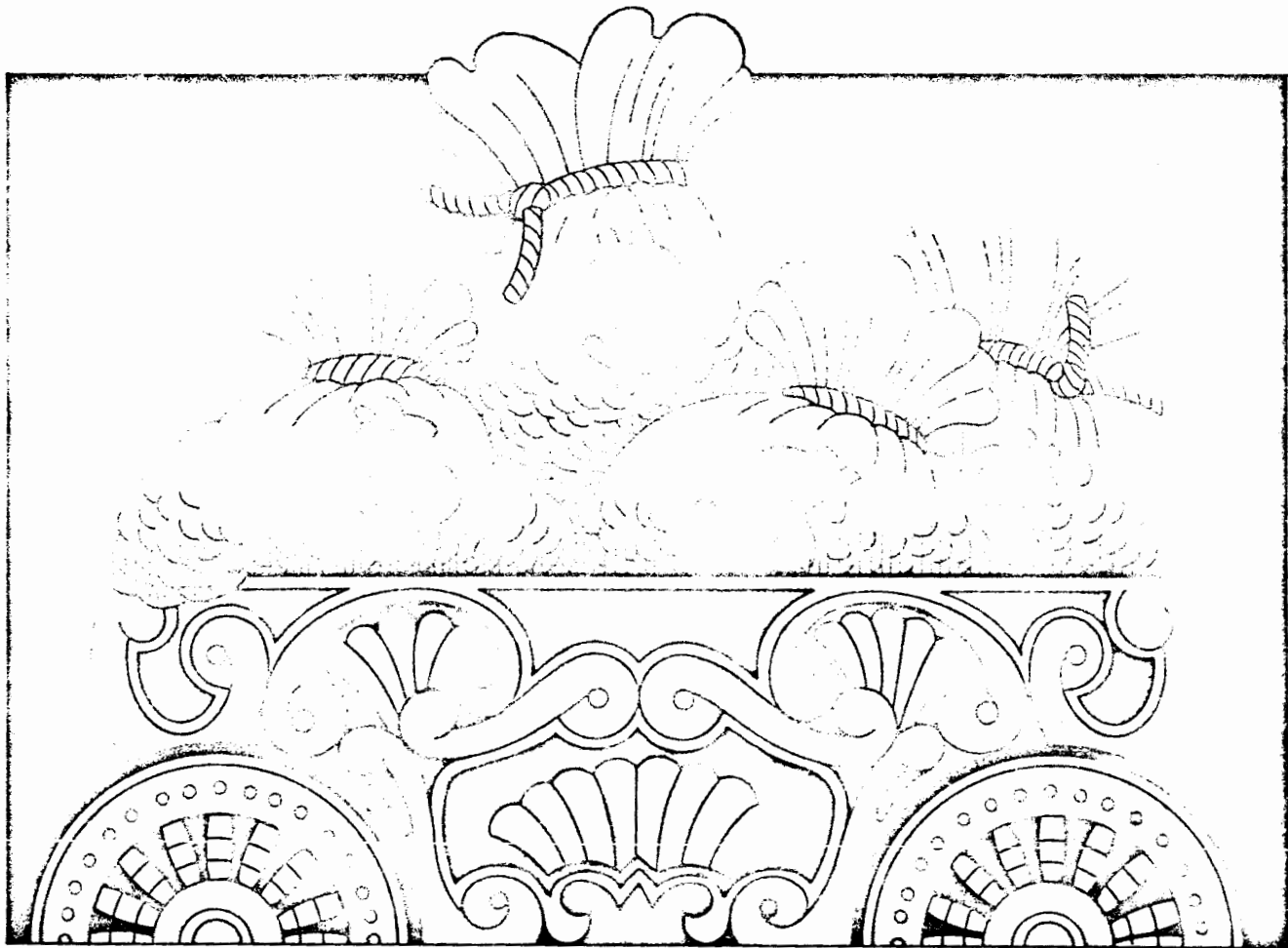
form of some kinds of programing aids and perhaps a relatively low-level time-sharing system. Of course, there's maintenance of the hardware, but this is far short of the support that is expected from the manufacturer of a general-purpose computer.

None of this kind of support applies in any way to microprocessors, except possibly in the form of higher-level programing languages like PL/M [*Electronics*, 6/27/74 p. 103]. Like the average passenger automobile, a microprocessor remains economically feasible, even though it may be "parked" 90% of the time. Its program is likely to be wholly in a read-only memory, making it even more dedicated than a dedicated mini. It doesn't even need hardware maintenance, because it

can't be patched the way a board or chassis can.

For users wholly unacquainted with the art and science of computer application, extensive support from the vendor will be necessary. By and large, this support is available now—in the form of users' manuals, programing manuals, application notes, and similar documentation—and it shows no signs of abating. But in all probability, using microprocessors won't be a wholly new experience for most people. Many engineers have already used minicomputers in some form or have enlisted the aid of various computer-aided-design programs. And, as indicated previously, most of those who have already tried microprocessors haven't found the software a serious problem.

Processors and product costs



How microprocessors boost profits

by William Davidow, Intel Corp., Santa Clara, Calif.

When the first single-chip microprocessors were introduced two years ago, few designers or project managers could foresee how massively these devices would influence creation of new products and services. To most, the microprocessor was merely another interesting LSI device to be evaluated and built with some memory and interface chips into prototype equipment.

But as designers became familiar with the early microprocessors and equipment began to benefit from them, respect grew for their capability and versatility. Rapidly there arose an awareness throughout the electronics industries that the microprocessor was indeed a significant extension of computer technology. Suddenly, the concept of distributed computer power became a reality, applicable to a host of new equipment.

There are compelling and fundamental reasons for the dramatic success of today's microprocessors:

- Manufacturing costs of products can be significantly reduced by designing around microprocessors.
- Development costs and time are reduced.
- Products can be rushed to the market faster, which enables a company to seize advantages in sales and market share.
- Product capabilities are enhanced, and manufacturers can economically add features that boost profits.
- Product reliability is increased, leading to a corresponding reduction in both the cost of service and warranties.

Microprocessors enable designers to replace hardware with software. Using programed logic, they can

ROM size (bits)	Gates replaced	ICs replaced
2,048	128 - 256	13 - 25
4,096	256 - 512	25 - 50
8,192	512 - 1,024	50 - 100
16,384	1,024 - 2,048	100 - 200

now substitute a handful of ICs for a large number of conventional random-logic networks. In such a system, the information about logical sequences and the output responses provided from input signals are stored in a few memory chips instead of in relatively expensive interconnect patterns on printed-circuit cards.

Use of microprocessors saves money and time at every stage of the product's life cycle. These savings are passed on to the customer in products with greater capabilities and higher reliability than has ever before been attainable. Microprocessors are not only improving the performance of established products, they are bringing about completely new products. They are beginning to permeate every walk of life.

Memory replaces random logic

If microprocessors were fast enough with their programmable techniques, they could replace all hard-wired logic. But as the speed of new generations of microprocessors is increased, they will move into more and more designs now implemented with conventional ICs. And although each new application has its unique structure, it's possible to estimate the package reduction that accrues when hard-wired random logic is replaced by programmable techniques.

Again, the microprocessor replaces logic by storing program sequences in memory, rather than implementing these sequences with gates and flip-flops. While it is impossible to prove quantitatively, designers use a rough rule that they can replace one gate by using 8 to 16 bits of memory. Therefore, if the average hard-wired logic circuit contains on the order of 10 gates, Table 1 indicates that a single 4,096-bit read-only-memory can replace 50 MSI packages. Each new 16,384-bit ROM save as many as 200 IC packages in every design. No wonder system designers are being so quickly convinced of the capability of microprocessors to reduce IC complexity.

Reducing manufacturing costs

Clearly, reduced IC complexity translates directly into reduced product costs. Table 2, which presents a detailed analysis of the sources of these surprisingly high costs, shows that the average sale price of an integrated circuit today is approximately 50 cents. Incoming inspection and testing cost an average of 5 cents more.

Many companies are now buying aged and tested circuits for their applications to increase system reliability, and this adds about 15 cents to unit costs. A simple printed-circuit card may cost as little as 25 cents for each IC position, but the average cost in most appli-

IC	\$.50
Incoming inspection	.05
Pc card	.50
Fabrication	.05
Board test and rework	.10
Connector	.05
Discretes	.05
Wiring	.10
Power	.10
Cabinetry, fans, etc.	.10
Total	\$ 1.60

cations for high-quality cards is closer to 50 cents. (In some systems, costs of sophisticated multilayer cards can go as high as \$1 per location, and if wire-wrap assemblies are used, the cost per IC position can reach the \$2 mark.)

Next, board test and rework add another dime to system cost, while the cost of a connector, divided by the number of ICs per printed-circuit card, frequently exceeds 5 cents. Then the system requires such components as resistors, capacitors, and power-bus bars, which add another 5 cents per IC.

Systems frequently average one wire or more per IC position, and the wires—even those installed by automatic equipment—frequently cost more than 10 cents each. Finally, the cost of power supplies and mechanical packaging add another 20 cents. Altogether, the minimum system cost approaches \$2 per IC.

Table 3 shows the potential system saving in manufacturing costs that can be achieved by using a microprocessor. The savings are derived by assuming that the typical manufacturer can save \$1.50 to \$3 by displacing a single IC, after which the cost of implementing an equivalent system with a microprocessor is taken into account. In moderate volumes, a system such as the MCS-4, made up of 16,384 bits of ROM, a processor, and a minimal amount of RAM, can be purchased for less than \$100. This system has the potential of displacing \$150 to \$600 of manufacturing cost in a system.

Reducing development time

Use of microprocessors simplifies nearly every phase of product development. Because of the extensive design aids and support supplied with microprocessors, it

ROM size (bits)	IC replaced	Savings
2,048	13 - 25	\$19.50 - \$78
4,096	25 - 50	\$37.50 - \$150
8,192	50 - 100	\$75.00 - \$300
16,384	100 - 200	\$150.00 - \$600

	Conventional system	Programed logic
Product definition		Simplified because of ease of incorporating features
System and logic design	Done with logic diagrams	Can be programed with design aids (compilers, assemblers, editors)
Debug	Done with conventional lab instrumentation	Software and hardware aids reduce time
Pc card layout		Fewer cards to lay out
Documentation		Less hardware to document
Cooling and packaging		Reduced system size and power consumption eases job
Power distribution		Less power to distribute
Engineering changes	Done with yellow wire	Change program in PROM

is relatively easy to develop applications programs that tailor the devices to the systems and then implement these systems in very short turn-around times. Indeed, a principal reason for the increasing popularity of microprocessors is the speed with which products can be developed, designed, and rushed to the market. Discussions with system designers indicate development cycles have frequently been shortened by as much as six to 12 months to only a few weeks.

Table 4 tabulates a number of the steps in a system-development cycle and the effects of the microprocessor on the design-cycle time—designing becomes easier, faster and less costly. Surprisingly, product definition is frequently speeded up as soon as the decision is made to use a microprocessor because the incremental cost for adding features to the system is usually small and can be easily estimated.

For example, adding such features as automatic tax-computation to an electronic cash register may require only the addition of a single ROM, which has a minimal effect on total system cost, power dissipation, and packaging requirements. But adding the same function by means of IC logic might require two or three fairly large pc cards filled with SSI and MSI logic packages.

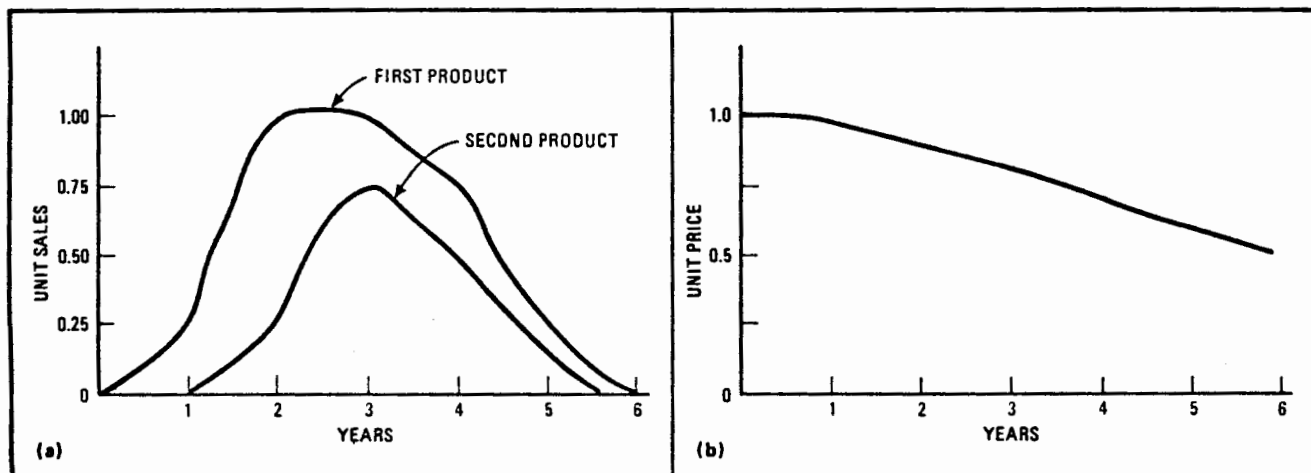
Building around microprocessors also reduces the time needed for system design. When the engineer de-

cides to use a microprocessor, he designs by programming—potentially a more organized and faster way to design than by using logic diagrams. What's more, the ready availability of extensive software aids, such as simulators, assemblers, editors, compilers, and monitors, reduces the cost of program development. These aids also reduce the time needed for system debugging. Pc-card layout time is reduced simply because fewer cards need to be laid out.

Getting to market fast

When product-design cycles can be shortened, obviously new products can be rushed to the market faster. This permits companies to either beat out the competition or effectively respond to competitive moves. Figure 1 shows what typically happens in a competitive program when one company beats another to the market. Assuming that both companies have about the same marketing capability, the company that introduces the product first usually can gain a greater share of the market (Fig. 1a) and reach a mature sales volume more quickly.

Figure 1(b) shows the price erosion characteristic of most products during their life cycles. This erosion means that the company introducing the product first will not only sell more but will sell at a higher price. In



1. Market jump. Microprocessor design helps get equipment to the market fast, resulting in a greater share of market than second entry (a) as product matures. What's more, first product shows a slower rate of price erosion (b).

FIRST PRODUCT			
Year	Price	Unit sales	Income
1	\$ 1.00	.25	\$.25
2	.90	1.00	.90
3	.80	1.00	.80
4	.70	.75	.52
5	.60	.25	.16
Total			\$ 2.63
SECOND PRODUCT			
Year	Price	Unit sales	Income
1	\$ 1.00	0	\$.00
2	.90	.25	.23
3	.80	.75	.60
4	.70	.50	.35
5	.60	.10	.06
Total			\$ 1.24

	Without microcomputers	With microcomputers
Sales	100%	100%
Cost of goods sold	-55	-45
Gross margin	45%	55%
Development		
Engineering	8 %	6%
Documentation	1.5	1
Warranty	1.5	1
Marketing	20	20
G & A	3	3
Engineering and marketing costs	34%	31%
Before-tax profit	11%	24%

this hypothetical example, the first product to the market generates about twice the total income that the second product does (Table 5). As a result, the advantage gained by application of a microprocessor to achieve early product introduction can have a much greater impact than merely reducing manufacturing costs.

Again, since product features can be added to equipment built around microprocessors simply by adding more program storage, many manufacturers are taking advantage of this characteristic to increase the value of their products. For instance, makers of point-of-sale equipment are adding automatic tax-computation to cash registers by merely increasing the ROM size. Instrument makers are adding automatic calibration to their instruments. Makers of vehicular-traffic-light controllers are adding automatic sensing of traffic loads to their basic equipment and adjusting the duration of the signals. From a profitability point of view, these optional features, many of which are requested by the customer, are frequently sold at 10 to 20 times the cost of adding them. Some companies have been able to earn sizable profits from marginal products and services through the application of microcomputers.

Because the danger of their failure is eliminated by replacing many ICs, the use of a microprocessor can significantly increase system reliability. A digital system fails most frequently because interconnects fail. The use of a typical 16-pin IC will introduce approximately 36 interconnections in a system (16 interconnections from the chip to the lead frame, 16 from the lead frame to the pc card, two interconnections from the pc card to the back plane, and two interconnections from back-plane point to back-plane point).

If one ROM eliminates 50 ICs, then it eliminates approximately 1,800 interconnections. While little data exists to prove the point, it is believed that the reliability of the electronic portion of a system can be increased by a factor of 5 to 10 by use of microprocessors.

Finally, consider the bottom line. Table 6 presents a comparison based on information from users of the profit-and-loss statements of a hypothetical product line

before and after the use of microprocessors. The product using the microcomputer has a smaller final cost because the manufacturing costs of systems containing microcomputers are generally lower than those built with conventional ICs, and the enhanced capability of many microprocessor-system products enables manufacturers to charge more for their equipment.

In addition, the shortening of development cycles and the elimination of much documentation can save a company another 2.5%. Warranty and service costs, such as those associated with stocking spare parts and training service engineers, can also be greatly reduced. The net effects of all these savings can frequently increase product-line profits by 10% to 20%.

The challenge is here. The design and cost advantages of putting computation and decision-making into equipment are clear messages to product-planning managers for all kinds of manufacturers, many of whom have been outside of the orbit of the electronics industries. These technical managers are finding that the use of microprocessors can affect such basic ingredients of corporate success and failure as manufacturing costs, market share, development costs, time, system reliability, and serviceability. The advantages of microprocessors have been demonstrated already. The challenge now is to use them wisely. □

Want to learn more about microprocessors?

Here are some additional articles on microprocessors that have been published in *Electronics*:

Kildall, Gary, "High-level language simplifies microcomputer programing," June 27, p.103.

Altman, Laurence, "Single-chip microprocessors open up new world of applications," April 18, p.81.

Shima, Masatoshi, and Faggin, Federico, "In switch to n-MOS, microprocessor gets a 2-μs cycle time," April 18, p.95.

Young, Link, Bennett, Tom, and Lavell, Jeff, "N-channel MOS technology yields new generation of microprocessors," April 18, p.86.

Tarui, Tadaaki, Namimoto, Keiji, and Takahashi, Yukiharu, "Twelve-bit microprocessor nears minicomputer's performance level," March 21, p. 111.

Electronics staff, "The minicomputer comes on," Oct. 25, 1973, p.98.

Gladstone, Bruce, "Designing with microprocessors instead of wired logic asks more of designers," Oct. 11, 1973, p.91.

SESSION OUTLINE

1. FRED SYSTEM : DESIGN
PHILOSOPHY AND SYSTEM
ARCHITECTURE
2. TV INTERFACE : DESIGN
AND USE

FRED

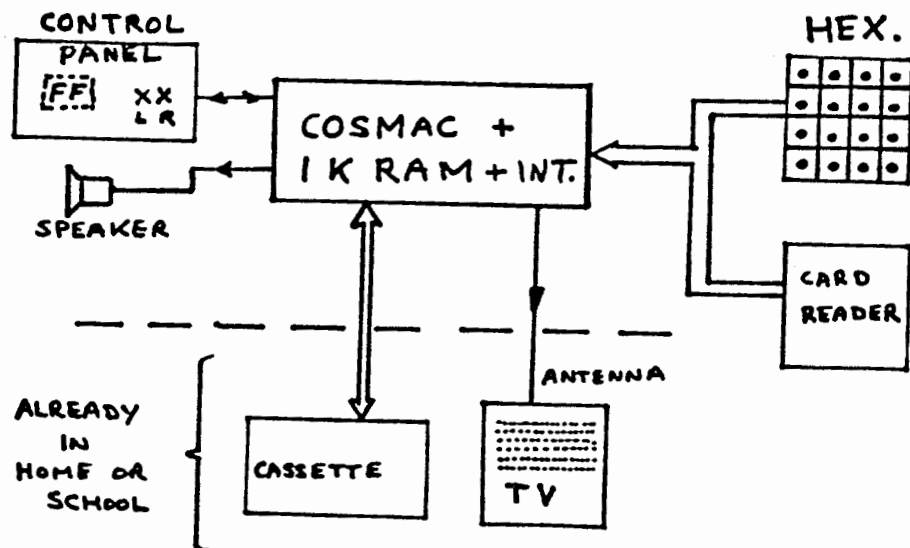
- FLEXIBLE
- RECREATIONAL
- Educational
- DEVICE

I-2

FRED

MOTIVATION

- MINIMUM COST
(CONSUMER)
- EASE OF OPERATION
(ALL AGES)



FRED SYSTEM

2-4

USES

- EDUCATIONAL - MATH/READ.
- GAMES: UNLIMITED TV GAMES (WITH SOUND EFFECT)
- ANIMATION
- CALCULATOR

MINIMUM COST

- CONSUMER OWNED PERIPHERALS
(TV, CASSETTE)
- MINIMAL CONTROL PANEL
(2 HEX DIGITS, 2 SWITCHES)
- LOW COST INTERFACES, & DEV.
- LSI TECHNOLOGY

I-7

HEX KEYBOARD

- 16 SWITCHES (HEX DIGIT)
- HEX OR BYTE MODE
- SIMPLE LOGIC (ONE CHIP)
- USES EXTERNAL FLAGS TO
COMMUNICATE WITH CPU

I-8

HEX MODE

2 KEYSTROKES \Rightarrow BYTE
(E.G., E7)

BYTE MODE

1 KEYSTROKE \Rightarrow BYTE
(PRESS "E", GET "OE")

I-7A

FRED

KEYBOARD USES

- PROGRAM LOADING/
MODIFICATION
- INPUT DEVICE DURING
PROGRAM EXECUTION

I-7B

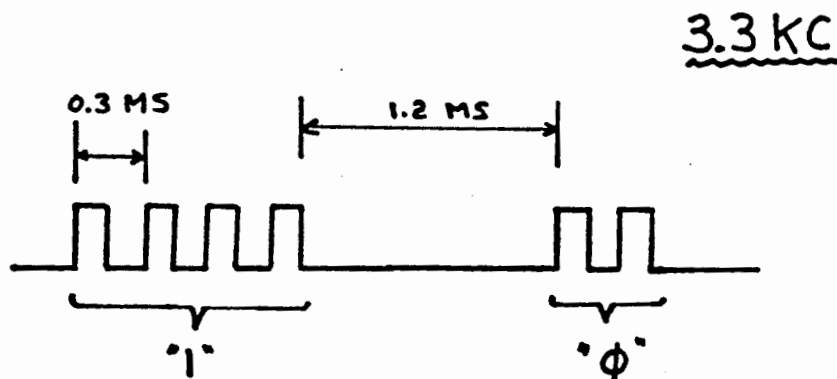
CASSETTE CE

- STANDARD AUDIO CASSETTE PLAYER
- READ/WRITE LOGIC
- DATA STORED AS BURSTS OF 3.3 KC PULSES
- 500 BITS/SEC. (EQUAL DISTRIBUTION OF 1'S AND ϕ 'S)
- PROGRAM LOAD VIA DMA

E-10

FRED

TIMING



500 BITS/SEC
(50% " ϕ ", 50% "1")

I-11

BIT ENCODING

	<u>WRITING</u>	<u>READING</u>
"0"	2 PULSES	1-2 PULSES
"1"	4 PULSES	3-4 PULSES

I-12A

FRED

CASSETTE USES

- PROGRAM LOADING
- SOUND EFFECTS DURING PROGRAM EXECUTION
- I/O DEVICE FOR CODE/DATA DURING PROGRAM EXEC.

CONTROL PANEL

- START/STOP PROGRAM EXEC.
- PROGRAM LOADING
- BYTE OUTPUT (2 HEX DIGITS)
- INDICATORS
 - ERROR IN DATA /CODE
 - PROGRAM LOAD ONGOING
 - ETC

I-12 c

FRED

TV INTERFACE

- STANDARD TV SET (B&W OR COL.)
- DOT MATRIX FORMAT
(32X32, 16X64, 32X64)
- DISPLAY REFRESHED VIA DMA
(CYCLE STEALING)
- USED AS OUTPUT DEVICE
DURING PROGRAM EXECUTION

TV INTERFACE

- HARDWARE STRUCTURE/
PHILOSOPHY
- SOFTWARE

x-φ

TV

INTERFACE CHARAC.

- DOT MATRIX
- DOTS EITHER "ON" OR "OFF"
(NO GREY SCALE)
- PARALLEL OPERATION WITH CPU
(DMA CHANNEL)
- LOW RESOLUTION
(1024/2048 DOTS)

RESULTING IN

- EFFICIENT USE OF MM (128/256 BYTES)
- SIMPLE LOGIC (LSI)
- MINIMAL REDUCTION IN CPU THRU PUT (2-15 %, DEPENDING ON CPU CYCLE TIME)

2-2

TV

DISPLAY REFRESH

- EVERY 1/60 SEC, CPU INTERRUPT
 - CPU RESETS $R\phi$ AND RESUMES NORMAL INST. EXECUTION
- A FEW MS AFTER INTR., CYCLE STEALING BEGINS TO REFRESH DISPLAY

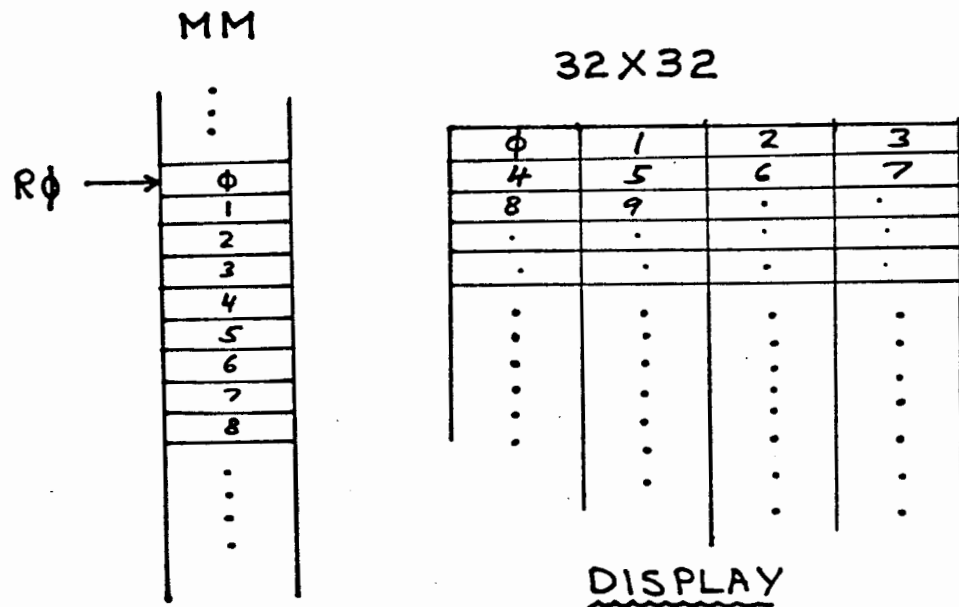
DOT MATRIX

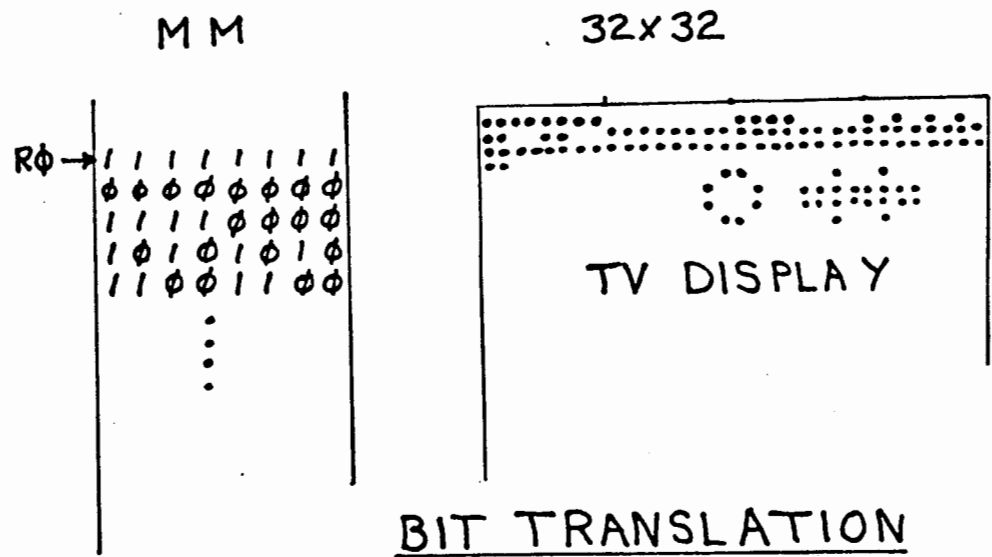
- ANY CONTIGUOUS 128/256 BYTE MM AREA CAN BE DISPLAYED
- BYTES DISPLAYED SEQUENTIALLY L TO R, TOP TO BOTTOM
- "1" BIT \Rightarrow WHITE DOT
- " ϕ " BIT \Rightarrow NO DOT

(DISPLAY IS WHITE ON BLACK)

I - 10

TV





I-12

TV

DISPLAY RESOLUTION

- MEMORY REQUIREMENTS
- CPU THROUGHPUT

% LOSS OF CPU THRU PUT

- LET T = MACHINE CYCLE
- ASSUME 32×64 FORMAT
(256 BYTES)

\Rightarrow NEED 256×60 CYCLES / SEC

TOTAL # OF MACHINE CYCLES
EXECUTED / SEC = $1/T$

II-14

TV

$$\begin{aligned} \% \text{ LOSS} &= \frac{15,360}{1/T} \times 100 \\ &= 1.536 \times T \times 10^6 \end{aligned}$$

IF $T = 10 \mu\text{SEC}$, $\% \text{ LOSS} = 15.36\%$

IF $T = 3 \mu\text{SEC}$, $\% \text{ LOSS} = 4.61\%$

$\% \text{ LOSS}$ IS HALVED WHEN USING
 32×32 OR 16×64 FORMATS

MAX T TO REFRESH DISPLAY

$$100 = 1.536 \times T \times 10^6$$

$$\Rightarrow T_m \doteq 65 \mu\text{SEC}$$

FOR HIGHER RESOLUTION DISPLAY,

T_m IS SMALLER

I-16

MEMORY REQ.

IF RESOLUTION (DOTS/INCH)
CHANGED BY FACTOR "N",
AMOUNT OF MEMORY NEEDED
CHANGES BY N^2 .

E.G., IMPROVING RESOLUTION
BY FACTOR OF 4 WOULD
INCREASE MEM. TO 4096
BYTES (FROM 256) FOR 32X64.

TV

THUS, FOR 4 TIMES BETTER RESOL.

NEED 60×4096 CYCLES / SEC.

$$\Rightarrow \% \text{ LOSS} = 24.58 \times T \times 10^6 \%$$

$$8. T_M = 4.07 \mu\text{SEC}$$

AND, WITH $T = 3 \mu\text{SEC}$,

$$\% \text{ LOSS} = 73.7 \%$$

II-1P

TV

TV I/O INSTRUCTIONS

- SELECT TV ($6\Phi : M(R(X)) = \Phi 2$)
- TURN TV ON ($62 : M(R(X)) = \Phi 1 / \Phi 2 / \Phi 3$)

$\Phi 1 : 32 \times 32$

$\Phi 2 : 16 \times 64$

$\Phi 3 : 32 \times 64$

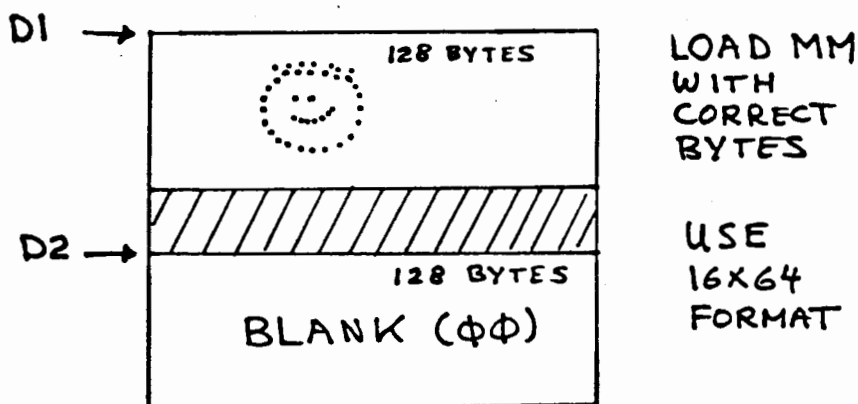
- TURN TV OFF ($62 : M(R(X)) = \Phi \Phi$)

NOTE:

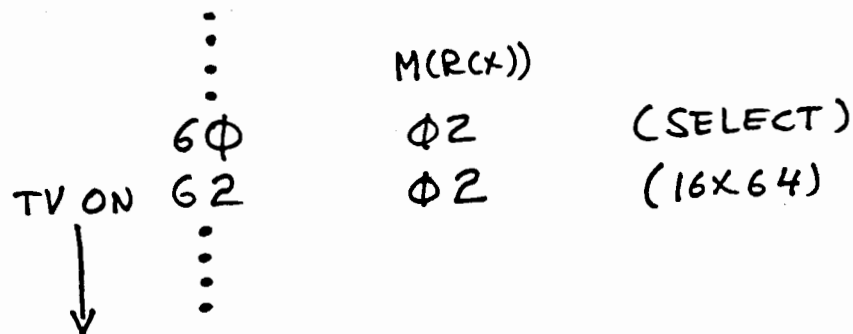
- NO FLAGS USED (LOWEST PRIORITY DEVICE)
- CPU INTERRUPTED 60 TIMES/SEC
- CE INTR. FF RESET BY S3.TPA
(POTENTIAL CONTENTION PROBLEMS)

I-21

TV

EXAMPLE: BLINK TV PATTERN

WANT: BLINK ON/OFF EACH SEC.
⇒ NEED TIMER ROUTINE

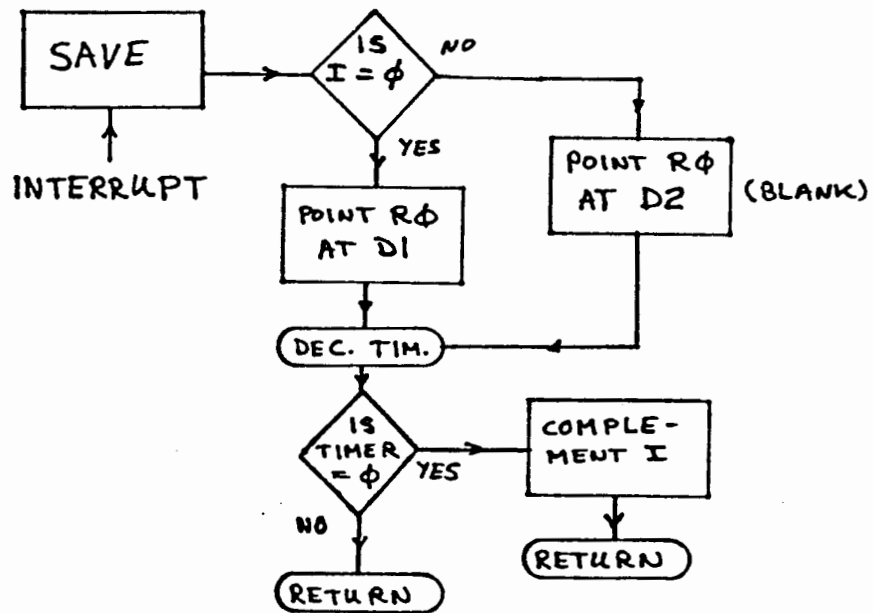
MAIN PROGRAM

II-23

TV

INTERRUPT ROUTINE

- POINT RΦ AT D1 OR D2
- TIME OUT DISPLAY
 - SINCE INTERRUPTS OCCUR EVERY 16.7 MS, CAN COUNT THEM TO GET DESIRED TIME INTERVALS
- LET I = "Φ" ⇒ DISPLAY FIGURE (D1)
 = "1" ⇒ BLANK SCREEN (D2)



I-25

TV

TIMER ROUTINE

- COUNT TV INTERRUPTS (16.7 MS)
- EG., WANT 1 SEC DELAY,
 - SET RC TO 003C (60₁₀)
 - (NOTE 60 x 16.7 MS ≈ 1 SEC).
- EACH TIME INTERRUPT ROUTINE IS ENTERED, DEC RC
- WHEN RC = 0000, TIMER (1 SEC) HAS ELAPSED

DOT MATRIX

PROS • FLEXIBLE (GRAPHICS /ALPHA.)

• ANIMATION

T_{EM} { • CPU CONTROLS EVERY DOT
• CAN DISPLAY ANY CONTIGU-
OUS MM AREA (DOT PER BIT)

CONS

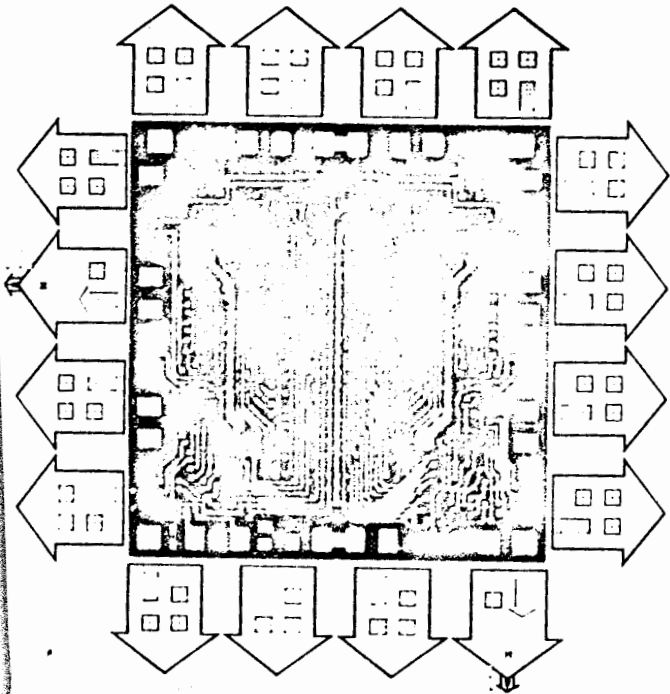
• LESS EFFICIENT USE OF MM
FOR ALPHANUMERICS

• GIVE UP MORE CPU CYCLES
THAN EQUIV. ALPHA. DSPLY

II-27

SUMMARY

- TV DISPLAY VERY FLEXIBLE AND
USEFUL
- CAN DO ANIMATION, GRAPHICS, ETC.
- CAN REVERSE DISPLAY (DOTS → NO
DOTS & VICE VERSA)
- DOT MATRIX FORMAT VERY USEFUL
IN T_{EM} (DISPLAY MIRROR IMAGE OF
MM).



The mention of low-cost computers usually evokes one of two images. Some of us see a super calculator; others picture a large data-base processor. The system described here is a more modest machine that could sell for under \$500 in the relatively near future. Not much has been written on practical computers of this size. Nevertheless, prototypes of this low-cost, mass-market, pre-standing computer system have been constructed, programmed, and operated in a home environment over the past several years.

A PRACTICAL, LOW-COST, HOME/SCHOOL MICROPROCESSOR SYSTEM

Copyrighted IEEE 1974. Reprinted, with permission, from COMPUTER magazine, August 1974

Joe Weisbecker
RCA Laboratories

Meet Fred

Despite the recreational and educational potential of stored-program computers, the single factor of cost has kept them out of the economic reach of most people. But with the advent of LSI microprocessor and memory chips, this may all change - particularly if we take a more modest applications approach and place reasonable limitations on hardware capability.

Of course, if we need conventional input, output, and bulk storage devices, or if our applications are playing chess, printing pages of data, or accessing large on-line digital/video data bases, the cost will still be prohibitive. However, prototypes of such modest home/school systems have

been constructed and in use for several years.

This system, called FRED (Flexible Recreational and Educational Device), has been developed using the RCA COSMAC microprocessor.

A computer of this type could have major social value. As an interactive, open-ended, adaptive, recreational and educational device, it could stimulate the development of analytical and other intellectual abilities. One can easily imagine the formation of a whole new group of computer hobbyists, complete with user groups and publications for the exchange of programs and ideas. In short, the inexpensive home/school computer could open the door to an entirely new environment that stimulates experimentation, analysis, and creativity.

Application and System Overview

In schools, FRED could provide a powerful educational tool. It could be used to drill and test students from first grade on. It could be used in educational games, simulation exercises, and reading readiness, as well as in teaching programming, as an adjunct to math courses, and as an accessible student tool in almost any subject. FRED could be used to set up stimulating demonstrations and experiments in a wide variety of areas, to help correct learning disabilities, and to stimulate the development of creative abilities. Cost per student hour would be measured in pennies.

In the home, FRED has already functioned as a sophisticated entertainment center for the whole family. It provides a variety of games, simulates a calculator, and even provides a controllable TV puppet for the youngest member of the family. FRED permits a number of creative activities, including TV picture drawing, low-fidelity music synthesis, and programming at a variety of skill levels. FRED also provides a shooting gallery, a variety of puzzles, and animated TV greeting cards for holidays.

Since FRED is a stored-program computer, it requires a program to be loaded into memory before use. Program loading is performed with an inexpensive audio cassette player which also gives the computer its voice, music, and sound effect capabilities. Pre-recorded program cassettes can be loaded in less than 30 seconds.

After a program cassette is selected and loaded, FRED is operated with a small 16-position keyboard. For a game, the player presses appropriate keys to indicate the moves. Overlay cards are provided so that keyboard labeling can be changed for different programs.

FRED is attached to the antenna terminals of any TV set. This provides an inexpensive, flexible, dynamic output display which is ideally suited for home/school use. Numbers, words, or simple pictures can be displayed on the TV screen in the form of dot patterns.

The basic FRED system comprises the RCA COSMAC microprocessor, 1024 bytes of RAM, a simple hex keyboard, an inexpensive audio cassette player, and the user's own TV set. One would be hard-pressed to imagine a less expensive free-standing computer system. This system is supported by a library of cassette programs in the same way that a photograph is supported by a record library. A continuing supply of new programs could be provided by the manufacturer of the system together with a selection of optional hardware attachments.

Adding a \$25 punched card reader and \$10 manual punch to the basic system increases its usefulness and provides more sophisticated users with the ability to prepare and save short parameter lists or programs. Adding a module for recording the contents of memory on cassettes turns the basic FRED system into a user-programmable computer for serious hobbyists. Other possible attachments include light guns, extra memory (RAM), pre-stored programs or tables (ROM), and output relays for control uses.

Design Considerations

Two different approaches can be taken toward developing an under-\$500 home/school computer. One approach involves specifying a desired set of system characteristics

and then attempting to achieve cost objectives. The danger in this approach is the tendency to overspecify the hardware so that price targets can't be met. The approach we took in developing FRED was to define a minimum cost, non-trivial hardware system that could easily meet our low price goal, and then testing its usefulness. This approach ensures that applications development effort won't be wasted. Applications developed for our minimum system are easily transferred to a larger system.

Any free-standing computer must include a (CPU, main memory, input and output device(s), and bulk storage device(s).

The FRED system required the development of a system philosophy that was consistent with utilizing minimum cost hardware. This system philosophy included asking what we could do with cheap devices instead of asking what types of devices we might ideally want. Because of our low price goal, the implications of each of our choices were magnified in importance.

For our purposes, competitive cost-performance ratios were largely ignored. Reliability was also sacrificed to some extent to achieve low cost. Of course, permanent system failures cannot be tolerated, but occasional transient errors, provided they are not catastrophic, may be permitted during program loading. The need for memory and processor parity checking was also felt to be an unaffordable luxury.

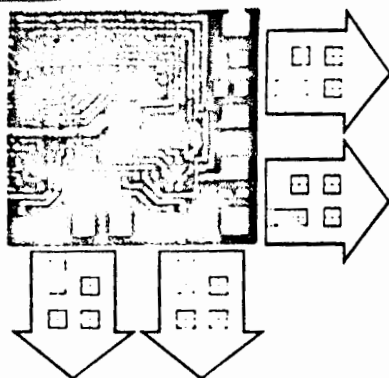
Ease of use is a primary requirement. So that the system would not appear overly complex to the novice user, a turn-key system philosophy was adopted. The user simply loads a program from a library to obtain a desired function. He is not expected to program the machine. This approach eliminates the need for program debugging and maximizes ease of use. The need for an expensive control and diagnostic panel is also eliminated. In fact, only two switches are required for basic use: LOAD and RUN. Utility routines and optional hardware are provided for the sophisticated user who wants to develop his own machine language programs. The minimum system also provides for small user-generated programs via a variety of simulation languages.

A block diagram of the basic system is shown in Figure 1. System considerations will be included in the discussion of individual elements.

Central Processing Unit and Memory

As mentioned earlier, the advent of single chip LSI microprocessors has made the under-\$500 system possible. Suitable microprocessor chips should be available for less than \$25 within the next several years. The choice of a microprocessor has a large influence on total system chip count and system cost. This influence is an important consideration since the microprocessor itself is only a small part of total system cost. The COSMAC architecture immediately eliminates the need for a read only memory (ROM) in the minimum system. Only one supply voltage is required. COSMOS circuits further reduce system power supply costs. A self-contained direct memory access (DMA) channel facilitates initial program loading and display refresh. A single-phase clock is another minimum cost feature. High output drive capability eliminates external buffer circuits.

The 8-bit COSMAC architecture is compatible with the intended uses of the system. The short instruction format permits compact programs with small memory require-



ments. Since the average user will never see the processor micro-instruction set, ease of programming is secondary to efficient memory utilization.

A complete description of the COSMAC microprocessor has appeared previously^{1,2} and will not be repeated here. This architecture has demonstrated its advantages in prototypes of the low cost home school system.

Due to the nature of our application, RAM is required for both program and data storage. It's well known that programs tend to expand to fill available memory space. Providing a 4096 byte memory only insures that no program will be written requiring a smaller memory. Even projecting a cost of 2¢ per byte would yield a cost of \$82 for a 4096 byte memory. This size memory would add \$200 or more to the selling price of the system. Instead of asking how much RAM we could use, we provide 1024 bytes

in the minimum system. This is consistent with keeping memory cost equal to projected microprocessor chip cost. Should LSI memory costs drop below 2¢ per byte we can increase minimum system capacity to 2048 bytes or lower the price of the 1024-byte system. Based on current trends, we can safely predict one microsecond LSI RAM costs of 2 to 3¢ per byte. Dynamic RAM chips are at this cost level now, while static, single voltage RAM chips are currently available at 7 to 8¢ per byte.

The challenge of a 1024-byte memory seems to stimulate cleverness in programming and makes a future 2048-byte memory seem large by comparison. If we had initially provided a 4096-byte memory, subsequent size reduction to meet cost targets would have been extremely difficult. Assuming 4x1024 bit RAM chips are available within the next several years, a minimum system would require only two chips for memory.

Limiting the minimum system memory to 1024 bytes also provides several system cost advantages. Power supply cost is reduced, memory address drivers are eliminated, and printed circuit board space is saved. A less obvious system implication is the effect of memory size on program loading costs.

In general, the user should be able to load a program in half a minute or so. This coincides with observed user-patience factors.

An occasional error requiring reload can be tolerated for short load times, so that lower reliability loading devices may be used. To load a 1024-byte memory in 30 seconds only requires a serial transfer rate of 300 bits/second. This assumes a parity bit for each byte. For a 4096-byte memory, the required rate jumps to 1200 bits/second. The required transfer rate influences the choice of a program loading technique. Lower rates can generally be translated into lower costs and better reliability.

It's in the area of input, output, and bulk storage that we encounter the major cost problems. The choice of I/O and bulk storage techniques also has a major effect on the

range of possible system applications. Obviously, a single switch input and single light output would achieve minimum cost but would also result in a trivial system relative to use.

Output Display

Fortunately, an ideal, low-cost output device for home/school applications already exists: a standard TV set provides a flexible, dynamic output display device which most users already own.

The choice of a TV display format involves a number of system considerations. These include types of applications, display refresh memory requirements, and complexity of control circuits. A low resolution, black and white dot matrix was chosen for maximum flexibility at minimum cost. An array of white dots is displayed on a black background. The black background avoids potential picture noise problems. Arrays of 32x32, 16x64, and 32x64 dots are provided. Figure 2 illustrates the flexibility of this format for displaying small game boards, simple pictures, words, numbers, or symbols. Each dot represents the state of a main memory bit. If the bit is "1" the dot is on; if the bit is "0" the dot is off.

Changing the memory bit pattern immediately changes the TV picture accordingly. Bit patterns are readily moved in and out of the displayed memory area by normal programming procedures. Simple animation can be achieved by modifying memory bit patterns at appropriate time intervals. Any contiguous 128 or 256-byte section of memory can be selected for display by setting a microprocessor address pointer. This display pointer can be modified at any point in a program, thus allowing the user to step through various memory display areas at any desired rate. It is easy to flash selected portions of a picture by alternating between two display areas in memory.

For 32x32 and 16x64 displays only 1024 bits (128 bytes) of memory are required for display refresh. This is only 12.5 percent of the minimum 1024 byte memory. The 32x64 display option utilizes 25 percent of the minimum system memory but provides a larger area picture when required. It is also useful in expanded memory systems. It should be emphasized that no ROM is required for TV display in the minimum system and that frame refresh storage is provided via main memory.

The TV control unit (CU3 in Figure 1) contains the circuits for generating TV sync signals and for requesting memory bytes via the COSMAC DMA channel as required for display refresh. The individual bits of each byte are used to generate a video signal. The composite sync and video signal modulates the output of a simple RF oscillator. This modulated RF can be applied to the antenna terminals of any standard TV set.

Figure 3 illustrates the detailed timing for displaying dots on the TV screen. A magnified view of four dots is shown. Each dot is two horizontal TV lines high with a two-line space between dots. An 8-byte row buffer is provided in the TV control unit. Each TV line time is 65 microseconds. During the two blank line times, between rows of dots, up to 8 bytes (64 bits) are retrieved from main memory and stored in the row buffer. During the next two TV line times the bits in the row buffer modulate the TV beam to display the proper dot row pattern. By spreading the dots as shown, the low resolution display fills up the TV screen without requiring a high refresh rate.

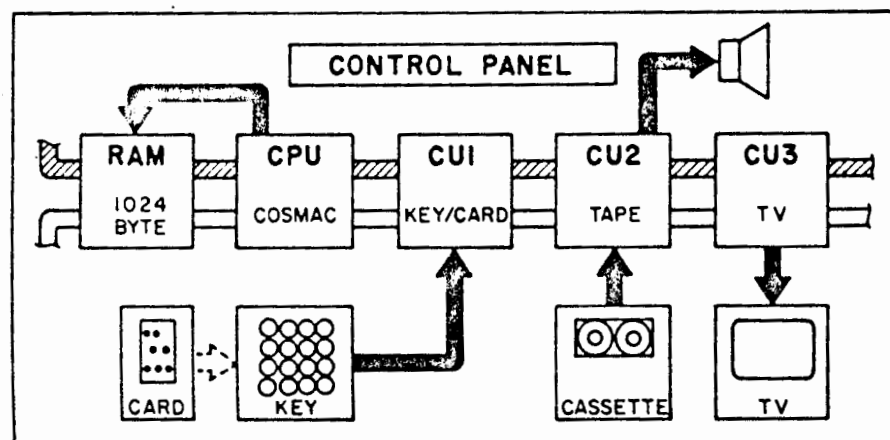


Figure 1. Basic System

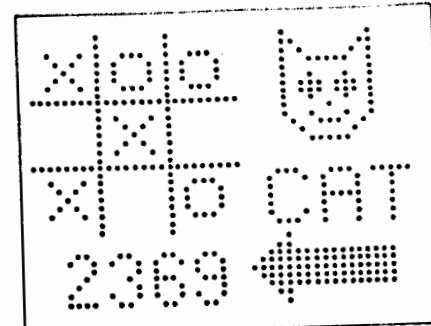


Figure 2. Display Flexibility

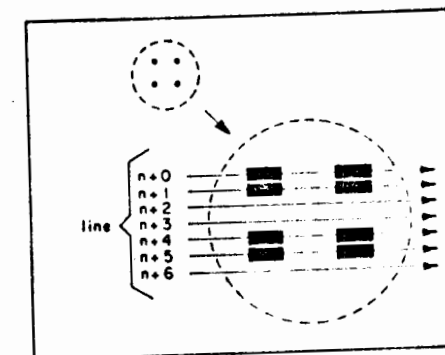
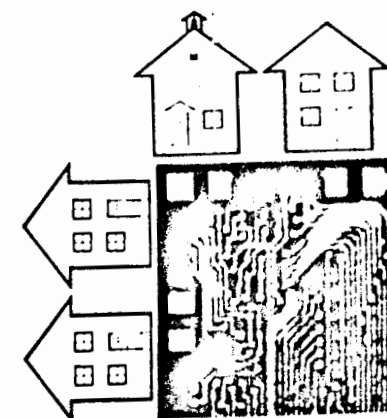


Figure 3. TV Dot Detail



Where can I find out about micros and minis in process control?

Come to COMPCON '74 Fall and find out.

The TV control unit also generates a program interrupt signal at the beginning of each TV frame. This interrupt permits the program to initialize the microprocessor display address pointer at the appropriate time. Since TV program interrupt occurs 60 times/second, a free real-time clock exists when needed. This clock capability is useful for timing purposes in a number of applications.

Bulk Storage

Program library storage and loading presents another major problem area in a low cost system. The high cost of existing computer devices such as floppy discs and digital tape units immediately rule out their use. Paper tape is awkward and still fairly expensive. Conventional punched card readers are expensive and inconvenient.

This problem was solved by using another existing, inexpensive consumer device - the audio cassette recorder. Suitable portable units sell for under forty dollars. A built-in unit could be provided for less than \$20. Several methods for storing bit serial digital data on audio cassettes have been described^{3,4} and others are possible. We developed a proprietary, pulse counting technique that yields a 50 byte per

second transfer rate, tolerates nussing or extra pulses, and permits tape speed variations of 30 percent. This system works well even for cheap portable audio units. Only single track capability is required in the system.

Since errors can be expected on occasion, a parity bit is added to each byte on tape. The cassette control unit checks the parity of input data read from tape and turns on an error light for incorrect parity. Reloading a program when an error occurs is a simple and quick procedure.

Figure 4 shows the single track, cassette tape format which was used. Digital or audio blocks are always framed by 4 kHz stop tones (1). The stop tone detection circuit is designed to respond only to long (.5 sec) continuous tones so that voice or music frames will not cause false triggers.

Figure 5 shows how a standard cassette player is used in the system. Most cassette recorders provide an external speaker or earphone output jack. This output is connected to the control unit as shown. Stop tones and digital data are detected via this cassette output line. A relay is also provided which permits the cassette output to be connected to a speaker under program control. This permits selected tape frames to be passed inaudibly.

The majority of inexpensive cassette recorders have a remote start-stop control jack. This is designed for use with a microphone or foot switch. For use in our system the cassette remote jack is connected to a program controlled relay. This gives the computer the ability to start and stop tape, providing the user has previously placed the cassette recorder in its PLAY mode.

The primary system operating controls comprise two toggle switches - LOAD and RUN. The LOAD switch activates the cassette control unit (CU2 in Fig. 1). The desired program cassette is selected by the user, rewind, and the recorder set to PLAY. When the first stop tone is encountered the data reading circuits are automatically turned on. Waiting for this stop tone eliminates possible noise problems at the beginning of tape. The digital data representing the program is loaded sequentially into memory at 50 bytes/second. The second stop tone automatically stops the tape via the tape control relay. Turning off the LOAD switch resets the computer. The RUN switch initiates execution of the program which was just loaded.

During program execution the tape can be automatically restarted so that the user will hear audio frame #1 at a desired time. The stop tone following audio frame #1 will automatically stop the tape. The program can monitor the state of the control relay to determine when the end of

data/audio frames occur. This permits synchronizing audio material on cassettes with a program.

The provision for program controlled audio segments has an important system implication. The ability to provide instructions, questions, or other data in the form of voice frames on tape minimizes the need for a high resolution, alphanumeric TV display with its attendant requirement for large refresh and back-up digital storage capacity.

The speaker provided for use with the cassette recorder provides a useful output device. A flip-flop which can be set and reset by program drives the speaker when it's disconnected from the cassette output. Programs can, therefore, create many audible sequences of tones.

Input Devices

The primary input device for our system is a 16 position keyboard. A number of 55 to \$10 keyboards of this type have been developed for use in pocket calculators. A flat, printed circuit type was chosen to facilitate an overlay feature. A slight modification of the keyboard permits insertion of a printed card above the switch array. Various cards are provided to relabel the switch array for different programs. For educational programs keys can be labeled with colors, pictures, words, or possible answers to questions. For other programs, the keys might be labeled with direction arrows for manipulation of the TV display. The variable label keyboard is fundamental to meeting the ease-of-use criterion for this type of system.

Unfortunately, the flat keyboard which is ideal for variable labeling has no tactile feel. This objection was overcome by taking a systems approach. Since a speaker already exists, switch depressions need only be coupled into this speaker to provide an audible "click." This has proven to be an adequate substitute for tactile feel. The scanning approach used to decode the switch panel minimizes the cost of this approach. Specific programs can also generate various tones for switch depressions which again substitute for tactile feel.

The 16-position keyboard normally causes an 8-bit byte to be stored in memory for each key depression. The most significant four bits (digit) are normally 0000. A shift switch pressed in conjunction with a hex key causes the most significant four bits to be 0001. The least significant four bits of a stored byte represent the code for one of the 16 possible hex digits shown in Figure 6. The hex keyboard in conjunction with a shift switch permits entry of 32 different codes.

An alternate mode of keyboard entry is also provided. In this mode two key depressions per byte are required. The first key specifies the most significant hex digit of the byte to be entered. The second key provides the least significant hex digit of the byte. This mode provides the sophisticated user with a convenient way to manually load his own machine language programs. It's also a useful mode for certain turnkey programs.

The hex keyboard control unit (CU1 in Figure 1) also supports the addition of an inexpensive card reader to the minimum system. This unique device uses 3-inch x 5-inch punched cards. Data is punched in the form of rows of holes. Figure 7 shows four such rows (A,B,C,D) punched on one side of a card (both sides can be punched). Each row represents the 4-bit code for one hex digit. A fifth hole is added and encoded with odd parity. At least one hole will

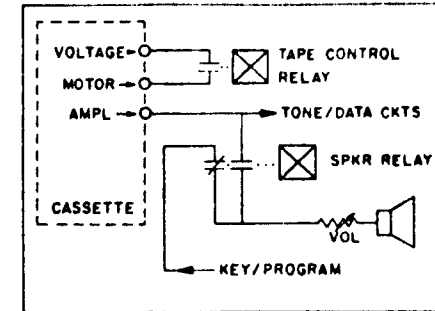


Figure 5. Cassette Attachment

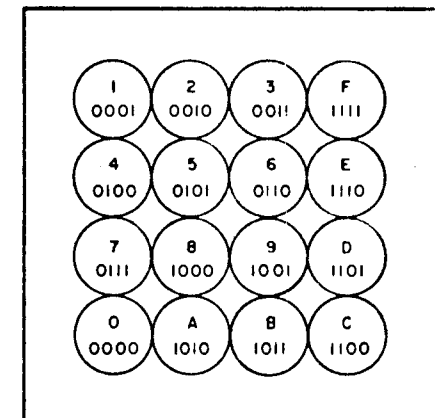


Figure 6. Keyboard Control

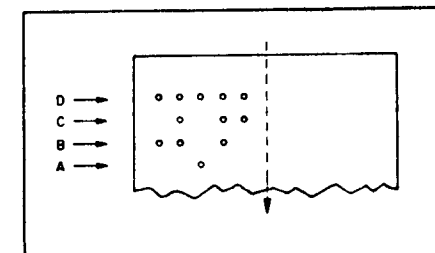


Figure 7. Punched Card

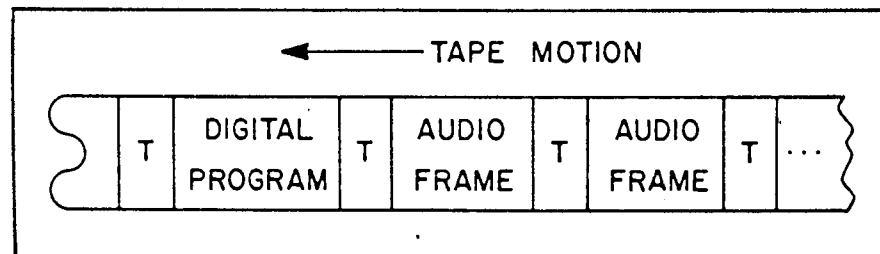
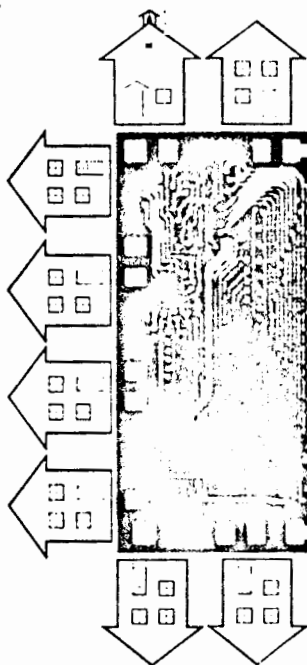


Figure 4. Cassette Tape Format



be punched for each of the possible hex digit codes. Cards are read by dropping them into a 3-inch slot. They fall past a light source and six photodiodes. One photodiode senses the presence of the card and conditions the control unit circuits accordingly. The other five photodiodes read the self-clocking hex digit codes into the system. Hex digits are paired to form bytes before storage in memory.

By limiting the information content of a card to 16 hex digits per side, the mechanical tolerances of the reader can be considerably relaxed. The reader has no moving parts, and photodiodes can drive the COS MOS control unit circuits directly. These factors combine to provide a very low cost input device (\$25 or less). A simple, manual card punch can also be provided permitting users to punch their own cards.

The low cost card reader can be used to enter short lists of parameters or short user prepared programs. In a classroom, the teacher might use parameter cards to set up test/drill programs. Picture cards used by the student could contain the spelling of the word pictured for checking by the computer. The cards also facilitate certain simulation languages and permit users to save simulation language programs that they develop.

Another low cost, optional input device is a simple light gun. This contains a lens system and a photodiode. The computer detects when the gun is pointed at any lighted area of the TV screen. The light gun facilitates various computerized target shooting games. By alternately flashing portions of the TV display a program can determine the

area at which the light gun is pointed. This permits the user to indicate various types of choices by pointing the gun at appropriate portions of the display.

Applications Philosophy

The open ended aspect of a stored program computer differentiates it from other types of recreational and educational devices. Any number of special purpose devices such as TV games, shuffleboard tables, electric football games, and educational toys are ideally suited to their intended function. None of these, however, will change their characteristics as user moods or interests change. Many of these special purpose devices are seldom used after their initial novelty expires. The stored program computer is a general purpose device. New programs can adapt it to changing moods and interests without the expense of new hardware. It can satisfy the needs of young and old and can grow with individual abilities.

The real value of the home school system lies in its ability to stimulate and develop human capabilities that are often ignored or discouraged by conventional recreational and educational devices. The computer system provides an environment that stimulates experimentation, analysis, and creativity. For example, contemporary TV encourages passive viewing. However, the computer attached to a TV set enables the user to interact and play a game with the TV set. As the games played increase in sophistication, the user is encouraged to improve his analytical abilities. The user can subsequently be encouraged to experiment via specific programs or eventually to write his own programs.

For a child, the computer may initially provide arithmetic or spelling drills. Even this kind of memory development can be made more interesting via interaction with the computer. However, the child will eventually begin to wonder about the computer. Programs are made available which stimulate this curiosity and let him experiment with changing game rules. He can even begin to formulate and develop his own simple programs in a variety of simulation languages. While the initial use of the computer involves memory skills, it eventually encourages experimentation and the development of analytical and other capabilities.

The creation of programs that stimulate the user to develop mentally is a challenging task with a high payoff in terms of satisfaction. We have only begun to explore this area of use for very small, inexpensive, practical computers of the type described here. Even so, the number and richness of uses for this type of system are surprising. Those of us who are experienced with 64,000 byte main memories and large disc files may be inclined to dismiss a 1024 byte memory system as unuseable. But, in fact, such a system can be adapted to a wide range of uses. Over 80 specific applications of the inexpensive home school system will be listed in the following sections. Many represent classes of programs which could be developed.

Four general areas of use are identified in Figure 8. These areas will be discussed individually although there is a high degree of overlap between them. Most of the listed uses only require the basic system. Reference 18 also describes a number of uses (mostly games) that have been programmed on larger computers with hard copy output. Many of these are readily adapted to the low-cost computer.

Utility Applications

This category of applications involves use of the computer to achieve specialized functions such as those listed in Table 1.

- *Four Function Decimal Calculator
- Hex Binary Calculator
- Game Score Keeper
- *Number Base Converter
- Weight/Measure Converter (Metric)
- Secret Code Computer
- Logic Machine
- Classification Computer
- Gambling Strategy Computer
- Other Specialized Calculators (temperature conversion, interest, etc.)
- Electronic Dice
- Random Number Generator
- Simulation Game Computer
- Bar Graph
- Interactive Audio-Visual Toy
- *TV Greeting Card
- *Electronic "Etch a Sketch"
- TV Puppet
- *Audio Visual Demonstrator
- Mind Reading Computer
- Party Compatibility Computer
- Programmed Timer/Controller
- Stop Watch/Game Timer
- Simple Electronic Organ
- Metronome
- Advertising Display

*Already developed for the COSMAC miniprocessor.

Table 1. Utility Applications

A simulated four-function decimal calculator has been implemented on the basic 1024 byte memory system. This includes display refresh, digit pattern tables, and decimal arithmetic algorithms with 20-digit operand and result capability. A 2048 byte memory would permit development of a programmable calculator with multi-line display. Optional ROM chips could provide a permanently resident calculator capability if desired.

A variety of specialized calculators can be implemented on the basic system. Programs to provide scorekeeping for card, war, or commercial games could be provided. Children

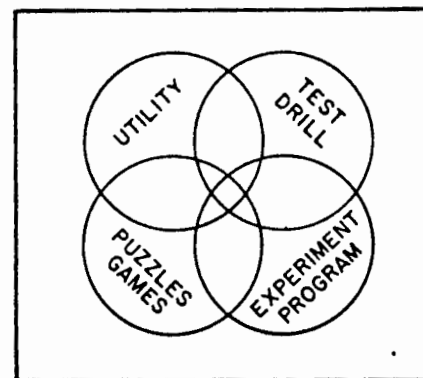


Figure 8. Areas of Use

August 1974

could have their own secret code computer. For several years a plastic toy rock identification computer has been on the market. Certain tests are performed (color, hardness, etc.) on a mineral sample. The plastic computer and a set of cards is then used to identify the sample. The basic home/school system could readily be programmed as a classification computer of this type.

Logic machines have held a certain fascination for years.⁴ The computer readily simulates a variety of machines of this type. It can also be programmed to simulate gambling algorithms. A pair of dice is easily simulated for use in a number of games. Random number generating machines find use in various school courses and experiments. Serious war game fans can use computer generated battle results and score keeping to advantage. The leading magazine in the field, "Strategy & Tactics," has over 20,000 subscribers indicating a wide interest in this type of activity.

For very young children the computer simulates a variety of interactive, audio-visual toys that make sounds and change TV pictures in response to key depressions. Customized, animated, TV Greeting Cards/Decorations for Birthdays, Christmas, or Halloween can be provided. Simple, key operated TV puppets are possible. Stepping a spot around the screen permits drawing TV pictures.

The ability to synchronize audio tape frames with programs permits programmed audio visual tutorials for home and school or eye-catching advertising displays. The basic system real-time clock facilitates key operated game timer or stop watch capability. The program controlled speaker turns the computer into a simple electronic organ or metronome. TV display can be included with the sound generation.

Test and Drill Applications

These are probably the first types of uses that come to mind when education is mentioned. Drills involve the development of memory or conditioned reflexes. Testing can involve the development of other skills, as well. The infinite patience of a computer makes it ideal for drills. Interactive capability adds interest and motivation. Some specific examples are included in Table 2.

- *TV Arithmetic Drill
- *Word Spelling Drill
- *Word Recognition Test
- *Pattern Recognition (Superimposed, Complex)
- Electronic Flash Cards
- Classroom Group Games
- Preschool Shape/Color Recognition
- Up Down, Left Right Discrimination
- Sound Picture Matching
- Reading Readiness Skill Drills
- Logical Aptitude Test*
- *Number Base Conversion Drill
- Flap Board Simulator
- Morse Code Drill
- Reflex Testing
- *Logical Deduction Test (21 Questions)
- Logix*
- Memory Training (Sobriety Test)
- Individual Testing & Scoring Aid
- Change Making Drill
- X-Y Curve Plotting Drill
- Time Sense Development

*Already developed for the COSMAC miniprocessor.

Table 2. Test and Drill Applications

COMPUTER

The number of different programs possible in this area is virtually unlimited. Programs of this type are ideal for individual use to overcome specific weaknesses. Unlike the teacher or parent, the computer does not make value judgments about the child during a drill. The drills can also be made to appear as games with the computer providing added motivation.

A simple arithmetic drill might appear as shown in Figure 9. Addition problems are randomly generated on the TV screen. The child must enter correct answers via the keyboard in time to prevent the boat from completely sinking. The rate at which the boat sinks can be preset by the teacher and changed from session to session to maintain challenge as the child's speed and accuracy improves. The computer displays the child's score when the teacher enters a special code (key card).

Spelling drills can be implemented in several ways. A cassette voice could ask for the spelling of a word which the student then spells via the keyboard. The TV display area or audio tone responds if he is right. The computer again keeps score and times the answers. A simple word recognition drill involves displaying a word on TV and asking for the corresponding picture via keyboard or card input. Patterns can be superimposed on the TV screen and the student asked to identify the components of the picture. Simple pre-specified shape, color, or sound recognition programs are possible. Up-down, left-right concepts are readily presented via taped voice and animated TV displays. Most test and drill programs become classroom games. Team members take turns answering computer questions and the computer announces the winner.

The computer can be programmed to momentarily flash a picture, word, pattern, or group of symbols on the TV screen to develop perception skills. Reflexes or time sense can be developed by requiring a specific keyboard response following programmed sounds or TV displays. Scoring adds a motivating game element to these types of drills. Morse code is taught by requiring the translation of tape voice passages into key depressions. The computer checks accuracy and gradually increases speed.

Reading readiness skills include simple shape recognition, word configuration recognition, and ability to maintain fixation on a moving object. The latter could involve having a slide, press direction-changing keys to prevent a moving TV spot from hitting obstacles. While not explicitly designed for this purpose, many of the games played with the system also develop reading readiness skills.

The computer can easily simulate logical aptitude testing devices, existing simple educational aids, or games. The dot array TV display format is ideal for X-Y plotting practice. The computer can be used for individual testing and scoring in any subject area and at any grade level. The test questions are provided in printed page or booklet form.

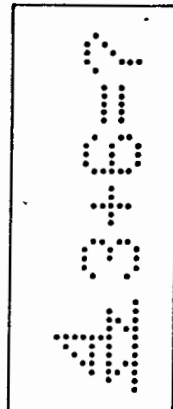
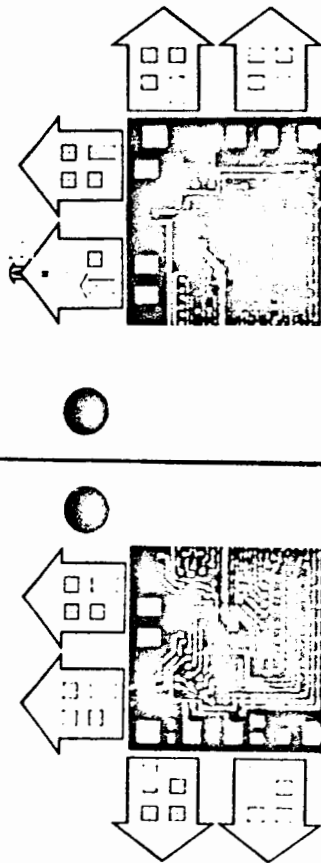


Figure 9. Add Drill Display



The computer specifies which questions are to be answered via taped voice or TV display. Answers can be in the form of multiple choice, numbers, or words which the computer can check against a prestored table of correct answers.

The ability to skip audio frames on tape via the program controlled speaker relay provides added flexibility for test and drill applications. Two sequential voice frames could be provided per question. One frame would "tell" the student that his answer was right. The other frame would "tell" him that his answer was wrong, and why. For each student response the computer "plays" the appropriate frame and "skips" the other.

Games and Puzzles

Games and puzzles are normally associated with recreation. We have already seen that a number of utility programs have recreational aspects. The educational as well as recreational aspects of games and puzzles will be discussed here. Some of the possible uses of the computer in this area appear in Table 3.

*TIC TAC TOE
*Hexapawn
*Shooting Block Puzzles
*Slide Change Games Puzzles ¹⁰
*Pooling ¹¹
*Football ¹¹
*Minkreg
*Target Shoot (Optional Gun)
*One Armed Bandit
*Network Games
*Twenty One
*Cell Matching Games
*Race Tracing (Invisible, Changing)
*Place Games (Against Time)
*Space War
*Bombs Away
*Combinational/Sequential Puzzles ¹²
*Dodge Games (Space Ship & Asteroids)
*Fish Card Game
*Moon Landing
*NIM Games (Static/Dynamic)
*Invisible Counter Board Games
*Simulation Games
*Game Forms of Utility/Test/Drill Programs

*Already developed for the COSMAC minicomputer.

Table 3. Games and Puzzles

The whole area of using small computers in simulation games requires further investigation. Reference 13 contains brief descriptions of over 500 such games currently available for educational purposes. The availability of an inexpensive computer for referee, controller, and randomizing functions should be welcome in this area.

Many of the utility, test, and drill uses previously discussed also provide the basis for games or puzzles. A widely available, mass market computer will undoubtedly stimulate the invention of many new games and puzzles for recreational and educational use.

Experimentation and Programming Uses

This area might be thought of as primarily educational characteristics of a computer that make it ideal for experimentation also provide the basis of a fascinating hobby. Developing programs for your own computer embodies both educational and recreational aspects. Some specific experimentation and programming possibilities are listed in Table 4.

*LIFE ¹⁴
*Penny Matching Computer ¹⁵
*Turing Machine
*Tutorial Computer
*Picture Computer
*Sound Computer
*Machine Code Programming
*Simulations
*Variable Rule Games
*Logic Simulator
*Learning Machines
*Probability & Monte Carlo Experiments
*Heuristic Program Design

*Already developed for the COSMAC minicomputer.

Table 4. Experimentation and Programming Uses

A classic example of experimentation via computer is provided by Conway's game of LIFE, described in Reference 14. The hours of bodiless computer time devoted to this program at computer centers all over the country are a testament to its recreational value. LIFE simulates a succession of generations for a colony of cells (cell birth, survival, and death are controlled by algorithms in the program. Watching the patterns of cell change for each generation on the TV screen is addictive. The program is extremely rich in experimental possibilities. New starting patterns that yield interesting and sometimes surprising life histories are constantly being discovered. The availability of an inexpensive computer would permit even unsophisticated users to experiment via programs of this type.

Heuristic programs for simple games such as Hexapawn and Penny Matching¹⁵ let the user develop experimental learning curves. This approach has been used to add interest to grade school math even without the availability of a computer.¹⁷ Letting the user modify program behavior via keyboard parameters stimulates more creative and sophisticated experimentation, even TIC TAC TOE becomes a fascinating educational device when the user is allowed to modify the rules that the computer uses.

The computer can provide a variety of simple simulations that encourage experimentation. A simple moon landing game or race-car game with acceleration and deceleration controls are examples. A logic simulator would permit experimenting with arrays of logic elements. Commercial hardware logic trainers are quite expensive. A random number generating program facilitates experimental development and understanding of probability curves. Game theory experiments are a natural application. None of these uses requires programming ability on the part of the user.

The area of programming provides the richest and most valuable recreational and educational experiences. Programming capability can be provided at several levels. A simple set of simulated instructions to move a spot around the TV screen could be provided via card or key symbols. Programming this picture-drawing computer could be introduced as early as second or third grade. The ability to program sequences of audible tones for musical via a simple simulation language is also easily provided.

At a slightly higher level of sophistication, various tutorial computers can be simulated. A simple fixed word, decimal computer was simulated on the basic 1024 byte system. This included ten instructions, 100 words of user memory, and a simulated control debug panel. Teenage children were able to write and debug their own programs with as little as one hour of instruction. Simple, simulated, tutorial computers open the door to a variety of interesting educational projects. What better way is there for a student to learn than by teaching his computer.

The construction of a hardware Turing machine model for educational purposes is described in Reference 16. The authors list several disadvantages inherent in the alternative approach of computer simulation:

- A. Computer time too expensive.
- B. Students have to learn how to operate the computer which has nothing to do with the simulation.
- C. Graphic output display is too expensive and printed output is too slow and inconvenient.

It is interesting to note that the home school system readily overcomes all three objections. How many other valuable educational tools might be easily provided via simulation if this type of inexpensive computer was made widely available?

For the sophisticated hobbyist the area of machine code programming will be of major interest. All that is required is to add circuitry for writing cassette tapes. This is an inexpensive option. The use of a small set of programming conventions together with specialized subroutines permits the sophisticated user to develop, debug, and save his own machine code programs.

Conclusions

A practical, inexpensive, free standing computer system based on the RCA COSMAC microprocessor has been described. The fact that over 30 programs are already running on prototypes demonstrates its viability. Over 80 users are listed to dispel any notion that limited capability machines of this class are necessarily trivial. Several basic system enhancements were mentioned and many others are possible.

A system of this type would, for the first time, permit widespread access to computers. Much of the public awe and confusion relative to computers would be dispelled. The creation of a group of home computer hobbyists would stimulate invention and development of new computer devices and applications. Educational benefits are unlimited.

Computers are considered to be useful tools with which to achieve a specific end result such as processing a payroll or calculating a trajectory. This view of computers has often carried over into educational applications with the computer cast in the role of teacher tutor. The low-cost home school system described here is intended as a flexible playing thing which encourages experimentation and stimulates a desire to learn. This approach may be more significant than the improvement of teaching methods for unmotivated students.

Acknowledgements

A. R. Marcantonio, C. I. Wu, and B. J. Cail have made a number of contributions to the development of this low-cost computer system. A. D. Robbi and P. Russo have provided continuing moral support, ideas, and major assistance in developing and evaluating the system. Most of all, R. O. Winder deserves credit for making the development of the system possible. ■



Joe Wesbecker has been developing new computers, communications terminals, I/O devices, and computer related commercial games for almost 20 years. He held engineering, advanced development, and product planning positions within RCA's Computer Systems Division prior to joining the Lab in 1970. Mr. Wesbecker is a graduate of Drexel University, has written several articles, received two RCA Laboratories Achievement Awards, and holds 21 patents, with others pending. He is a Senior Member of the IEEE and a member of the Computer Society. The computer described in this article has been accepted as a member of the family by Jean (his wife), their two children (Dorise and Jean) and even their dog who punches random number cards with her teeth.

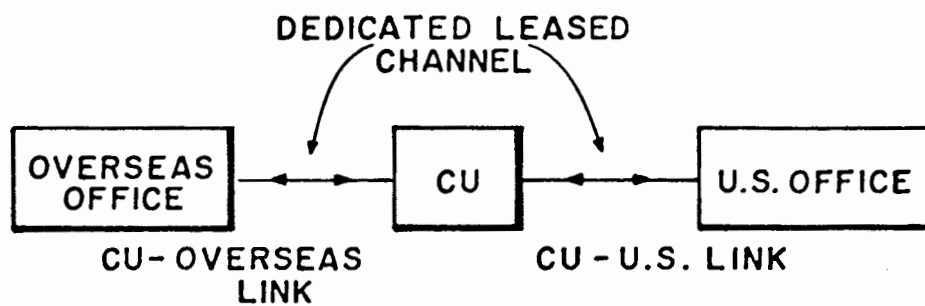
What are the software requirements for this computer? Does it need an Apple II clone?

Come to COSMAC CON '74 Fall and find out.

COMPUTER

References

1. J. Wesbecker, "A Simplified Microprocessor Architecture," *Computer* (March 1974).
2. N. Sadeh and J. Wesbecker, "COSMAC - A Microprocessor for Minimum Cost Systems," 1974 ISHRCN, Session 17/2.
3. "Putting Data on an Ordinary Audio Recorder," *The Electronic Engineer* (May 1972) p. 18-9.
4. J. Wolf, "Ratio Recording for Lower Cassette Recorder Cost," *Computer Design* (December 1972) p. 76.
5. M. Gardner, *Logic Machines and Diagrams*, McGraw-Hill, 1958.
6. A. Opler, "Testing Programming Aptitude," *Datamation* (October 1963) pp. 28-31.
7. J. Jones, "The Flipboard: A Simple Diagnostic and Remedial Tool," *Educational Technology* (January 1973) pp. 59-61.
8. H. Nurge, "I opales," *Popular Electronics* (November 1973) pp. 63-66.
9. J. L. Hughes and K. J. Ingold, "Hesapawn: A Learning Demonstration," *Datamation* (March 1968) pp. 67-73.
10. B. L. Schwartz, "Mathematical Theory of Think-A-Dot," *Mathematics Magazine* (September-October 1967) pp. 187-193.
11. R. J. Graf and G. J. Whalen, "Electronics Football Lets You Play Like the Pros," *Popular Mechanics* (October 1967) pp. 147-149, 228.
12. J. W. Cusack, "The Princess Puzzle," *Popular Electronics* (May 1971) pp. 27-32.
13. D. W. Zuckerman and R. E. Horn, *The Guide to Simulations/Games for Education and Training*, Information Resources, Inc., 1973.
14. M. Gardner, "John Conway's New Solitaire Game LIFE," *Scientific American* (October 1970), pp. 120-123.
15. D. W. Hagelbarter, "Seer, A Sequence Extrapolating Robot," 181. *Transactions on Electronic Computers* (March 1956), pp. 1-7.
16. J. Gilbert and J. Crib, "A Simple Hardware Model of a Turing Machine: Its Educational Use," *Proceedings of the ACM Annual Conference* (August 1972) pp. 324-329.
17. J. Ackerman, "Computers Teach Math," *The Arithmetic Teacher* (May 1968), pp. 467-468.
18. D. H. Ahl, Ed., *101 Basic Computer Games*, Digital Equipment Corp., 1973.



TYPICAL LEASED CHANNEL SYSTEM

LCCU FUNCTION

- SPEED/CODE CONVERSION
- MESSAGE STORAGE
- PLAYBACK/ANSWERBACK
- CHARACTER EXPANSION
- CHANNEL CONTROL

SPEED/CODE CONVERSION

- OVERSEAS — 5 BITS/CHAR
(BAUDOT)
 - SLOW SPEED (TTY)
(50-300 BITS/SEC)
- DOMESTIC — 8 BITS/CHAR.
(ASCII)
 - HIGHER SPEEDS
(300-4800 BITS/S.)
 - VOICE GRADE

I-3

MESSAGE STORAGE

- TIME DIFFERENCES
- BUFFERING FOR SPEED
DIFFERENCES

EXAMPLE OF CHARACTER EXPANSION

ASCII BAUDOT

\$ —→ DOLLAR

—→ NUMBER

I-7

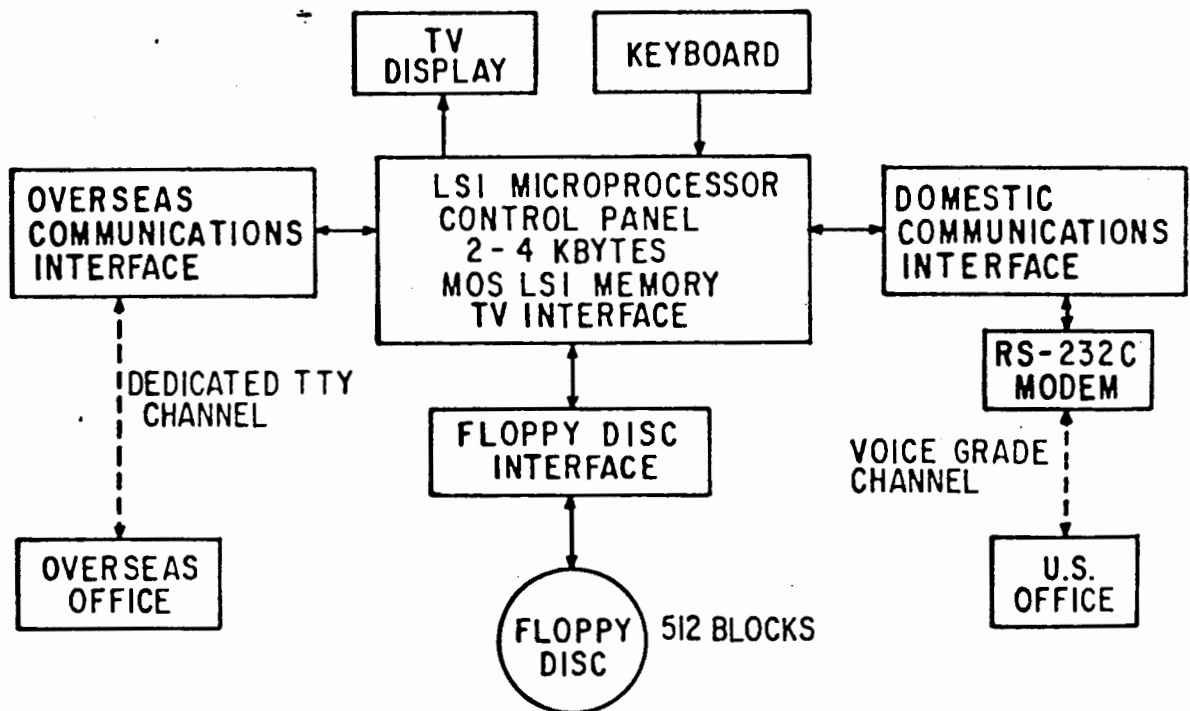
LCCM

PRE-~~R~~P IMPLEMENTATION

- HARDWARE CODE CONVERSION
- HARDWARE ANSWERBACK
- TAPE LOOPS (MIN. OF TWO)
- CUSTOMIZATION IN HARD.
 - CHAR. EXPANSION
 - CONTROL CHAR.
 - CHANNEL CONTROL

LCCU

- COSMAC
- FLOPPY DISC & INTERFACE
- COMMUNICATIONS INTERFACES
- OTHER



LEASED CHANNEL CONTROL UNIT

COMMUNICATIONS INTERFACES

- USE UAR/T LSI CHIPS
- TTY/RS-232
- FULL DUPLEX
- SOFTWARE CONTROL
 - SPEEDS
 - CHARACTER CODES & FORMATS
 - PARITY
 - CHARACTER EXPANSION

ADVANTAGES OF MICROPROCESSOR IMPLEMENTATION

- SYSTEM CUSTOMIZING VIA SOFTWARE CHANGES
- DYNAMIC MEMORY ALLOCATION
- PROGRAMMABLE SYSTEM SPECS (SPEEDS, CODES, ETC.)
- MESSAGE PRIORITY
- LOWER COST, HIGHER PERFORMANCE
- MAINTENANCE
- RAPID SYSTEM REGENERATION

FUNCTIONS

- SERIAL TO PARALLEL, PARALLEL TO SERIAL
- CHARACTER SYNCHRONIZATION & BUFFERING
- PARITY GENERATION & CHECKING
- ABNORMAL CONDITION DETECTION
- STATUS/OPERATOR DISPLAYS

S-17

LCCM

COMMUNICATIONS CE'S

- INTERRUPT CPU
- FLAGS ENCODED
- PRIORITY IN SOFTWARE
 - READ BEFORE WRITE
 - DOMESTIC (HIGH-SPEED)
BEFORE OVERSEAS (LOW-SPEED).

1224

INTELLIGENT COMMUNICATIONS SUBSYSTEMS

MICROPROCESSOR-BASED

- CONCENTRATORS
- MULTIPLEXORS
- FRONT ENDS
- CODE/SPEED CONVERTERS
- ANY COMBINATIONS OF THE ABOVE

1224

FLOPPY DISCS

- FLOPPY DISCS
- IBM DISKETTE
 - SOFT/HARD SECTORING
 - DOUBLE FREQ. ENCODING
- CDS-110 FD (USED IN SYST.)
- CE - ORGANIZATION
 - I/O INSTRUCTIONS

FLOPPY DISCS

- LOW COST ($\sim .03$ ¢/BIT)
- RANDOM ACCESS (TO BLOCK)
- NON-VOLATILE

FLOPPY DISC (FD)

- FLEXIBLE OXIDE COATED DISC
- IN JACKET
- DATA ON ONE SIDE ONLY
- HOLES FOR
 - INDEXING
 - SECTORING (OPTIONAL)
- HEAD CONTACT WHEN
READING/WRITING

TYPICAL SPECS

- CAPACITY \Rightarrow 1-3 MBITS
- TRACKS \Rightarrow 64-77
- TRANSFER RATE \Rightarrow 33-250 KBITS/S.
- DISC LIFE \Rightarrow 1-10 MILLION PASSES
PER TRACK
- (HARD SECTORS \Rightarrow 8-32 /TRACK)

II-3

FD DRIVE

- SLOW SPEED (90-375 RPM)
- HEAD LOAD/UNLOAD
- HEAD STEPPED TO DESIRED
TRACK (6-10 MS/STEP)
- STABILIZATION DELAYS
(HD LOAD, LAST STEP, ETC.)

II-4

IBM DISKETTE

- 77 TRACKS
- 360 RPM
- 250 KBITS / SEC.
- USE ONLY ONE SIDE OF DISC
- SINGLE INDEX HOLE (SOFT)
- * • 32 SECTOR HOLES
- * HARD SECTORING STD.

X-5

LCCU

DISKETTE CAPACITY

- 3.1 MBITS RAW CAPACITY
- HARD SECT. 128 BYTES/BLOCK

$$32 \times 77 = 2464 \text{ BLOCKS}$$

$$\text{DATA } \left\{ \begin{array}{l} \Rightarrow 315,392 \text{ BYTES} \\ \Rightarrow 2.5 \text{ MBITS} \end{array} \right.$$

- SOFT. SECT. \simeq 2 MBITS.

HEAD CONTROL

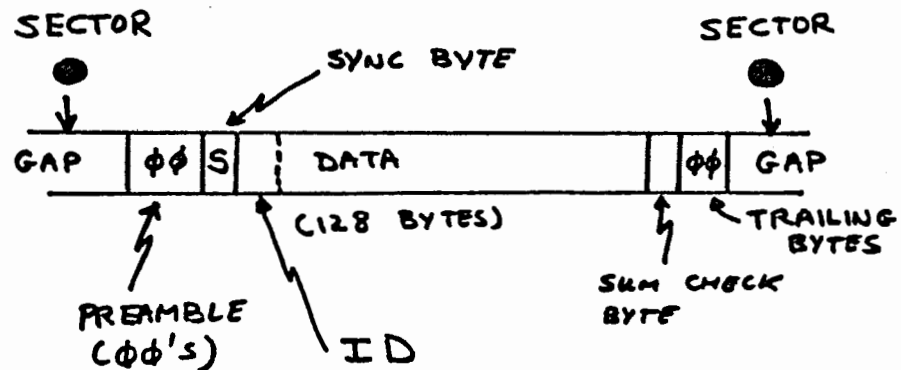
- HEAD MOVED VIA STEPPER
(6-10 MS STEPS)
- HEAD LOADED/UNLOADED
- STABILIZATION DELAYS
(HEAD LOADED, LAST STEP, ETC.)

II-7

DISC SECTORING

- SOFT (SOFTWARE)
 - ONE INDEX HOLE
- HARD (HARDWARE)
 - ONE INDEX HOLE
 - SECTOR HOLES PHYSICALLY
IDENTIFY START OF SECTOR

DISC FORMAT



II-10

LCCM

HARD SECTORING (VS SOFT)

- 1/2 REV. AVG. LATENCY (1 FOR S.)
- SIMPLER HARDWARE
- LESS CPU ACTION
- FIXED LENGTH RECORDS (128/256 BYTES)
- HIGHER CAPACITY (2.5 VS 2 MBITS)

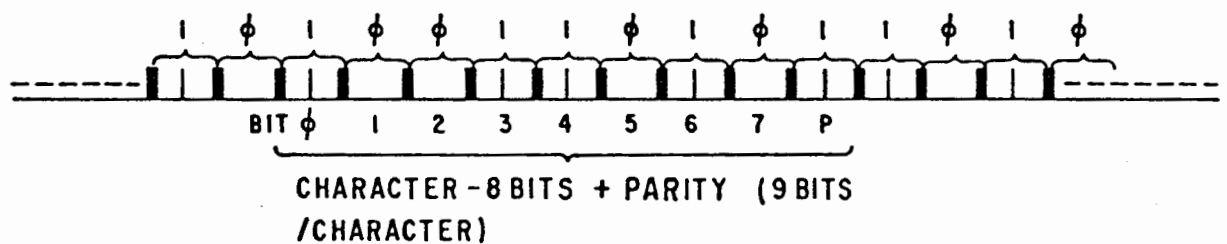
HARD

SOFT
(IBM FORMAT)

NOTE:

GAPS BETWEEN BLOCKS USUALLY
ALLOW FOR DATA CHAINING
"ON THE FLY".

LOCK



LEGEND

| SYNC PULSE

| DATA PULSE - PRESENCE \Rightarrow "1", ABSENCE \Rightarrow " ϕ "

P DESIGNATES PARITY BIT (ODD PARITY)

} ALL PULSES ARE OF THE SAME
WIDTH - WIDER SYNC PULSES
ARE ONLY FOR EMPHASIS

BIT STREAM WITH DOUBLE FREQUENCY ENCODING
AND ODD PARITY BIT (9 BITS / CHARACTER)

LOCK

DATA RATE VS CPU CYCLE

250 KBITS / SEC

\Rightarrow 31.25 KBYTES / SEC

I.E., ONE DMA CYCLE NEEDED
EVERY 32 μ SEC.

\Rightarrow NEED CPU WITH CYCLE
TIME $< 32 \mu$ SEC.

(NOTE: NO CYCLE STEALING DURING
ACCESS TIME)

A microprocessor implementation of a dedicated store-and-forward data communications system

by P. M. RUSSO and M. D. LIPPMAN

*RCA Laboratories
Princeton, New Jersey*

INTRODUCTION

The stored-program approach to data communications system design is not new. The past several years have witnessed a large and ever-increasing number of minicomputers and larger processors dedicated to the implementation of a variety of data communications functions. To date, however, the use of computers has been relegated primarily to medium-sized and larger systems where highly complex data communications requirements justify reasonably large investments in hardware and software. In many low-end applications, however, the high cost of minicomputers and their associated peripherals cannot be justified. This is especially true in a dedicated system where two terminals (or groups of terminals) communicate over a dedicated communications channel.

The advent of low-cost LSI microprocessors and mass storage devices (e.g., floppy discs) is having a significant impact on the design of new low-end data communication systems. A multitude of systems that, until recently, would have required a hard-wired logic implementation with logic speeds far in excess of the system requirements, can now realize the many advantages of the stored program approach. These advantages include, among others, lower cost, programmability (flexibility), improved reliability, ease of maintenance, and the addition of many new system functions hitherto impractical to implement via hard-wired logic. The many advantages of micro-processor implementations of data communications systems are discussed more fully below.

In this paper we will describe a dedicated store-and-forward system that may prove suitable for international data communications. The system is configured around the RCA COSMAC LSI microprocessor and the Century Data Systems CDS-110 floppy disc, with suitable disc, keyboard, display and communications interfaces. The system architecture, disc interface organization, COSMAC microprocessor, data structure and the system functional capability will be detailed. Emphasis will be placed on various new functions achievable with stored-program control. Finally,

other potential applications of microprocessors in the data communication field will be briefly discussed.

LEASED CHANNEL SYSTEM

The dedicated store-and-forward data communications system that we have implemented is functionally related to currently commercially available international leased channel systems. Hence, for the purpose of this paper, we will also refer to the microprocessor based system as a leased channel system. An international leased channel is a dedicated communications link between a customer's domestic and foreign offices. Figure 1. Typically, this link is made via a leased channel control unit (CU) that performs a variety of functions. Many CU's are usually located in a single centralized control room. The CU acts as an interface between domestic and foreign communications networks. This includes both electrical interfacing and message format interfacing, such as code and speed conversion. Typically, the international link employs 5-level baudot character encoding whereas the domestic link uses 8-level ASCII. The control unit also handles character expansion, handshaking, playback/answerback control, message switching and message storage. Message storage is often desirable both because of existing time differences between distant offices and because of transmission speed differences on the international and domestic links.

A typical current implementation of a CU consists of a dedicated rack of special-purpose hardware specifically tailored to a given customer's requirements. The principal sub-assemblies are a message switch with appropriate communication interfaces, code converters, and relatively expensive tape-loop storage media. Also a minimum of two tape loops are required per system since internationally and domestically bound traffic cannot share the same loop. Finally, use of serial storage media necessitates that messages be transmitted in the same order they are received.

The principal undesirable features of hard-wired leased channel implementations are high cost, difficulty in custom-

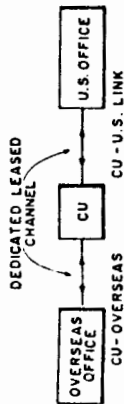


Figure 1—Typical leased channel system

izing a design (e.g., answerback sequences, character expansions, character codes and formats are hard-wired), and high maintenance costs with high mean times to repair. Furthermore, many desirable system functions are prohibitive to implement via random logic.

MICROPROCESSOR IMPLEMENTATION

The microprocessor based based channel control unit described in this paper is presented in Figure 2. It consists of three basic components: an LSI microprocessor, a floppy disc drive and its associated processor interface, and a pair of communications line interfaces. The communications interfaces support simultaneous full duplex asynchronous data transmission. Modular design minimizes the hardware changes required to support a wide variety of communications channels and devices. Parameters such as data rate, transmission mode, character length, and parity are programmable. The microprocessor, called COSMAC, is an RCA proprietary, byte-oriented, parallel structure machine. Even though COSMAC can directly address 65 Kbytes of random access memory, only 4 Kbytes are used in our current design. The TV display and keyboard are not essential parts of the system but can be used to display status information and to provide the system with operator interaction and diagnostic capability when required. The floppy disc used in the current design is a Century Data Systems CDS-110 and as implemented, has a storage capacity of 118 Kbytes.

The leased channel system depicted in Figure 2 can

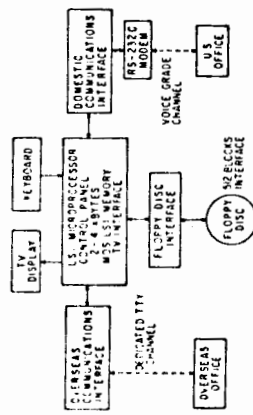


Figure 2—Leased channel control unit

TABLE I Advantages of Microprocessor Implementation

1. Rapid Implementation of Customer Requirements
2. Dynamic Memory Allocation
3. Message Priority Queuing
4. Programmable System Specifications
 - Data Rates
 - Character Formats and Codes
 - Character Expansion
 - Playback Answerback Control
 - Error Detection Correction
5. Maintenance
 - Single Unit Backup
 - Fault Isolation
 - Diagnostics
6. Data Message Logging and Archival Storage
7. Improved Cost Performance

duplicate all the system functions available in a hard-wired design at much lower cost. Additionally, many new and highly desirable system functions and features are now available at no extra cost in hardware. These are summarized in Table I.

Notice that the functions and features given in Table I are in addition to the functional capabilities of typical current implementations.

Rapid implementation of customer requirements

Since system control resides in a RAM it is extremely flexible because the base system can be easily tailored to individual customer needs by software modification. Thus the time needed to get a new customer on-line is greatly reduced. Furthermore, system upgrading in the field is made possible by modular software design.

Dynamic memory allocation message priority queuing

As previously discussed, where message storage is required, two tape loops are currently employed—one for the domestic bound and one for the overseas bound traffic. Traffic flow imbalances between domestic and overseas traffic often result in heavy loading in one direction. This is currently resolved by adding an additional tape loop storage unit to the heavy traffic direction. Meanwhile, the tape loop associated with the lighter traffic remains essentially unused. The use of a random access storage medium (floppy disc) results in the dynamic allocation of memory wherever it is needed—thus if the traffic is heavier in one direction, that direction will be assigned more memory.

An additional desirable feature available for free is that of message priority queuing. Since messages are stored on a random access device, one can tag each message with a priority. Thus when messages are requested from the system, those with highest priority can be transmitted first. Contrast this with the first-in-first-out (FIFO) requirement associated with tape loops. Finally, individual messages can be retransmitted as needed without having to retransmit the entire stream.

Programmable system specifications

System specifications such as domestic and overseas data rates, character codes and formats, control characters, playback answerback sequence generation, detection, and character expansion can all be implemented by simple software changes. Character expansion is necessitated by the use of 5 byte transmit codes in international data communications. Thus many ASCII characters have no standard counterparts and are represented by sequences of valid characters. For example, a customer may request that ASCII "H" expand into DOLLAR (hardwired). Finally, error detection/correction algorithms can be implemented or modified by changes in the software.

Maintenance

Systems tailored to specific customer requirements will differ only in the software. Thus only single unit backup needs to be maintained. When a system component fails (e.g., tape drive, etc.), simply plug in a new component, reload the software and go. To facilitate the latter function, the disc interface supports a bootstrap function which permits system regeneration in seconds. This feature will be discussed more fully in a following section on the disc interface organization.

A system structure based around a chip lends itself readily to self-testing. Diagnostic programs can be run to isolate and identify faulty system modules. A keyboard and TV display permit the operator to interact with the diagnostic programs and rapidly determine which portion of the hardware is inoperative. Note that this diagnostic testing can be done off-line, since the load of the faulty system can be taken up by an identical hardware when backup system.

COSMAC

The COSMAC microprocessor was designed for implementation on a single 40-pin, MOS LSI chip. It provides a flexible, powerful, building block for a variety of stored program products, including device controllers, terminals, and computers. Special features such as "on-chip" DMA

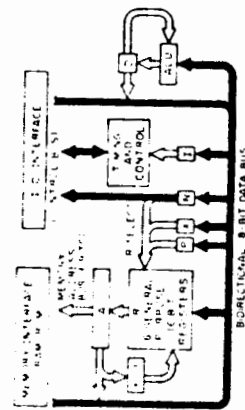


Figure 3—COSMAC microcomputer architecture

channel minimize added circuits for complete systems. A parallel internal structure using state circuits provides maximum reliability, speed, testability, and application flexibility. Proprietary architecture, utilizing a one byte instruction format, minimizes program memory requirements.

Figure 3 illustrates the microcomputer architecture. It represents an array of system-level circuit purposes registers. This is essentially a 16×16 bit RAM. P, X, and N are three 16-bit registers. The contents of P, X, or N select one of the 16 R registers. R(N) will be used to denote the specific R register selected by the 16-bit data contained in the X register. R(N) denotes the low order 8 bits (byte) of the R register selected by N. R(N) denotes the high order byte. The contents of a character 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(RN) 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. D is an 8-bit register that functions as an accumulator. The ALU is an 8-bit logic network. It is a four-bit instruction register. Bytes can be read onto the contents data line 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 Figure 4. The instructions are summarized below where [XN] contains two hex digits and represents the instruction byte.

Register Operations

- [1 N] Increment R(N) by 1
- [2 N] Decrement R(N) by 1
- [8 N] Transfer R(N) to D
- [9 N] Transfer R(N) to D
- [A N] Transfer D to R(N)
- [B N] Transfer D to R(N)
- [C N] Transfer D to R(N)

Memory Operations

- [4 N] Load D from M(RN) and increment R(N)
- [5 N] Store D in M(RN)

Miscellaneous Operations

- [0 N] Idle
- [3 N] Branch
- [6 N] Input/output byte transfer
- [7 N] Interrupt control
- [D N] Set P to value in N
- [E N] Set X to value in N
- [F N] ALU operations

For the miscellaneous operations, N no longer selects one

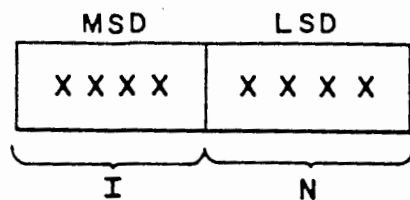


Figure 4—One byte instruction format

of the R registers, but is divided as needed. For example, for instruction "3", N selects the type of branch instruction desired.

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 an input/output byte can be stored at M(R(X)). 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. COSMAC can directly address up to 65K bytes of RAM or ROM, has a program execution speed of up to 100,000 instructions/second and can achieve a DMA burst transfer rate of up to 200,000 bytes/second. Additional and more detailed information on the COSMAC microprocessor architecture and its instruction set is available in References 1 and 2.

Our experience, to date, indicates that COSMAC is indeed very well suited to the types of processing required in data communications systems (table look-up, interrupt driven software, data management). Multiply and divide must be done in software, but these operations are uncommon in low end data communications systems.

COMMUNICATIONS INTERFACES

An interface links the microprocessor to each communications channel. The domestic interface connects to a voice-grade telephone line via an RS-232C compatible modem or data set. The overseas interface connects directly to overseas channel terminal equipment which accepts TTY current-loop signals.

Both interfaces may be active simultaneously supporting full-duplex asynchronous data transmission. They perform serial-to-parallel and parallel-to-serial conversion, parity generation and checking, and character synchronization and buffering. For convenience, status display registers are also included in the communications interfaces. To maintain

maximum flexibility, all other communications functions, such as code conversion and control character recognition, are performed in software. More detailed information on the operation and implementation of the communications interfaces is presented in a companion paper.³

FLOPPY DISC INTERFACE

Floppy discs

Since the commercial introduction of floppy discs in the latter part of 1972, the number of announced drives has increased from two (CDS and Memorex) to almost a dozen. Use of floppy discs in systems such as the IBM 3740 Data Entry System attests to the floppy disc drive's basic simplicity, low cost (potentially much lower) and applicability to many low end systems.

Properties common to most floppy disc drives include the following: The recording medium is a non-volatile, flexible, small (7.5" dia.) oxide coated disc, usually packeted in an envelope. Data is recorded on only one side of the disc. Typically, a one inch recording band is accessible through an aperture in the envelope. During reading or writing, contact is usually made between the head and medium (in some cases, a "fragile" air bearing exists). Since some contact exists, precise "flying head" and mechanical stability problems are avoided; however, head and medium wear do occur and must be accounted for (usually by maintaining head disc contact only during reading and writing). The disc rotates at slow speeds ranging from 90 to 400 RPM, can store typically 0.5 to 2.5 Mbits, has an average access time of about 500 milliseconds, can transfer data at 33-250 Kbits/second, and costs about \$5. Changing cartridges can be accomplished in seconds. More

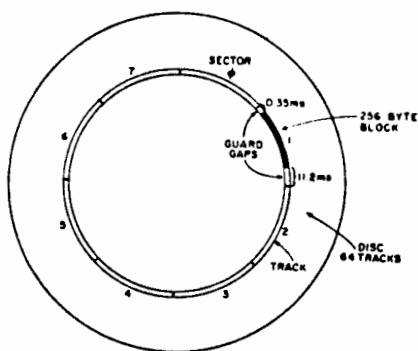


Figure 5—Floppy disc data structure

information on floppy discs is available in References 4 through 6.

Interface organization

Appendix A presents a block diagram of the floppy disc (FD) interface and discusses the hardware implementation. The philosophy behind the interface architecture was motivated by our leased channel project. A substantial portion of the CPU's processing power will be needed to support the communications interfaces. Thus it was decided to dedicate the CPU's Direct Memory Access (DMA) channel to the FD whenever I/O to/from the disc is required. Of course, the DMA is available to other devices (such as the TV display) whenever the FD is not busy. With the above philosophy, the CPU need only issue a few instructions and then check, periodically, to see if the data transfer is completed.

Four instructions need to be issued by the CPU to effect data transfer. These will select the FD, load appropriate status information into a two-byte buffer (2 instructions) and start the I/O operation. When block transmission is completed, a flag is raised which can be tested by the CPU.

Each disc sector (8 sectors/track, 64 tracks/disc) will contain one 256 byte block, see Figure 5. Each block consists of a 16 byte synchronization pattern, 232 data bytes and an 8 byte trailing pattern, Figure 6. The total disc capacity is thus 512 blocks or 131,072 bytes (118,784 data bytes). There are 9 bits/byte—8 data bits plus parity. This block per sector approach greatly simplifies the interface electronics and seems to be a useful organization for most low cost random access bulk storage systems envisioned.

The selection of a 256 byte block size in conjunction with the present design allows for an 11 msec guard gap following the data block before the next sector is reached. Thus there is enough time for the software to access the very next sector should data chaining be desired.

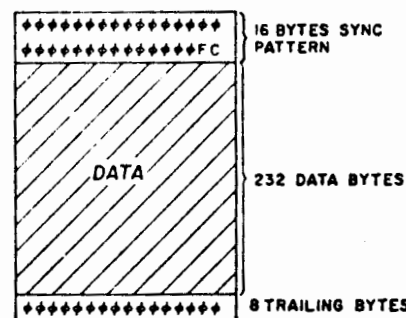


Figure 6—Data block structure

The present interface is equipped with a control panel which, among other things, resets the head over track #0. A bootstrap feature enables the user to enter a loader program (resident on the disc) into the CPU's memory at the flick of a switch. This loader can then load the CPU's memory with any other program residing on disc, and render the system completely self-contained modular as Initial Program Loading (IPL) is concerned.

Control buffer

Since the FD interface must work independently of the CPU, it must initially be provided with status information. This status information must include the following:

- track information
- sector information
- read/write information

Status information is stored in the interface in a two byte control buffer. The bit assignments are presented in Figure 7.

Use of the control buffer in the FD interface enables it to work completely independently from the CPU since all the information the interface needs is continuously available to it.

Disc related CPU instructions

In our current implementation, a 61 instruction "block" of the peripheral device to or from which information is to be transferred. Device selection is accomplished by assigning each I/O device a "device number" and ensuring that M(R(X)) contains the desired device number when the "61" instruction is executed. The five CPU instructions needed for disc CPU communication are as follows:

I	M(R(X))	Function
61	08(hex)	Select FD
62	11XXXXXX	Load Buffer A
62	01XXXXXX	Load Buffer B
62	10XXXXXX	Start I/O
62	00XXXXXX	Turn FD off

When a start I/O instruction is issued, the head moves over the desired track and the correct sector is located. Simultaneously, the head is loaded (contact with disc is made) and all suitable stabilization delays are generated. When the desired sector reaches the head and all stabilization delays have elapsed, the interface will raise the IN REQ or OUT REQ lines of the DMA either wishing to store a byte in M(R(0)) or requesting a byte from M(R(0)). See References 1 and 2 for details on the operation of the CPU's DMA channel. When one data block has been trans-

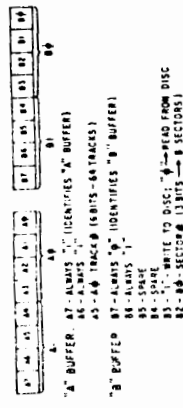


Figure 7. Control buffer

mitted to from the disc, an external flag is raised which can periodically be tested by the CPU. If the block transfer is complete and if the operation was a disc read, the CPU can then test a different external flag to determine whether a parity error has occurred. If the software desires to fetch the block of data stored in the next sector for any other sector on the same track, the software can issue a 62 instruction with M(RX) = 01XXXXXX to update control buffer "R" followed by a 62 instruction with M(RX) = 0XXXXXX to start another block transfer. The 11.2 ms. guard gap following the data block in each sector is sufficiently large to enable the software to catch the data block on the very next sector. If the software no longer needs the disc, a 62 instruction with M(RX) = 00XXXXXX is issued to turn the device off.

I/O Transfer rate

The present design results in an average access time (seek + latency) of 500 ms and a data block transfer time of about 70 ms for a total average transfer and access time of 570 ms. Raw data transfer between the FD interface and the disc itself occurs at 33 Kbits/sec. Insofar as the CPU is concerned, the critical I/O rate is in bytes/second. Between the CPU and interface, the burst transfer rate is given by $33 \times 9 = 27$ Kbytes/second. Hence less than 2 percent of CPU cycles are stolen during disc data transfer.

Bootstrap loader

In the following it is assumed that sector 0 of track 00 contains a bootstrap program which can be used to load programs previously stored on disc into the main memory. The programming convention required for the programs is as follows: on disc is given in Appendix B. The structure of the control buffers is such that when they are cleared (by a CPU reset), they point at sector 0, track 00 and are in the "read" mode. The disc interface is organized such that the block of a switch will select the disc and issue a false "start I/O" instruction which will bring the loader into main memory. Any program residing on disc can then be loaded via this loader program eliminating the need for auxiliary

program load devices such as cassettes or paper tape and greatly simplifying system regeneration after a crash.

CONCLUDING REMARKS

The advent of low cost LSI microprocessors and bulk storage devices will bring about a profound change in the architecture of next generation low end data communications systems. Microprocessor based data communications, typified by the leased channel system described in this paper, will begin to proliferate before the end of this decade. Many desirable system functions and features will be made economically viable via the stored program approach. Functions such as programmability, dynamic memory allocation, message priority queuing, etc. have already been discussed. However, system features such as ease of maintenance, which are of prime importance to the operator, rarely receive sufficient attention from system designers. A microprocessor based system, with modular hardware, can recover from a failure in minutes instead of days, as is sometimes necessary in the repair of customized hard-wired logic. Simple substitution can identify the faulty hardware, and the system can be regenerated in seconds.

Looking into the future, many other microprocessor based data communications systems can be envisioned. Intelligent multiplexers, buffers, concentrators, code speed converters, and/or any combination of the above are but a partial list. These systems are certainly technically feasible and may operate either on a stand-alone basis or as adjuncts to larger computer based data communications systems.

The advent of commercially available all digital communications channels may encourage the development of intelligent repeaters where error detection correction algorithms, speed code conversion, buffering, multiplexing, concentration and routing can all be implemented (and readily modified) via software. As new applications and mass markets emerge for low end microprocessor based systems of all types, new LSI CPUs, and matching low cost peripheral devices will certainly become available. These in turn will spur the development of multi-microprocessor systems where many of the peripheral device controllers will themselves consist of dedicated CPUs. Thus it seems reasonable that this decade will witness the introduction of complete low-end computer systems, including CPU, display, simple keyboard and mass storage, having manufacturing costs of under \$1000. Therein, perhaps, lies the germ for yet another technological mini-revolution that will more than rival the current calculator explosion.

ACKNOWLEDGMENTS

The authors are indebted to R. O. Winder, A. Longo, and J. A. Wislender, the inventor of the CUSMAC architecture, whose constant support and many hours of discussion have been invaluable in the successful evolution of this

system. Finally, the authors wish to thank D. Nichols for his time work in building the prototype hardware and for his patience and responsiveness in our many design changes.

REFERENCES

1. Wislender, J. A. "A Simplified Microcomputer Architecture." *Computer*, February 1971.
2. Saia, N. and J. A. Wislender. "CUSMAC: A Microprocessor for Minimum Cost Systems." *Proceedings of the IEEE International Conference on Communications*, New York, N.Y., March 26-29, 1971.
3. Lippman, M. D. and P. M. Rhee. "A Microprocessor Controller for International Leased Data Channels." *Proceedings of the IEEE International Conference on Communications*, Minneapolis, Minn., June 17-19, 1971.
4. Boser, G. D. "Selecting a Mass Storage Memory." *EE Systems Engineering Today*, June 1973, pp. S184.
5. "Focus on Disc and Drum Memory." *Electronics Design*, Vol. 20, No. 10, May 11, 1972, pp. C10-C12.
6. Davis, Solovey. "Disc Storage for Microcomputer Applications." *Computer Design*, June 1973, pp. 55-56.

APPENDIX A—DISC INTERFACE HARDWARE IMPLEMENTATION

A detailed block diagram of the FD interface logic is presented in Figure 8. Six basic functions can be identified. The control logic is activated either by CPU (normal operation) or by control panel (initial start-up) signals. Except for initial start-up, the control panel is only used to display status information. The control logic acts on information previously stored in the control buffer, simultaneously positioning and loading the head and generating all the required settling delays. When the head is over the desired sector, and all required delays have elapsed, disc reading/writing is initiated. When a disc block read is desired, the sync pattern detector locates the beginning of the data stream and 256 bytes are framed, checked for parity, and transmitted to the CPU via the DMA channel. An external flag is raised to signal end of transmission. When it is desired to write a block to disc, output bytes are immediately requested from the DMA channel. These bytes are, in turn,

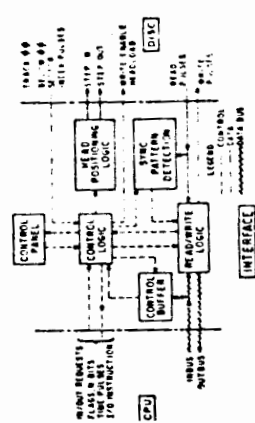


Figure 8. Floppy disc interface logic diagram

serialized and appended with an odd parity bit. When 256 bytes have been written, a flag signaling end of transmission is raised. The end of transmission flag can be tested by the CPU, which can then either modify if needed the control buffer and issue another start I/O instruction, or issue a "turn FD off" instruction which will, among other things, unload disc head to minimize head medium wear. Head disc contact is maintained only when the disc is active.

The current disc interface design can readily handle several disc units by adding simple multiplexing logic.

APPENDIX B—DISC PROGRAMMING CONVENTIONS

Programs must start at memory address hex 0000 (M(0000)). M(0000) to M(0007) can be used during program execution as temporary storage, or as the TV display area. However, program execution must not depend on the initial contents of this area. Utility programs, including the disc loader, will be executed in M(0000) to M(0007). The last function performed by any utility program is to issue an unconditional branch over the utility code to permit normal program operation.

Linking microprocessors to the real world

**A proper interface serves as communications traffic cop,
setting priorities and directing the flow of messages**

Microcomputers promise the engineer new design freedom. But, to harness the potential power of tiny computer chips, he has to enter an often unfamiliar world where software and circuitry must be skillfully combined. In forging the connections between various pieces of microcomputer system equipment, the engineer faces a task that demands the full use of these skills.

The box on this page reviews some of the basic terms used to describe this interconnection, or interfacing process.

Starting on the next page, Paul Russo and Michael Lippman describe how they designed the interfaces for a microcomputer-based store-and-forward communications system. Their experience illustrates how interfacing techniques can be combined to meet the requirements of a particular system design.

Taking a more general view, it seems important to consider the overall role of a microcomputer interface. The basic job of such an interface is to allow the transfer of information, back and forth between the processor section of the microcomputer system and various devices such as communication lines, keyboards, CRT displays, large memories, data collection devices, and control actuators.

Since the processor usually talks to all its peripherals over only one or two main interconnecting busses, the interface must insure that processor outputs reach only the intended peripheral. In the reverse direction, the interface must provide a means for information from each peripheral to reach the processor without interfering with other units hanging on the system busses. In addition, the interface must reconcile any differences between microprocessor and peripheral timing. The microprocessor runs on its own internal clock. Peripherals may, or may not, have internal clocks of their own.

Howard Falk Senior Associate Editor

What is a peripheral interface?

A *Microcomputer system* centers around a *Microprocessor* unit, capable of performing logical functions under the control of sequences of software instructions. Closely tied to the microprocessor is a *Memory* unit, capable of storing data and programmed (software) instructions.

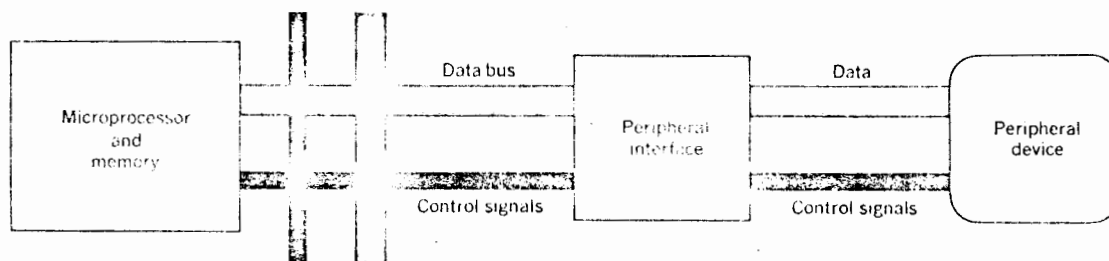
The rest of the system is made up of peripheral units. Devices such as keyboards, teletypes, tape readers, CRT displays, disk memories, and even communications links, are all considered to be peripherals, when they are connected to the processor.

Data flows between the processor and the peripherals over a *Data bus*. Individual, binary data bits, travel on this bus in groups called *bytes*. For most microprocessors, a byte consists of 8 bits (however, there are also 4-, 12-, and 16-bit processors). One of these can be a *parity* bit, which may be added to

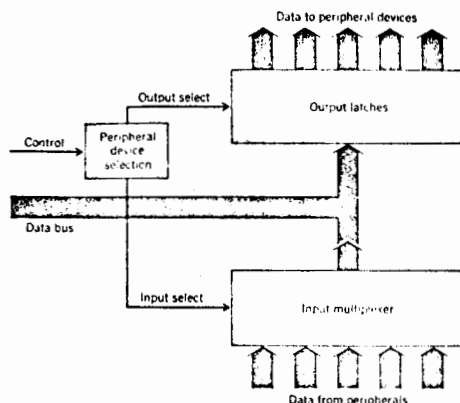
make the sum of the 8 bits in the byte either an odd or an even number. This process can then be used to check for possible errors in the data, caused by noise or system malfunction.

The *Peripheral interface* is necessary to convert the data from the processor format to one that is acceptable to the peripheral device, and also to perform the required conversion from peripheral to processor data formats. The interface also reconciles timing differences and relays processor instructions in the form of control signals to the peripheral.

Flags—usually flip-flops—in the interface, are set to inform the processor of significant current, peripheral conditions. *Interrupts* are signals generated by the interface to force the processor to take immediate action when the peripheral must have quick service.—H. F.



[1] Simple microcomputer input output interface. Output latches, an input multiplexer, and their controls provide the elements needed to connect peripheral devices.



The interface usually handles timing problems by temporarily storing data in shift registers or flip flops. Then, when the processor is ready to take the data from a peripheral, the bits can be "clocked" out of this temporary storage by the processor clock.

Beyond the problem of reconciling data-transfer timing, the interface provides means for the processor to control peripheral actions and to get status information from the peripherals. Most microprocessors also provide one or more interrupt lines that the peripheral devices can yank, when they have an urgent need for attention from the processor.

Latches and multiplexers are basic

A simple input-output interface arrangement is shown in Fig. 1. Here, the bus from the processor transfers data to the peripherals through groups of flip-flops, called latches. A control signal from the processor selects the flip-flop group in which each segment of output data is to be stored, and each of these groups is connected to a different peripheral device.

Data coming into the processor from the peripherals is fed into a multiplexer. Using input select sig-

nals the processor can choose which input it wishes to connect to its data bus.

Multiplexing is generally done by hanging a set of logic gates on the data bus connected to the output of each peripheral device. When enabled, a given group of gates connect the desired peripheral output to the data bus.

Tri-state gates are increasingly used for this function. In addition to the usual input and output signal lines, tri-state gates have a special control line input. When the control line is ON, the gate looks like any other logic gate—that is, its outputs can be either in the "1 or 0 state." The added feature comes in when the control line is OFF. Then the output of the gate has a very high impedance, and looks almost like an open circuit.

For the engineer who wants to connect many different devices directly to a single, common bus, the tri-state gate is indeed a boon. It virtually eliminates the need to deal with complicated impedance loading calculations, and substitutes simple control line selection of devices, for what might otherwise be a more cumbersome multiplexing procedure.

However, many logic designers don't yet feel com-

fortable connecting outputs directly together, in the way made possible by tri-state gates.

Interfaces on a chip are appearing

Most I/O interfaces for microcomputer systems are built up on integrated logic circuit packages, but complete interface packages on a single chip are beginning to appear.

Designing and producing a large-scale integrated (LSI) chip is expensive, but many powerful features can be packed into a small space. The idea is to provide one part that can be set to serve many different interface functions. Then each peripheral device can interact with the microprocessor through its own interface chip. With one chip for each peripheral, the volume use of the chips make the use of LSI economical.

The *Peripheral Interface Adapter (PIA)*, shown in Fig. 2, was designed by Motorola Semiconductor Products Inc. to serve peripheral devices. Data from the microprocessor reaches the peripheral through either of two *Peripheral Interface Registers* that contain the necessary latches. Data from the peripheral to the processor is gated directly onto the processor

Case history: store and forward

Here, in one system, are interfaces for communications, a floppy disk, and a TV display

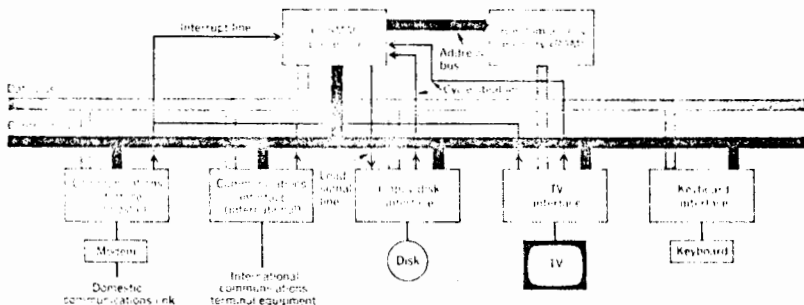
Interfaces were a central concern in our design of a microprocessor-based store-and-forward system at RCA for international leased line communications.

We found it desirable to make a number of interface parameters program-selectable, or program-

able. For example, in our communication link interfaces, transmission characteristics such as data rate, stop-bit length, character length, and parity are programmable and can all be set by simple software instructions.

Paul M. Russo, Michael D. Lippman
RCA Laboratories

[A] System-interface interconnections. Linking the processor and the peripheral interfaces are a data bus, a control bus, and special lines for interrupts, cycle stealing, and initial program loading.



Just how our design finally took shape will become evident as we present a description of the system's interface hardware and software.

Moving messages through the system

Incoming messages enter the system through one of two communications interfaces. Here the messages are converted from a stream of bits into characters, each contained in an 8-bit data byte. These bytes are transferred, one at a time, into the system's semiconductor random access memory (RAM). When 252 bytes accumulate, they form a *block of data*.

The data block is then moved into the larger disk memory, where it is held until needed for retransmission. Outgoing data blocks move from disk, to RAM, to the appropriate communications line.

The RCA COSMAC microprocessor controls this sequence of events with programs written to fill the requirements of the overall store-and-forward communication process.

The entire microcomputer system (Fig. A) consists of a large scale integrated microprocessor, a 4096-byte RAM, and five peripheral interfaces, each of which use a group of integrated circuit packages, and serve to connect different "devices" to the system.

The microprocessor makes both a data bus and a control bus available to the peripherals. These busses carry almost all the information that flows between the processor and the peripherals.

Since several different interfaces are connected to these busses, there must be a clear way to indicate which interface is permitted to be active at any given moment. The *Select* instruction performs this assignment function.

Each interface has its own, unique selection number. For example, a *Select* instruction together with

the number 08 on the data bus, will activate the floppy disk interface. Once an interface is selected, it is free to act on further processor instructions.

To control certain peripheral functions, such as disk startup and head location, or communications transmission speed, the processor issues a *Set* instruction. Other processor instructions are used to test the state of external flag lines. These lines are connected to flip-flops, set by the peripheral interfaces to indicate such conditions as readiness to read or write, as well as faults and error conditions.

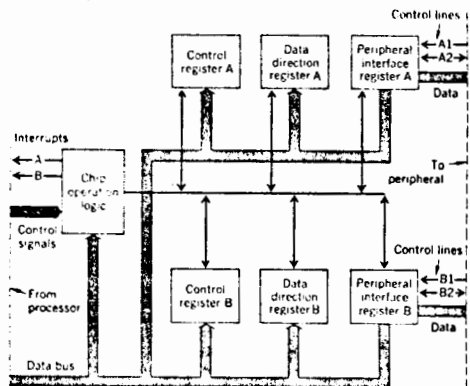
Three special lines allow the peripheral interfaces to initiate system actions, without first getting permission from the processor. By using the *Interrupt line*, the communication interfaces demand immediate handling of incoming data as it arrives on the communications links, and an immediate supply of outgoing data from the RAM, as it is needed for transmission over the links. With the *Cycle steal lines*, the floppy disk memory and TV display gain direct access to the RAM so they can write into the memory, or read from it, without software instructions. Finally, via the *Load line*, the system can be reset and restarted, using a disk-stored program—after a catastrophic failure or loss of power.

Inside the communications interface

When a communications interface has received an incoming character from its communications link, it raises the microprocessor interrupt line. At the same time, this unit raises an external flag, *EFI*, to indicate that a received character is available.

At the microprocessor, the interrupt causes the ongoing program to branch to a special software routine designed to service interrupts. Since there is only one interrupt line, the routine must first find out

[2] Interface on a chip. This flexible and sophisticated interface was designed to connect a wide variety of peripherals to the Motorola M6800 microprocessor system. Two sets of lines are used to send and receive peripheral data.



which of the two communications interfaces sent the interrupt, and then determine what kind of service is needed. This information is obtained by testing to see which external flags are raised.

The actual programmed sequence in the interrupt routine includes a *Select* instruction for each communications interface. After an interface is selected, its external flags are tested to determine the presence and cause of the pending interrupt. Knowing the device and its flagged condition, the processor issues an appropriate instruction to service that condition. For example, in response to flag EF1, indicating a received byte of data is available, a *Read* instruction would be issued to call for the transfer of a byte of data from the communications interface to the pro-

cessor.

The processor selects the peripheral device it wishes to talk to, by sending *chip select* control signals to the PIA *Control and Select Logic*. Every peripheral interface chip in the system is selectively wired so that these signals will activate only the interface that is selected.

Other control signals from the processor allow the processor data bus to reach any one of the six PIA registers shown in Fig. 2. There are also signals that properly time the peripheral interface outputs to the processor and that reset the interface circuits when system power comes on. Two interrupt lines allow the PIA to initiate needed processor activities.

The PIA is capable of a wide variety of different operations, including several powerful, automatic modes. For example, a single command from the processor can send a data byte through the PIA to its peripheral on a handshaking basis, and the PIA will do all the necessary details of housekeeping completely automatically.

To get this kind of operation, the PIA must first be set up, by loading appropriate control bits in its *Control* and *Data direction* registers. These registers are

each set up by a single control byte (8 bits of data) sent from the processor on the system data bus. Separate control signals to the *Chip operation logic* specify the register that will receive the first control byte. When such a byte is received, its bits set the registers for the kind of peripheral operation that is desired.

For example, the *Data direction register* control bits define whether data can flow into, or out of, the system data bus lines connected to *Peripheral interface registers A and B*. The data bus includes eight parallel bit-lines and the direction of each of these can be independently set.

Usually the "A" data lines are set to send data in from the peripheral to the processor, while the "B" lines are used to send data out to the peripheral. Indeed the A and B sides of the PIA are specifically designed to handle these data transfers efficiently. The *Control registers* are set up to select the use of the interrupt lines to the processor and the control lines to and from the peripheral. One combination of control bits in *Control register B* might set up control line B2 (see Fig. 2) to go low, right after a data word from the processor is loaded into *Peripheral interface register B*. The same control setup could also specify that B2

would remain low until a signal from the peripheral—on line B1—indicates that the data has been received. Finally, the same control setup could relay the B1 signal back to the processor, on interrupt line B, to call for another data word to the peripheral. With this kind of automatic operation, the programmer can set up the PIAs to handle peripherals with very simple and fast software routines.

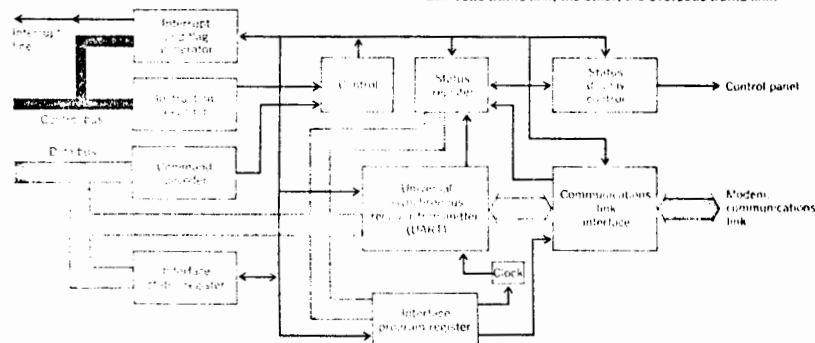
The PIA has two "interrupt" lines going to the microprocessor. These can be set up as flag lines—presenting information about the status of the peripheral to the processor—or they can be set to be used as interrupts, which demand the immediate attention of the processor.

For situations where input-output needs don't have to be served immediately, polling techniques are often used. In that case, the microprocessor is programmed to test the *Control registers* every so often, for given logic levels on the flag lines, and when these are found, it can leave its ongoing program temporarily, to serve the peripheral device that needs attention. When peripherals require more immediate attention, interrupts from the peripheral are used to force the processor to branch from its ongoing pro-

cessor (for transfers in the opposite direction, a *Write* instruction is used). On receipt of the *Read* instruction, the interface places the received byte on the incoming data bus, and the processor clocks the byte into RAM memory.

The system allows up to four external flags (EFs) for each peripheral interface, and the meaning of each of these flags—or combinations of flags—can be different for each interface. For the communications interfaces, EF4 is set in conjunction with EF1 if the received character is erroneous (bad parity). When an interface is transmitting data, EF2 is set to indicate that the next character can be transferred. Finally,

[B] Internal functions of the communications interface. Two of these interfaces are used in the system. One handles the domestic traffic link; the other, the overseas traffic link.



EF3 is used to indicate special conditions, such as abnormal communication line operation.

While a given interrupt is being serviced, all other interrupts must wait their turn. Priorities for servicing interrupts are established in the processor's software interrupt routine. For example, the domestic communications interface is always selected first by the routine. Domestic data rates are usually higher than international rates, and therefore the penalty for keeping the domestic line waiting is greater. Likewise, *Read* interrupts are always given priority over *Write* interrupts, because failure to read may result in loss of data, but the worst penalty for failure to write is time lost on an idle transmission line.

The interrupt driven form of data transfer is well suited to the communications function. Competing functions and devices are easily queued; each input character can be examined and processed as it is received; new devices are easily added; and existing device priorities are easily changed.

The hardware that communicates

Communications interface hardware centers on a large-scale integrated circuit contained in a single 40-lead chip that handles several key functions. This circuit, called a *Universal asynchronous receiver-transmitter (UART)*, converts data bytes from the microprocessor into a stream of serial bits for the communications link. The UART also performs the opposite conversion, taking serial bits from the communications link and shaping them into characters for the microcomputer system. Actually, each character consists of up to 8 data bits, and an odd-even parity bit may be added. When called for, these parity bits are generated by the UART as it transmits characters, and the UART also checks the parity of in-

coming characters from the communications link.

Surrounding the UART are a number of functional blocks implemented in transistor-transistor logic (see Fig. 3). The *Interface program register* provides the means for microcomputer program instructions to control the *Clock* and the communications mode (half- or full-duplex). Flip flops in this register select speed control lines into the clock to allow program selection of bit-per-second transmission and reception rates. The type of parity (even or odd), length of characters, and number of stop bits used in serial transmission are also under program control.

To connect the transistor-transistor logic (TTL) circuits in the interface to communications link circuits, some matching is necessary. This is done in the *Communications link interface* circuitry. For links using RS 232 industry standard data interfaces, voltage-level shifting of TTL signals is needed. For tele-type links, output currents must be controlled to specified levels. Some communications links require that there be no direct ground connections. For these, optical isolation devices must also be included.

Instructions to the communications interface from the processor are received over the system's control bus and routed through the *Instruction decoder* which interprets microprocessor codes and generates logic signals that can be used to control the interface hardware. Since some of these microprocessor codes come over the system data bus, there is a *Command decoder* to interpret this added information. The *Interface status register* indicates whether the interface is, or is not, currently selected and thus allowed to communicate with the processor.

The *Control logic* distributes all the appropriate information to the *Interface program register*, the *UART*, and other parts of the interface. The external

gram to one that will serve the peripheral as soon as possible.

Interrupts and more interrupts

Most present-day microprocessors make some form of interrupt capability available. But there are interrupts and interrupts. Some processors just give the user a single general interrupt signal to work with, and that is often far from adequate. To handle the interrupt properly, the system usually needs to know where it originated and why it occurred.

Other processors, like the Intel 8008, use a single interrupt line, but offer a somewhat more elaborate capability. The user can code three bits, which allow him to reach specified main memory locations. The idea is to store the first instruction of an appropriate interrupt-handling routine at each of these locations. With three binary bits, up to eight different interrupt routines can be addressed. The capability is called a *vector* interrupt.

Some of the newer processors offer more than one interrupt line. The National Semiconductor IMP-16C microprocessors offer two lines—one is a vector interrupt; the other, a general interrupt. Motorola's M6800

processor provides two interrupt lines, and the Toshiba TLCS-12 processor has eight independent lines, with a hardware-implemented priority scheme.

Of course, the system designer can usually make his application work with a single, general interrupt line, but there may be extra costs attached to his design. He may use external circuitry to handle priority when several different interrupts occur together. That will mean added equipment costs. He may be able to solve the priority problem with software. But an interrupt-handling subroutine takes time to handle the job, and time can be a critical design parameter.

The capabilities of the interrupt lines give only part of the design picture. Important also is the sequence of events that occur when an interrupt takes place.

At that time, the processor is probably churning away at ongoing program tasks. When an interrupt occurs, the processor is supposed to complete the current ongoing program step, and then drop everything to take care of the interrupt.

The problem is, that after the interrupt has been served, the processor is supposed to continue the ongoing program just where it left off. To return to this task smoothly, it is always necessary to store

something. Ideally it would be best to store the entire state of the machine including the contents of the arithmetic accumulator, and all registers, as well as all of the flags that indicate the statuses of various system operations. In practical terms, storing the contents of a few key registers may be adequate.

Having stored the ongoing program status, the processor is free to handle the interrupt. When finished, it can retrieve the status information and smoothly resume its ongoing tasks. This store-and-go operation should, ideally, be as fast and simple as possible.

Some of the newer processors, like the Intel 8080, Motorola 6800, and the RCA COSMAC, take care of this store and go operation automatically, but with other machines, the process may be more difficult.

For example, a simple store-and-go procedure is not possible in the Intel 8008. After an interrupt, the 8008 registers that are used to reach the microcomputer memory continue to hold address values needed by the interrupted program. The user can get around this shortcoming by reserving two of the processor's general purpose registers to hold the contents of these main storage address registers during the interrupt. But this is awkward, because in addition to the time

lost for the necessary program steps, two of the system's seven registers are then not available for processing the interrupt.

A more acceptable alternative is to add enough external circuitry to supply a register that can hold three bytes of information. By passing the contents of the two storage address registers through the 8008's accumulator register, the contents of all three of these registers—all the information needed to resume the interrupted program—can be stored in the new external register.

Following the interrupt servicing process, the end registers can be reloaded by using an input instruction to the external register. This process can be quite efficient, in terms of the number of program steps required, if the external register is in the form of a push down stack that can be both "pushed" and "popped" by a single instruction. However, it does mean that added hardware—the external register—must be used.

One-chip communications interfaces

In minicomputer and microcomputer systems, interfaces with communications lines have usually been

flags that inform the processor of the interface current status are placed on the system's control bus by the *Interrupt and flag generator*. This information comes from the UART and Status register, which is used by the UART and other circuits to record the occurrence of such fault conditions as parity errors and open communications lines.

The third major element of the system, in addition to the communications interface and the microprocessor RAM combination, is the floppy disk interface.

Stealing cycles for more efficient operation

The key feature of the floppy disk interface is its direct access to RAM memory, without need for detailed microcomputer program control. Using this direct memory access feature—built into the COSMAC processor—the disk can put data bytes into the RAM, or take them out, without receiving even a *Select*

command. In fact, it can transfer this data while the processor is occupied with other tasks, such as talking to a communications interface.

The direct memory access mechanism used here is called *cycle stealing*. There are normally two microprocessor cycles for each program instruction: a fetch cycle, followed by an execute cycle. When the cycle steal line comes up, say during a fetch, the processor will complete that fetch, and the corresponding execute cycle, and then hold its breath for a one cycle interval before moving on with the next instruction, fetch cycle. It is during these stolen one cycle intervals that data bytes are moved between the RAM and the disk.

Before the cycle stealing can begin, a direct memory access address register must be loaded with the

first RAM memory address of the data block to be read from, or written on, the disk. Then the register is automatically incremented at each succeeding stolen cycle, until an entire block of 252 data bytes has been transferred. And all the processor sees is a slight slowdown, usually less than a 1-percent reduction in the time available for its ongoing program activities.

Controlling the floppy disk

The floppy disks used in this system are 7.5 inches in diameter. Data is recorded on one side of the disk—coated with oxide material—which is made of flexible plastic. The disk is packaged in a paper envelope, and a recording band, about one inch wide is accessible through the envelope.

During reading and writing the disk is "loaded," so that physical contact is made between the read-write head and the disk surface. Since this causes wear, it is desirable to unload the disk as soon as possible. The disk rotates at 90 r/min, can store 1.4 Mb, and costs about \$5. Data can be transferred from the disk at a rate of 33 kb/s, and it takes an average of 560 ms to reach a desired specific data block on the disk.

Each of the disk's 64 concentric data tracks can hold 8 complete blocks of data. And each of these blocks begins with 16 bytes of synchronization information, followed by 232 bytes of data, and capped off by 8 bytes of trailing zeros.

This structure provides an 11-ms gap between adjacent blocks on the same track, and this gives the system enough time to process the blocks one right after another.

Data from the disk is always transferred to the system's RAM memory, before going elsewhere, and all data stored on the disk comes to it from the RAM.

As with the communications interfaces, a *Select* in-

struction—including the disk's identifying number—is used to initiate contact between the processor and the disk. Four additional instructions are used to control disk functions. *Locate A* and *Locate B* specify the disk storage location for each block of data; *Start* loads the disk head and allows data flow to or from the disk when the desired location is reached; *Stop* unloads the disk when the desired data transfer is completed.

To follow these disk control operations in more detail, refer to the interface function diagram in Fig. C. The *Locate A* instruction sends the desired track location number to the interface *Control Logic*, and *Locate B* sends the corresponding sector location number.

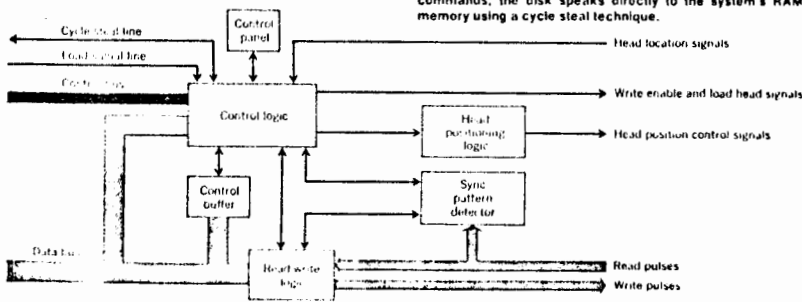
Actually, the *Locate B* instruction serves a double purpose since it also tells the control logic whether data is to be written onto the disk or read from it. This information, along with the current track and sector location data, is stored in the *Control Buffer*, where it is available until updated by new *Locate A* or *Locate B* instructions.

When a *Start* instruction is received, the *Control Logic* activates the *Head Positioning Logic* to move the head into the position stored in the *Control Buffer*, and simultaneously loads the head into contact with the disk.

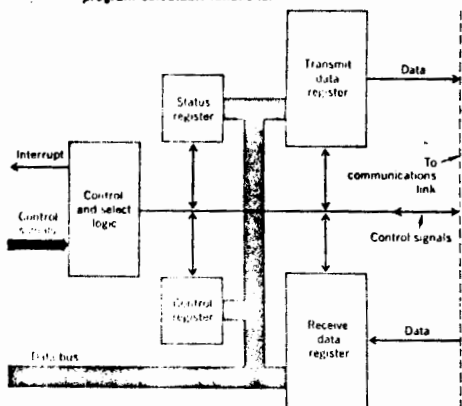
After the disk mechanisms have had time to settle to steady-state operation (delays to guarantee this are generated by the *Control Logic*), the actual data transfer is initiated by the interface itself. No processor instructions are needed during this transfer, which is not complete until an entire block of data has been moved.

When the disk is to be read, the disk head is first loaded, the desired location is reached, and the read

[C] Floppy disk interface. Once set in motion by processor commands, the disk speaks directly to the system's RAM memory using a cycle steal technique.



[3] Asynchronous Communications Interface Adapter (ACIA). On a single chip, this device provides a variety of program-selectable functions.



constructed of dozens of integrated circuit logic packages mounted on large boards. Development of standard Universal asynchronous receiver-transmitter (UART) chips, each replacing 20 to 25 logic packages, has considerably simplified this type of equipment. The communications interfaces described by Russo and Lippman in this article, illustrate a UART-based design.

Recently, single-chip communications interfaces have been announced and are expected to be on the market soon. These large-scale integrated circuits combine, in a single package, all of the functions needed to connect a microprocessor system to a communications link.

Motorola Semiconductor seems to have the most sophisticated of these new single-chip devices. Called the Asynchronous Communications Interface Adapter (ACIA), this device operates with the Motorola M6800 microcomputer system to provide software control of a variety of interface functions. Serial data flows from the communications link into a shift register in the ACIA's *Receive data register* section (see Fig. 3).

Here, incoming bits are assembled into bytes, to be

circuits are activated somewhere within the 16-byte synchronizing pattern that prefaces the desired block of 252 data bytes. The *Sync Pattern Detector* then searches for the synchronizing pattern to determine the exact moment when the first data bit in the block is about to move into position under the read/write head. This mode of operation allows large design tolerances on both the location of the head and on the timing for reading and writing.

The *Read/Write Logic* frames the data bytes while the disk is read. That is, it defines the beginning and end of each byte as it appears in the serial stream of one bit read pulses from the disk. When it is time to transfer a byte of data to the microcomputer RAM, the *Control Logic* raises one of the cycle steal lines, and the byte is then placed on the system data bus. When data is being recorded on the disk, the *Read/Write Logic* takes in each data byte and converts it into a stream of one bit write pulses to the disk head.

At the present time, the *Read/Write Logic* adds a parity bit after every 8 bits it sends to the disk. When the disk is read, these parity bits are checked to get indications of errors that may have occurred during the disk read operation. Since disk errors tend to occur in bunches, or bursts, future system design plans call for the use of burst error-detecting coding techniques rather than parity bits. The savings in available storage space should be substantial.

In addition to coordinating all the other disk interface operations, the *Control Logic* sets the external flags that notify the processor about disk conditions. For example, a flag is raised when the interface finishes transferring a complete block of data to the RAM, or when a complete block of data has been written on the disk. By testing this flag, the microprocessor program can decide when to change *Locate*

instructions, and initiate new *Start* instructions to read or write additional data blocks.

As a convenience feature, the disk stores a bootstrap program that can restart the entire microcomputer system from scratch after power is lost, or after some other unanticipated condition puts the system out of commission. Any other program residing on the disk can then be loaded using this bootstrap program, eliminating the need for auxiliary program load devices such as cassettes or paper tape, and greatly simplifying system regeneration after a crash.

Less vital to the system than the floppy disk, but still interesting from an interfacing viewpoint, is the TV display.

Talking to a TV display

The TV can display text indicating communication-link fault conditions, and other system status conditions. But experience has shown that its most useful function is to display memory patterns; bit patterns on the data bus, and other information for diagnosis, test, and maintenance of the system.

A standard, unmodified TV set is used for the display which is refreshed from 128 bytes of RAM memory. This storage space is dedicated to the display, and these data provide 1024 dots for the display. Like the floppy disk, the TV display uses the direct memory access (cycle-stealing) capability of the system.

Every 60th of a second, the TV interface interrupts the processor and asks for new information. The interrupt routine then points to the beginning of the 128 bytes of RAM memory that contain the TV display data. These data are then sent to the TV interface on a cycle-stealing basis.

Since only one peripheral device can be served at a time by the direct memory access capability, the disk

sent to the microprocessor on the system data bus. Outgoing data moves from the system data bus to the *Transmit data register*, where it is shifted out as a stream of serial bits—with necessary added trimmings, such as start bits and parity bits—onto the communications link.

Operation of the interface is set up by software, in the form of a single byte of control information stored in the *Control register*. Among the communications parameters controlled by the contents of this register are: the word length, parity (even or odd), and number of stop bits for each transmitted or received character.

It is interesting to note that Paul Russo and Michael Lippman (see companion article below) also made programmable parameters a feature of their communication system interfaces. In fact, after completing their equipment, they felt that their floppy disk interface would also have benefitted from the use of programmable parameters.

They found that this approach is preferable, in most applications, to one that requires manual hardware modification, whether this consists of logic modification or simple strap selection. For their disk, they felt

and the TV cannot operate simultaneously. The disk, vital to the main communications function of the system, is given absolute priority over the TV. The result is that when disk and TV needs conflict, the TV display may flicker or show a reset pattern for up to about 1/2 second.

The lowest-priority peripheral in the system is the manual keyboard used to enter data bytes (in the hexadecimal code internally used by the system) into the RAM memory. This provides a means to debug and modify programs—for example, to change track and sector numbers manually for floppy disk operations.

Paul M. Russo (M) joined RCA Laboratories, Princeton, N.J., in September 1970. There, he has done research in the areas of computer architecture, program behavior, computer system performance evaluation, microprocessors, and data communications. During the 1969-70 academic year, he taught circuit theory and circuit optimization at the University of California, Berkeley. Dr. Russo was born in Plevlje, Yugoslavia, in 1943. He received the B.Eng. degree in Engineering Physics from McGill University in 1965, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley, in 1966 and 1970. He is a member of ACM, Eta Kappa Nu, and Sigma Xi.

Michael D. Lippman (M) has been a member of the Technical Staff of the RCA Laboratories, Princeton, N.J., since 1966. He has done research on magnetic recording, vapor transport processes, and graphics data compression. He is currently engaged in the development and design of microprocessor-based data communications systems. Mr. Lippman is a member of Tau Beta Pi and Eta Kappa Nu.

parameters such as parity, block length, character length, sector size, and head stepping-time could all have been made program-selectable. The interface would then have been capable of controlling a multitude of different disk drives with only minor modifications to its hardware.

The ACIA contains its own clock, to time incoming and outgoing data, and the rate of this clock is set by control register bits. Here, control is limited to the basic clock rate and rates of 1/4 and 1/8 of the basic rate. The control register contents also determine whether or not an interrupt will be generated when the receive data register is ready to communicate with the microprocessor. Finally, the control register provides for optional transmission of a break level (space) on the communications link, sets the level of request-to-send signals for controlling a communications modem, and enables or disables a ready-to-transmit interrupt to the microprocessor.

The *Status register* stores the flags that notify the microprocessor of important conditions at the interface. These include indications that data has been received from the communications link, and that the transmitter is ready for data from the microprocessor, as well as such error indications as overrun (data coming in faster than it is being read) and parity error.

The ACIA can be operated on a polling basis, in which case the microcomputer program checks status flags and initiates all transfers of data to and from the interface. It can also be operated on an interrupt basis. Interrupts to the microprocessor are generated when the *Receive data register* contains a full byte of data for the processor, and also when the presence of a carrier is first detected on the communications link.

A second one-chip communications interface is the Telecommunications Data Interface (TDI) designed for use with Rockwell International's PPS microprocessor systems. Like the ACIA, the TDI accepts serial bits from a communications link, converts them into bytes for the microprocessor, and vice-versa, while taking care of formatting, parity, and other communications housekeeping chores.

A unique feature of the TDI is the inclusion of a full modem on the same chip as the interface circuitry. The 1200-b/s modem is designed to drive a telephone line through an operational amplifier.

The TDI generates interrupts when the transmitter register is empty and when the receiver register is full. These must be followed by microprocessor instructions to test the source of the interrupt. From one to eight characters may be transmitted or received within a single pair of start and stop bits, allowing very flexible formats.

Parameters like bit-rates, parity, and word length are set by wired-in circuit straps and cannot be changed—as they are in the ACIC—by program instructions.

Reprints of this article (No. X74-091) are available at \$1.50 for the first copy and \$0.50 for each additional copy. Please send remittance and request, stating article number, to IEEE, 345 E. 47 St., New York, N.Y. 10017, Attn: SPSU. (Reprints are available up to 12 months from date of publication.)

BITS/SEC VS BAUD

BAUD : #TIMES LINE CONDITION CHANGES
PER SECOND

ASYNC : BPS = BAUD

SYNC : BPS = N X BAUD

$N = 1, 2, 4, 8, \dots$

<u>EX.</u>	PHASE ϕ°	$\phi\phi$	IF BAUD = 50 BPS = 100
	90°	$\phi 1$	
	180°	1ϕ	
	270°	$1 1$	

ASCII

START BIT
8 DATA BITS
STOP BIT

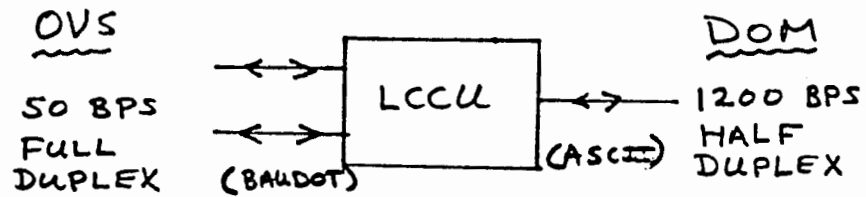
1200 BPS
= 120 CPS

BAUDOT

START BITS (1, 1.5, 2)
5 DATA BITS
STOP BIT

50 BPS =
7.1, 6.7, 6.3 CPS

FINAL



$$\begin{aligned} \text{PEAK TRAFFIC} &= \overbrace{2 \times 2 \times 7}^{\text{OVS}} \\ &+ \underbrace{120}_{\text{DOM}} = 148 \text{ CHAR/SEC} \end{aligned}$$

MOST LCCU'S < 100 CHAR/SEC.

2-3

FINAL

FLOPPY DISCS


- SEEK TIME
- LATENCY
- DATA TRANSFER TIME

2-4

SEEK TIME = TIME FOR HEAD
TO REACH DESIRED TRACK

$$S = (\# \text{ TRACKS}) \times \text{STEP TIME (P)} \\ + \text{STEP. STAB. DLY (D)}$$

$$S = N \times P + D$$


 # TRACKS

I-5

EX. IBM DISKETTE P=10MS
D=10MS

$$\begin{array}{l} \text{WORST CASE: } S = 76 \times 10 + 10 \\ \text{SEEK} \qquad \qquad \qquad = 770 \text{ MS} \end{array}$$

$$\begin{array}{l} \text{AVERAGE : } S = 38 \times 10 + 10 \\ \text{SEEK} \qquad \qquad \qquad = 390 \text{ MS} \end{array}$$

LATENCY (L) = TIME TO
ACCESS BLOCK ON TRACK

$L \propto \text{RPM OF DISC}$

FINAL

EX. IBM DISKETTE (360 RPM)

WORST CASE: $L = 166.7 \text{ MS}$

$$\begin{aligned}\text{AVERAGE: } L &= \frac{1}{2} \times 166.7 \\ &= 83.4 \text{ MS}\end{aligned}$$

FINAL

DATA TRANSFER TIME (X)

= TIME TO TRANSFER A
BLOCK OF DATA

FOR IBM DISKETTE (31.25
KBYTES/SEC)

$$X = \frac{\text{BLOCK LENGTH}}{31.25} \text{ MS}$$

EX. 128, 256 BYTE BLOCKS

$$X_{128} = \frac{128}{31.25} = 4.1 \text{ MS}$$

$$X_{256} = \frac{256}{31.25} = 8.2 \text{ MS}$$

FINAL

$$\underline{\text{ACCESS TIME}} = \text{SEEK} + \text{LATENCY}$$

$$\underline{\text{DISC THRU PUT}} = \frac{\text{BLOCK SIZE}}{\text{ACCESS} + X}$$

~~FINAL~~

WORSE CASE THRU PUT

$$= \frac{\text{BLOCK SIZE}}{\text{MAX. ACCESS} + X}$$

EX. IBM DISKETTE , P = 10MS

$$\frac{\text{WORSE CASE}}{\text{THRU PUT}} = \frac{\text{BLOCK SIZE}}{0.760 + 0.167 + X}$$

$$(128 \text{ BYTES / BLK}) = \frac{128}{0.927 + 0.004} = 137$$

BYTES/SEC

$$(256 \text{ BYTES / BLK}) = \frac{256}{0.927 + 0.008} = 274$$

I-12

FINAL

NOTE: AVG. THRU PUT \approx

2 X W.C. THRU PUT

(128 BYTES) AVG. = 274 BYTES/SEC

(256 BYTES) AVG. = 548 BYTES/SEC

IN DATA COMM. SYSTEMS,
STORE 1 CHAR. PER BYTE

•• UPPER BOUND ON

WORST CASE SYSTEM THRUPT
= 137 CHAR/SEC (128 B/B)
= 274 CHAR/SEC (256 B/B)
= 548 CHAR/SEC (512 B/B)

* RANDOM ACCESS MODE

I-14

FINAL

WHY NOT LARGE BLOCK SIZE ?

1 MSG PER BLOCK IS IDEAL

IF MSG LENGTH \ll BLOCK SIZE

\Rightarrow INEFFICIENT USE OF
DISC STORAGE

\Rightarrow "USEFUL" THRUPT
UNCHANGED

I-15

RECALL: ALL CALCULATIONS
ASSUMED $P = 10 \text{ MS.}$!

WORSE CASE
THRU PUT
(128 BYTES/BLOCK)

$P = 6 \text{ MS}$ $P = 10 \text{ MS}$

202 137

↖ CHAR/SEC ↗

✓
FINAL

IMPROVE THRU PUT

1. LARGER BLOCKS
2. SMALLER STEPPING TIME
3. DISC MANAGMENT
 - RELATED INFO. ON ADJ. TRACKS
 - MULT. MSG. PER BLOCK
 - OPTIMAL I/O SEQUENCE WHEN TRANSFERRING MULTIPLE BLOCKS

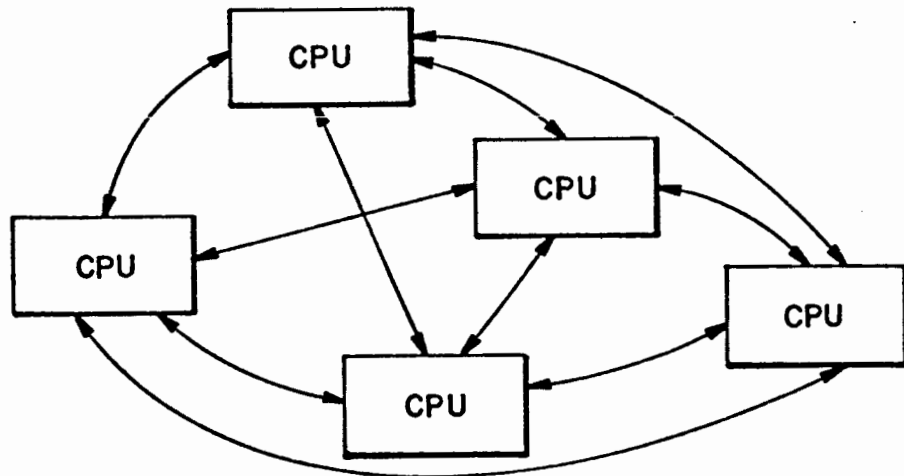
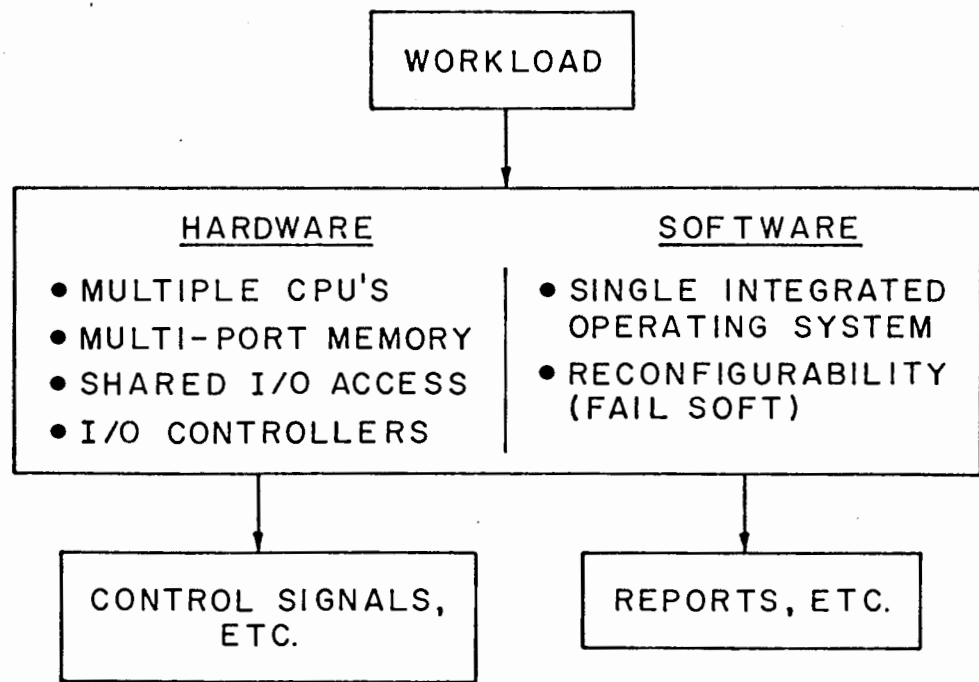
✓
FINAL

WITH $T = 5 \mu\text{SEC}$, 128 BYTES/BLOCK

WE HAVE $\frac{100,000}{137} = 730 \text{ INST./CHAR}$

MORE THAN ENOUGH FOR CHAR.
PROCESSING

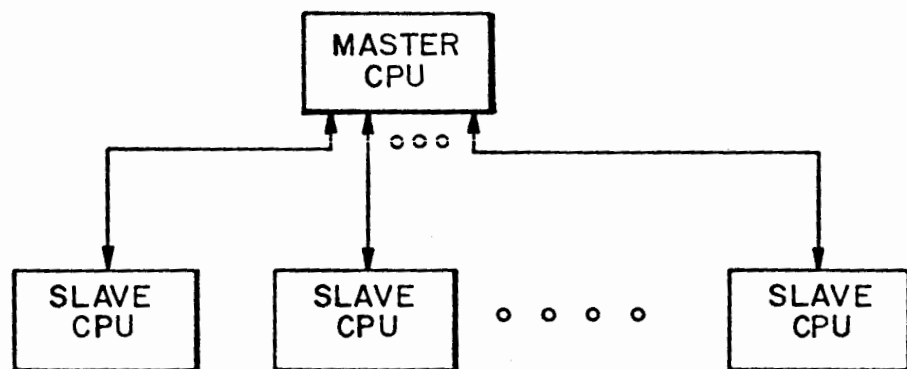
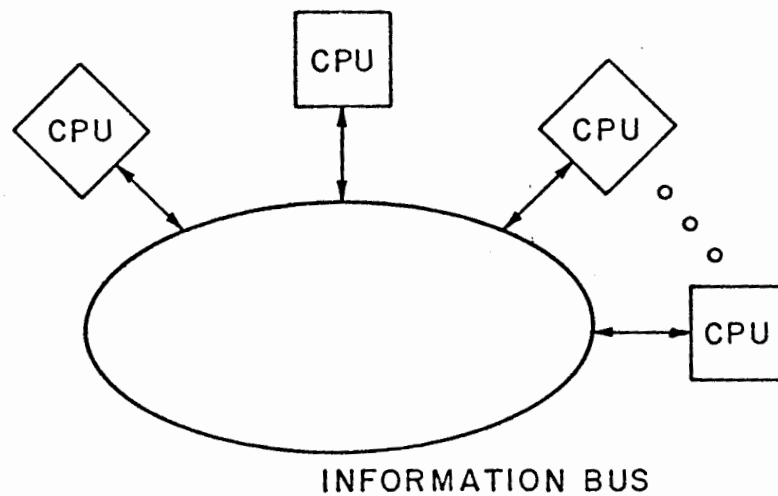
⇒ FD IS BOTTLENECK !



FINAL

DRAWBACK

ALL CPU'S MUST SUPPORT
COMPATIBLE INTER-PROCESSOR
INTERFACES AND INSTRUCTIONS



MASTER - SLAVE ORGANIZATION

I-4, FINAL

ADVANTAGE

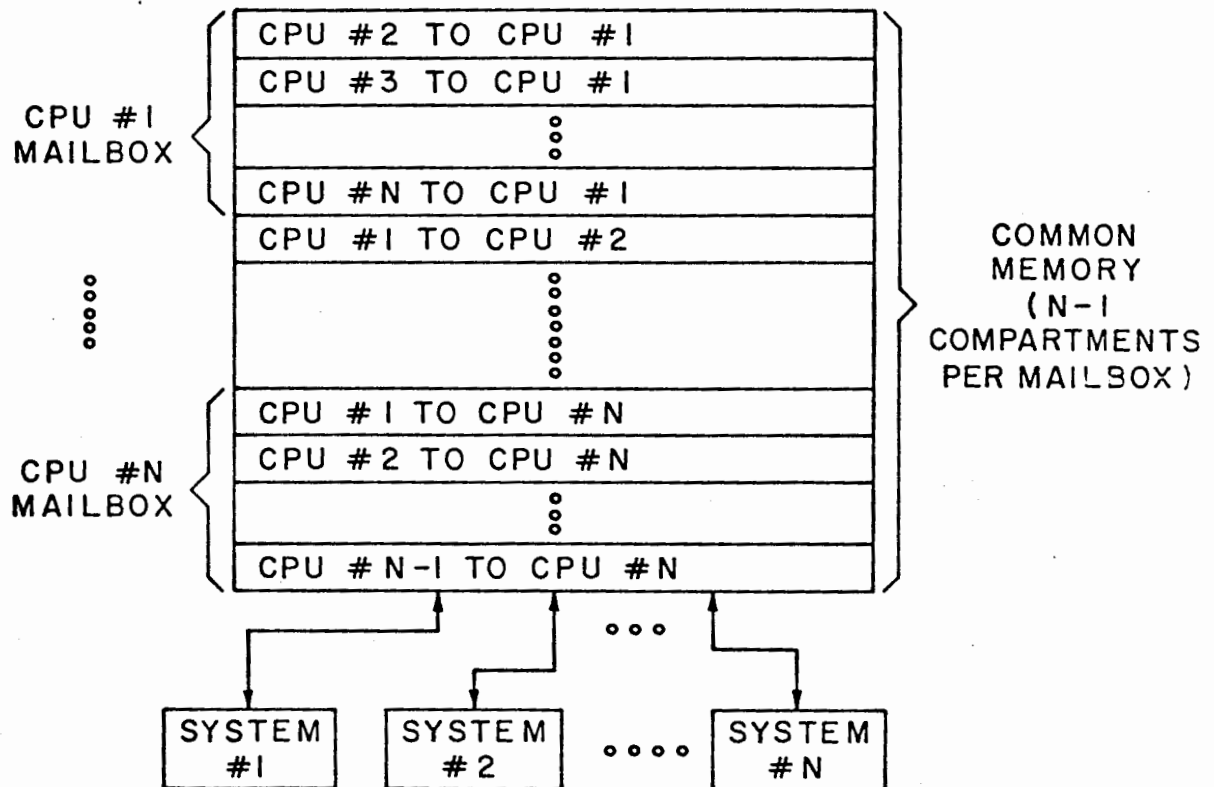
EACH INTERPROCESSOR INTERFACE
CAN BE TAILORED TO CONTROLLER.
RESULTS IN EFFICIENT USE OF
I/O CODES.

NOTE : FOR LOCAL INTELLIGENCE,
NEED PROCESSING (CPU) AND
MEMORY (RAM).

THUS, EACH SLAVE CPU HAS
OWN RAM.

- NO LIMIT ON # CPU'S (MM
CONTENTION)
- SIMPLER LOGIC

II-5

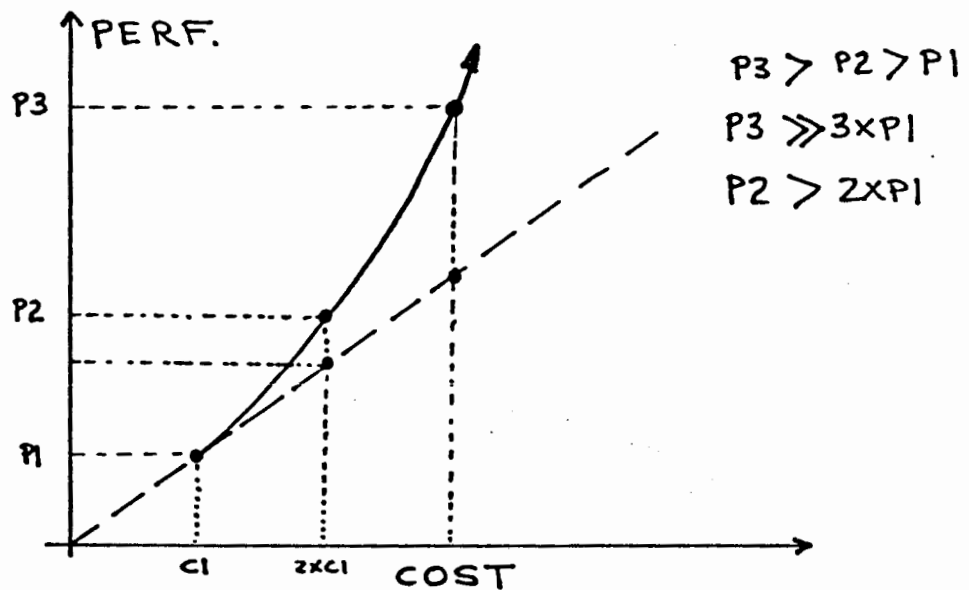


WHEN MULTI-CPU'S ?

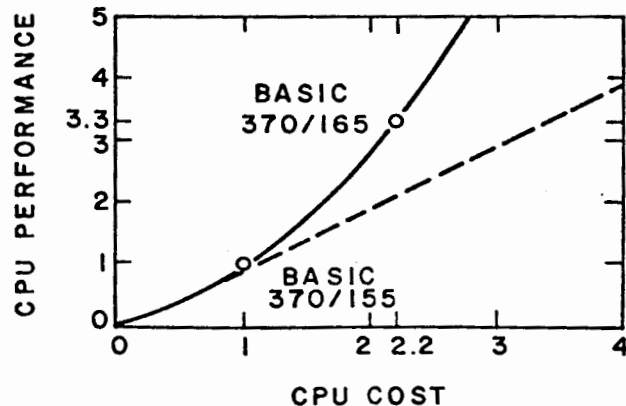
- NEED MORE CPU POWER
AND TASKS CAN BE
EFFECTIVELY PARTITIONED.

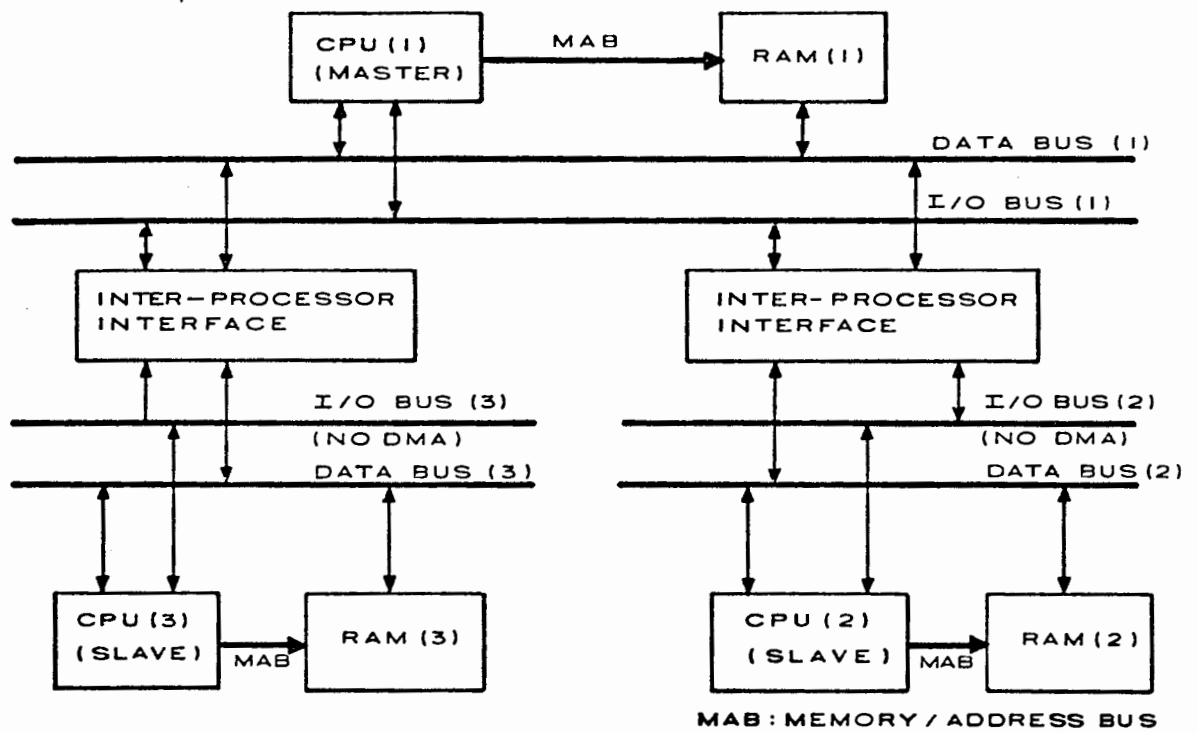
MULTI
FINAL

MICROPROCESSOR PERF. vs COST

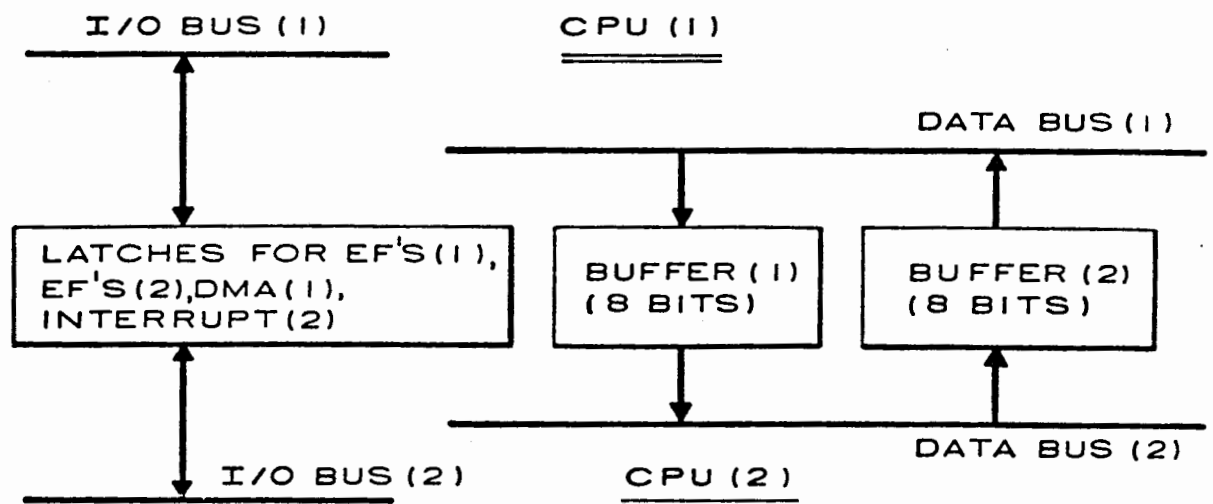


A-7

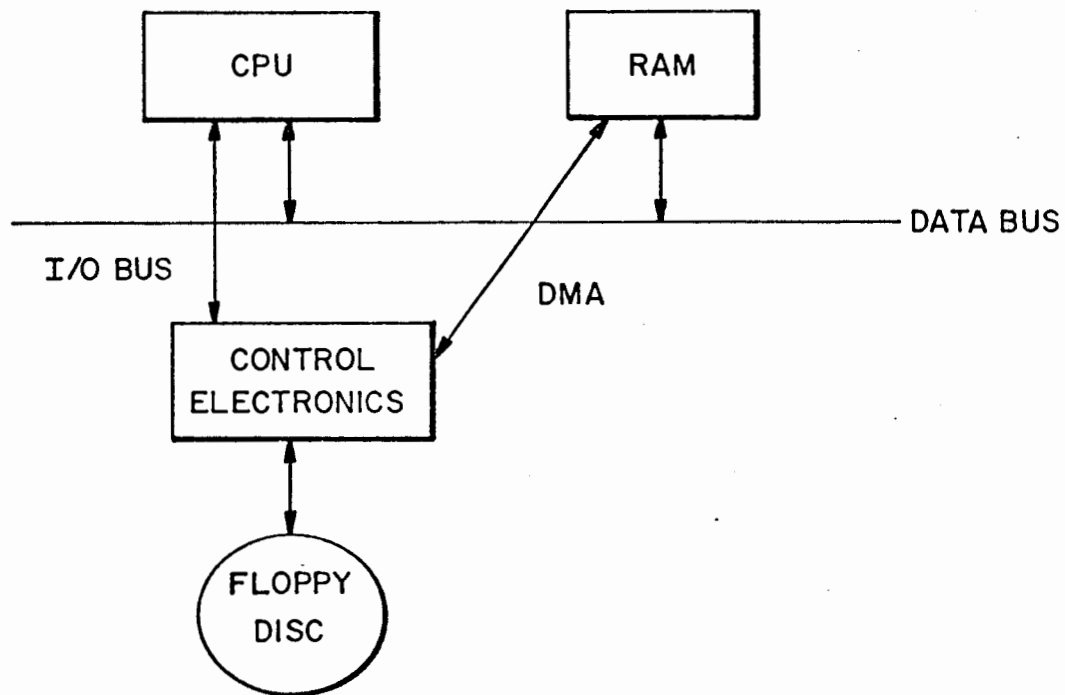




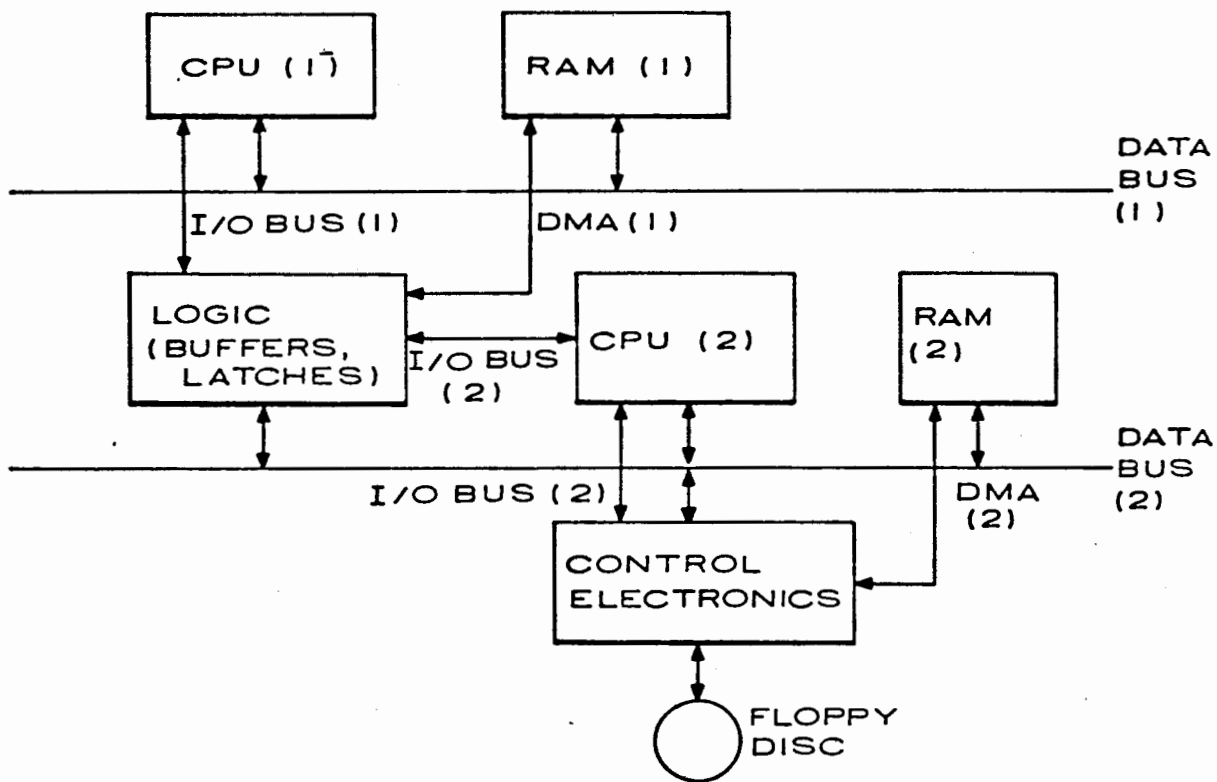
MULTI-MICROPROCESSOR HARDWARE



INTER-PROCESSOR INTERFACE



SINGLE CPU SYSTEM



TWO CPU SYSTEM

MICRO USES

- MINI REPLACEMENTS
- HARDWIRED LOGIC REPLACEMENT (LCCU)
- NEW APPLICATIONS (FRED)

FINAL

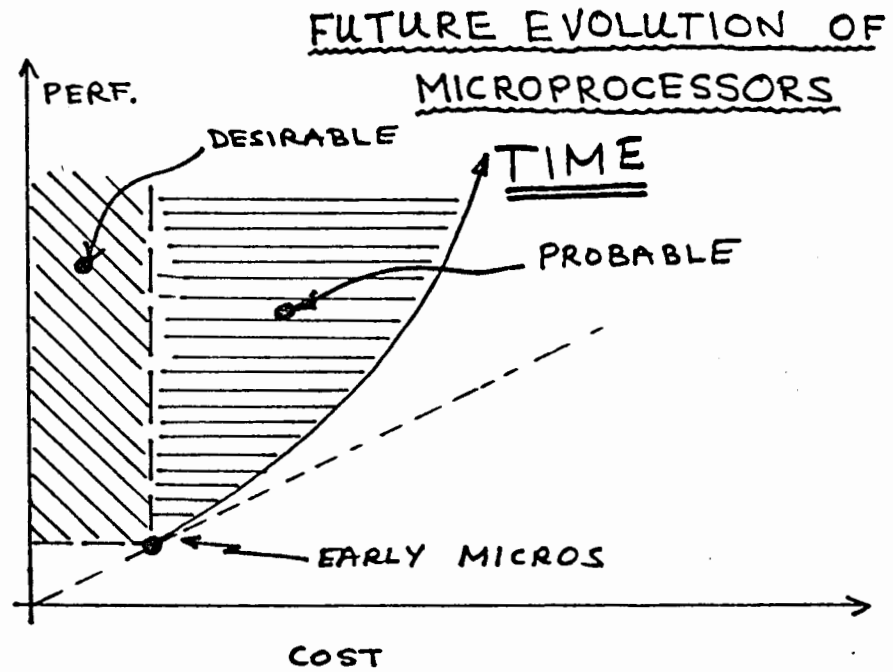
MICROPROCESSOR APPLICATIONS

- DEDICATED CONTROLLERS
 - INTELLIGENT TERMINALS
 - DATA COMMUNICATIONS
 - PROCESS CONTROL
 - NUMERICAL CONTROL
 - INTELLIGENT PERIPHERALS
 - INSTRUMENTS
 - AUTOMOTIVE APPLICATION

MICROPROCESSOR APPLICATIONS

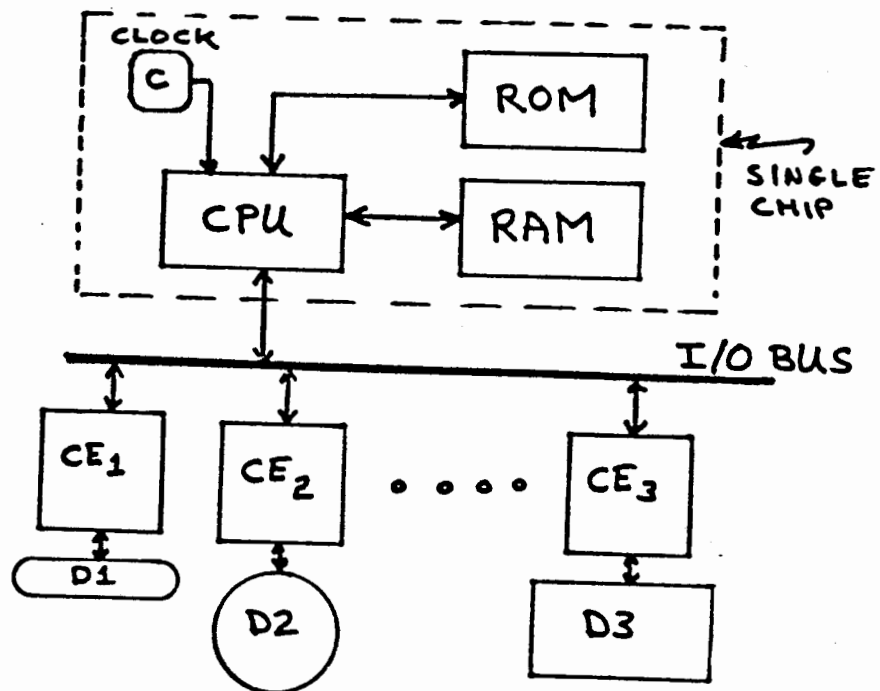
- STAND ALONE MICROCOMPUTER SYSTEMS
 - HOME
 - SCHOOL
 - ARCADE

FINAL



II-15

FINAL



ADVANTAGES

- HIGHER RELIABILITY (FEWER PARTS)
- LOWER COST
- MORE PINS FOR I/O INTERFACE
- EASIER TO USE

⇒ WOULD OPEN MORE NEW APPLICATIONS
THAN FASTER CPU'S & RAMS WITH
IMPROVED PRICE/PERF.

CCD'S

I-17

- IMAGERS
- ANALOG SIGNAL PROCESSING
- BLOCK-ORIENTED MEMORIES

FINAL

COURSE SUMMARY

- WHAT ARE MICROPROCESSORS
- COSMAC (IN DETAIL)
- ONE & TWO LEVEL I/O STRUCTURES
- APPLICATIONS
 - FRED
 - DATA COMM.
- FLOPPY DISCS
- MULTI-MICRO STRUCTURES
- FUTURE — MORE LSI, CCD'S, ETC