Field Guide to Computers, Their Habits & Habitats
Part I: The Nature of the Beast

LRL mathematician Jim Baker, author of the series which begins here, is currently acting head of Berkeley's Mathematics and Computing Group. Baker majored in mathematics at UCLA and Pomona College, later did graduate work at UC. In 1952, he joined the Laboratory to work on applications of computers to experimental and theoretical physics problems. Since 1960, he has assisted Kent Curtis in the administration of the Mathematics and Computing Group. He was named acting group leader in August, 1963, when Curtis left for a year's assignment with the AEC in Washington.

Back in 1940—which is not so very long ago as the history of science is measured—an LRL research scientist with a problem in mathematics on his mind could take his questions to any one of the five computing machines which the UC physics department proudly maintained and made available to the faculty. Of course, one couldn't be sure of getting an electric machine (there were, after all, only two of those in the whole department); most likely, our scientist would settle for a hand-operated model and count himself lucky that he wasn't living back in the days when scientists did their own calculations with pencil, paper, and a "head for figures."

By 1950, this picture had changed but little. The hand-operated machines had, no doubt, been replaced by electric models, and the first generation of electronic digital computers—ENIAC and its relatives—had already appeared on the national scene. But the working scientist at LRL still did most of his calculations with the help of computing machines not very different in principle from the ones he had been using ten years earlier.

The Machines Multiply

By 1960 all this had changed. Today, this Laboratory ranks as one of the greatest concentrations of automatic computing power to be found anywhere on earth. Within the past twelve years, automatic digital computing machines and their associated systems have become central to LRL research programs, and their uses have been extended to fields as diverse as high energy physics and payroll accounting, nuclear device design and library documentation.

This is the first of a series of MAGNET articles which will explore the design and uses of computers, with particular attention to their pertinence to LRL research programs. Among the questions which we shall seek to answer are these:

1. What is an automatic digital computer?
2. How does a computer work?
3. What sorts of problems can you do with a computer?
4. What kinds of computers does the Laboratory have?
5. What will be done with computers in the future?

Definitions

The things we want to talk about in this series are called Automatic Stored-Program Digital Computing Machines. Let us start, then, with a word-by-word analysis of this rather sonorous title.

A *computing machine* is any device which allows us to deduce from certain numbers that we know, certain other numbers that we want to know.

All computing machines may be classified either as analogue machines or digital machines. *Analogue machines* compute by measuring; *digital machines* compute by counting. An example of an analogue computer is the ordinary slide rule. The operation of the slide rule depends on the principle that says that if you lay two sticks end-to-end, then the distance from the left end of the first stick to the right end of the second stick is the sum of the lengths of the sticks; slide rules work by adding and subtracting distances.

The Abacus

An example of a digital computer is the abacus. This is an ancient oriental device which allows one to compute by moving beads on wires. A typical abacus may have eight vertical wires and seven beads on each wire. The right-most wire is the units wire, the second right-most wire is the tens wire, the third right-most wire is the hundredths wire, and so-on; this is a decimal computing machine. On a given wire, the five bottom-most beads are worth one unit apiece, while the two top-most beads are worth five units apiece; this is called the biquinary encoding system. One operates the abacus by moving the beads up and down on their wire. If a bead is pushed up as far as it can go, then it counts its whole value. If it is down as far as it can go, then it counts zero. One essentially operates the abacus by counting.

Manual vs. Automatic

Digital computers may again be divided into two classes: manual and automatic. *Manual digital computers* require the presence and intervention of a human operator at each step of the calculation which is being performed; *automatic digital computers,* on the other hand, can perform a large number of calculations without any human intervention.

An example of a manual digital computing machine is an ordinary electric desk calculator. This machine requires its operator to enter each number by hand and then to press the appropriate button indicating to the machine the operation that it is to perform. An example of an automatic digital computing machine is the IBM type 602A electronic calculator; in this machine, the user indicates the sequence of arithmetic operations to be performed in advance by inserting wires in a plug board. This plug board is then mounted on the machine, input data in the form of tabulating cards are placed in a hopper belonging to the machine, the operator presses the START button, and the machine then proceeds automatically to perform a fairly lengthy series of calculations on the input data without further operator intervention.

Fixed vs. Stored Program

All automatic digital computing machines may again be subdivided into two classes: fixed and stored program.

The fixed program machines employ a fixed sequence of operations for each problem that they do. This sequence of operations is either permanently built into the hardware of the machine (such a machine is called a *single-purpose machine*) or it is specified by a medium such as a plug board or sequence of tabulating cards which the machine itself cannot alter. In a *stored program machine,* on the other hand, the sequence of operations to be performed is specified by "instructions" which are stored in a part of the machine called the memory which the machine itself can alter.

The distinction between a fixed program and stored program automatic computer is much more fuzzy than the distinction between a manual digital computer and an automatic digital computer; *the essence of this distinction is, however, that the sequence of operations in a stored program computer is much more easily altered than the sequence of operations in a fixed program computer, and that it is very much quicker and easier to insert a new sequence of instructions in a stored program computer than it is in a fixed program computer.*

An example of a fixed program automatic digital computer is the IBM 602A Electronic calculator cited above. An ex-

ample of a stored program automatic digital computer is the IBM type 1401 computer.

The sequence of "instructions" which specifies the operations that the computer is to perform is called a *Program.*

Parts of a Computer

Every automatic stored-programmed digital computer may be divided into four functional units. These units are called:

1. *The memory*
2. *The arithmetic unit*
3. *The control unit*
4. *The input-output section*

The *memory* of a digital computer is used to hold data which is being processed, intermediate results, and the instructions which tell the machine which operation to perform next.

The basic information unit in any computer is called the *word.* A word is nothing but a string of numeric digits or alphanumeric characters with an algebraic sign (+ or −) in front of it. Associated with each computer is a definite *word size;* one computer might have a word size of five decimal digits together with an algebraic sign. A typical word in such a computer might be −21376. In another computer, the word size might be 11 binary digits (we will talk about the binary number system later) and an algebraic sign. An example of a word in such a computer would be +10110010-110. A computer whose words contain decimal digits is said to be a *decimal computer.* A computer whose words contain binary digits only is said to be a *binary computer.* A computer whose words contain decimal digits and alphabetic characters is said to be an *alphanumeric computer. The important thing is that associated with each computer is exactly one word size.* For example, the IBM 650 is a decimal computer and its word consists of an algebraic sign and ten decimal digits. The IBM 7094 is a binary computer whose word size is an algebraic sign and 35 binary digits. The IBM 1401 is an alphanumeric computer whose word size is one alphanumeric character.

Computer memories are divided up into cubbyholes — much like postoffice boxes—each one of which is large enough to hold exactly one word. So that we can keep track of the cubbyholes in memory where we are storing particular pieces of data or particular instructions for the computer, we give each cubbyhole a name or label. This label is called the *address* of that cubbyhole or the address of that memory cell. The label is usually a number; so if, for example, we have a machine with a one-thousand word memory, the addresses of cells in that memory would run from 000 to 999.

By the *cycle time* of a computer memory, we mean the time that is required to retrieve one word from the memory and be ready to retrieve another word. Hence, if we have a computer whose memory has a cycle time of 10 microseconds, we will need 100 microseconds to retrieve 10 words from that memory.

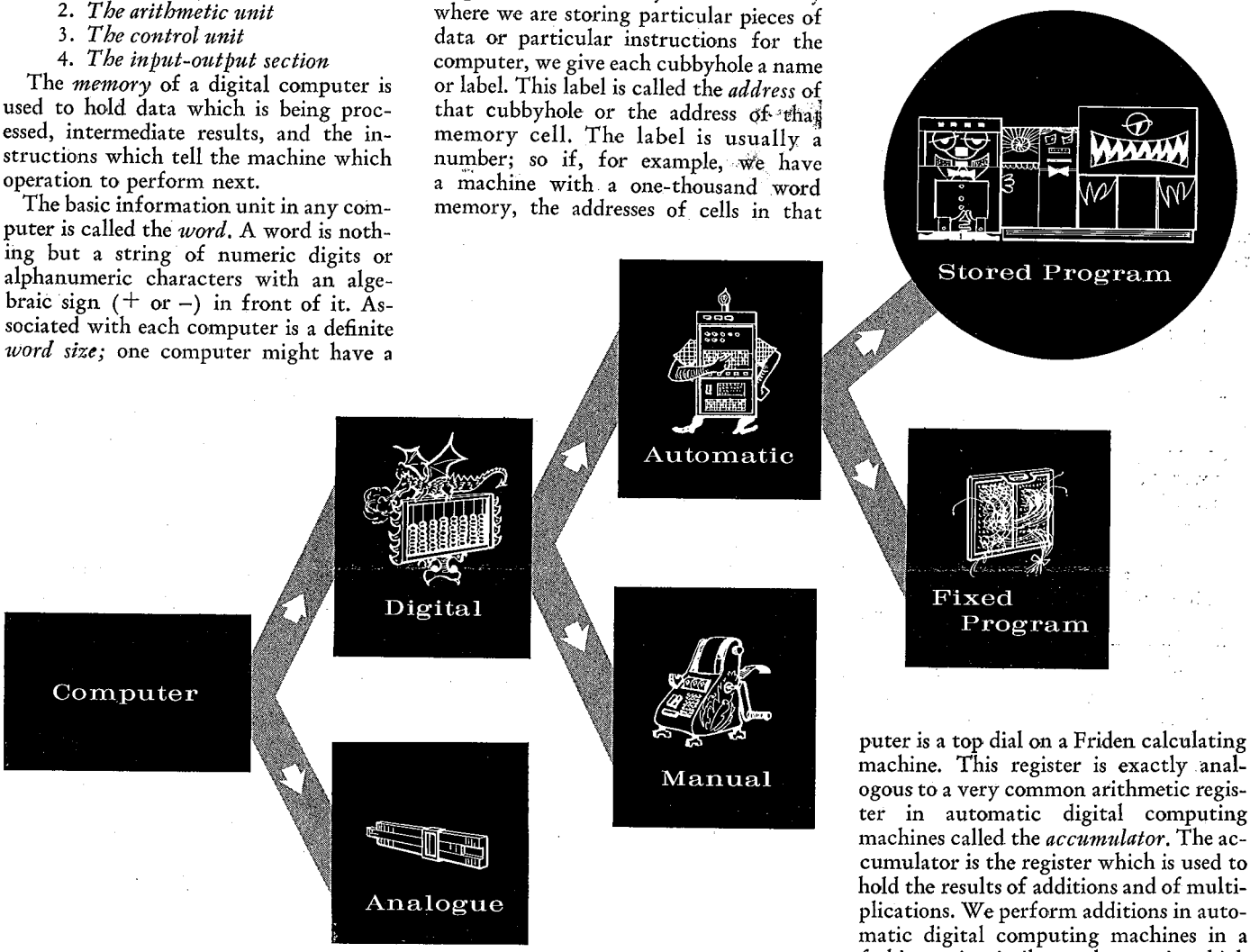
Arithmetic Unit

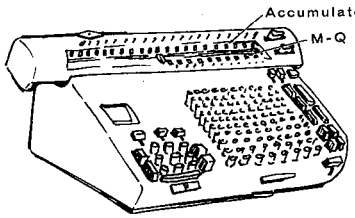
The *arithmetic unit* (pronounced with the accent on the "met") of a digital computer is the part where all of the work gets done. This unit contains hardware which enables the computer to add, subtract, multiply, divide and perform certain other logical functions. As users, we need not be concerned with the mechanisms which are used to perform these arithmetic operations; the thing that we need to be concerned with is where the results of these operations end up.

The results of arithmetic operations in many digital computers end up in devices which are called *registers.* A register is simply a bin or cell or cubbyhole in the arithmetic unit which is large enough to hold one or more words. A good example of an arithmetic register in a digital computer is a top dial on a Friden calculating machine. This register is exactly analogous to a very common arithmetic register in automatic digital computing machines called the *accumulator.* The accumulator is the register which is used to hold the results of additions and of multiplications. We perform additions in automatic digital computing machines in a fashion quite similar to the way in which we perform them on an ordinary adding machine (the Friden, for example); we first set the accumulator to zero, then we place in it a number which up until now has been stored in a cell in memory; we then add to it a second number which was perhaps stored in another cell in memory. At the conclusion of this operation, the sum of these two numbers remains in the accumulator register. Often, to save space, we refer to the accumulator as the *A register* or, starkly, as *A.*

Another arithmetic register which occurs in quite a large number of digital computers is called the *Multiplier-Quotient* register or for short, the "M-Q" register. This register, as its name implies,

Continued on Page 9





A Field Guide to Computers . . .

Continued from Page 7

is used to hold one of the factors in a multiplication and to hold the result of a division. It corresponds to the second dial from the top on the Friden.

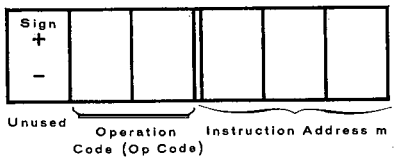
All computers have an arithmetic section, but not all computers have arithmetic registers. In some machines, the IBM 1401 for example, results of arithmetic operations are stored directly in the memory without stopping in an arithmetic register along the way.

The Control Unit

The *control unit* of an automatic digital computer is that portion of the machine that interprets the instructions that the machine is to execute and, in general, tells the machine what to do next. The control unit almost always contains two registers, called the *instruction register* and the *control counter* (IR and CC, for short). The instruction register, which is usually one word long, contains the command which is *currently being executed.* The control counter, which is just large enough to hold one address, holds the address of *the next instruction to be executed.*

It should be clear by now that the instructions or commands which tell the machine which function to perform next are nothing but machine words (which is to say, numbers of a certain length); thus, there is no effective way of distinguishing an instruction word from a data word. In fact, we occasionally inadvertently try to execute as an instruction a data word—this usually leads to absurd results. On the other hand, we often intentionally do arithmetic with instruction words as we shall see later.

An instruction word, in a digital computer whose word length was algebraic sign and five decimal digits, might look something like this:



The sign in this particular machine is not interpreted as part of the instruction word. The *operation code,* sometimes abbreviated op code, is composed of two decimal digits. This operation code tells the machine what function it is to perform next; for example, an operation code of 01 might tell the machine to add, an operation code of 02 might tell the machine to subtract, and so on. The last three digits of the instruction word in this machine are to be used for an address. We will use the letter "m" to designate this instruction address in future discussions. The *address part of the instruction* tells the machine the location of the data on which it is to perform the operation specified by the operation code. So, for example, the instruction word +01256 might mean to add (operation code 01), the number that is stored in address number 256 to the number that is already in the accumulator, and leave the result in the accumulator.

Some machines have registers called "B registers" or "index registers" in their control sections. These registers are used in a special way to modify the address parts of instructions. We will discuss them in more detail later.

Input-Output Section

The *input-output section* of an automatic computer is a very important one from the viewpoint of the user. It is this section which allows the user to communicate with the computer (this is called input) and the computer to communicate with the user (this is called output).

Input-output may be accomplished through a very wide variety of media. The simplest, least expensive, and slowest input device is a set of *switches* on the console of the computer. By throwing these switches appropriately, the operator may feed information into the computer. While almost every computer has facilities for this type of input, these facilities are seldom used except by engineers who are trying to repair the machine.

The simplest kind of output device is a set of *display lights* on the console of the machine. These lights may display the contents of the various arithmetic and control registers and may also have the capability of displaying the contents of various cells in the memory. Obviously, the machine must be stopped before we can read the contents of the lights; hence, these devices are not frequently used either.

A step ahead of console switches and lights as an input-output device is the *typewriter.* This device may be used for input by its keyboard or for output onto a piece of paper. Because of its slowness, it should not, however, be used for large volumes of either input or output.

Now we come to a class of input-output media which includes *paper tape, tabulating cards,* and *magnetic tape*—all of which make input-output a two or three step process. Consider, for example, paper tape, which is a very common medium on low-cost computers. If we wish to prepare some data for input to our computer, we may punch this data onto a paper tape using a typewriter-like device such as a Flexowriter or teletype machine. We then take this paper tape containing our data over to our computer and read it in. The important thing here is that the slow-speed human process of punching the paper tape is separated from the relatively high-speed automatic computer processes of reading paper tape and computing. The situation on the output side is very similar. The computer punches out our answers on paper tape at relatively high speed. Since we cannot read this paper tape directly, we take it over to a printing device which looks something like a typewriter, insert the paper tape into this device, and our answers are then printed out on a sheet of paper.

Paper tape and cards are generally used as input-output media on low or medium price machines. Magnetic tape, which is very much faster, is used as the basic input-output medium on all high speed computers today.

In the next article in this series, all the above material will be illustrated in a small, fictional, but realistic computer called "the SMAC" (Simple Minded Automatic Computer).

Yuval Ne'eman . . .

Continued from Page 2

study of the strong interaction), archaeology, and the philology of Middle Eastern languages. He hopes some day to take time off from physics to work on a comprehensive theory of ancient Near Eastern history, drawing on data from archaeology, the Bible, and other historical sources.

Ne'eman's success in his new profession brought him a measure of publicity in his native land—and led to what certainly must go down in history as the most unusual tribute ever tendered a physicist. A few days after the news of the omega hyperon discovery made the Israeli papers, the English-language Jerusalem *Post* carried a small, display-type advertisement on its front page. The ad (which Ne'eman clipped and kept as a reminder of how science can be all things to all men) reads as follows:

OMEGA MINUS . . .

The latest discovery which can coordinate the various theories about the structure of the atom.

MAYONNAISE PLUS . . .

(plus lemon juice) the housewife's latest discovery for improving food. Telma mayonnaise plus lemon juice adds that piquant flavor to salads, fish, meat, hard-boiled eggs, etc.

Field Guide to Computers, Their Habits & Habitats
Part II: Simple-Minded Computer

In which our Correspondent James Baker, of Berkeley Mathematics and Computing, comes Face to Face with SMAC, a dim-witted but benevolent Representative of an Alien Species.

These days, digital computers can solve calculus problems at about the level of a college sophomore; they can translate from Russian into English at perhaps the college freshman level; they can produce numerical solutions to very difficult mathematical problems.

Last month we described the four principal parts of an automatic digital computer. We saw that the computer must perform all its complicated tasks by doing a few relatively simple arithmetic and logical operations at a very high rate of speed.

This month, we are going to give an example which will illustrate all the features and functions described in last month's article. This example computer, called the SMAC (or Simple Minded Automatic Computer), is a realistic machine; it is more complicated than some computers which are presently installed at the Laboratory. If you can understand how SMAC works, then you should be able to understand how almost any digital computer works.

Type of Arithmetic

The SMAC is a *decimal machine* — which is to say that it uses the same number system that you and I use. SMAC's word length is five decimal digits and an algebraic sign. It represents negative numbers the same way that you and I represent negative numbers. For example, in SMAC the number −14 would be represented as −00014. This is an advantage over most hand calculators, which would tend to represent −14 as 9999999986 (as you can easily verify by subtracting 14 from zero on one of them). The SMAC's way of representing numbers is called the *sign and magnitude system.* The hand calculator's way of representing negative numbers is called the *tens compliment system.*

Anatomy of SMAC

Memory. SMAC has a one-thousand word memory. Each word in the memory has an address consisting of three decimal digits; thus, addresses run from 000 up to 999. Each of the one thousand memory cells is, of course, just big enough to contain one word consisting of five decimal digits and an algebraic sign.

Arithmetic Section. SMAC has a very simple arithmetic section indeed. It consists of exactly one arithmetic register: the accumulator. We will often refer to the accumulator as the A register or simply as A. SMAC's accumulator is just one word long — five decimal digits and an algebraic sign. It is the register that will hold the results of all arithmetic operations performed in SMAC.

Control Section. The control section of SMAC contains three registers. The *instruction register,* or IR, is a one-word (five decimal digits and sign) register; it holds the command which is currently being executed. The *control counter,* or CC, is a three-digit register; it contains the address in memory where the next command to be executed is located. The *index register* (which we will call the B register or B) is another three-digit register. It is used to modify the address portions of certain commands. Its use will be illustrated later.

Input-Output Section

SMAC has a paper tape reader for input and a paper tape punch for output. It can read one word in a forward direction from the paper tape which is currently in the tape reader, and then place this word in a memory location specified by the command which is being executed. Similarly, it can punch out onto the paper tape which is currently in the tape-punching mechanism one word from an address designated by the current command.

Paper tapes which are to be read into the computer must first be prepared in the proper form. Sometimes, they are punched on a Flexowriter or Teletype machine away from the computer; sometimes they have been punched by the computer itself in a previous run. When we wish to find out what information is on an output tape, that has been punched by the computer, we must take that tape over to a Flexowriter, which then prints the contents of the tape on a piece of ordinary paper.

Instruction Format

Every computer word that is brought from the memory into the instruction register is interpreted by SMAC as an instruction word. The two decimal digits at the extreme left (which we sometimes call the two most significant digits) are interpreted as an *operation code,* or OP code. This operation code tells the machine what function it is to perform next. The three digits at the extreme right of the instruction word are interpreted by the machine as an *address.* This address, in general, gives the machine the location of the data upon which the function specified by the OP code is to be performed. So, for example, if the operation code for addition is 01 and the word +01223 comes into the instruction register, the computer will add the word which is presently stored in memory location 223 to the number which is currently in the accumulator, leaving the sum in the accumulator.

The sign of the instruction word tells the computer what to do with the B register. If the sign of the current instruction word is +, then the B register is ignored. If the sign of the current instruction word is −, then the contents of the B register (a three-digit number) is subtracted from the address part of the instruction word (the three digits on the right) before the instruction is executed. We often refer to the address part of the current instruction as m.

SMAC EXECUTES A COMMAND
(Assume that the execution of a command has just been completed, then start at Step 1.)

1. C(C(CC))→IR
2. I+ C(CC)→CC
3. If the sign of IR is +, go to Step 5; if the sign of IR is −, go to Step 4.
4. Subtract C(B) from m (the rightmost 3 digits of IR) and replace m by the result.
5. Execute the instruction in IR.
6. Go to Step 1.

Figure 1

Notation

In order to be able to describe more concisely how the computer works, we must make some notational conventions. Let us agree that $C(s)$ is to mean the *contents of s* if *s* is the name of either *a memory location, an arithmetic register, or a control register;* similarly, $C(m)$ will mean the word that is currently stored in memory location m; $C(A)$ will mean the contents of the accumulator; $C(CC)$ will mean the contents of the control counter (remember that the contents of the control counter is a three-digit number).

We use the arrow (→) to indicate "goes to"; C(m)→A, for example, means that the contents of the memory register m goes to the accumulator (that is, that the contents of m will replace the current contents of the accumulator). C(A)→m means that the current contents of the accumulator will replace the current contents of m.

How the Machine Works

Each time that SMAC executes an instruction, it goes through five or six steps under the direction of the hardware in its control section. These steps are summarized succinctly in Figure 1.

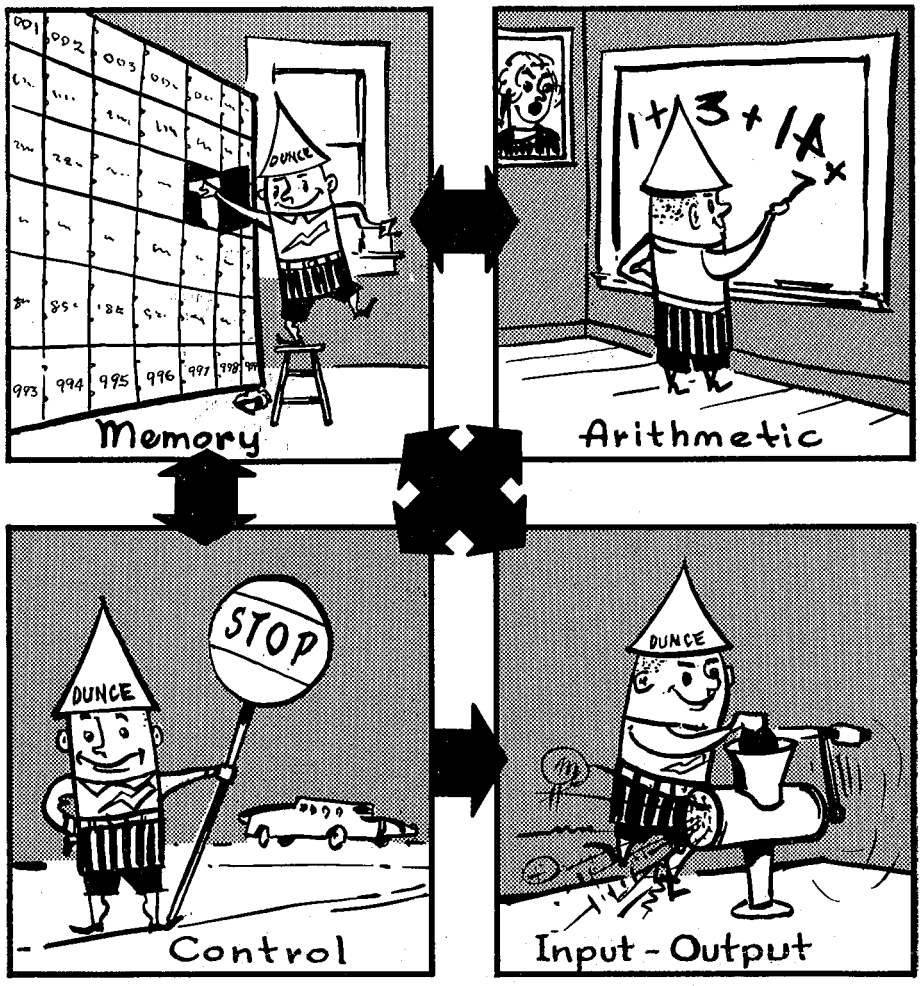
1. $C(C(CC)) \rightarrow IR$. The purpose of this step is to take the next command from the location specified by the *control counter* and load it into the *instruction register*. (The command actually tells the contents of the contents of the control counter to "go to" the instruction register.) Remember that the control counter is a three-digit register — just big enough to contain one address. So, in fact, C(CC), the contents of the control counter, is an address in SMAC's memory, and C(C(CC)) is the *contents* of that address in the memory. In fact, the contents of that address is to be used as the next instruction to be executed by SMAC.

2. $1+C(CC) \rightarrow CC$. The effect of Step 2 is to add *one* to the control counter. For example, if during Step 1 the control counter contained 173, after Step 2 it would contain 174. In performing this operation, SMAC takes successive commands from successive locations in memory.

3. *If the sign of IR is +, go to Step 5; if the sign of IR is −, go to Step* 4; and,
4. *Subtract C(B) from m (the last three digits of IR), and replace m by the result.* Steps 3 and 4 have to do with the *index register* (the B register). They say that if the sign of the next instruction is minus, we should subtract the contents of the B register from the address part of the next instruction before executing that instruction. On the other hand, if the sign of the next instruction is positive, then we should simply ignore the B register.

(It may be somewhat reassuring for you to know that you are not yet supposed to understand the purpose of the B register; this will be explained when we start to do a programming example.)

5. *Execute the instruction in IR.* Step 5 says that we are now to execute the instruction as it currently appears in the instruction register. At this stage of the game, the memory address portion of the instruction, m, may have been altered by having the contents of the B register subtracted from it. However, we will continue to call this altered address m.



SMAC
(Simple Minded Automatic Computer)

After we have finished executing the current instruction, we return to Step 1 and start the whole process over again.

Command Repertoire

In Figure 2, the SMAC's repertoire of instructions is listed in concise form. The first column gives a three-letter mnemonic abbreviation which is supposed to remind us what each operation code does. The second column lists the actual operation codes; these codes are the ones that will actually appear in the two left-hand digits of instruction words. The third column in Figure 2 lists exactly what each operation code tells SMAC to do.

The OP Codes

The first operation in the table has operation code 00. Its mnemonic is CLA, which stands for *clear and add*. The effect of this instruction is that the contents of the address designated by m, (the address portion of this instruction), should be placed in the accumulator. Whatever was in the accumulator before is lost. However, the contents of the memory address number m remains un-

Continued on Page 7

SMAC'S INSTRUCTION REPERTOIRE

Mnemonic	OP Code	Explanation
CLA	00	C(m)→A. Clear and Add.
ADD	01	C(A)+C(m)→A. Add.
SUB	02	C(A)−C(m)→A. Subtract.
STO	03	C(A)→m. Store.
TRA	04	m→CC. Unconditional Transfer.
TMI	05	If C(A) <0, then m→CC; otherwise proceed normally. Transfer on minus.
TZE	06	If C(A)=0, then m→CC; otherwise proceed normally. Transfer on zero.
LXA	07	Rightmost 3 digits of C(m)→B. Load Index from Address.
SXA	08	C(B)→Three rightmost digits of m. Store index in address.
TIX	09	C(B)−1→B; if C(B) ≠0, then m→CC; otherwise, proceed normally. Transfer on index.
INP	10	Read the next word from the tape in the paper tape reader into address m.
OUT	11	Punch onto paper tape the word in address m.
HLT	12	STOP. Halt.

Figure 2

SMAC: Simple-Minded Automatic Computer . . . Continued from Page 5

changed. It is generally true that the only way in which we can destroy the contents of a register is by storing something on top of it.

The next two operation codes, 01 (ADD) and 02 (SUB), are the codes for *add* and *subtract*, respectively. They cause the contents of m to be added to or subtracted from the contents of A and the resulting sum or difference to be left in A. Again, the contents of m remains unchanged.

Operation Code 03 (STO) is the *store* operation. The execution of an instruction word which has this operation code causes the contents of the accumulator to be placed in address number m. The previous contents of m are lost and the contents of the accumulator remain unchanged.

Transfer of Control

Operation Code 04 (TRA) is the *unconditional transfer of control* operation. In this operation, the number m (itself a three-digit number) replaces the current contents of the control counter. Notice that this operation is different from all the preceding operations. In the others, we were always doing something with the *contents* of m. In the transfer of control operation, however, the contents of m are undisturbed. Instead, we are simply instructing SMAC that *its next command is to be picked up from address number m* instead of from the usual spot.

Operation Code Numbers 05 (TMI) and 06 (TZE) are the *transfer on minus* and *transfer on zero* operations, respectively. They tell SMAC to go to location m for its next command in the event that the accumulator is either negative (TMI) or zero (TZE).

Operation Code 07 (LXA) is the *load index from address* operation code. This code tells the computer to take the three right-hand digits from the memory cell whose address is m, and place those digits into the B register. The contents of m is unaffected by this operation.

Store Index

Operation Code 08 (SXA) is the *store index in address* operation. It is the reverse of the LXA operation above. Its effect is to place the contents of the B register into the three right-hand digits of the memory cell whose address is m. The left-hand two digits and the sign of the cell number m are unaffected by this operation.

Operation Code 09 (TIX) tells the computer to perform the *transfer on index* operation. This is a very complicated operation; first of all, the computer subtracts 1 from the current contents of the B register. Then, if the new contents of the B register is different from zero, the computer takes its next command from location m. However, if the new contents of B is equal to zero the computer goes ahead and takes its next command from the normal location designated by the control counter.

Operation Codes 10 (INP) and 11 (OUT) are the input and output operations. They tell the computer to read the next word from paper tape into location m or to punch the next word from location m onto paper tape, respectively.

The final operation code is number 12 (HLT). It is the code for Halt. When SMAC executes this command, it stops computing and turns on a light on its console which says PROGRAM STOP.

Getting Started

A question that very often worries beginners in computing is "How does the machine get started?" On the SMAC the starting operation is accomplished through the use of a very convenient button which is labeled "LOAD PAPER TAPE." The depression of this button causes the following sequence of actions to take place:

1. The computer reads the next four words from the paper tape in the tape reader into memory cells 000, 001, 002, and 003.
2. The control counter is set to 000.
3. The computer proceeds to operate automatically, taking its first command (of course) from location 000. Location 000 contains one of the four words that have just been read in from paper tape.

At the beginning of his paper tape, the programmer will have punched a "Loader" program which will load itself into the computer's memory and then in turn load his program in. In next month's article we will see an example of such a loader.

Next month, we will write down three programs to do the same simple calculation. We will see how we can economize on the use of memory by using different sections of our program a number of times, how a program can modify its own commands, and (at last) what useful purpose the B register serves.

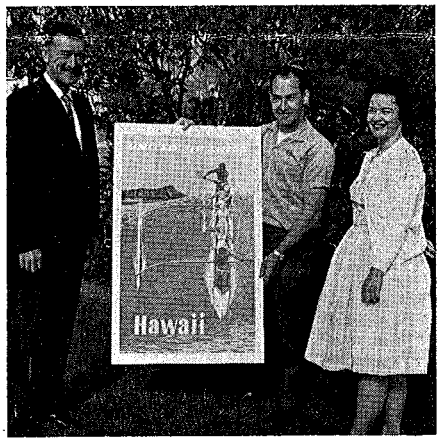
Lucky Blood Donor Wins Hawaii Trip

Livermore Mechanical Technician David Wood was the lucky winner of a 12-day, all-expense-paid trip to Hawaii offered by RLRA as a door prize at last month's Blood Drive at Livermore.

Wood's name was chosen by lot from among those who donated blood at the Building 130 cafeteria on April 22.

A record total of 334 donors turned out for the drive, with 249 pints of blood going to the LRL Blood Bank and the remainder to special funds.

The happy winner Dave Wood is shown at right, flanked by RLRA representative Roy Moffitt and Livermore Blood Drive coordinator Pat Jordan.



Materials Symposium to Meet at UC

LRL's Inorganic Materials Division, headed by Leo Brewer, will play host to participants in the Second International Materials Symposium, to be held at UC on June 15-18.

Keynote speaker at the Symposium will be Professor J. Friedel, of the University of Paris, and the principal topic will be progress made to date in understanding the basic strengthening mechanisms in metallic, ceramic, and polymeric materials.

The entire proceedings will be published in book form shortly after the symposium and will be part of a series, sponsored by the Inorganic Materials Division, on the subject of materials research.

LRL and UC scientists participating in the symposium include LRL Director Edwin McMillan, Inorganic Materials Division leader Leo Brewer, and scientists Earl Parker, Gareth Thomas, Victor Zackay, Alan Searcy, Lies Finnie, Arthur Kip, John Dorn, and Jack Mitchell.

The Inorganic Materials Division is financially supported as a fundamental research organization of the AEC, but operates as an interdisciplinary research laboratory for portions of the faculties of several UC academic departments.