KLEIN

NATIONAL BUREAU OF STANDARDS REPORT

1984

PROGRAMMING AND CODING HANDBOOK FOR SEAC

by

Joseph H. Levin

NBS

U. S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

U. S. DEPARTMENT OF COMMERCE

Sinclair Weeks, *Secretary*

NATIONAL BUREAU OF STANDARDS

A. V. Astin, *Director*

THE NATIONAL BUREAU OF STANDARDS

The scope of activities of the National Bureau of Standards is suggested in the following listing of the divisions and sections engaged in technical work. In general, each section is engaged in specialized research, development, and engineering in the field indicated by its title. A brief description of the activities, and of the resultant reports and publications, appears on the inside of the back cover of this report.

Electricity. Resistance and Reactance Measurements. Electrical Instruments. Magnetic Measurements. Electrochemistry.

Optics and Metrology. Photometry and Colorimetry. Optical Instruments. Photographic Technology. Length. Engineering Metrology.

Heat and Power. Temperature Measurements. Thermodynamics. Cryogenic Physics. Engines and Lubrication. Engine Fuels. Cryogenic Engineering.

Atomic and Radiation Physics. Spectroscopy. Radiometry. Mass Spectrometry. Solid State Physics. Electron Physics. Atomic Physics. Neutron Measurements. Infrared Spectroscopy. Nuclear Physics. Radioactivity. X-Ray. Betatron. Nucleonic Instrumentation. Radiological Equipment. Atomic Energy Commission Radiation Instruments Branch.

Chemistry. Organic Coatings. Surface Chemistry. Organic Chemistry. Analytical Chemistry. Inorganic Chemistry. Electrodeposition. Gas Chemistry. Physical Chemistry. Thermochemistry. Spectrochemistry. Pure Substances.

Mechanics. Sound. Mechanical Instruments. Fluid Mechanics. Engineering Mechanics. Mass and Scale. Capacity, Density, and Fluid Meters. Combustion Control.

Organic and Fibrous Materials. Rubber. Textiles. Paper. Leather. Testing and Specifications. Polymer Structure. Organic Plastics. Dental Research.

Metallurgy. Thermal Metallurgy. Chemical Metallurgy. Mechanical Metallurgy. Corrosion.

Mineral Products. Porcelain and Pottery. Glass. Refractories. Enameled Metals. Concreting Materials. Constitution and Microstructure.

Building Technology. Structural Engineering. Fire Protection. Heating and Air Conditioning. Floor, Roof, and Wall Coverings. Codes and Specifications.

Applied Mathematics. Numerical Analysis. Computation. Statistical Engineering.

Electronics. Engineering Electronics. Electron Tubes. Electronic Computers. Electronic Instrumentation. Process Technology.

Radio Propagation. Upper Atmosphere Research. Ionospheric Research. Regular Propagation Services. Frequency Utilization Research. Tropospheric Propagation Research. High Frequency Standards. Microwave Standards.

● Office of Basic Instrumentation

● Office of Weights and Measures.

NATIONAL BUREAU OF STANDARDS REPORT

NBS PROJECT		NBS REPORT
1102-50-5126	September 30, 1952	1984

PROGRAMMING AND CODING HANDBOOK FOR SEAC

by

Joseph H. Levin

Contents

Introduction

The construction of SEAC is one aspect of the National Bureau of Standards' digital computer program. Known in the pre-operation phases of its development as the NBS Interim Computer [1], this machine was constructed by the Electronics Division for the Applied Mathematics Division with a major part of the financial support coming from the Office of the Air Comptroller. The project was undertaken in an effort to provide a modest scale automatically-sequenced electronic digital computer to fill the need for computing facilities during the time that larger scale machines were reaching their completion stages. Emphasis was placed on simplicity in design and on earliness in completion. Thus, for example, there was to be no division instruction, no absolute comparison instruction, no logical transfer instruction, etc. But during the course of the development it was realized that with comparatively little additional effort, such facilities could be added. The completed machine contains all of these features and others, and is in fact a full scale computer. It took less than 18 months for the planning, design and construction of the SEAC and less than 6 months for testing. The machine performed its first integrated computational operation on April 7, 1950. It carried out some miscellaneous mathematical exercises, such as the factorization of numbers into primes, the computation of sine-cosine tables, and the solution of some diophantine equations, during the months of April and May. On May 9, it carried out its first significant computation, the tracing of optical rays through lens systems, for the NBS

Optics Division; and on June 30, 1950 the SEAC was officially announced and dedicated [2].

Since that time the SEAC has been in continuous use 24 hours per day, 7 days a week. At the present time approximately 70 per cent of this time is spent on problems of importance in mathematics, physics, engineering, management, etc. In particular, a high percentage of the computing effort on the SEAC is devoted to problems of importance in national defense. Approximately 30 percent of the SEAC time is spent on scheduled engineering work; viz., research on components, installation of new equipment, and preventive maintenance. Unscheduled maintenance amounts to about 20 per cent of the total time assigned to computing. [9].

The programming for and operation of the SEAC is done in the Computation Laboratory of the Applied Mathematics Division. Most of the calculations on the SEAC are performed as technical services in line with the role of the Applied Mathematics Division as a central computing laboratory and mathematical service facility for Government agencies.

This Handbook is directed toward two classes of readers - one includes those engineers, mathematicians, physicists, etc., who are potential customers for SEAC services. For such persons a general familiarity with machine characteristics and coding principles will facilitate the formulation of problems and will make it possible to produce realistic estimates of coding time and computing time. The second class of reader is composed of employees of the Computation Laboratory and others who in one way or another become engaged in actual coding for SEAC. Part I of this report is addressed primarily to the first class of persons and Part II to the second class. Part II is pitched on the level of the new employee who is an average college graduate with a major in mathematics. These employees may be aware of number systems other than the decimal, but in general they have no working knowledge of such systems; hence Part II begins with a discussion of the binary system which is used by the SEAC and the hexadecimal (base 16) number system which is more convenient than the binary for coding. Part II also contains a discussion of conversions between number systems and then goes into the details of coding for SEAC.

The intention is to make this report as self-contained as possible. Hence general descriptions are given of certain SEAC components which are described in great detail elsewhere. For the reader who wishes to go into greater detail on certain aspects of SEAC or other computing machinery design or operation, a list of references is given. In particular, Reference 4, listed at the end of Part I, gives a comprehensive treatment of the subject and contains a very complete bibliography up to the year 1950. It does not, however, contain SEAC design features. Reference 8 deals with devices in terms of design, philosophy, and internal organization, rather than in terms of specific mechanisms or circuits, and therefore achieves a generality not likely to be outdated by new developments.

The first codes were prepared for SEAC in the Machine Development Laboratory of the Applied Mathematics Division in 1949, considerably before construction of the machine was completed. It was during that period that the conventions for standard subroutines described in Part II, Sect. 19, were established. The fact that these conventions are still used and are found to be highly convenient attests to the reasonableness of the decisions made at that time. In those days of coding for a still non-existent machine (in the operational sense) the Machine Development Laboratory staff prepared a basic library of fundamental subroutines, most of which are listed among the references at the end of Part II, and the incorporating routine described in Part II, par. 19.4. The members of the Machine Development Laboratory staff who contributed in this effort under the direction of Dr. E. W. Cannon are Dr. Merle M. Andrew, Mr. Ira C. Diehm, Mrs. Florence Koons, Dr. Samuel Lubkin, Mrs. Ethel C. Marden, Mrs. Ida Rhodes and Mr. Otto Steiner.

At the time that the SEAC was put into use, programming, coding, and operation were taken over by the Computation Laboratory. An early report, "A Manual For Coding On the SEAC", prepared by Dr. M. M. Andrew, was issued in May 1950, and contained in particular the basic library of subroutines referred to above. As work with the SEAC got under way, practical programming and coding experience was acquired and new techniques were developed. The staff also grew and classes were organized whenever necessary for the instruction of incoming personnel. Instruction notes were prepared in mimeographed form by members of the C. L. Staff, under the direction of the undersigned. Some of these notes have been used extensively in the preparation of this Handbook. In particular, the undersigned wishes to express his appreciation to Mr. Ira C. Diehm for his assistance in the writing of Sections 13, 14 and 18 of Part II, to Mrs. Viola D. Hovsepian for her assistance in the writing of section 17 of Part II, and to Mrs. Ethel C. Marden for her contributions in the writing of Section 19 of Part II. In addition, indebtedness is acknowledged to Dr. Samuel Lubkin whose dittoed notes on number systems have been used almost verbatim in Section 10 of Part II.

J H Levin

J. H. Levin

SUMMARY OF SEAC SPECIFICATIONS [3]

1. Number System — Binary

2. Word Length — 45 Binary Digits (44 digits of numerical information, 1 digit for sign)

3. Memory

Type	Capacity	Average Access Time
(a) Acoustic	512 words	216 microseconds
(b) Electrostatic	512 words	60 microseconds
Total:	1024 words	

4. Intermediate Storage

Type	No. of Channels	No. of Units	Maximum Capacity	Maximum Speed
(a) Reel-less magnetic tape unit	1	4 (at present)	10,000 words/unit	7000 words/min.
(b) Servo-magnetic tape unit[1]	5	1	10,000 words/channel	4100 words/min.

5. Input-Output

Type	Maximum Speed
Keyboard	(manual)
Teletype	30 words/min.
Magnetic Wire	3100 words/min.

6. Type of Arithmetic Unit — Serial

7. Modes of Operation — (a) 4-Address; (b) 3-Address

8. Numbers of Orders Available — $2^4 = 16$

[1] Currently being installed

9. Speed of Operation (includes access time)

Type of Operation	Acoustic Memory (Average Time)	Electrostatic Memory
(a) Addition	900 microseconds	200 microseconds
(b) Multiplication	3000 microseconds	2400 microseconds

10. Basic Repetition Rate — 1 megacycle

11. Power Requirements — 15 kw.

12. Number of Tubes — 1200 (approximately)

13. Number of Crystal Diodes — 19000 (approximately)

14. Net Floor Space — 150 square feet

PART I

DESCRIPTION OF SEAC

1.0 General Characteristics. The SEAC is a general-purpose automatic binary digital electronic calculator. It is general-purpose in the sense that it is capable of any computation which can be formulated in terms of numerical operations; i.e., addition, subtraction, multiplication, etc. A complete list and description of the operations performable on SEAC are given later in Part II, Sect. 13 of this report. The SEAC is automatic in the sense that it is independent of a human operator once computations are begun. The automatic feature implies that at any stage in a computation without any intervention from an operator the machine "knows" what it is to do next. This in turn requires: (a) the existence of a means of representing "instructions" within the machine; (b) the ability of the machine to "store" both instructions and numbers; (c) the presence of "control" facilities to interpret and execute the instructions in their proper sequence; (d) the presence of facilities for carrying out certain arithmetic and other operations; (e) the presence of communication facilities between the machine and its operator.

1.1 Words, Instructions, and Numbers. A single instruction is a command to the machine to execute a simple operation such as, addition, subtraction, multiplication, division, comparison, reading, writing, etc. Instructions are written in a numerical code and are indistinguishable in appearance as well as in method of storage from numbers. The generic term "information" is used to include both instructions and numbers. The unit of information is the "word" which for the SEAC is defined to be a sequence of 45 binary digits. The least significant digit position is reserved as a sign digit, zero for plus and one for minus. Any word may be interpreted either as an instruction or as a number. The particular interpretation made depends on whether within the machine the word is channeled when called for from the component in which it is stored into the "control unit", where words are interpreted as instructions, or into the "arithmetic unit", where words are interpreted as numbers. A given word may be interpreted as an instruction at one stage of a calculation, and as a number at another stage of the calculation.

A negative number is represented by its absolute value together with a negative sign. The location of the binary point in numbers is between the second and third binary positions from the left[1].

1.2 <u>Storage.</u> The storage function is performed by various storage devices which may be classified into two categories: internal, or "fast" storage, and external, or "slow" storage (cf. Ref. 4, Chap. 14). The internal storage or "memory" consists of both acoustic and electrostatic systems each of which is capable of storing 512 words. There is thus a total internal memory capacity of 1024 words. The memory locations (usually referred to as "cells", or "registers") are numbered serially, 0,1,...,1023, and these numbers are called "addresses".

The external storage media consist of teletype tape, magnetic tape, and magnetic wire. These media are essentially identical with the "input" and "output" media which are required for purposes of communication between the

[1] This choice was influenced by a number of factors, one being the convenience of being able to handle commonly used numbers such as 1, 2, π, e, etc., in unscaled form. For a full discussion of this choice see Ref. 7.

machine and the operator.

Since stored information must be readily accessible at any time, a measure of the efficiency of a storage system is "access time", i.e., the time required to withdraw a word from storage. The access time is much shorter for internal storage than for external storage - on the order of microseconds as compared with hundredths or tenths of seconds, or seconds.

1.3 Routines. The sequence of words (instructions and numbers) that must be introduced into the memory in order to carry out a computation is called a "routine". The process of drawing up such a sequence of instructions and numbers is called "programming". The process of translating these numbers and instructions into the coded form required for feeding into the machine is called "coding". Within the machine, instructions specified in the routine are interpreted in the control unit, which also supervises their execution. The execution involves withdrawing the numbers to be operated upon from storage into the arithmetic unit. Because of the dual aspect of words mentioned in par. 1.1, a word which is an instruction at one stage of a calculation may be called into the arithmetic unit at another stage to be operated upon as a number, and vice versa. This ability on the part of the SEAC to modify or change its own instructions is an essential characteristic of this type of machine.

1.4 Four-Address and Three-Address Modes. The SEAC was originally constructed as a 4-address machine. That is, normally, each coded instruction includes four address references as well as the operation to be performed: (1) address of first operand; (2) address of second operand; (3) address where the result is to be stored; and (4) address of the next instruction to be performed. Circuitry was later added (March, 1952) to enable it to operate in the 3-address mode. In this mode, item (4) above, the address of the next instruction to be performed is not exhibited. The control normally proceeds through the instructions in their address-sequence. Conditional transfer of control can be performed as described in Part II, Sect. 20.

2.0 Memory. There are many sources where one may obtain as complete and as detailed a discussion of memory devices as is desired (cf., for example, Ref. 4). However, in order to make this report reasonably self-contained and because some conception of the principles behind the memory devices aid in understanding many features of the coding process and of the machine operation, the essential nature of these devices on the SEAC will be here described.

It is possible to represent numbers by a series of disturbances (usually electrical or sonic pulses) following each other at a single point along a path or a circuit at successive times. It is on the other hand possible to represent numbers by conditions (usually static voltages) at a series of points all occurring at the same time. A pure time distribution is termed "serial" while a pure space distribution is termed "parallel". The former in general is more economical as concerns equipment but involves time duration not required for the latter.

2.1. Acoustic Memory. The acoustic memory system exemplifies the serial type of number representation in which numbers are represented as a succession of digits available at a given place in a time sequence. The basic unit of the acoustic storage system is the acoustic delay line (cf. Ref. 4, pp. 341-348). This consists essentially of a mercury filled glass tube closed at the ends with quartz crystals. These crystals have the property of becoming slightly deformed under the influence of an electrical pulse. Conversely, when one of these crystals is subjected to a mechanical distortion it emits an electrical impulse. In other words, electrical energy is transformed by these crystals into mechanical energy, and mechanical energy into electrical energy. This effect is known as a "piezoelectric" effect.

Electrical pulses representing a sequence of binary digits are introduced at one end of the tube or "tank". These are transformed by crystals through piezoelectric contractions and expansions into sound waves and transmitted through the mercury. The receiving crystal at the exit end of the tube vibrates under the influence of the sonic signals from the mercury, generating electrical pulses which through appropriate circuitry are amplified, reshaped and then re-circulated through the mercury. Since the velocity of sound in mercury is very low (about .06 in. per microsecond) compared with the velocity of an electrical impulse this type of device provides means for packing a great many pulses into a small linear range.

In the SEAC in fact each tube is 24 inches long and is capable of storing 384 pulses or 8 groups of 48 pulses each. Only the first 45 of any 48 pulse group represent information and these 45, as seen before, constitute a word. Thus a single delay line or "tank" is capable of storing 8 words, and since there are althogether 64 tanks, this means an acoustic memory capacity of 512 words. The contents of a tank are available, i.e. may be read one binary digit at a time, after leaving the tube. The time for the complete contents of a tank (i.e., eight words) to pass a given point in the circuit is called one "major cycle". Since the pulse rate is one megacycle a major cycle is

384 microseconds. One-eighth of this time is called a minor cycle. That is, the time for the contents of a single word to pass a given point is 48 microseconds. The interval between the time that a word is called for and the time that it has been read is called the "access time". The word nearest the exit end of the delay line has the shortest access time and that nearest the entrance has the largest, the average access time being 216 microseconds. This value can sometimes be materially reduced for a specific problem by special planning of coding taking into account the time of each operation, but this is not always practical (cf. Part II, Sect. 18).

2.2 Electrostatic memory. In the SEAC the parallel type of storage is realized in the electrostatic memory (cf. Ref. 4, pp. 354-370). This system is composed essentially of a bank of 45 cathode ray tubes. Storage in one of these tubes depends on electrical charges placed at discrete points on the target surface of the tube. Means are provided for charging these points to desired potentials corresponding to digits 0 or 1, under the influence of the electron beam in the tube. Deflection circuits are furnished for aiming the beam at any desired point. The pattern for the charges is chosen to be a rectangular lattice. At the present time this lattice consists of 32 points along one side and 16 along the other, so that a single tube may store 16 x 32 = 512 binary digits. The states of corresponding lattice points, one from each tube, represent a set of 45 binary digits, which constitute a word. The scheme described is thus capable of storing 512 words. Reading and writing circuits are provided which make it possible to record or call for all the digits of a word simultaneously, thus giving the parallel feature. The effective access time for this memory is about 60 microseconds[1] as compared with an average access time of 216 microseconds for the acoustic memory[2].

The use of cathode ray tubes in this kind of memory was first demonstrated by F. C. Williams [5], of the University of Manchester, England. The tubes are frequently called "Williams tubes", and the memory is usually referred to as the "Williams memory" (cf. Ref. 4, pp. 366-370).

3.0 Input and Output Equipment. In addition to the two types of internal storage just described, the SEAC has various means of external storage which are more appropriately discussed under the heading of INPUT and OUTPUT equipment.

In order for the machine to carry out a sequence of computations it is necessary to be able to communicate to it the routine to be followed together with all necessary numerical data. Similarly it is necessary to have facilities for communicating results of the machine's computation to the operator. This communication involves the translation (via one or more intermediate stages) from one type of word representation convenient for the coder or operator to a representation suitable for the machine, and vice versa. Typical input information has successively the forms of characters on a typewritten page, holes in a punched paper tape, magnetized areas on a wire, electrical and sonic pulses in the acoustic memory, or charged spots on the

[1] During this 60 microseconds there have really been 5 references of 12 microseconds each to the electrostatic memory: one cycle of reading or writing, and four cycles for regeneration of the pattern.

[2] As of this writing memory check circuits are being installed both in the acoustic and electrostatic memories.

cathode ray tubes in the electrostatic memory. Output information takes these forms in the reverse order. It is clear that all of these forms represent types of storage. As listed above they are in increasing order of accessibility to the machine. The first three are varieties of "external storage", while the last two, which have already been discussed, are examples of "internal storage". Historically, Teletype apparatus was the first input-output equipment to be used with SEAC, but with the inauguration of faster magnetic media it has assumed a secondary role. It remains, however, a necessary intermediate stage in passing from the printed page to the magnetic wire; i.e. input information must first be manually punched on Teletype tape which is then fed into an "inscriber" to make a wire recording. This recording may be filed away and stored indefinitely until needed on the SEAC. For reading wire-recorded information into the SEAC memory the cartridge on which the wire is wound is plugged into an input receptacle in the control panel. An output receptacle is also provided to accommodate a second cartridge for recording output information. When recording is completed, the cartridge may be removed and plugged into an "outscriber" which reads the wire and makes a punched paper tape. Output information may be fed back into the SEAC by plugging the output cartridge into the input receptacle. It is to be noted that the inscriber and outscriber are items of auxiliary equipment not connected directly with the SEAC. Thus the SEAC can generally devote most of its time to actual computations and relatively little time need be spent on input and output. A variation on the foregoing procedure consists in punching input information on punched cards which are then fed into a "card-to-wire" recording device[1]. This recording is again ready for filing

[1] Currently under construction.

or for feeding into the SEAC. The SEAC is also provided with a number of magnetic tape units which are intended for intermediate storage. Each of these units is equipped with a reading, a writing, and an erasing head, so that information read out to these tapes at any stage of the calculation are available for recall into the internal memory at any subsequent stage of the calculation. These tapes are accessible only through the machine. All of the input and output units may be operated under machine control. The selection of the unit to be employed at any stage of the computations is controlled by the code and the switch settings on an "external selector" panel which is associated with the manual controls (cf. par. 6.0). A schematic diagram showing the possible paths for flow of input and output information, and the interrelation between the SEAC and the various items of auxiliary equipment is given in Fig. 1.

4.0 Arithmetic Unit. (cf. Ref. 4, Chap 13.) All of the arithmetic operations are performed in the arithmetic unit; namely the usual operations of addition, subtraction, multiplication, and division. The arithmetic unit is also capable of making either an algebraic or absolute comparison between two numbers and determining which is the largest. Finally it performs the operations of extraction and logical multiplication. All of these operations are described in greater detail in Part I, Sect. 7 and Part II, Sect. 13.

The binary point is fixed by the arithmetic unit between the second and third binary positions from the left (see Footnote 1, pg. 8). Addition is the fundamental operation in the arithmetic unit and is performed serially; i.e., corresponding digits of the addend and augend are added one pair at a time together with any carry which may have come from the previous steps.

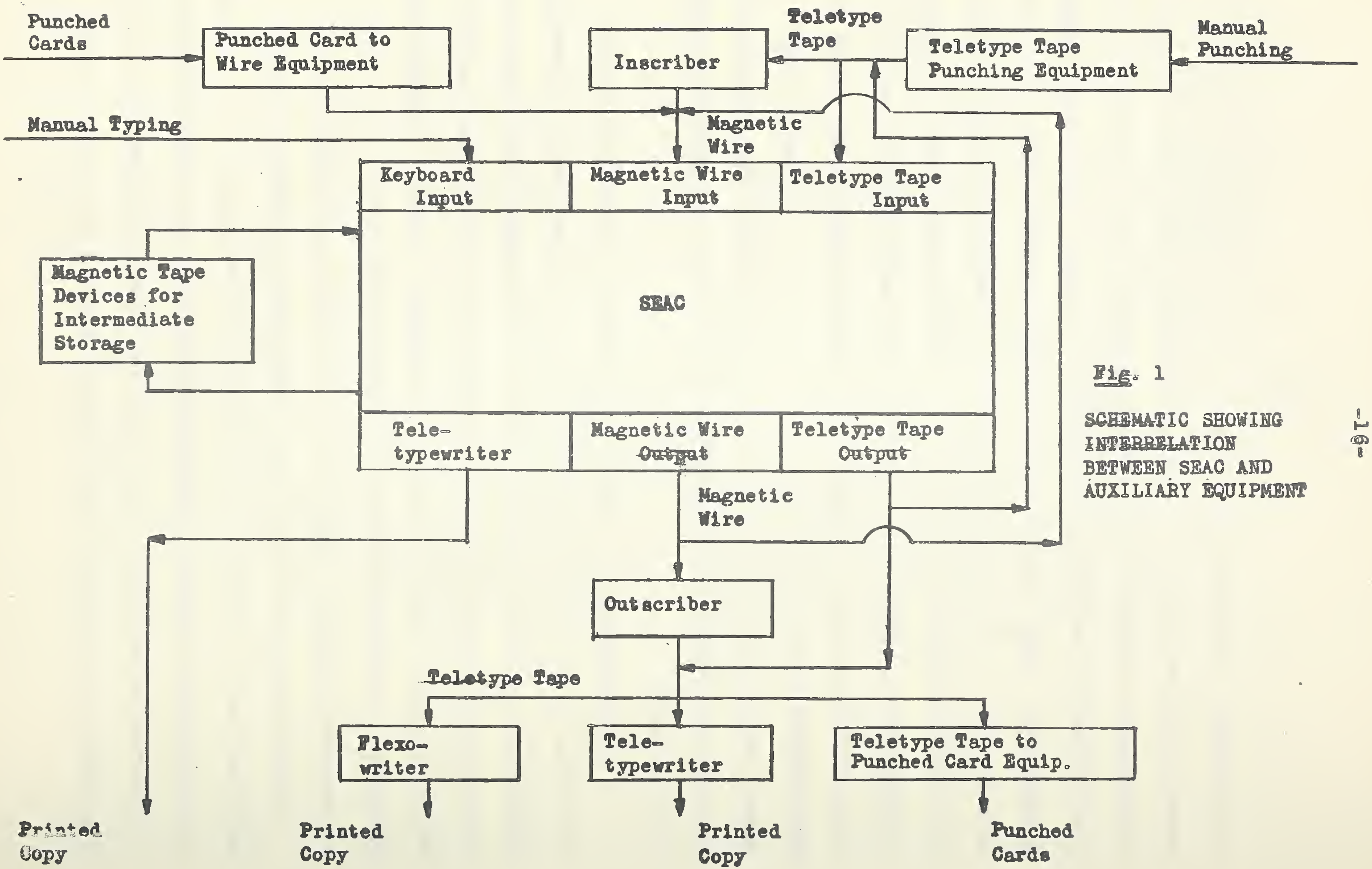


Fig. 1

SCHEMATIC SHOWING INTERRELATION BETWEEN SEAC AND AUXILIARY EQUIPMENT

Subtraction, multiplication, and division make use of addition, and hence are also performed in a serial fashion. For this reason the SEAC is said to have a "serial" type arithmetic unit. In contrast a parallel arithmetic unit would simultaneously add all corresponding digits of addend and augend, generating the carries. Another parallel operation simultaneously adds all these carries to the initial sum. This may result in further carries in which case the process is repeated.

5.0 <u>Control Unit.</u> The function of the control unit is to guide the operation of the machine under the influence of the coded instructions stored in the memory. Under the supervision of this unit an instruction about to be performed is transferred from the memory into the "instruction register" where it is held during the time that it is being interpreted and executed. (The transfer into the instruction register does not clear the memory cell from which the instruction was taken.)

6.0 <u>Manual Controls.</u> Nearly all the operations of the SEAC may be controlled from a central group of control panels. These panels include facilities for clearing the memory and control circuits, for starting and stopping the machine, and for causing the machine to proceed one step at a time through a program. One of the panels of this group is the "external selector" panel mentioned earlier. Another contains the receptacles for input and output cartridges together with necessary control switches. Adjoining the control panel is Teletype equipment for input and output. Near by are the various magnetic ~~magnetic~~ tape devices together with their control switches.

7.0 Operations. The operations that the SEAC is capable of performing are described more fully in Part II, Sect. 13. Here they are classified into five main categories.

7.1 Arithmetic Operations.

a. Addition and Subtraction.

b. Multiplication. The product P of two 44 binary digit numbers

$A = a_{-1}\, a_{0} .\, a_1\, a_2 \cdots\, a_{42}$, and

$B = b_{-1}\, b_{0} .\, b_1\, b_2 \cdots\, b_{42}$

(where each a_i and b_i is 0 or 1) is an 88 binary digit number:

$P = p_{-3}\, p_{-2}\, p_{-1}\, p_{0} .\, p_1\, p_2 \cdots p_{42}\, p_{43} \cdots p_{84}$,

where each p_i is 0 or 1. On the SEAC there are three types of multiplication operations:

(i) The "high order" product of A and B is

$P_H = p_{-1}\, p_{0} .\, p_1\, p_2 \cdots p_{42}$

(ii) The "high order rounded" product of A and B is

$P_R = P_H + p_{43} \cdot 2^{-42}$

(iii) The "low order" product of A and B is

$P_L = 00.p_{43}\, p_{44} \cdots p_{84}$

Note: In all cases the digits p_{-3} and p_{-2} are lost. Also, in all cases the sign of the result is that of the high order product.

c. Division. This operation yields the unrounded quotient. The remainder does not appear.

N. B. There is no automatic over-flow indication for any of the above operations.

7.2 Comparison. Both algebraic and absolute.

7.3 Logical Operations.

a. Logical transfer. This instruction enables one to replace any designated portion of a word by the corresponding portion of another word.

b. Logical multiplication. Digit by digit multiplication of two numbers.

7.4 Input and Output Operations. Tape or Wire Movement Operations. Words may be read into the internal memory from any of the input-output media, or vice versa, in one word or in eight word blocks. Input and output media have been described in par. 3.0. The selection of a particular medium is indicated by the code and the switch settings on the external selector panel.

7.5 Base Operation. This operation has different meanings depending on whether it is used in the four address mode or three address mode. In the 4-address mode, it has the effect of translating references to certain addresses by any prescribed amount, and also provides a means of switching control. In the 3-address mode it provides a way of switching control only. The base operation in the two modes will be described more fully in Part II, Sects. 13 and 20.

8.0 Coding. Coding for the SEAC may be done either in the 4-address or in the 3-address system. These systems will now be described.

8.1 Coding in the 4-Address System. In the 4-address system an instruction word has the following composition:

Symbol	α	β	γ	δ	Op	Sign
No. of Binary Digits	10	10	10	10	4	1

where each of the 4 Greek letters represents an address in the memory and Op stands for an operation. The plus and minus are represented by 0 and 1 respectively. Thus an instruction consists of a sequence of 45 binary digits, the extreme right hand digit indicating the sign. Since each address is represented by 10 binary digits, there are $2^{10} = 1024$ addresses possible, which coincides with the size of the memory. Similarly, there are $2^4 = 16$ possible operations. For operations falling into the categories of pars. 7.1 and 7.3, the interpretation of the above instruction is: the operation Op is applied to the contents of α and β; the result is put into address γ; the next instruction is found in δ. The comparison operations in par. 7.2 provide a way of selecting between two possible paths for the control to follow. For example, in algebraic comparison, if the number in α is less than that in β, then the control goes to γ; otherwise it goes to δ. In the case of an absolute comparison, the comparison is made between absolute values. For the input and output instructions in par. 7.4 the operations are performed on blocks of one word or eight words depending on whether β is odd or even, respectively. In the case of an odd β one word is called into γ on an input instruction or read out of γ on an output instruction. For operating on a block of eight words γ must be a multiple of eight[1] and then the addresses affected by the order are γ , $\gamma + 1$ ..., $\gamma + 7$. The designation of input or output unit called for is done partly by the code and partly by the external selector switches. A minus sign after an instruction is normally a signal to the machine to halt after completing the instruction. However, with appropriate switch settings on the manual panel it is possible either to over-ride all coded halt signals, or to over-ride all halt signals except those associated with input and output instructions.

8.2 Coding in the 3-Address System. The 3-address system for SEAC was designed with a special "floating address feature", which will be described below. A 3-address instruction has this composition:

Symbol	α	β	γ	a	b	c	d	Op	S
No. of Binary Digits	12	12	12	1	1	1	1	4	1

In this system the α , β , and γ addresses play the same role as in the 4-address system. The absence of a δ address is explained by the fact that the control normally proceeds sequentially through the addresses of the memory. The control may be transferred or "jumped" however, to the γ address of an appropriate comparison instruction just as in the 4-address system. Associated with the control are two "address counters", C_0 and C_1. The 0 and 1 values of the single binary digit d are used to select the first or the second of these counters. Normally each time an

[1] The eight addresses γ, $\gamma + 1$, ..., $\gamma + 7$, where $\gamma = 0 \pmod{8}$, belong to a tank (see par. 2.1). The tanks are numbered (in decimal) 0,1,...63. The number of the tank to which an address belongs is the integral part of the quotient upon division by 8.

instruction is performed the counter selected is advanced one unit indicating where the control goes next. Normally, also, C_0 is the counter selected. In transferring control by means of a comparison (either algebraic or absolute) instruction, the counter selected by d is set to the <u>value</u> γ of the comparison instruction, and the control goes to the <u>contents</u> of this γ. This is the only way to select C_1 initially. Thereafter C_1 remains selected as long as d = 1. The control returns to C_0 immediately after an instruction appears in which d = 0, no comparison instruction being necessary in this case.

The digits a,b,c, may be called "floating address indicators" corresponding to addresses α, β, γ, respectively. For example, if a = 0, then in the control unit α is interpreted as an absolute number, while if a = 1, then α is interpreted relative to the address in which the instruction being executed appears. Similarly for b and c.

The fact that 12 binary digits are provided in the 3-address system for the α, β, and γ addresses means that with this system it is possible to expand the memory of the SEAC from the present 2^{10} words to 2^{12} words. The floating address feature, and the presence of the two address counters C_0 and C_1 facilitate the handling of subroutines in the 3-address system. This will be treated in more detail in Part II, Sect. 20 of this manual.

8.3 <u>Getting a Routine into the Machine</u>. The question remains how to get a routine into the machine. A feature of SEAC design is that when the START button is pressed after the memory and control circuits have been cleared preparatory to starting a new problem, the first instruction performed is a call for eight words of information. These eight words would normally consist of a short routine for reading in the main problem routine and then sending the control to this main routine.

9.0 Subroutines. Certain processes such as conversion of numbers from binary to decimal and vice versa, calculation of certain common transcendental functions (e.g. sin x, e^x, etc.), arithmetic operations with numbers expressed in floating binary point form, etc. occur in a great many problems. For processes of this sort it is convenient to maintain a library of precoded "subroutines". Such a library is in existence for the SEAC and is being steadily augmented [6]. The subroutines making up this library are punched on short strips of teletype tape and filed for ready use. Programmers are thus relieved of the task of coding processes of the type described above when they occur, but need merely to draw the required subroutines from the file and copy them by means of mechanical reperforating equipment onto the master routine tape. Duplication of coding effort is thus avoided, the possibility of coding errors is reduced, and much time is consequently saved. The library subroutines have been carefully checked so that the users may feel secure in using them.

Subroutines are coded without reference to the actual positions they will occupy in the memory. Necessary changes in subroutine instructions to make them conform to their actual positions are made by a special pre-coded preparatory routine (see Part II, Sect. 19 of this report).

Associated with each subroutine in the library is a description, including a list of specifications which briefly summarizes the pertinent information concerning the subroutine, such as title, file number, number of cells occupied, where the argument goes, where the result appears, etc. Thus the user can determine quickly what the subroutine does and how to use it. A more detailed explanation together with the code itself is also included in the subroutine description, so that a more careful study may be made if necessary. A list of SEAC subroutine descriptions is included in the bibliography at the end of Part II.

References

1. NBS Interim Computer, Technical News Bulletin, National Bureau of Standards, v. 33, pp. 16-17, (Feb. 1949).
2. SEAC, The National Bureau of Standards Eastern Automatic Computer, Technical News Bulletin, National Bureau of Standards, v. 34, pp. 1-8 (September, 1950).
3. NBS Electronic Computers Laboratory Staff, The Operating Characteristics of the SEAC, Mathematical Tables and Other Aids to Computation, v. IV., no. 32, pp. 229, 230 (October, 1950).
4. Engineering Research Associates, High-Speed Computing Devices, McGraw Hill Book Co., 1950.
5. Williams, F. C. and T. Kilburn, A Storage System for Use With Binary Digital Computing Machine, Proceedings of the Institute of Electronic Engineers, Part III, pp. 81-100 (March, 1949)
6. Levin, J. H., Construction and Use of Subroutines for the SEAC, Proceedings of the Association for Computing Machinery, (May 1952).
7. Lubkin, S., Decimal Point Location in Computing Machines, Mathematical Tables and Other Aids to Computation, v. III, no. 21, pp. 44-50 (January, 1948).

8. Hartree, D. R., _Calculating Instruments and Machines_, The University of Illinois Press, 1949.

9. Alexander, S. N., The National Bureau of Standards Eastern Automatic Computer, _Review of Electronic Digital Computers, Joint AIEE-IRE Computer Conference_, Feb. 1952.

PART II

PROGRAMMING AND CODING FOR SEAC

10.0 Systems of Number Representation. Several number systems come into play in connection with SEAC programming. The principal ones, of course, are the decimal and the binary. In addition, however, the "octal" (base 8), and "hexadecimal" (base 16) have a number of convenient features[1]. These systems will be described in this section.

10.1 Decimal System. In the usual manually performed arithmetic, numbers are represented in the decimal system. In this system a number is represented by an ordered sequence of characters, called digits (in remembrance of the system's origin). Successive characters occupy successive horizontal positions and have unequal weights in the ratio of 10 to 1 for the same character while, in any one position, ten different characters are used to present the numbers 0 through 9. A similar system can be used in machines. For example, ten levels of voltage can be used to represent the numbers 0 through 9 on each of a group of lines with each of which is associated a power of 10.

10.2 Binary System. A more common number system for digital computers is the "binary". In this system (used on the SEAC), there are only 2 characters, 0 and 1, representing the numerical values of zero and unity, and different positions are associated with different powers of 2. The use of only 2 types

[1] The words "octal" and "hexadecimal" are hybrid words which have become established through usage. Though actually "octaral" or "octadic", and "sedecimal" or "hexadecadic" would have been preferable, this Handbook will conform to the established parlance.

of digits makes this system particularly convenient for representation by certain types of electrical and mechanical devices; e.g. the digits 1 and 0 may be made to correspond to the conducting and non-conducting states of a vacuum tube, to the presence or absence of a pulse, to the closed and open states of a relay, etc. Moreover, rules of computation are particularly simple for the binary system in view of the existence of only 2 types of digits and can, therefore, be easily mechanized.

10.3 Comparison of Decimal and Binary Systems. To emphasize some similarities as well as differences between the decimal and binary systems consider the following examples of number representation in the two systems, respectively:

$$\text{(a)}\ 38.701 \equiv 3\cdot10^{1} + 8\cdot10^{0} + 7\cdot10^{-1} + 0\cdot10^{-2} + 1\cdot10^{-3}$$

$$\text{(b)}\ 1011.01 \equiv 1\cdot2^{3} + 0\cdot2^{2} + 1\cdot2^{1} + 1\cdot2^{0} + 0\cdot2^{-1} + 1\cdot2^{-2}.$$

In (a) the point is called a "decimal point", and the "base" or "radix" is 10. In (b) the point is called "binary point", and the base or radix is 2. In both cases the decimal or binary point serves to separate the integral and fractional parts of the number. In arithmetic operations in the binary system, the binary point is handled quite analogously to the decimal point, as will be illustrated in the examples which follow. Systematic methods for converting from binary to decimal and vice versa will be discussed in Sect. 11.

10.4 Arithmetic Operations in the Binary System. Addition in the binary system uses the following relations:

$$0 + 0 = 0$$

$$1 + 0 = 0 + 1 = 1$$

$$1 + 1 = 10 \text{ (0 and carry 1).}$$

With these rules for individual digits, it is simple to add any two binary numbers. For example, 11001.01 may be added to itself as follows:

$$\begin{array}{rl} 11001.01 & \\ \underline{11001.01} & \\ \underline{11\ \ 1\ \ 1\ \ \ \ } & \text{carries} \\ 110010.10 & \text{sum} \end{array}$$

This result could, of course, be obtained immediately by noting that the addition of a zero to the right of an integer in the binary system causes multiplication by 2 analogous to the multiplication by 10 resulting from a similar operation in the decimal system.

Subtraction is equally simple, using the relations

$$0 - 0 = 1 - 1 = 0$$

$$1 - 0 = 1$$

$$0 - 1 = 1 \text{ and borrow 1 .}$$

For example, .1101 may be subtracted from 1.1010 as follows:

$$\begin{array}{rl} 1.1010 & \\ \underline{.1101} & \\ \underline{1\ 1\ 1\ \ \ } & \text{borrows} \\ .1101 & \text{difference} \end{array}$$

The Multiplication Table in the binary system is much simpler than in the decimal system and is given by

$$0 \times 0 = 0 \times 1 = 1 \times 0 = 0$$

$$1 \times 1 = 1$$

Except for this simple table and the above rules for addition, multiplication in the binary system is performed in the same way as in the decimal system. For example, 1.101 may be squared as follows:

1.101	
1.101	
1101	First partial product
11010	Second and third partial products
1101	Fourth partial product
10.101001	Final product

Division is also accomplished in a manner similar to that used in the decimal system. For example, 1010.1100 is divided by 11.01 as follows:

11.01) 1010.1100 (11.01

1101

10001

1101

10000

1101

11

giving a quotient of 11.01 and a remainder of .0011.

It should be noted that since 2^n is an unrepeated factor of 10^n the process of conversion of a binary fraction into decimal form terminates with the same numbers of decimal digits after the decimal point as there were binary digits after the binary point. However, 10 is not a factor of any power of 2; hence, in general, the binary fraction equivalent to a given decimal fraction does not terminate after a finite number of digits. It a limited number of digits were to be retained and the remainder dropped, it would result in making most numbers too small with an average systematic error of about half in the last retained digit. The same difficulty arises whenever a value is given to more digits than can be retained. The process of multiplication, for example, produces more digits, in general, than were present in either of the original factors. The same is true in division; in fact, most divisions yield quotients which cannot be exactly expressed in any finite number of digits. To terminate a number after a given number of digits, the familiar "Round-Off" procedure is commonly used. This consists of choosing that number having the desired number of digits which is closest to the original unrounded value, and, in the special case where the latter lies exactly midway between two consecutive numbers of the desired type, of choosing the higher in absolute value. This is particularly simple in the binary system since it merely requires that, for round-off to n digits, we take the first n digits and add unityto the last place if the (n+1)st digit is a one, adding nothing if it is a zero. Another way of stating this is that we add the (n+1)st digit to the last place of the number formed by the first n digits. The bias remaining after such round-off is negligible[1]. For example, round-off of the binary number 1011.00011100... by this method gives, for 2 through 6 digits after the binary point:

1011.00

1011.001

1011.0010

1011.00100

and 1011.000111 respectively.

---

[1] For a full discussion of round-off error, cf. Ref. 2.

10.5 Octal and Hexadecimal Systems. One difficulty of the binary system is the large number of digits required as compared to the decimal system where equivalent accuracy is desired. Since $\log_2 10 \simeq 3.3$, the former requires about 3.3 times as many digits as the latter. The fact that only the two digits 0 and 1 occur, while advantageous arithmetically and in the design of a computing machine, makes typographical errors likely when typing a long sequence of digits. It is, therefore, advantageous to group several binary digits together for convenience of handling. If each 3 binary digits are handled as a unit, we have the number base $2^3 = 8$ and call the resulting number system the "octal number system". This system has only about 1.1 times as many digits as the decimal for equal accuracy, involves no new digits, and conversion between it and the binary system is trivial, using the following equivalences:

Binary	Octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Obviously, no 8 or 9 is used in the octal system. The arithmetic for this system is very similar to that for the decimal system, noting that 8 (decimal) = 10 (octal), 9 (decimal) = 11 (octal), 10 (decimal) = 12 (octal), etc. For many purposes, however, it is preferable to consider each octal digit as merely a shorthand representation for 3 successive binary digits.

Similarly if each 4 binary digits are treated as a unit, we have the number base $2^4 = 16$. The corresponding number system is the "hexadecimal number system". This system requires approximately .83 times as many digits as the decimal system for a given accuracy. Sixteen characters are required in the hexadecimal system for number representation. We introduce 6 new characters, A, B, ..., F, in addition to the usual decimal characters and have the following equivalences:

Binary	Hexadecimal	Decimal
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

Binary	Hexadecimal	Decimal
1100	C	12
1101	D	13
1110	E	14
1111	F	15

For purposes of SEAC coding the hexadecimal system has been found to be more useful than the octal and has been generally adopted.

11.0 Conversion Between Number Systems. It is convenient and useful to discuss this subject first in terms of a number system with base p and another system with base q. Two distinct cases of conversion from p-ary to q-ary arise: (a) using p-ary computation; and (b) using q-ary computation. It can then be seen at once that the method embodied in case (a) applies to (1) conversion from decimal to binary using decimal computation; and (2) binary to decimal using binary computation. Likewise case (b) applies to (1) conversion from decimal to binary using binary computation; and (2) binary to decimal using decimal computation. Since a human computer working at a desk calculator generally prefers the decimal system for computing purposes, he would convert from one system to the other by methods (a,1) or (b,2). But the SEAC being a binary computer, would use methods (a,2) or (b,1). In any event only two of these four methods are essentially distinct, as noted above. The integral parts of the numbers will be separated from the fractional parts, and each part will be treated separately.

11.1 Conversion From p-ary to q-ary Using p-ary Computation. Consider first the conversion of an integer P. This number may be represented in q-ary as follows:

(1) $$P = a_k q^k + \ldots + a_2 q^2 + a_1 q + a_0, \quad a_k \neq 0,$$

where the a_i are integers in the q-ary system, and are to be determined together with k. The calculation is accomplished by successive divisions by q in the following manner:

$$P/q = n_0 + m_0/q = a_k q^{k-1} + \ldots + a_3 q^2 + a_2 q + a_1 + a_0/q$$

$$n_0/q = n_1 + m_1/q = a_k q^{k-2} + \ldots + a_3 q + a_2 + a_1/q$$

$$n_1/q = n_2 + m_2/q = a_k q^{k-3} + \ldots + a_3 + a_2/q$$

.

.

.

$$n_{k-1}/q = n_k + m_k/q = a_k/q$$

In each step, the quotient n_i and remainder m_i are integers, and $m_i < q$. Then $a_i = m_i$ (expressed in q-ary) since all terms on the right except for the last are integers. The process terminates when for some $i = k$, $n_k = 0$.

Example 1. Convert the decimal number P = 213 to binary using decimal computation.

Solution. In this case p = 10, q = 2. The calculation may be arranged as follows:

		Remainders:
2)	213	
2)	106	1
2)	53	0
2)	26	1
2)	13	0
2)	6	1
2)	3	0
2)	1	1
	0	1

The result of the conversion is therefore 11010101. This conversion is an example of case (a,1) in par. 11.0.

Example 2. Convert the binary number P = 11010101 to decimal using binary computation.

Solution. Here, p = 2, q = 10 (= 1010 in binary).

1010)	11010101	**Remainders:**
1010)	10101	11 (= 3 in decimal)
1010)	10	1 (= 1 in decimal)
	0	10 (= 2 in decimal)

The result of the conversion is 213. This is an example of case (a,2) in par. 11.0, and illustrates how the conversion would be accomplished in the SEAC.

The conversion of a fractional quantity is performed similarly. A fraction P may be written in q-ary as follows:

(2) $$P = a_{-1}q^{-1} + a_{-2}q^{-2} + a_{-3}q^{-3} + \dots$$

In this case the a_i are determined by successive multiplications by q thus:

$$Pq = a_{-1} + a_{-2}q^{-1} + a_{-3}q^{-2} + \dots \equiv a_{-1} + n_{-1}$$

$$n_{-1}q = a_{-2} + a_{-3}q^{-1} + a_{-4}q^{-2} + \dots \equiv a_{-2} + n_{-2}$$

$$n_{-2}q = a_{-3} + a_{-4}q^{-1} + a_{-5}q^{-2} + \dots \equiv a_{-3} + n_{-3}$$

etc.

At each step the corresponding a_i is obtained as the integral part of the product. Only the fractional part of this product is used in the next multiplication.

Example 3. Convert P = .213 to binary using decimal computation (p=10, q=2).

Solution. The calculation is as follows:

Integer carry:		.213
	x	2
0		.426
	x	2
0		.852
	x	2
1		.704
	x	2
1		.408
	x	2
0		.816
	x	2
1		.632
	x	2
1		.264
	x	2
0		.528
	x	2
1		.056

etc.,

so that .213 = .001101101... in binary form.

Example 4. Convert the binary number P = .001101101 to decimal using binary computation ($p=2$, $q=10$).

Solution:

Integer carry:	$.001101101$
	$\underline{\times \quad 1010}$
10 (=2 in decimal)	$.001000010$
	$\underline{\times \quad 1010}$
1 (=1 in decimal)	$.010010100$
	$\underline{\times \quad 1010}$
10 (=2 in decimal)	$.1110010$
	$\underline{\times \quad 1010}$
1000 (=8 in decimal)	$.1110100$

The converted result is thus .213 to 3 decimal places.

11.2 Conversion From p-ary to q-ary Using q-ary Computation. The integer P in (1) would have the following form in p-ary:

$$P = b_s p^s + b_{s-1} p^{s-1} + \ldots + b_1 p + b_0 = \sum_{i=0}^{s} b_{s-i} p^{s-i} .$$

This number could also be written

$$P = p(\ldots\ p(p(p \cdot b_s + b_{s-1}) + b_{s-2}) + \ldots + b_1) + b_0 .$$

Using the q-ary representations of p and b_i $(i=1,\ldots,s)$ and performing the indicated operations starting with the innermost parentheses, the q-ary representation of P is obtained.

Example 1. Convert the decimal number P = 213 to binary, using binary computation.

Solution. Here, $p = 10$, $q = 2$; $b_2 = 2$, $b_1 = 1$, $b_0 = 3$. The calculations are given in the left hand column of the following table:

Binary	Decimal	Notation
10	2	b_2
x 1010	x 10	$\times\ p$
10100	20	$b_2\ p$
+ 1	+ 1	$+\ b_1$
10101	21	$b_2\ p + b_1$
x 1010	x 10	$\times\ p$
11010010	210	$p(b_2\ p + b_1)$
+ 11	+ 3	$+\ b_0$
11010101	213	$p(b_2 p + b_1) + b_0$

Thus the binary equivalent of 213 is 11010101. This examples illustrates the method used by SEAC in decimal to binary conversion [9.4].

The computation may be laid out in the following convenient arrangement:

m	b_{2-m} (Dec.)	$\sum_{i=0}^{m} b_{2-i}\ p^{m-i}$	$p \sum_{i=0}^{m} b_{2-i}\ p^{m-i}$
0	2	1010	10100
1	1	10101	11010010
2	3	11010101	

Example 2. Convert the binary number P = 10111 to decimal.

Solution. Here $p = 2$, $q = 10$; $b_4 = 1$, $b_3 = 0$, $b_2 = 1$, $b_1 = 1$, $b_0 = 1$. The calculations are as follows:

m	b_{4-m}	$\sum_{i=0}^{m} b_{4-i}\, p^{m-i}$	$p \sum_{i=0}^{m} b_{4-i}\, p^{m-i}$
0	1	1	2
1	0	2	4
2	1	5	10
3	1	11	22
4	1	23	

The result is thus 23.

If P is a fractional quantity, as in (2), we proceed in a similar manner. Supposing that P is a terminating fraction when written in p-ary, it may be represented in the following ways:

$$P = b_{-1}p^{-1} + b_{-2}p^{-2} + \ldots + b_{-s}p^{-s} = \sum_{i=0}^{s-1} b_{-(s-i)}p^{-(s-i)}$$

$$= 1/p(\ldots\ 1/p(1/p(b_s/p + b_{s-1}) + b_{s-2}) + \ldots + b_1).$$

Again, making use of the last of these representations, and starting with the innermost parentheses, the indicated calculations are performed using q-ary representations.

Example 3. Convert the binary number P = .001101101 to decimal (p=2, q=10).

Solution:

m	$b_{-(10-m)}$	$\sum_{i=0}^{m-1} b_{-(9-i)}p^{-(m-i)}$	$1/p \sum_{i=0}^{m} b_{-(9-i)}p^{-(m-i)}$
1	1	1	.5
2	0	0.5	.25
3	1	1.25	.625
4	1	1.625	.8125
5	0	0.8125	.40625
6	1	1.40625	.703125
7	1	1.703125	.8515625
8	0	0.8515625	.42578125
9	0	0.42578125	.212890625

The result of this conversion is .212890625.

Example 4. Convert the decimal number $P = .213$ to binary ($p=10$, $q=2$), giving the result to 9 places.

Solution.

m	$b_{-(4-m)}$	$\sum_{i=0}^{m-1} b_{-(3-i)}p^{-(m-i)}$	$1/p \sum_{i=0}^{m-1} b_{-(3-i)}p^{-(m-i)}$
1	3	11	.0100110011
2	1	1.0100110011	.0010000101
3	2	10.0010000101	.001101101

The result is .001101101.

11.3 Conversion Using Known Equivalents. The number to be converted may be broken up into a sum of simple numbers whose q-ary representations are known or tabulated. For example a conversion table for the decimal digits and for positive and negative powers of ten would in many cases be a highly useful tool. To convert 213, one would then merely have to determine from the table that 3 (decimal) = 11 (binary), 10 (decimal) = 1010 (binary), and 100 (decimal) = 1100100 (binary). Then 213 = 2 x 100 + 1 x 10 + 3; i.e.,

$$\begin{array}{r} 11001000 \\ 1010 \\ \underline{11} \\ 11010101 \end{array}$$

so that 213 = 11010101 in binary.

The examples in the foregoing paragraphs have all been in terms of the decimal and binary systems. For purposes of hand computations, however, it is easier to work in terms of decimal and octal, or decimal and hexadecimal, the latter being more appropriate for the SEAC. The conversion from hexadecimal or octal to binary, or vice versa, is immediate.

11.4 Exercises.

1. Using a decimal computation convert the following integers to hexadecimal: 1023, 18756, 783113.
2. Using decimal computation check the results of Ex. 1 by converting back to decimal.
3. Using hexadecimal computation convert the integers of Ex. 1 to hexadecimal.

4. Using hexadecimal computation check the results of Ex. 1 by converting back to decimal.

5. Using decimal computation convert the following numbers to hexadecimal: .1, .0023577, 73.40961.

6. Using decimal computation check results of Ex. 5 by converting back to decimal.

7. Using hexadecimal computation, convert the numbers of Ex. 5 to hexadecimal.

8. Using hexadecimal computation check the results of Ex. 5 by converting back to decimal.

9. Using the decimal - hexadecimal equivalences 10 = A, 100 = 64, 1000 = 3E8, 10000 = 2710, 100,000 = 186A0, convert the following numbers to hexadecimal: 1023, 18758, 783113

10. Using the hexadecimal - decimal equivalences 10000= 65536, 1000 = 4096, 100 = 256, 10 = 16, 1 = 1, check the results of Ex. 9 by converting back to decimal.

Answers:

1. 3FF, 4944, BF309

5. 1999... , .009A83A, 49.68DC3

12.0 Representation of Information on Input and Output Media.

As indicated in Part I, the unit of information is a "word" which may be either an instruction or a number. It is convenient in writing a word to group the 45 binary positions into a sequence of 11 hexadecimal positions, followed by a sign position. The reasons for this are: (1) Hexadecimal notation is much more compact, more rapidly written, and less subject to errors in writing or transcription than binary notation. (2) Hexadecimal notation is well adapted for use with the auxiliary equipment, particularly the Teletype equipment which uses a 4-hole code; i.e., information is punched on Teletype tape in groups of 4 binary digits. The Teletype keyboard characters are also in hexadecimal. Fig. 2 shows how successive hexadecimal characters as well as plus and minus signs, space, and carriage return symbols look when punched on Teletype tape. The tape engages with a sprocket in the reading unit by means of the small perforations to the left of the center. Note that a plus or minus sign is represented by a single binary position.

TELETYPE	BINARY	KEYBOARD CHARACTER
o	0000	0
o O	0001	1
o O	0010	2
o O O	0011	3
o O	0100	4
o O O	0101	5
o O O	0110	6
o O O O	0111	7
O o	1000	8
O o O	1001	9
O o O	1010	A
O o O O	1011	B
O o O	1100	C
O o O O	1101	D
O o O O	1110	E
O o O O O	1111	F
O o	0	+
O O o	1	-
O o		Space
O O o O O O		Carriage Return

Fig. 2. Representation of keyboard symbols on Teletype tape

In input or transcribing operations the Teletype tape moves into the reader in the direction of the arrow, so that in the figure, the bottom characters would be read first.

In representing numbers on input and output media it must be kept in mind that the SEAC has a "built in" fixed binary point, between the second and third binary positions from the left (See Footnote 1, Part I, pg. 8). This means that only numbers less than 4 in absolute value can be represented in "unscaled" form. The location of the binary point is immaterial in addition or subtraction operations, but it is significant in multiplication and division. In the representation of instructions, it is ignored (cf. Sect. 13).

12.1 Teletype Tape Preparation. In preparing Teletype tape, it is necessary to punch 13 characters per word: a space character, followed by 11 hexadecimal characters, and a sign. Any character at all may be punched in the space position. It does not affect the contents of the memory. However, for use with auxiliary equipment it is desirable to punch either a carriage return symbol, or a "plus" symbol in this position. Fig. 3 illustrates the punching of two numbers on Teletype tape: the number on line 3 of Fig. 4, and the number ADF85 458A2 C- (the sign is customarily written after the hexadecimal or binary representation to conform to the representation within the machine), whose decimal equivalent is -2.71828 18284 590 $\cong$ -e (cf. par. 12.2). In Fig. 3 a plus sign has been used in the space position in the first word, and a carriage return in the second word.

```
  0                       0 0                       0
     0 0   0 0 0 0           0 0 0 0 0       0 0   0 0
o o o o o o o o o o o o o o o o o o o o o o o o o o o
     0     0 0           0   0   0 0   0 0 0       0
          0   0 0 0 0   0   0 0   0           0 0
       0   0 0         0 0   0   0 0   0   0
```

Fig. 3. Number representation on Teletype tape

Teletype punching is a preliminary step required for putting information on any input medium. If the input medium is to be magnetic wire, the information must first be punched onto Teletype, then read from this onto the wire by means of auxiliary apparatus (the "inscriber").

12.2 "Machine Hexadecimal". Care must be taken in preparing hexadecimal representations of numbers for punching on Teletype tape, inasmuch as the representation required for Teletype punching differs from the "true hexadecimal" representation obtained by the methods of Section 11. For example, the number π = 3.14159 26535 898 appearing in line 1 of the tabulation in Fig. 4, when converted by any of the methods already discussed, either to binary or to "true hexadecimal", yields the number shown on line 2 or line 4, respectively. Assembling the 44 binary digits on line 2 in groups of 4 as shown (ignoring the binary point), and substituting for each group the corresponding hexadecimal digit, the representation on line 3 is obtained, and this is what would be punched on Teletype tape for purposes of feeding into the Teletype input or for transcribing onto magnetic wire. This is also what would be printed by the output typewriter if the machine were asked to print the contents of some address in which the number on line 2 were stored. It is referred to in the NBS Computation Laboratory as the "machine hexadecimal" representation as distinguished from the "true hexadecimal" representation in which the "hexadecimal point" separates the integral and fractional parts of the number.

1	Decimal	3.14159 26535 898
2	Binary	11.00 1001 0000 1111 1101 1010 1010 0010 0010 0001 0111
3	Machine Hexadecimal	C 9 0 F D A A 2 2 1 7
4	True Hexadecimal	3.2 4 3 F 6 A 8 8 8 5
5	Binary Coded Decimal	0011 0001 0100 0001 0101 1001 0010 0110 0101 0011 0110

Fig. 4. Forms of number representation

The binary point does not appear explicitly in any punching or printing operation, but it is always understood to be between the second and third binary positions from the left. Ignoring the binary point it is seen that the machine hexadecimal form may be obtained from the true hexadecimal by multiplying the latter by 4, and vice versa.

To convert a number from decimal to machine hexadecimal it is convenient to use a slight adaptation of the method of par. 11.1: Multiply by 4 to get the first hexadecimal digit, and by 16 to get succeeding hexadecimal digits. The converse problem is to interpret decimally a hexadecimal form printed out by the output typewriter. To perform this conversion, the "hexadecimal point" may be assumed to be at the extreme left. Convert this fractional number to decimal by any of the methods discussed in Sect. 11, and multiply the result by 4. This is the desired decimal equivalent.

Exercises:

1. Convert the following numbers to the machine hexadecimal form for punching on teletype tape: 3.209125, 10^{-3}, $1/2\pi$= .15915 49431 218

2. Convert the following numbers, assumed to have been printed out by the typewriter, to decimal: 10907 DC192 5, 00A3D 70A3D 7, 66666 66666 6.

Answers:

1. CD624 DD2F1 B, 00106 24DD2 F, 0A2F9 836ED 2

2. .25881 90451 000, .01000 00000 000, 1.60000 00000 000

12.3 Binary Coded Decimal Representation. In general it is desirable to have facilities for feeding input data, and receiving output data in decimal form. This may be done by representing decimal numbers in "binary coded decimal" form. In this representation, each decimal digit is represented by a group of four binary digits. For example, the number on line 1 in Fig. 4 in binary coded decimal form would be written as shown on line 5. Since decimal and hexadecimal notations are identical for digits 0, 1, ...,9, the Teletype apparatus would here print 3 14159 26536. It should be noted that in a single cell a number can be represented in binary coded decimal form to at most 11 significant decimal digits. On the other hand, the 44 digits of a number stored in binary form, as on line 2, are equivalent to slightly over 13 (i.e., $44 \div 3.3$) decimal digits (line 1). A subroutine (decimal to binary conversion [9.4]) is available to convert from binary coded decimal form to binary. Another subroutine (binary to decimal conversion [9.4]) converts from binary to binary coded decimal.

13.0 Instruction in 4-Address System. In this and the following sections the discussion will be primarily in terms of the 4-address system, since that is the system, which was first installed, and which has been used almost exclusively to date. Most of the discussion, however, will also apply to the 3-address system. The 3-address system has been described in Part I, par. 8.2. It will be discussed further in Sect. 20.

In the 4-address system the form of an instruction word is:

	1st address	2nd address	3rd address	4th address	Operation	Sign
Usual Notation	α	β	γ	δ	Op	$\pm$
No. of Binary Digits	10	10	10	10	4	1

The binary point has no significance in an instruction. The sign is usually plus. A negative sign at the end of an instruction will cause the machine to halt after the execution of the instruction, provided the switches on the manual panel are properly set. The point at which such a halt is programmed is called a "breakpoint".

The symbols α, β, γ, δ stand for 10 binary digit addresses, ten positions being required to refer to $2^{10} = 1024$ addresses. Operations in both the 3- and 4-address systems are represented by a 4 binary digit (1 hexadecimal digit) code. This code is fully described in the accompanying table. In this table and elsewhere in this Handbook, the notation (x) means "the contents of address x".

Operation Symbol	Operation	Result of Operation (Next Instruction Always in δ Unless Otherwise Specified)
0	Input	If β is odd, 1 word is read from Input medium into γ. If β is even, and $\gamma \equiv 0 \pmod 8$, then 8 words are read into $\gamma, \gamma + 1, \ldots, \gamma + 7$. Designation of Input medium is by means of 5 digit binary code number occupying last 3 positions of α and first two positions of β, and by switch settings on External Selector Panel (cf. Refs. 8.1 and 8.2).
1	Logical Transfer (Extraction)	Digits of (γ) in positions which correspond to 1's in (β) are replaced by corresponding digits of (α).
2	Clear	Clear (γ).
3	Logical Product	(γ) is the digit by digit product of (α) and (β). Hence, the logical product of two numbers either of which is positive, is positive; the logical product of two negative numbers is negative.
4	Subtraction	$(\gamma) = (\alpha) - (\beta)$. (Cf., also par. 14.3)
5	Addition	$(\gamma) = (\alpha) + (\beta)$. (Cf., also par. 14.3)

Operation Symbol	Operation	Result of Operation (Next Instruction Alsays in δ Unless Otherwise Specified)
6	File Order	Has meaning in 3-address system only. (γ) = NFFFFUVWXYZ, the letters being hexadecimal digits. N is 0 or 1, indicating address counter in operation, UVW is contents of address counter C_0, and XYZ is contents of address counter C_1.
7	Reverse	For magnetic tape only. Reverse through one word or eight words, according as β is odd or even, respectively.
8	Multiplication (high order)	$(\gamma) = p_{-1}\, p_0 .\, p_1 \cdots p_{42}$ (Cf., also, par. 14.3)
9	Multiplication (rounded)	$(\gamma) = p_{-1}\, p_0 . p_1 \cdots p_{42} + p_{43} \cdot 2^{-42}$ (Cf., also, par. 14.3)
A	Multiplication (low order)	$(\gamma) = 00.p_{43} \cdots p_{84}$ Sign is that of high order product.
B	Division	$(\gamma) = (\beta)/(\alpha)$. The quotient is rounded off, never rounded up. If overflow occurs, the result, though it could be determined in any given case, is not useful (cf. par. 14.3). If $(\alpha) = (\beta) = 0$, then (γ) = FFFFF FFFFF F+.

where $(\alpha) = a_{-1} a_0 . a_1 \cdots a_{42}$, $(\beta) = b_{-1} b_0 . b_1 \cdots b_{42}$, $(\alpha)\cdot(\beta) = p_{-3} p_{-2} p_{-1} p_0 . p_1 \cdots p_{84}$, the a_i, b_i, p_i being binary digits.

Operation Symbol	Operation	Result of Operation (Next Instruction Always in δ Unless Otherwise Specified)
C	Algebraic Comparison	Control goes to γ for next instruction if $(\alpha) < (\beta)$, and to δ if $(\alpha) \geq (\beta)$. This operation treats -0 same as +0.
D	Absolute Value Comparison	Control goes to γ for next instruction if $\lvert(\alpha)\rvert < \lvert(\beta)\rvert$, and to δ if $\lvert(\alpha)\rvert \geq \lvert(\beta)\rvert$. This operation treats -0 same as +0.
E	Base Operation	<u>In 4-address</u>: If $\alpha \geq \beta$, control goes to δ, and β is stored in Base Counter. (Note: References in this statement are to α and β, <u>not</u> (α) and (β).) All references in subsequent instructions to odd addresses are interpreted relative to β, while even addresses are unchanged, until an E instruction is reached where $\alpha < \beta$. Base Counter is then cleared, and control goes to γ. <u>In 3-address (see Sect. 20)</u>: If $\alpha \geq \beta$, and quantity g is in operating address counter, then according as b is 0 or 1, either $\beta + 1$ or $\beta + 1 + g$ is set up in address counter designated by d, and the routine proceeds from that counter setting. If $\alpha < \beta$, then γ is set up in address counter designated by d.
F	Output	If β is odd, 1 word is read from γ onto output medium. If β is even, and $\gamma = 0 \pmod 8$, then 8 words are read from γ, $\gamma + 1$, ..., $\gamma + 7$. Designation of Output medium is

Operation Symbol	Operation	Result of Operation (Next Instruction Always in δ Unless Otherwise Specified)
F (cont'd)	Output	accomplished in same way as for input instruction.

Notes: (1) -0 may be produced by operations:

(a) 8, 9, A, or B where the result is zero (possibly because of overflow) and the signs of the operands differed;

(b) 4 or 5, where the true result is -4;

(c) 1 and 3.

(2) Whenever the result of an add or subtract operation is zero, the machine gives + 0.

(3) It has been pointed out in par. 12.0 that the fixed binary point location is immaterial in addition and subtraction operations, but is significant in multiplication and division. It is helpful to notice also that low order multiplication may be considered as conventional multiplication with the binary point at the extreme right, with the special feature that the two extreme left hand digits in the product are always zero (cf. par. 14.5).

13.1 Ways in Which an Instruction May be Written. An instruction may be written down in a number of different ways, all equivalent, depending on the tastes of the coder. Some of these are illustrated in Fig. 5. Line 2 illustrates an instruction in binary form, and it is seen to conform to the composition given in par. 13.0. This representation corresponds most closely to the form in which the instruction is stored within the machine; i.e., 1's and 0's corresponding to pulses or no pulses, respectively. However, in coding a routine, it is inconvenient to write instructions directly in binary. The binary representation is long and it is easy to make errors in writing down, reading, or transcribing a lengthy series of 0's and 1's. It is more convenient to number the addresses and to code in decimal or in hexadecimal. It is easily seen, for example, that the instruction on line 2 could be written as on line 5 in decimal, or as on line 4 in hexadecimal. In any case, however, before punching the code on teletype tape, it would first be necessary to rewrite the code in appropriate form. Just as in the case of numbers, this appropriate form is referred to as "machine hexadecimal", and is illustrated on line 3. It is obtained as in par. 12.2, by dividing the word into eleven groups of four binary digits each, and a sign, and substituting for each group its hexadecimal equivalent. This representation is also often called the "composed form" of the instruction. The "true hexadecimal" representation on line 4, in which each address is written as a group of 3 hexadecimal digits, is also frequently called the "decomposed form" of the instruction.

1	Notation	α	β	γ	δ	Op	S
2	Binary	0111001001	0111001001	0100101000	1111100110	0101	+
3	"Machine Hex."	7 2 5	C 9	4 A 3	E 6	5	+
4	"True Hex."	1C9	1C9	128	3E6	5	+
5	Decimal	457	457	296	998	5	+

Fig. 5. Forms of instruction representation

The rewriting in machine hexadecimal may be seem like unnecessary duplication of labor, and the coder may prefer to code directly in this ultimate system. While this is quite practical with a little experience, it may seem rather awkward at first. For example, it will be noticed in the foregoing illustration that, inasmuch as 4 is not a factor of 10, the last two binary digits of the α address may be combined with the first two binary digits of the β address to form the third hexadecimal digit on line 3. Similarly, the last two digits of the γ address must be combined with the first two digits of the δ address to form the eighth hexadecimal digit on line 3. Thus the machine hexadecimal form of the instruction does not exhibit as readily as the other forms given the actual addresses being referred to. Indeed, it will be noted in the example that although the α and β addresses are identical, this fact is not immediately evident in the machine hexadecimal representation. (A similar statement would apply for identical γ and δ addresses.) Upon closer inspection, however, it will be observed that the first five hexadecimal digits in line 3 may be obtained from the α and β addresses in line 4 by the formula $2^{10}\alpha + \beta$.

Similarly the next five hexadecimal digits may be obtained from line 4 by the formula $2^{10}\gamma + \delta$. The procedure for converting an instruction from pure hexadecimal to machine hexadecimal may be reduced to a simple rule:

(1) Multiply the α (or γ) address, expressed in hexadecimal, by 4.

(2) Align the right hand hexadecimal digit of this product with the left hand hexadecimal digit of β (or δ), and add.

The sum gives the first five (or second five) digits of the machine hexadecimal representation. It has been found convenient to use an auxiliary multiplication table to facilitate the hexadecimal multiplication by 4.

Some of the awkwardness of coding directly in machine hexadecimal may be alleviated by coding on a form with a double column down the side. One column lists all the addresses serially, in hexadecimal. The second column is 4 x (1st column) so that this form also constitutes a convenient multiplication table. Each instruction is written on the line corresponding to its address, and the rule just given for composing instructions in machine hexadecimal may be readily applied. An example of the use of such a form is given in the next section.

14.0 Coding of Routines. The solution of any problem requires the introduction into the machine of an appropriate routine, consisting of instructions and numerical information. The planning of a routine generally involves many considerations apart from the mathematical formulation; e.g., considerations relating to the range of magnitudes of quantitites involved in the calculations, cell capacity, memory capacity, number of significant figures maintained during calculations, time of computation, etc. Some of these matters will be discussed in the following paragraphs.

14.1 Example of a Routine. To illustrate the preparation of a simple routine the following exercise will be proposed: To produce the first 100 triangular numbers. (The nth triangular number is defined to be the sum $\sum_{k=1}^{n} k$).

The routine may be written for any part of the memory. In this case, it will be put in successive cells starting with 008. The routine is as follows:

4 x Address	Address	Instruction in Machine Hex	Instruction in True Hex α β γ δ Op	Explanation
020	⌈008	03C0E 03C09 5	00F 00E 00F 009 5	$n = 1 + (n-1)$
024	009	0400F 0400A 5	010 00F 010 00A 5	$\sum_1^n k = n + \sum_1^{n-1} k$
028	00A	00001 0400B F	000 001 010 00B F	Print $\sum_1^n k$
02C	00B	03C0D 0200C D	00F 00D 008⌋ 00C D	$n < 100?$ Yes $\longrightarrow$ 008 No $\longrightarrow$ 00C
030	00C	00000 00000 C-	000 000 000 000 C-	Halt
034	00D	00000 00006 4		$100 \cdot 2^{-42}$
038	00E	00000 00000 1		$1 \cdot 2^{-42}$
03C	00F	[00000 00000 0]		n
040	010	[00000 00000 0]		$\sum_1^n k$

The instructions have first been written in true hexadecimal, and the conversion to machine hexadecimal is facilitated by the presence of the two columns to the left. Each word in the routine is explained in the right hand column.

Though extremely simple, this example illustrates a number of points about SEAC coding. It will be observed that at the end of the fourth instruction the control is sent back to address 008, provided that $n < 100$. Each time through this cycle of four instructions a new triangular number is generated. This iterative type of computation is typical of most, if not at all, routines. An iteration is frequently enclosed in half brackets, ⌈ ⌋ , as in this example, to make it more conspicuous. No entries have been made in the last four lines of the fourth column, because the words in these addresses are numerical quantities and would not normally be written in the true hexadecimal form of instructions. Brackets, such as in the last two lines of column 3, mean "subject to change". In this case, 00F and 010 initially contain zeros, but are used in the course of the computation to store the successive values of n and $\sum_{1}^{n} k$. Since these quantities quickly become greater than 4, all integers are multiplied by the scale factor 2^{-42} so that they can be accommodated in the storage. With this choice of scale factor the 100th triangular number clearly does not overflow. At the end of the 100th iteration 00F will contain $100 \cdot 2^{-42}$, so that the comparison in address 00B will send the control to 00C. The instruction in 00C clearly does not affect the contents of any cell in the memory, but since it is followed by a negative sign, it will cause the machine to halt (with proper initial settings of the switches on the manual panel). In order to make this routine complete it is necessary to add proper input instructions, and this will be explained in the following paragraph.

14.2 Method of Getting Information Into Machine. As explained in Part I, par. 5.0, instructions are held in the instruction register while being interpreted and carried out. When the computer is halted, the last instruction executed remains in the instruction register. If the MEMORY CLEAR button on the manual panel is now depressed the contents of the memory will become all zeros, but the instruction in the instruction register remains unaffected. It the START button is now pressed, the δ of that instruction specifies the address of the next instruction. But since the memory has been cleared this is (when written in decomposed hexadecimal form):

(1) 000 000 000 000 0+.

The interpretation of this instruction is: "Read eight words into cells 000 through 007, and go to 000 for the next instruction." For the example of the foregoing paragraph, it is readily seen that this next instruction must be an input instruction:

	4 x Address	Address	Instruction in Machine Hex	Instruction in True Hex α β γ δ Op	Explanation
(2)	000	000	00000 02008 0	000 000 008 008 0	Read 8 words into cells 008 - 00F

As a result of this instruction the routine of par. 14.1 will be read in and the control will then be sent to the first instruction of this routine. The remaining seven words called in by the instruction (1) could be anything, since they are never referred to. The components of the routine would be punched on Teletype tape in the following order:

(a) the input instruction (2),

(b) seven words of arbitrary information,

(c) the first eight words in column 3 of the routine in par. 14.1.

Clearly the routine of par. 14.1 could have been coded to go into tank 0 of the memory; i.e., into addresses 000-007 (see Footnote, Part I, par. 8.1). It could then have been read into the memory at once by activating the instruction (1) in the manner described, and no other read-in instruction such as (2) would have been necessary. If, however, the routine of par. 14.1 were to consist of a number of tanks of information then an equal number of read-in instructions such as (2) would be required, and it would be convenient to use tank 0 for the purpose of storing or generating (cf. Sect. 17) these instructions. After the information has been read into the memory, the read-in instructions are no longer needed, and the addresses they occupy are available for other purposes. It is a generally recommended practice to keep tank 0 available for contingencies that arise in the course of operating the machine. For example, one may wish to insert corrections, additions, or other modifications to the routine after it has been fed into the memory. Or, at some stage of the computation it may be decided to "dump" the contents of the memory onto magnetic wire for storage or for checking. To accommodate the instructions for doing such things as these and other auxiliary operations, it is necessary at least to have address 000 available, and preferably all of tank 0. The examples in this Handbook conform to this recommendation.

The design of the manual panel makes it especially convenient to reserve tank 0 for the uses described. For, by simple manipulations of the switches it is possible to clear the instruction register, thus activating the instruction (1). It is likewise possible to activate the instruction for a single word input: 000 001 000 000 0.

This last facility makes it easy to introduce a single word into any desired address at any time. For example, suppose it is desired to read a word W into address 12A, and to then send the control to 086. This may be done by first activating the single word input instruction above, and then typing on the keyboard the instruction 00001 4A886 0+ (i.e., 000 001 12A 086 0+, in decomposed form). This is another single word input instruction which will put the word W in its assigned location. The actual manipulations of the controls for performing these and other operations will not be described here. Basic instructions for operations of the manual controls are given in Refs. 9.1, 8.1, 8.4, 8.5.

14.3 Overflow. In operations 4,5,8 and 9, binary digits which overflow are lost. In the B operation, if overflow occurs (that is, if $|(\beta)/(\alpha)| \geq 4$), the result of the operation, although it could be determined in any given case, is not useful.

14.4 Scaling. If quantities larger than 4 appear in the data or may appear in the computation, difficulties may frequently be avoided by "scaling". The data may be multiplied by a constant small enough to make all quantities less than 4 before they are read in, or the multiplication may be done internally. In the triangular number routine, a scaling factor of 2^{-42} was used. In other cases it is desirable to scale in the other direction; i.e., to make quantities larger. In nearly all cases the scaling factor must be kept in mind as the coding is done. If the ranges of the quantities appearing in a computation are very large, it may be necessary to resort to storage of numbers in double precision form, or in floating binary point form. The ordinary arithmetic operations can then no longer be performed by single instructions, but require the use of rather elaborate subroutines (cf. Sect. 19, and Refs. 9-13).

14.5 Low Order Multiplication. The appearance of the 2 zeros to the left of the binary point in low order multiplication should be borne in mind each time that instruction is used. Thus (using the symbol A to denote low order multiplication), $(p \cdot 2^{-m})\ A\ (q \cdot 2^{-n}) = pq \cdot 2^{42-(m+n)}$ provided $pq < 2^{(m+n)-42}$. Otherwise the integral part of the low order product, $pq \cdot 2^{42-(m+n)}$, is lost. Examples of the results of this instruction are:

(1) $2^{-42}\ A\ 2^{-1} = 2^{-1}$ (2^{-42} acts as a unity element),

(2) $p \cdot 2^{-28}\ A\ 2^{-32} = p \cdot 2^{-18}$ for $p < 2^{+18}$ (a left shift of 10 places),

(3) $(4-2^{-42})\ A\ 2^{-42} = 1-2^{-42}$ (because of the dropping of the two 1's to the left of the binary point).

14.6 Shifting. Right and left shifts are frequently needed in coding. Right shifts may be accomplished by multiplication by a negative power of 2. A left shift of one binary place may be accomplished by adding the quantity to itself. Other left shifts may be made by low order multiplication (as in the second example of the preceding paragraph) if the two most significant binary digits are not needed, or by division, if no overflow will occur. If the two left hand digits are needed, and overflow may occur, the shift cannot be made in a single instruction.

14.7 Example. The D and F type bars on the Teletype printer or Flexowriter may be replaced by a decimal point and space respectively. This facility permits the coder to improve the appearance of results by inserting decimal points, and suppressing unwanted information.

Assume that a coded decimal quantity is in cell 100, in the form

$$d_1 \cdot 2^{-26} + d_2 \cdot 2^{-30} + d_3 \cdot 2^{-34} + d_4 \cdot 2^{-38} + d_5 \cdot 2^{-42},$$

that the decimal point is between d_1 and d_2, and that d_4 and d_5 are not significant. Prepare a routine for printing numbers of this form out as far to the left as possible, beginning the instructions in cell 030. The printed copy should exhibit the decimal point, and any non-significant figures are to be suppressed.

Solution: The code is as follows:

4 x Address	Address	Instruction in Machine Hex	Instruction in True Hex α β γ δ Op	Explanation
0C0	030	0D900 0F431 A	036 100 03D 031 A	Shift left 20 places
0C4	031	0E037 0F432 1	038 037 03D 032 1	Insert dec. pt. and spaces
0C8	032	0E500 0F033 B	039 100 03C 033 B	Shift left 24 places
0CC	033	0F035 0F434 1	03C 035 03D 034 1	Insert d_1
0D0	034	00001 0F400 F-	000 001 03D 000 F-	Read Out
0D4	035	F0000 00000 0		$F \cdot 2^{-2}$
0D8	036	00000 10000 0		2^{-22}
0DC	037	0F00F FFFFF F		Extractor
0E0	038	0D00F FFFFF F		Dec. Pt. and spaces for extraction
0E4	039	00000 04000 0		2^{-24}

14.8 Selection of Input and Output Media. There are at the present time (September 1952) 12 units from which it is possible to read information into the memory, or onto which it is possible to record information from the memory. These are (see also Summary of SEAC Specifications):

(1) 1 Teletype unit - includes tape, reperforator and typewriter,

(2) 2 magnetic wire units, one for input and the other for output,

(3) 4 reel-less magnetic tape units, single channel,

(4) 1 servo-magnetic tape unit, 5-channels (count as 5 units)[1].

Any combination of units, up to 10 units, can be used in a single routine. As pointed out earlier (Part I, pars. 3.0, 7.4; Part II, par. 13.0) the selection of the medium used in a given input or output instruction is determined both by the code and the external selector panel switch settings. For both input and output instructions the first 19 binary positions (constituting the α address and all but the unit's position in the β address) have not yet been assigned any significance. The 5 positions consisting of the 3 lowest order positions of α and the 2 highest order positions of β are used to identify correspondingly numbered switches (of which there are 10) on the panel. Each switch has 10 possible settings and each setting selects an input-output unit. When the tape reverse operation, 7, is used, the desired magnetic tape is selected in the same way as for input and output instructions. In addition to the facilities described above, a "Direct Selection" switch is present which enables one to override the selection specified by an instruction. By means of this switch one can substitute any desired unit in place of the unit indicated by the instruction word (even when the instruction word fails to indicate any active code number).

Details concerning the use of the external selector panel, and the selection code are given in Refs. 8.1 and 8.2. The reader should not attempt

[1] Currently being installed.

to prepare a complete program without first studying these references.

14.9 Compression: For many types of operation with magnetic media, input and output speeds may be materially increased by reducing the gaps between blocks (1 word or 8 words) of recorded information. A longer gap is required when the magnetic unit has to stop while the machine computes, and then restart, than when the input or output instructions follow each other in quick succession. The reduction of gap size is called "compression". It is accomplished by coding in α of all pertinent read-out instructions a "1" in the 4th binary position from the left. Input and output speeds may be increased by a factor of as high as ten by the use of the compression feature. The greater the number of blocks read in or out under compression, the greater the saving. A greater time saving is achieved by use of the compression feature on magnetic wire than on magnetic tape. The tape and wire speeds given in the Summary of SEAC Specifications are maximum speeds under compression.

Further details on the use of the compression feature are given in Refs. 8.2 and 8.4. Again the reader should not attempt to prepare a complete program without first studying these references.

15.0 Flow Diagrams. The first step in the programming of most problems is the preparation of a "flow diagram". This is a block diagram or schematic which presents an overall picture of the calculation procedure. It serves a number of purposes:

(1) Once a careful flow diagram has been worked out, writing the code is much simplified. Careful attention must nevertheless be given to such matters as scaling.

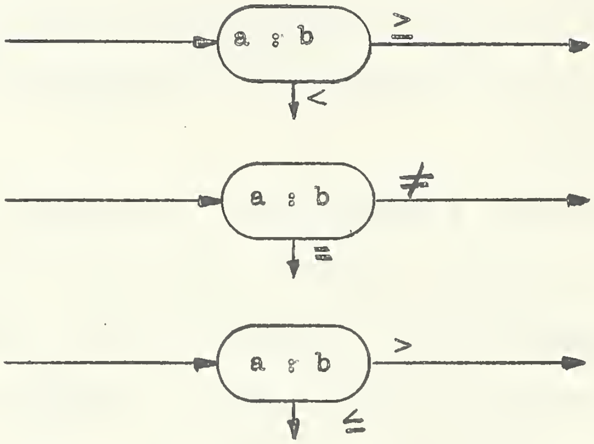
(2) The flow diagram presents a graphic view of the calculation process. It is much easier to grasp the logical organization of a program by looking at this diagram than by looking at the code. The flow diagram is therefore a convenient and useful means of communicating information.

(3) It is frequently necessary to correct, amend, or modify a code. A change in one part of a code generally entails changes in other parts also. Since the flow diagram shows more conspicuously than the code the inter-relationship between the various parts of the program, it is a recommended practice to first make any required changes in the flow diagram, then in the code.

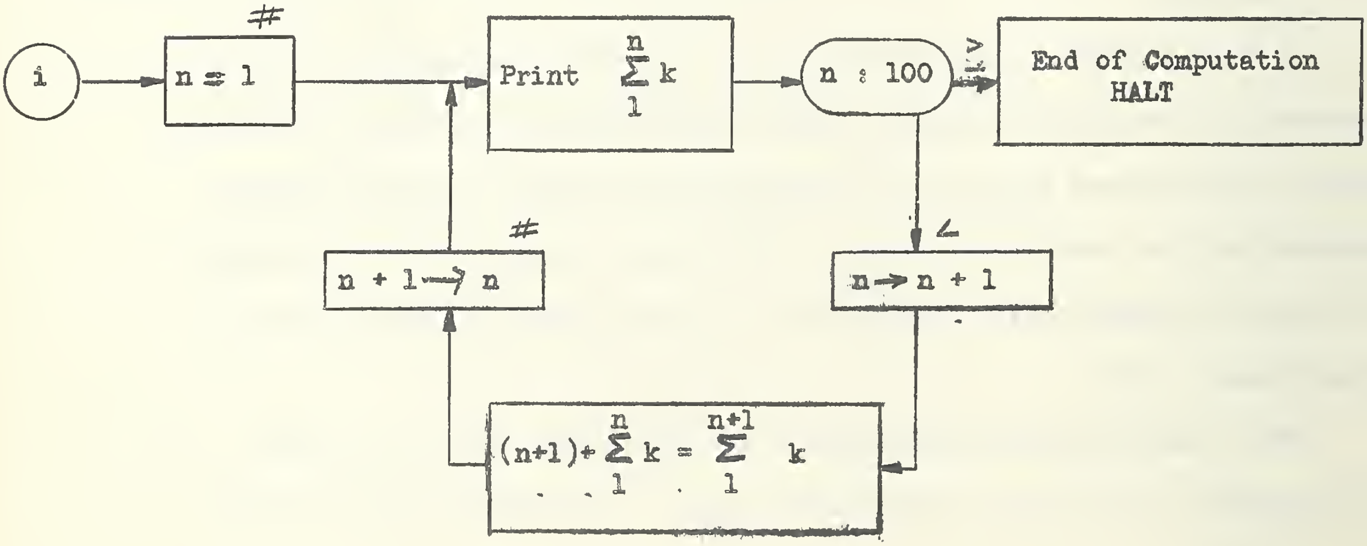
A flow diagram may be drawn up in as much detail as desired. At one extreme only the broad outlines of the calculation may be shown, or at the other extreme there may be a 1-1 correspondence between the flow diagram symbols and the coded instructions. Sometimes it may be found convenient to have two types of flow diagrams for a given program, a rough one and a more detailed one.

For a comprehensive discussion of the procedures for flow diagram preparation, the reader is referred to Ref. 1. The flow diagrams in this

Handbook for the most part use the conventions and notation described in that reference. One minor deviation is in the symbolism for the "comparison box". The flow diagrams included in this report indicate comparisons in the following ways:



As a simple example, the flow diagram for the triangular number routine in par. 14.1 would appear as follows:

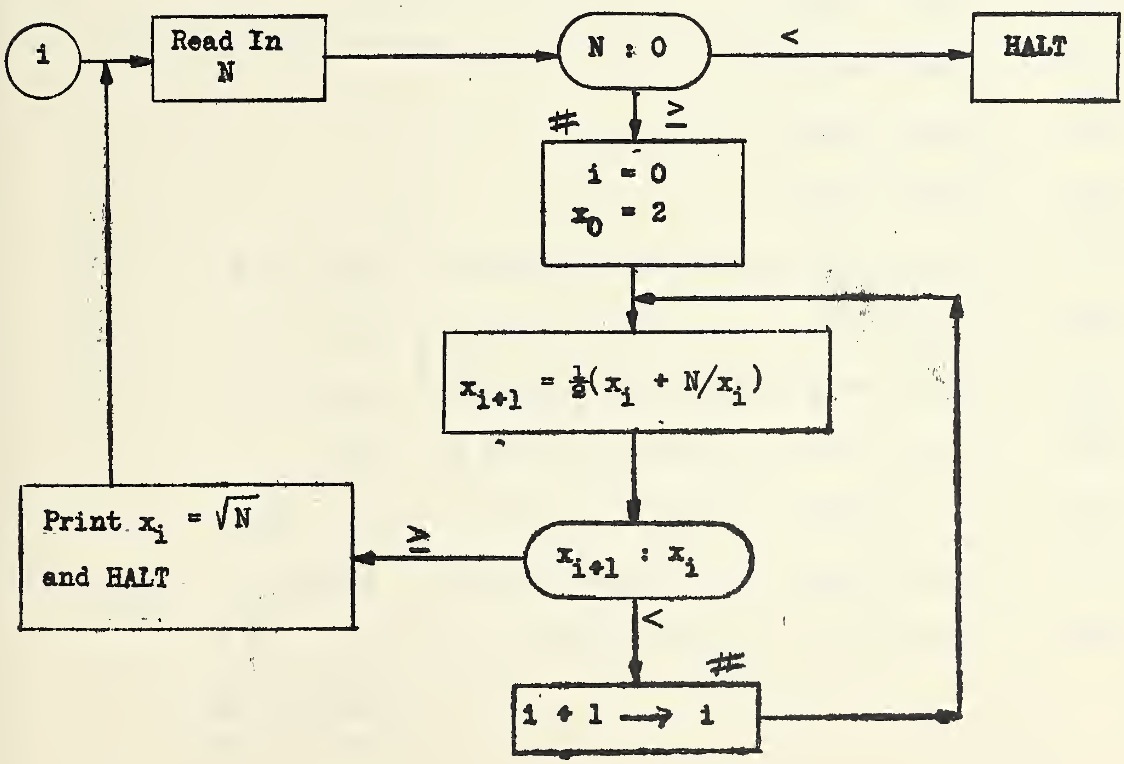


15.1 Example. Prepare a flow diagram and code for computing $\sqrt{N}$, $(N < 4)$, by solving the equation $x^2 = N$ using the Newton-Raphson method [7].

Solution. Denoting the i^{th} approximation to the root by x_i, the Newton-Raphson method yields for the $(i+1)^{th}$ approximation:

$$x_{i+1} = \tfrac{1}{2}(x_i + N/x_i),$$

An initial approximation near zero would produce a larger new approximation and there is danger of overflow. An approximation which is too large is preferable; and since $N < 4$ in this case, $x_0 = 2$ is satisfactory. With this choice of x_0 it follows that $x_{i+1} < x_i$. This is evident, for example, from a graphical interpretation of the recurrence relation. The flow diagram can be drawn as follows:



Note that both the flow diagram and code below are arranged to call in another argument after each computed result is printed. All that is necessary to continue from the halted condition is to press the START button. Note also that the printed results will be in binary form. The coding including read-in instructions may be done as follows:

4 x Address	Address	Instructions in Machine Hex	Instructions in True Hex α β γ δ Op	Explanation
000	000	00000 02001 0	000 000 008 001 0	Read in
004	001	00000 04008 0	000 000 010 008 0	Read in
008	002	00000 00000 0		
00C	003	00000 00000 0		
010	004	00000 00000 0		
014	005	00000 00000 0		
018	006	00000 00000 0		
01C	007	00000 00000 0		
020	008	00001 05809 0	000 001 016 009 0	Read in N
024	009	05813 0440A C	016 013 011 00A C	Is $N < 0$?
028	00A	04813 0500B 5	012 013 014 00B 5	Store x_o
02C	00B	05016 0540C B	014 016 015 00C B	N/x_i
030	00C	05015 0540D 5	014 015 015 00D 5	$x_i + N/x_i$
034	00D	04815 0540E B	012 015 015 00E B	$\frac{1}{2}(x_i + N/x_i) = x_{i+1}$
038	00E	05414 03C10 C	015 014 00F 010 C	$x_{i+1} < x_i$? Yes → 00F No → 010

4 x Address	Address	Instruction in Machine Hex	True Hex α β γ δ Op	Explanation
03C	00F	05413 0500B 5	015 013 014 00B 5	$x_{i+1} \rightarrow x_i$
040	010	00001 05008 F-	000 001 014 008 F-	Read out $\sqrt{N}$, Halt.
044	011	00000 00000 C-	000 000 000 000 C-	Halt.
048	012	80000 00000 0		+2
04C	013	00000 00000 0		+0
050	014	00000 00000 0		
054	015	00000 00000 0		
058	016	00000 00000 0		
05C	017	00000 00000 0		

16.0 Preliminary Coding in Terms of Symbolic Addresses. It is recommended that after completion of the flow diagram for a problem, the code be prepared first in terms of "symbolic addresses" instead of actual numerical addresses. The latter normally would not be assigned until nearly the last stage of the code preparation process. A common practice is to classify storage locations according to the kinds of words they are to store, and to assign a letter to each classification. Distinct storage locations within each classification are then identified by distinct subscripts. For example, it is usually found that certain addresses are used exclusively for constant quantities, others exclusively for instructions, etc. Still others are working positions, which may store different categories of words at different stages of the calculation. Such positions have been referred to among Computation Laboratory staff as "Temporaries". They may be left empty by the read-in instructions. Following are some storage classifications and symbols that have been used in the initial preparation of a number of SEAC codes:

Constants	$K_1, K_2, \ldots$
Temporaries	$T_1, T_2, \ldots$
Variables	$V_1, V_2, \ldots$
Instructions	$L_1, L_2, \ldots$

In practice it is feasible to mix actual addresses with symbolic addresses in the initial code. Care should be taken not to confuse these symbolic addresses with the contents of the addresses; e.g., the contents of the fixed address L_1 may be modified during the course of the calculation in ways to be described in Section 17. In general the choice of notation may be left to the discretion and taste of the coder.

There are several advantages in first coding in the manner described above: (1) The notation may be chosen so as to suggest the categories of quantities being worked upon. This makes it easier not only to write the code in the first place, but to read it afterwards and to check it visually. Thus, for example, using the illustrative notation above, the instruction

$$K_1 \quad V_2 \quad T_2 \quad L_3 \quad 8$$

clearly conveys the meaning, "the constant in address K_1 is multiplied by the variable in V_2, the result goes to the temporary storage cell T_2, and the next instruction is found in L_3". This is somewhat more information than can be gleaned from an instruction such as:

054 12A 239 176 8.

(2) It is usually desirable or convenient to assign a block of the memory to each classification of storage location; e.g., one block for constants, one block for variables, etc. However, it is usually not certain until the initial code is completed how large a block is required in each category. Once this is determined, numerical assignments may be made for the addresses. (3) Once a code in terms of actual addresses is completed, any correction in the address of, let us say, a constant would necessitate correcting all instructions where reference is made to this constant. If, however a preliminary code were prepared using a symbol K_1 for the location of this constant, no actual address assignment need be made until this preliminary code is completed. Decisions as to actual assignments made at this later stage are not very likely to require changing afterwards.

16.1 <u>Conversion to Final Code.</u> After the preliminary code has been completed and checked, and the actual numerical address assignments have been made, it can be converted directly to the machine hexadecimal form without first converting to true hexadecimal. This conversion is facilitated by the use of a form similar to that described in par. 13.1 but which contains a third address column on the left. On each line of this column is entered the symbolic address corresponding to the numerical address given (see, for example, codes in pars. 17.3, 17.4).

There are several advantages in first coding in the manner described above: (1) The notation may be chosen so as to suggest the categories of quantities being worked upon. This makes it easier not only to write the code in the first place, but to read it afterwards and to check it visually. Thus, for example, using the illustrative notation above, the instruction

$$K_1 \quad V_2 \quad T_2 \quad L_3 \quad 8$$

clearly conveys the meaning, "the constant in address K_1 is multiplied by the variable in V_2, the result goes to the temporary storage cell T_2, and the next instruction is found in L_3". This is somewhat more information than can be gleaned from an instruction such as:

054 12A 239 176 8.

(2) It is usually desirable or convenient to assign a block of the memory to each classification of storage location; e.g., one block for constants, one block for variables, etc. ~~However, it is usually~~ not certain until the initial code is completed how large a block is required in each category. Once this is determined, numerical assignments may be made for the addresses. (3) Once a code in terms of actual addresses is completed, any correction in the address of, let us say, a constant would necessitate correcting all instructions where reference is made to this constant. If, however a preliminary code were prepared using a symbol K_1 for the location of this constant, no actual address assignment need be made until this preliminary code is completed. Decisions as to actual assignments made at this later stage are not very likely to require changing afterwards.

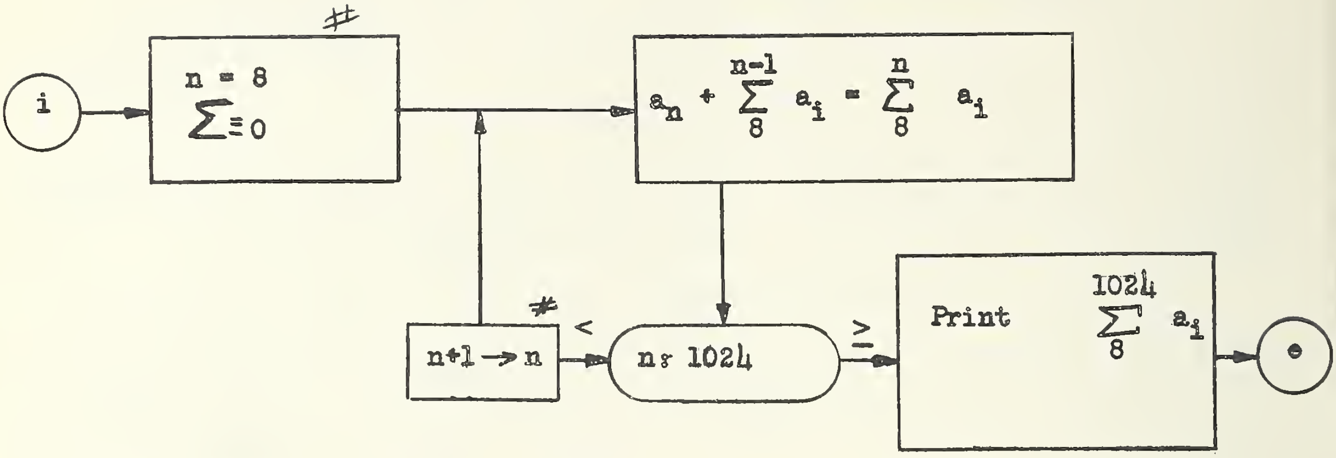
16.1 Conversion to Final Code. After the preliminary code has been completed and checked, and the actual numerical address assignments have been made, it can be converted directly to the machine hexadecimal form without first converting to true hexadecimal. This conversion is facilitated by the use of a form similar to that described in par. 13.1 but which contains a third address column on the left. On each line of this column is entered the symbolic address corresponding to the numerical address given (see, for example, codes in pars. 17.3, 17.4).

17.0 <u>Modification of Instructions.</u> If was pointed out in Part I, par. 1.3 that the ability of the SEAC to operate on and modify its instructions is one of its essential features. Much of the machine's flexibility and versatility springs from this facility. All except the most trivial of routines involves variable instructions. Some common cases will be described.

Frequently it is necessary to subject successive arguments of a given sequence of arguments to a prescribed arithmetic process. This may be done in two principal ways, both of which give rise to variable instructions. (1) The argument about to be worked upon is first transferred to a standard location as a starting point. A variable instruction is required to accomplish this transfer. (2) Each argument is worked on from its initial location. There is no transfer to a standard location and every instruction which refers to the initial locations must be modified in passing from one argument in the sequence to the next. The first method is frequently more economical, both time- and space-wise, when the processing consists of several references to the argument. The second procedure is more economical when each argument is to be operated on only once. This will become more evident in the following two examples.

17.1 <u>Example 1.</u> Prepare a flow diagram and code for summing the memory, starting with address 008, and printing the sum. Neglect all overflows.

Solution: Since the sum is supposed to begin with address 008, this leaves addresses 000-007 available for the summation routine. The summation process involves only a single operation on each quantity - that of adding it into a cumulative sum. It therefore appears that procedure (2) described in paragraph 17.0 is suitable. The flow diagram and code can be drawn up as shown below:



Note 1: $a_i \equiv$ contents of address i,

Note 2: In view of the fact that overflow is being neglected, all sums are understood to be modulo 4.

4 x Address	Address	Instruction in Machine Hex	True Hex α	β	γ	δ	Op	Explanation
000	⌈000	02007 01001 5	[008]	007	007	001	5	$a_n + \sum_{8}^{n-1} a_i = \sum_{8}^{n} a_i$
004	001	00002 00004 D	000	002	003	004	D	n < 1024? Yes → 003 No → 004
008	002	FF000 00000 0	3FF	000	000	000	0	Constant for Comparison
00C	003	00006 00000 5	000	006	000	000	5	Modify (000)
010	004	00001 01C00 F-	000	001	007	000	F-	Print $\sum_{8}^{1024} a_i$
014	005	00000 00000 0						
018	006	00400 00000 0	001	000	000	000	0	1 in α; constant for modification
01C	007	[00000 00000 0]						$\sum_{8}^{n} a_i$; (0 initially)

Note the instructions to be iterated, indicated by the half brackets. In this code, the instruction to be modified is in cell 000, and the modification is accomplished by adding 1 to its α address each time through the iteration. The memory sum is highly useful as a check number in determining whether a routine has gone into the memory correctly, and is almost always inspected before starting in on a computation. In normal practice the δ of the read-out instruction in 004 would be the address of the first instruction of the routine proper. With proper initial switch settings on the manual panel (HALT-PHASE switch on BREAKPOINT) the machine will halt after performing the negative instruction in 004. Then, after the memory sum is inspected and found to be correct, the START button is depressed and the control proceeds to the main computation. If the memory sum is in error, then the routine would normally be fed in again, and another check made.

17.2 Exercises.

1. Prepare a flow diagram and routine to go into tank 0 for filling the memory, starting with address 008.
2. Prepare a flow diagram and routine to go into tank 0 for dumping (i.e., reading-out) the contents of the memory, starting with address 008.

17.3 Example 2. Prepare a flow diagram and routine to compute the quantities

$$u_i = \frac{x_i(1-x_i)}{1+x_i+x_i^2}, \quad i = 1, \dots, 8.$$

Solution. In order to illustrate the techniques of Section 16 an initial code will be prepared using symbolic addresses instead of numerical addresses. The x_i will be assumed to be read into consecutive addresses $V_1, \ldots, V_8$, where we will require that $V_1 \equiv 0 \pmod 8$. Since each x_i must be referred to several times in computing u_i, the first of the procedures of paragraph 17.0 will be found to be most economical, both in terms of space and time. Two variable instructions are required here: one to transfer the x_i to a standard location; and a second to place the u_i, as they are produced, into the addresses where they are to be stored preparatory to printing. The flow diagram for this problem is similar to others that have already been presented. It will therefore not be drawn here but will be left as an exercise for the reader.

In the initial code the notation used for addresses will be the same as in paragraph 16.0. A convenient way to proceed with the coding is to begin with the computation of the successive u_i, and leave the preparatory and other auxiliary steps to be coded later. Brackets, [ ], are used to distinguish addresses which will have to be modified in the passage from i to i+1. Symbolic addresses ($K_1, K_2, \ldots$) for constants, and working positions ($T_1, T_2, \ldots$), are assigned as needed. The problem will be coded with variable instructions being transferred to the working positions for modification and execution.

Symbolic Address	Initial Code α	β	γ	δ	Op	Explanation
$\lceil T_1$	$[V_1]$	K_1	T_2	L_1	5	$x_i \rightarrow T_2$
L_1	T_2	T_2	T_3	L_2	9	$x_i^2 \rightarrow T_3$
L_2	T_2	T_3	T_4	L_3	4	$x_i - x_i^2 \rightarrow T_4$
L_3	T_2	T_3	T_3	L_4	5	$x_i + x_i^2 \rightarrow T_3$
L_4	K_2	T_3	T_3	T_5	5	$1 + x_i + x_i^2 \rightarrow T_3$
T_5	T_3	T_4	$[V_1]$	L_5	B	$u_i \rightarrow V_i$
L_5	T_1	L_6	L_7	L_6	D	$i < 8$? Yes $\rightarrow L_7$ No $\rightarrow L_6$
L_6	V_8	000	V_1	-	F-	Comparison constant; read out $u_1, \ldots, u_8$
L_7	T_1	K_3	T_1	L_8	5	Modify transfer instructions
L_8	T_5	K_4	T_5	$\underline{T_1}\rfloor$	5	
K_1	0					
K_2	1					
K_3	2^{-8}					+1 in α
K_4	2^{-28}					+1 in γ

Thus far the routine will compute the eight u_i, store them, and read them out. Note that the word in L_6 serves a dual purpose: it serves as a constant against which the instruction in L_1 is compared to determine when eight values of u_i have been computed; and it is also an instruction for reading out the 8 words in $V_1, \ldots, V_8$. The δ of this final instruction will be left blank for the time being and will be filled in later after the rest of the routine has been written. In the above routine, the α address of the instruction in T_1 will take on the values $V_1, \ldots, V_8$, and at the end of the series of calculations will stand at the value V_8; and similarly for the γ address of the instruction in T_5. If it is desired to compute the function u for a second set of eight x's it will first be necessary to set up the instructions T_1 and T_5 to their initial states. Likewise, it would be necessary to set up these instructions if during the course of calculating the u_i it is desired for any reason to restart the computing without having to read the routine in again[1]. These two possible contingencies illustrate why it is always wise, if not essential, to preset the initial state of a variable instruction before performing the instruction the first time. The presetting process can be done as follows:

Symbolic Address	α	β	γ	δ	Op	Explanation
L_9	L_{10}	K_1	T_1	L_{11}	5	Set up (T_1)
L_{10}	V_1	K_1	T_2	L_1	5	
L_{11}	L_{12}	K_1	T_5	T_1	5	Set up (T_5)
L_{12}	T_3	T_4	V_1	L_5	B	

[1]The instruction 00000 $00a_1a_2a_3$D+ (where a_1, a_2, a_3 are hexadecimal digits), typed manually on the keyboard or read in on teletype, will send the control to address $a_1a_2a_3$.

To make the routine complete and self-contained we shall add an instruction to read in the x_i:

L_{13}	000 000 V_1 L_9 0	Input instruction: $x_i \rightarrow V_i$, i=1 ..., 8

The δ of the instruction in L_6 will now be filled in as L_{13}. Thus, after reading out eight values of u_i, the routine will call in eight more x_i.

The method (2) of handling the instruction modification would have required four variable instructions instead of two. Each one of these would have had to be set up at the beginning, and each would have had to be modified in passing from u_i to u_{i+1}.

It is now possible to assign addresses in the foregoing code. To do this one need merely write beside the symbolic address for each instruction, independent variable x_i, constant, and working position, its assigned address. By inserting an additional column labelled 4 x Address, it is possible to go through the instructions and write them down in machine hexadecimal form directly without having to go through the intermediate stage of writing them in true hexadecimal. While in practice no rewriting of the initial code is necessary prior to writing the final code, this will nevertheless be done here for purposes of presentation. At the same time the addresses will be rearranged slightly in order to keep words of the same kind together in the memory. The read-in instructions for this routine will be left to the reader.

Assigned Address: 4 x Address	Assigned Address: Address	Symbolic Address	α	β	γ	δ	Op	Machine Hex	Explanation
020	008	L_{13}	000	000	V_1	L_9	0	00000 08009 0	
024	009	L_9	L_{10}	K_1	T_1	L_{11}	5	02815 0640B 5	
028	00A	L_{10}	V_1	K_1	T_2	L_1	5	08015 0680D 5	
02C	00B	L_{11}	L_{12}	K_1	T_5	T_1	5	03015 07419 5	
030	00C	L_{12}	T_3	T_4	V_1	L_5	B	06C1C 08011 B	Read-in x_i,
034	00D	L_1	T_2	T_2	T_3	L_2	9	0681A 06C0E 9	Compute u_i,
038	00E	L_2	T_2	T_3	T_4	L_3	4	0681B 0700F 4	and Read out.
03C	00F	L_3	T_2	T_3	T_3	L_4	5	0681B 06C10 5	
040	010	L_4	K_2	T_3	T_3	T_5	5	0581B 06C1D 5	
044	011	L_5	T_1	L_6	L_7	L_6	D	06412 04C12 D	
048	012	L_6	V_8	000	V_1	L_{13}	F-	09C00 08008 F-	
04C	013	L_7	T_1	K_3	T_1	L_8	5	06417 06414 5	
050	014	L_8	T_5	K_4	T_5	T_1	5	07418 07419 5	
054	015	K_1						00000 00000 0	Constants: 0
058	016	K_2						40000 00000 0	1
05C	017	K_3						00400 00000 0	$2^{-8}=1$ in α
060	018	K_4						00000 00400 0	$2^{-28}=1$ in γ
064	019	T_1							
068	01A	T_2							
06C	01B	T_3							Temporaries
070	01C	T_4							
074	01D	T_5							
080	020	V_1							Storage positions
⋮	⋮	⋮							for x_i and u_i.
09C	027	V_8							

An inspection of this code will reveal that no word in the block of cells from 008 to 018, inclusive, is changed in the course of the calculation. This fact can be used to check the machine by summing this block at intervals during the running of the problem.

17.4 Function Table Technique of Instruction Modification. Up to now we have been modifying instructions in a regular manner. It is sometimes necessary to set up an instruction as a function of an argument which varies in an unpredictable fashion. The instruction is made to depend on the argument by introducing the argument into one or more address components of the instruction. The technique is illustrated in the following example.

Example. A sequence of "quasi-random" numbers n_m, $m = 0, \ldots, M$ can be generated by low order multiplication of $n_0 = 5^{17}$ (with scale factor 2^{-42}) by itself to produce the first number, low order multiplication of this result by 5^{17} (again with scale factor 2^{-42}) to produce the next number, etc. Each number, n_m, obtained in the sequence becomes the multiplicand for the constant multiplier $5^{17} \cdot 2^{-42}$. Determine a frequency distribution for this sequence of numbers, using 32 equal class intervals.

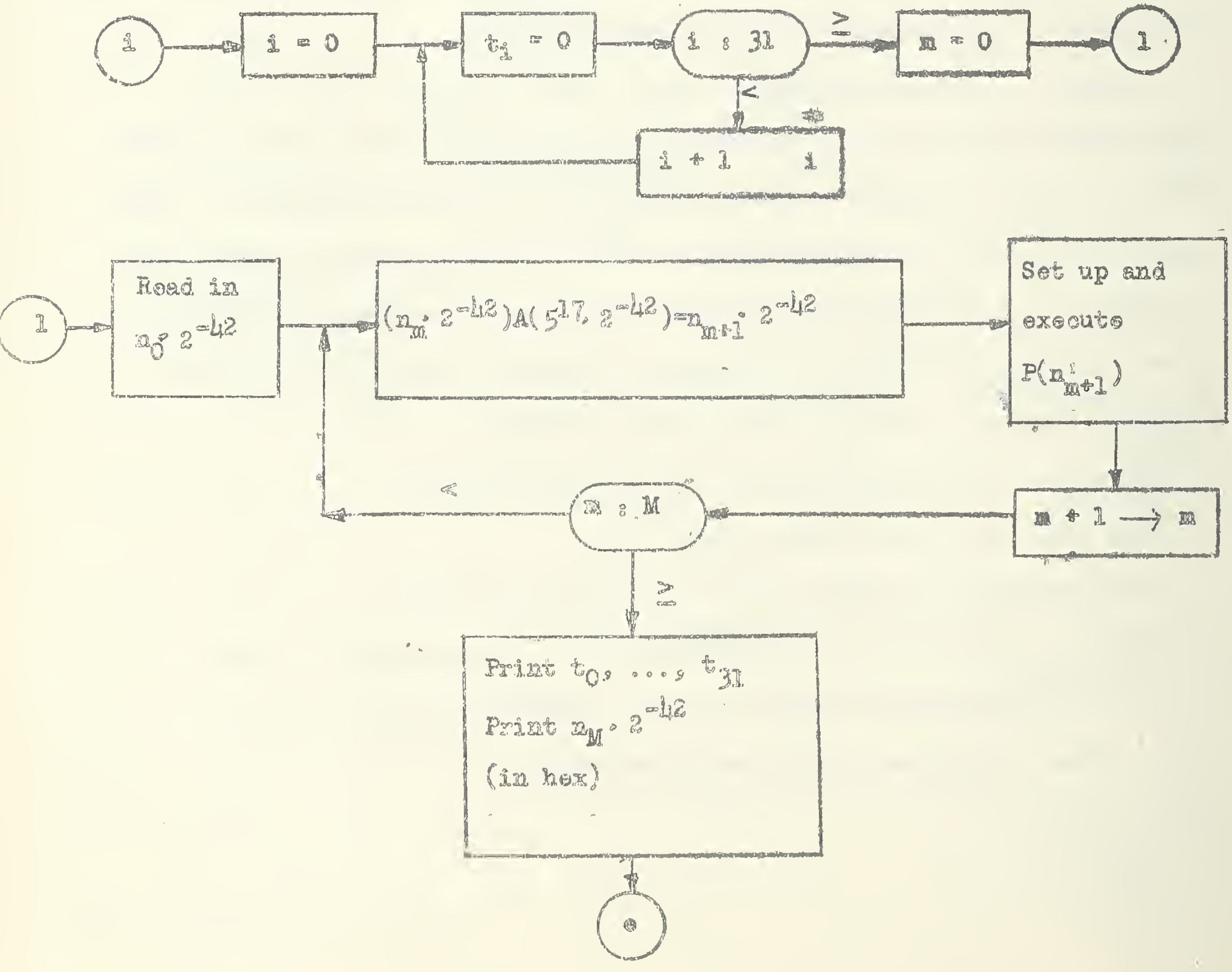
Solution. The procedure may be outlined as follows:

(1) Generate a quasi-random number, n_m.

(2) Determine the class interval to which it belongs.

(3) Tally one in cell corresponding to this class interval. The instruction which does this is a variable instruction and is in fact a function of the number generated.

The 32 class intervals are bounded by the points $n=0,\ 1\cdot 2^{37},\ 2\cdot 2^{37},\ \ldots,$ $31\cdot 2^{37},\ 32\cdot 2^{37} = 2^{42}$. Every integer, n'_m, composed of the five most significant binary digits of n_m, corresponds to one of these points, and in fact can be used to identify the class interval to which the quantity n_m belongs, since $n'_m\cdot 2^{37} \leq n_m < (n'_m + 1)\cdot 2^{37}$. The integer, n'_m, is the argument for setting up the variable instruction in (3), which we denote by the functional notation $P(n'_m)$. We shall write t_i, $i=0, \ldots, 31$, for the tally corresponding to the interval $i\cdot 2^{37} \leq n < (i+1)\cdot 2^{37}$. The instruction $P(n'_m)$ then changes $t_{n'_m}$ to $t_{n'_m} + 1$. All the tallies are set to zero before the actual computation is begun. Following is the flow diagram for this problem.



Legend:

(1) t_i is tally for class interval $i \cdot 2^{37} \leq n < (i+1) \cdot 2^{37}$, $i=0,\ldots, 31$.

(2) P(i) is instruction which adds 1 to tally t_i.

(3) Operation symbol A in the second box following the remote connection

(1)→ signifies low order multiplication.

(4) M is maximum value of m.

As in the example of par. 17.3 this problem will also be coded initially in terms of symbolic addresses. After this is done, the address assignments will be written alongside the symbolic notation, and the conversions to machine hexadecimal will be performed without rewriting the initial code. It is to be noted parenthetically that the process of clearing the tallies involves instruction modification, and this has been done by method (1) of par. 17.0. Note also that the storage locations $C_0, \ldots, C_{31}$ for the tallies must begin with $C_0 = 0 \pmod 8$ in order to make use of 8 word output instructions.

4 x Address	Address	Symbolic Address	α β γ δ Op	Machine Hex	Explanation
020	008	L_1	L_2 K_6 T_1 T_1 5	0241F 7F1FC 5	Clear tallies $t_0, \ldots, t_{31}$
024	009	L_2	000 000 C_0 L_3 2	00000 0A00A 2	
028	00A	L_3	T_1 K_8 L_4 L_5 D	7F021 02C0C D	
02C	00B	L_4	K_7 T_1 T_1 T_1 5	081FC 7F1FC 5	
030	00C	L_5	000 000 S_2 L_6 2	00000 7FC0D 2	Set m=0
034	00D	L_6	000 001 S_1 L_7 0	00001 7F80E 0	Read in n_0.
038	00E	$\lceil L_7$	K_2 S_1 S_1 L_8 A	06DFE 7F80F A	$(5^{17} \cdot 2^{-42})$ $A(n_m \cdot 2^{-42}) = n'_{m+1} \cdot 2^{-42}$
03C	00F	L_8	S_1 K_1 T_1 L_9 3	7F81A 7F010 3	$n'_{m+1} \cdot 2^{-5}$

4xAdd.	Add.	Symb. Add.	α	β	γ	δ	Op	Machine Hex		Explanation
040	010	L_9	K_3	T_1	T_1	L_{10}	9	071FC 7F011	9	$n'_{m+1} \rightarrow \beta$ and γ of T_1
044	011	L_{10}	L_{11}	T_1	T_2	T_2	5	049FC 7F5FD	5	Set up $P(n'_{m+1})$
048	012	L_{11}	K_4	C_0	C_0	L_{12}	5	07428 0A013	5	
04C	013	L_{12}	K_4	S_2	S_2	L_{13}	5	075FF 7FC14	5	$m+1 \rightarrow m$
050	014	L_{13}	S_2	K_5	L_7	L_{14}	D	7FC1E 03815	D	$m < M$? Yes $\rightarrow L_7$ No $\rightarrow L_{14}$
054	015	L_{14}	000	000	C_0	L_{15}	F	00000 0A016	F	Print $t_0, \ldots, t_{31}$
058	016	L_{15}	000	000	C_8	L_{16}	F	00000 0C017	F	
05C	017	L_{16}	000	000	C_{16}	L_{17}	F	00000 0E018	F	
060	018	L_{17}	000	000	C_{24}	L_{18}	F	00000 10019	F	
064	019	L_{18}	000	001	S_1	000	F-	00001 7F800	F-	Print n_M
068	01A	K_1						3E000 00000	0	$1-2^{-5}$
06C	01B	K_2						0B1A2 BC2EC	5	$5^{17} \cdot 2^{-42}$
070	01C	K_3						00020 08000	0	$2^{-13} + 2^{-23}$
074	01D	K_4						00000 00000	1	2^{-42}
078	01E	K_5								$M \cdot 2^{-42}$
07C	01F	K_6						00000 00000	0	0
080	020	K_7	000	000	001	000	0	00000 00400	0	2^{-28}
084	021	K_8	000	000	C_{31}	000	0	00000 11C00	0	Comparison Constant
0A0	028	C_0								Storage locations for $t_0, \ldots, t_{31}$
.	.	.								
.	.	.								
.	.	.								
11C	047	C_{31}								
7F0	1FC	T_1								Temporary

4xAdd.	Add.	Symb. Add.	α β γ δ Op	Machine Hex.	Explanation
7F4	1FD	T_2	[K_4 $C_{n'_{m+1}}$ $C_{n'_{m+1}}$ L_{12} 5]		Temporary: $P(n'_{m+1})$
7F8	1FE	S_1			n
7FC	1FF	S_2			m

Before sending the control to the above routine, the number M must be inserted in K_5, to control the number of quasi-random numbers to be generated. It can be seen that the process need not begin with $n_0 = 5^{17}$, but could just as well begin with any number in the sequence $\{n_m\}$. This is the reason for printing n_M at the conclusion of the run. On the next run, this can be the n_0, and the generation process can resume from where it left off the previous time. The tallies, of course, would all start from zero again.

18.0 Timing. Calculation of timing is done differently for the acoustic memory, because of its serial nature, than for the electrostatic memory, which is parallel in nature. The discussion of timing will first be carried out for each of these memories separately. It will then be simple to understand how timing is determined when both memories are used.

18.1 Addresses in Acoustic Memory Only. In the acoustic memory, 48 binary positions are required for the storage of each word, 45 of which are for information. The basic repetition rate of the SEAC is 1 megacycle (1 million pulses per second) so that 48 microseconds (millionths of a second) are required for a word to pass a given point. A tank in the acoustic memory holds 8 words, or $8 \cdot 48 = 384$ binary digits. Therefore 384 microseconds are required for a complete circulation of the contents of a tank. This unit of time is called one "major cycle". One eighth of this time, or 48 microseconds, is called a "minor cycle". The contents of all tanks are circulated synchronously. The number of minor cycles within a major cycle are counted by a "minor cycle counter" which takes one step each minor cycle, counting from 7 to 0. All words whose addresses have the same residue modulo 8 (i.e., whose addresses have the same remainder upon division by 8) move simultaneously into position to be read (though only one can actually be read at a time), and at this instant the reading in the counter is this residue.

The execution of an instruction (not input, output, or tape reverse) in the instruction register involves four phases. Starting from the halted condition these are:

(1) Location of new instruction, specified by δ (normally) or γ of instruction currently in instruction register, and bring new instruction

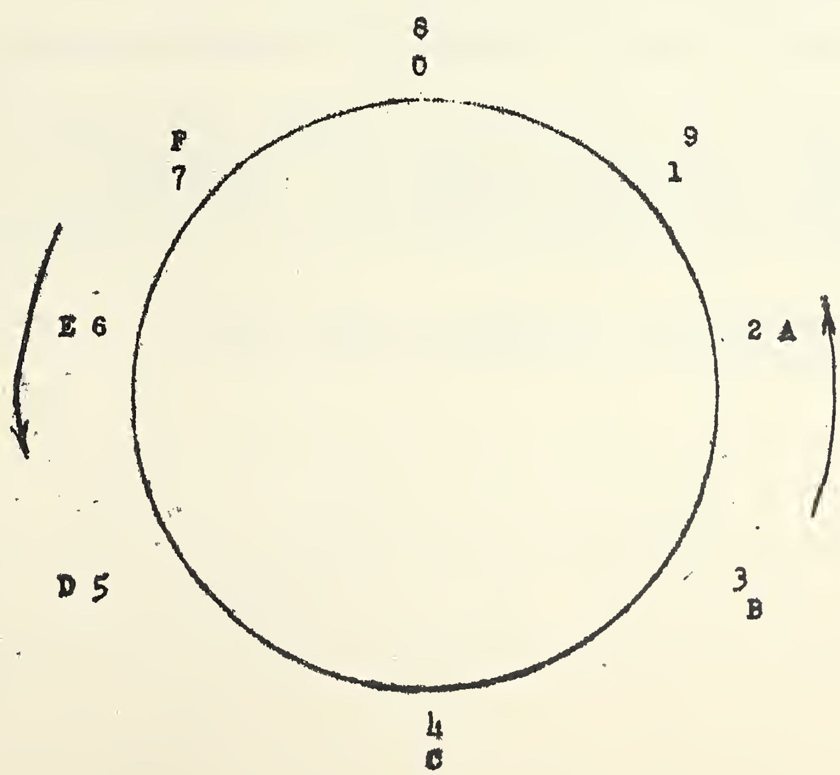
into the register.

(2) Location of β address of this new instruction.

(3) Location of α address of the new instruction and perform operation.

(4) Result of operation (for non-comparison operations) goes to γ of the new instruction. In comparison this is a "dummy" phase.

The time required in minor cycles for the execution of a phase is the number of steps taken by the counter during that phase. At least one step is taken during any phase. This implies that if the counter readings are the same at the beginning and end of a phase, then at least 8 steps were taken during that phase. With the exceptions (2) and (3) noted below, the counter reading at the end of a phase is always the least positive residue modulo 8 of the address associated with that phase. Counting the minor cycles corresponding to a specified phase is facilitated by the following figure:



Given the addresses in hexadecimal associated with 2 successive phases, count in the figure from the last digit of the first address to the last digit of the second address in the direction of the arrow. The number of steps obtained (which will vary from 1 to 8) normally is the time in minor cycles for the second of the two phases, for all operations. Exceptions to the foregoing rules are as follows:

(1) For multiplication and division operations, add 44 minor cycles to the time obtained by application of the above rule.

(2) In comparison operations, phase 4 is always 1 minor cycle, regardless of what γ is.

(3) In the base operation, E, phases 2 and 3 are always 1 minor cycle each.

The time for a complete instruction is the sum of the times for 4 successive phases, starting with phase 2.

The times for input and output instructions are obtainable from the Summary of SEAC Specifications preceding Part I.

Example 1. Find the time for executing the following instruction:

Address	α	β	γ	δ	Op
1A2	1A1	012	118	117	D

The calculation is conveniently exhibited in tabular form:

	Minor Cycle Counter Reading at		Time in Minor Cycles	Remarks
	Beginning	End		
Phase 2	2	2	8	
" 3	2	1	1	
" 4	1	0	1	Item (2) in list of "Exceptions"
1	0	0	8	Control goes to γ
		7	1	" " " δ
	Total Time:		18	Control goes to γ
			11	" " " δ

Example 2: Find the time for executing the following instruction:

Address	α	β	γ	δ	Op
017	100	115	10D	206	A

Calculation:

	Minor Cycle Counter Reading at		Time in Minor Cycles	Remarks
	Beginning	End		
Phase 2	7	5	2	
3	5	4	49	Item (1) in list of "Exceptions"
4	4	5	7	
1	5	6	7	
	Total Time:		65	

18.2 Addresses in Electrostatic Memory Only. If all addresses are in the electrostatic memory the rules for calculating timing are particularly simple:

(1) Phases (1) and (2) are always 1 minor cycle each.

(2) Phase (3) is 1 minor cycle for all operations except multiplication and division. These operations require 45 minor cycles.

(3) Phase (4) is 2 minor cycles for all operations except C,D,E.

(4) For operations C,D,E, phase (4) is 1 minor cycle.

18.3 Addresses in Acoustic and Electrostatic Memories. The rules given in paragraphs 18.1 and 18.2 are sufficient for this case. The method of calculation will be illustrated by some examples.

Example 1.

Address	α	β	γ	δ	Op
10A	34C	20D	019	018	3

In this example, addresses 34C and 20D are in the electrostatic memory. The calculation is as follows:

	Minor Cycle Counter Reading at		Time in Minor Cycles	Remarks
	Beginning	End		
Phase 2	2	1	1	Par. 18.2
" 3	1	0	1	Par. 18.2
" 4	0	1	7	
" 1	1	0	1	
		Total:	10	

Example 2.

Address	α	β	γ	δ	Op
125	3E3	09C	34E	1F8	8

	Minor Cycle Counter Reading at		Time in Minor Cycles	Remarks
	Beginning	End		
Phase 2	5	4	1	
" 3	4	7	45	Par. 18.2
" 4	7	5	2	Par. 18.2
" 1	5	8	5	
		Time:	53	

18.4 Remarks. It is generally not advisable to spend much time in trying to minimize the time for a routine by rearranging addresses, etc. Such a procedure is generally difficult, time consuming, and likely to give rise to coding errors. Moreover, the time saved on the machine by optimum timing is likely to be balanced by the time lost in coding. And finally, the overall time for a problem is often determined principally by speed of input and output, rather than by speed of computing. It is usually more desirable to reduce computing time by selecting an efficient method of solution and by reducing input and output time than by rearranging addresses. Some of the early SEAC subroutines were coded for the acoustic memory with a view to minimizing computing time. This practice was abandoned in the preparation of most later subroutines.

19.0 Subroutines. The need for certain types of mathematical calculations may be anticipated, and they may be coded, recorded, and filed in advance of their use. The series of instructions comprising such a calculation is called a "subroutine". The same series of instructions may be used many times in any problem, with varying arguments (or initial conditions). Some examples of SEAC subroutines are: conversion of numbers from decimal to binary, conversion from binary to decimal, generation of trigonometric functions, etc.

This procedure of pre-coding and filing results in a considerable saving of coding time. For before filing, these subroutines will always have been carefully checked, prepared on tape, and tested. Duplication of much work is thus avoided. Furthermore, these subroutines may be coded with special attention to economy in memory space as well as computing time.

A list and description of selected SEAC subroutines available as of this writing (Sept., 1952) is given at the end of this Handbook (Refs. 9-13).

19.1 Conventions for Standard Subroutines. In preparing SEAC subroutines certain conventions have been adopted which render these subroutines more convenient for general use and eliminate unnecessary duplication and consequent waste of memory space. Subroutines conforming to these conventions are referred to as "standard subroutines".

(1) The second tank (cells 008-00F) is reserved for temporary storage in all standard subroutines. Certain constants commonly used in subroutines and elsewhere are called "standard constants", and are assigned standard locations in addresses 010-027. It should be kept in mind that these constants must be read into their assigned locations whenever using the

subroutines. For this purpose they have been prepared on tape and are on file together with the subroutines. In paragraph 19.3 is appended a list of the standard constants. Additional constants needed in any subroutine are placedat the end of the subroutine.

(2) For convenience in use all standard SEAC subroutines are coded with the instructions beginning in cell 100. In general, however, the coder will want to specify a different set of locations for each subroutine used. This becomes a necessity, in any case, whenever more than one subroutine is used. The desired locations for the subroutines are determined by the coder, and the subroutines are accordingly read into those positions in the memory. Certain modifications are then required in the instructions to make them conform to their actual positions in the memory, and this is accomplished by the modifying routine described in par. 19.4.

(3) In all standard subroutines the argument is sent to 00D before entering the subroutine, and the result of the calculation is also sent to 00D. The control then goes to 00F for the exit instruction. Care must be taken that the exit instruction in 00F is properly set up before entering a subroutine.

19.2 <u>Transcription of Total Routine Onto Input Medium</u>. A problem involving subroutines may be transcribed onto the input tape or wire in various arrangements. One convenient arrangement might be as follows:

(1) Input instructions (000-007),

(2) Temporary storage (008-00F),

(3) Standard constants (010-027),

(4) (a) Instructions for incorporating subroutines (028-037),

(b) One word for each subroutine, followed by word containing all zeros (038, 039,...); (See par. 19.4),

(5) Subroutines in succession,

(6) Body of main routine.

19.3 List of Standard Constants:

Standard Address	Constant	Machine Hexadecimal Representation
010	-0	00000 00000 0-
011	-3	C0000 00000 0-
012	$+2$	80000 00000 0+
013	$+1$	40000 00000 0+
014	$+2^{-1}$	20000 00000 0+
015	$+2^{-2}$	10000 00000 0+
016	$+2^{-4}$	04000 00000 0+
017	$+2^{-8}$ ($+1$ in α)	00400 00000 0+
018	$+2^{-16}$	00004 00000 0+
019	$+2^{-32}$	00000 00040 0+
01A	$+2^{-42}$	00000 00000 1+
01B	$+2^{-38}$ ($+1$ in δ)	00000 00001 0+
01C	$+2^{-28}$ ($+1$ in γ)	00000 00400 0+
01D	$+2^{-22}$	00000 10000 0+
01E	$+2^{-20}$	00000 40000 0+
01F	$+2^{-18}$ ($+1$ in β)	00001 00000 0+

Standard Address	Constant	Machine Hexadecimal Representation
020	$+2^{-12}$	00040 00000 0+
021	$+2^{-10}$	00100 00000 0+
022	$+4 - 2^{-8}$(α extractor)	FFC00 00000 0+
023	$+2^{-8} - 2^{-18}$(β extractor)	003FF 00000 0+
024	$+2^{-18} - 2^{-28}$(γ extractor)	00000 FFC00 0+
025	$+2^{-28} - 2^{-38}$(δ extractor)	00000 003FF 0+
026	$+4 - 2^{-42}$	FFFFF FFFFF F+
027	+ [Reserved for Absolute values.]	

19.4 Method of Incorporating Subroutines. This code incorporates subroutines into the main routine without a manual modification of the orders making up the subroutines.

As pointed out earlier, all subroutines are written as though they are stored in the memory starting with hexadecimal address 100. However, when used in a problem they are read into blocks of memory locations assigned them by the coder, which will not usually begin with cell number 100. Any reference to addresses 100 and over in their instructions must therefore be modified by the difference between 100 and the beginning of the actually assigned block.

For example, suppose that the decimal to binary conversion subroutine [9.4] is placed in the block of memory cells 09C - 0A7. Then we have the following condition in the first two cells of the block:

Address	True Hex
09C	00D 013 103 10B D
09D	00D 016 008 107 A

In order to make the subroutine work in its assigned location, the above two instructions must be modified to read:

Address	True Hex
09C	00D 013 09F 0A7 D
09D	00D 016 008 0A3 A

For each subroutine the addresses of the first and last words to be changed are specified in cells 038, 039,... Constants and instructions not to be modified are placed at the end of the subroutines to which they belong. The routine for incorporating subroutines takes the subroutines one at a time and performs the required modifications.

It is to be noted that 027 is assumed to contain +0 to accomplish the logical transfer operation in the seventh instruction of the routine. This is permissible because +0 is read into address 027 with the standard constants, and since this is the first routine executed by the computer, 027 still contains +0.

Preliminary Information for Use of Routine for Incorporating Subroutines

1. Let a_i = address containing first word of ith subroutine.
2. Let b_i = number of words of ith subroutine to be changed.
3. Store $2^{-8}(a_i+b_i-1) + 2^{-28}a_i$ in locations 038, 039,...,037+n, where n is the number of subroutines; and store 0 in location 038+n.
4. The instruction to be executed after the completion of this subroutine is located in 007.
5. The number of addresses occupied by this preparatory routine is 17+n.
6. In order to restore a "+" sign to address 027 before performing calculations, make sure subroutines are arranged with positive sign for last instruction of last subroutine to be modified.

(Column headed mc indicates timing in minor cycles)

4 x Address	Address	Instruction in						mc	Explanation
		Machine Hex	True Hex α	β	γ	δ	op.		
0A0	028	0E024 0B432 1	038	024	02D	032	1	14	Set up correction instruction
0C8	032	0E019 02833 A	038	019	00A	033	A	55	Obtain $2^{-18}a_i$
0CC	033	0840A 02837 B	021	00A	00A	037	B	52	Obtain $2^{-8}a_i$
0DC	037	02822 0A434 1	00A	022	029	034	1	19	Set up storing instruction
0D0	034	02813 02829 4	00A	013	00A	029	4	11	Obtain $2^{-8}a_i-1 = 2^{-8}(a_i-100)$
0A4	029	00010 02036 5	$[a_i]$	010	008	036	5	12	Store word of subroutine
0D8	036	02035 09C2C 1	008	035	027	02C	1	10	Extract 2nd binary digits of addresses
0B0	02C	09C0A 02C2D 8	027	00A	00B	02D	8	63	Find correction term
0B4	02D	0200B 00031 5	008	00B	$[a_i]$	031	5	14	Correction instruction
0C4	031	0A438 0A82E C	029	038	02A	02E	C	15/11	Test for end of subroutine
0A8	02A	05C29 0A430 5	017	029	029	030	5	10	Modify (029) for next word
0C0	030	0702D 0B429 5	01C	02D	02D	029	5	15	Modify (02D) for next word
0B8	02E	0E410 0E02F 5	[039]	010	038	02F	5	16	Store $(a_i+b_i-1)2^{-8} + a_i 2^{-28}$
0BC	02F	05C2E 0B82B 5	017	02E	02E	02B	5	12	Modify (02E) for next subroutine
0AC	02B	04038 0A007 C	010	038	028	007	C	19/20	Test for last subroutine
0D4	035	40100 40100 0-	100	100	100	100	0-		Extractor
0E0	038								$(a_1+b_1-1)2^{-8}+a_1 2^{-28}$
0E4	039								$(a_2+b_2-1)2^{-8}+a_2 2^{-28}$
									.
									.
0DC+4n	037+n								$(a_n+b_n-1)2^{-8}+a_n 2^{-28}$
0E0+4n	038+n	00000 00000 0	000	000	000	000	0		

Time required is approximately $6\frac{1}{2}$ milliseconds per instruction requiring modification.

20.0 <u>Some Special Features of the 3-address System.</u> There are a number of advantages in the use of the 3-address system on the SEAC. One obvious advantage can be seen by recalling the composition of a 3-address instruction (cf. Part I, par. 8.2). Since each address is represented by 12 binary digits (exactly 3 hexadecimal digits) instead of ten, the decomposed (true hexadecimal) form of an instruction, is identical with its composed (machine hexadecimal) form.

Also, it was indicated in Part I, par. 8.2 that the handling of subroutines in the 3-address system is facilitated in 2 ways: (1) The necessity to print an exit instruction before entering a subroutine (see par. 19.1 (3)) can be eliminated by use of the binary digit d to select one or the other of the address counters C_0 and C_1. Characteristically, during the course of the main routine we would have d = 0, the control being thus referred to counter C_0. One would get into a subroutine by means of a comparison instruction in which $(\alpha) < (\beta)$, γ is the address of the first subroutine instruction, and d = 1. Counter C_1 would then be reset to γ, and the control would be established by C_1 as long as d = 1. The last subroutine instruction executed would have d = 0, thus causing the control to return to C_0, picking up where it left off. (2) The need for the incorporation routine of par. 19.4, arising from the fact that subroutines are usually precoded for blocks of cells beginning with a standard address, is eliminated by use of the digits a, b, c. References in the subroutine instructions to words stored in standard locations outside the subroutine (such as standard constants, and standard temporary locations) do not depend on the actual location of the subroutine in the memory. On the other hand, references to words within the subroutine (such as other instructions, or non-standard constants included with the subroutine) must be adjusted to conform to the actual subroutine location. For each address in the former category, the corresponding digit a, b, or c is made zero, indicating that the address is to be interpreted in an absolute sense. The effect of assigning the value one to any of these digits is to cause the corresponding address to be interpreted relatively to the address of the instruction itself. Thus, for example, letting k be the hexadecimal digit formed by the binary digits a,b,c,d, the instruction

Address	α	β	γ	k	Op
2C8	013	003	005	7	5

would be interpreted during execution as

Address	α	β	γ	k	Op
2C8	013	2CB	2CD	7	5

One complication that arises in this type of application is that the base with respect to which floating addresses are interpreted changes each time the control advances. Suppose, for example, that a subroutine in its originally coded form contains the two instructions

Address	α	β	γ	k	Op
009	00E	010	008	9	1
..	.	.	.	.	.
00B	013	00E	009	5	5

Suppose further that in actual use the above two words are stored in 057 and 059, respectively. Then α of the first instruction will be interpreted as 057+00E=065; and β of the second instruction will be interpreted as 059+00E=067. Thus, whereas it was originally intended that α of the first instruction and β of the second refer to the same location, this is no longer the case. The address that we really wish to transform 00E to is clearly 05C. The difficulty is resolved by rewriting the subroutine before use, so that every address to be interpreted relatively is decreased by the address of the instruction of which it is a component. If the application of this rule would yield a negative address, then add 1000 (in hexadecimal) to the address in question before subtracting the address of the instruction. Thus, in the example given, replace 00E in the first instruction by 00E-009=005; and in the second by 00E-00B=003. Then in the actually used locations, the interpretations will be 057+005=05C, and 059+003=05C, respectively, as desired.

A further complication in the use of 3-address coding as compared with 4-address is that much more care has to be taken in arranging instructions in proper sequence. While in the 4-address system the path of the control is completely flexible, in the 3-address system a change from the originally coded path can be achieved only by inserting an appropriate comparison instruction.

The standard constants in par. 19.3 were chosen primarily with convenience for 4-address coding in mind. These constants, while numerically the same regardless of the coding system used, sometimes have different applications in the two systems. For example, in 4-address, "1" in the α, β, γ, and δ addresses are, respectively, equivalent to 2^{-8}, 2^{-18}, 2^{-28}, and 2^{-38}. In 3-address, however, "1" in α, β, and γ are equivalent to 2^{-10}, 2^{-22}, and 2^{-34}, respectively. Similarly, in 4-address, the α, β, γ, and δ extractors are represented by $2^{2} - 2^{-8}$, $2^{-8} - 2^{-18}$, $2^{-18} - 2^{-28}$, and $2^{-28} - 2^{-38}$, respectively; whereas in 3-address the α, β, and γ extractors are given by $2^{2} - 2^{-10}$, $2^{-10} - 2^{-22}$, and $2^{-22} - 2^{-34}$, respectively. It will be noted that 2^{-34} and the 3-address extractors are not included among the standard constants.

Timing is calculated in the 3-address system exactly as in 4-address. It must only be noted that in phase 4, the location of the new instruction is specified by the appropriate counter, since there is no δ address.

The preparation of a code in 3-address will be illustrated by the following two examples.

20.1 Example 1. Prepare a subroutine for the calculation of the square root.

Solution. A flow diagram and 4-address routine for square root computation was given in par. 15.1. The same notation and method of computation will be used here. The subroutine will be coded beginning in address 000. Some of the assumptions for standard 4-address subroutines (par. 19.1) will be made here; namely, that the standard constants of par. 19.3 are in the memory, that the argument is in 00D, and that the result goes to 00D. Addresses 008-00F will be available as temporaries. The code is as follows:

Address	α	β	γ	κ	Op	Explanation
000	00D	010	009	3	C	$N < 0$?
001	013	010	00A	1	5	$x_0 = 1$
002	00A	00D	00C	1	B	N/x_i
003	00C	00A	00B	1	5	$x_i + N/x_i$
004	00B	014	00E	1	8	$\frac{1}{2}(x_i + N/x_i) = x_{i+1}$
005	00E	00A	007	3	D	$x_{i+1} < x_i$?

Address	α	β	γ	κ	Op	Explanation
006	00A	010	00D	0	5	$\sqrt{N} \rightarrow$ 00D
007	00E	010	00A	1	5	$x_{i+1} \rightarrow x_i$
008	000	001	002	F	D	Control $\rightarrow$ 002
009	000	001	00D	0	F-	Error halt

In this code, we have made d=1 on all instructions except that in 006, the last to be performed, and in 009 where it is rather immaterial since an error is indicated and the machine halts. The only instructions where addresses are to be interpreted relatively are in 000 (γ interpreted relatively, c=1), 005 (γ interpreted relatively, c=1), and 008 (α, β, γ interpreted relatively, a=b=c=1). If we rewrite these instructions according to the rule described in par. 20.0, we see that the instruction in 000 is unchanged and the instructions in 005 and 008 become

Address	α	β	γ	κ	Op
005	00B	00A	002	3	D
008	FF8	FF9	FFA	F	D

20.2 Example 2. Using the subroutine of Example 1, write a routine for the computation of

$$u_i = \frac{x_i^{\frac{1}{2}}}{1 + x_i^{3/2}}, \quad i = 1, \ldots, 8,$$

including read-in instructions, and read-out instructions for the u_i.

Solution. As in Example 1, addresses 008-00F will be used for temporaries, and the standard constants will be assumed to be in 010-027. The input instructions for the routine will be written in tank 0. The eight x_i will be stored in 048-04F, and the u_i will be sent to the same address. The subroutine of Example 1 will be stored in 038-041. As in the example of par. 17.3, all variable instructions will be preset to their initial states before entering the actual computation. This example illustrates, among other things, how the base operation, E, may be used as a tally (in addresses 001 and 032; see tabulation of operations in par. 13.0).

Address	α	β	γ	K	Op	Explanation
000	000	000	[008]	0	0	Read in 8 words
001	[020]	027	004	0	E	Read-in completed?
002	000	000	008	0	0	Constant
003	001	000	000	0	0	Constant
004	000	002	000	0	5	Modify (000)
005	001	003	001	0	5	Modify (001)
006	002	003	000	0	D	Control → 000
007	000	000	000	0	0	
008 . . . 00F						Temporaries
010 . . . 027						Standard constants
028	000	000	048	0	0	Read in x_i, i = 1,...,8
029	042	010	02C	0	5	Preset 02C
02A	043	010	031	0	5	Preset 031
02B	044	010	032	0	5	Preset 032
02C	[048]	010	008	0	5	x_i → 008
02D	008	010	00D	0	5	x_i → 00D

Address	α	β	γ	δ	Op	Explanation
02E	010	013	038	1	D	Enter subroutine
02F	00D	008	008	0	9	$x_i^{3/2}$
030	013	008	008	0	5	$1 + x_i^{3/2}$
031	008	00D	[048]	0	B	$x_i^{\frac{1}{2}}/(1 + x_i^{3/2}) = u_i$
032	[02F]	036	033	0	E	If $i < 8$, go to 033 If $i = 8$, go to 037
033	02C	021	02C	0	5	Add 1 to α of 02C
034	031	045	031	0	5	Add 1 to γ of 031
035	032	021	032	0	5	Add 1 to α of 032
036	010	013	02C	0	D	Control → 02C
037	000	000	048	0	F-	Read out u_i, $i = 1, \ldots, 8$
038	00D	010	009	3	C	
039	013	010	00A	1	5	
03A	00A	00D	00C	1	B	
03B	00C	00A	00B	1	5	
03C	00B	014	00E	1	8	Square Root Subroutine
03D	00E	00A	002	3	D	
03E	00A	010	00D	0	5	
03F	00E	010	00A	1	5	
040	FF8	FF9	FFA	F	D	
041	000	001	00D	0	F-	
042	048	010	008	0	5	Constant instruction
043	008	00D	048	0	B	Constant instruction
044	02F	036	033	0	E	Constant instruction
045	000	000	001	0	E	2^{-34}

Address	α	β	γ	κ	Op	Explanation
046	000	000	000	0	0	Blank
047	000	000	000	0	0	
048						Storage of x_i and u_i
.						
.						
04F						

21.0 Remarks. There are many aspects of SEAC programming and operating that have not been discussed in this Handbook. Such topics as the use of the manual controls, techniques of code-checking both on and off the SEAC, use of breakpoints, assembly and composition routines, instructions for use of auxiliary equipment, descriptions of subroutines, etc., have been omitted. However, information is available on these subjects in the Computation Laboratory Technical Memoranda, and the Electronic Computers Laboratory Technical Memoranda listed in the references below. These memoranda are prepared in an effort to maintain a record of current developments in SEAC operating and programming.

REFERENCES

1. Goldstine, H. H. and von Neumann, J., Planning and Coding of Problems for An Electronic Computing Instrument, Institute for Advanced Study, Princeton, N.J., 1947.
2. von Neumann, J., and Goldstine, H. H., Numerical Inverting of Matrices of High Order, Bulletin of the American Mathematical Society, pp. 1021-1099 (Nov., 1947).
3. Isaac, E. J., and Trabka, E. A., Machine Aids to Coding, Cornell Aeronautical Laboratory, Mar., 1952.
4. Wilks, M. V., Wheeler, D. J., Gill S., The Preparation of Programs For An Electronic Digital Computer, Addison-Wesley Press, Inc., 1951.
5. Leiner, A. L., Provision for Expansion in the SEAC, Mathematical Tables and Other Aids to Computation, pp. 232-237, (October, 1951)
6. Machine Development Laboratory Staff, The Incorporation of Subroutines into a Complete Program on the NBS Eastern Automatic Computer, Mathematical Tables and Other Aids to Computation, pp. 164-168 (July, 1950).

7. Scarborough, J. B., Numerical Mathematical Analysis, pp. 178-184, The Johns Hopkins University Press, 1930.

8. Electronic Computers Laboratory, National Bureau of Standards, Technical Memoranda.

   8.1 No. 16 M, External Selector Unit for SEAC, Oct., 1951.

   8.2 No. 19 M, Installation of External Selector Panel on SEAC, Oct., 1951.

   8.3 No. 30 M, The Outscriber, Mar., 1952.

   8.4 No. 42 M, Operating Instructions for Magnetic Input-Output Devices in SEAC, June, 1952.

   8.5 No. 43 M, Three-Address Operation of SEAC Control Panel, Jul., 1952.

   8.6 No. 44 M, E 1.1, E 1.2 (512), E 2.2 (1024) Dump Routines, Jul., 1952.

9. Computation Laboratory, National Bureau of Standards, SEAC Operating and Programming Notes, I, (April, 1952); includes following:

   9.1 Tech. Memo. No. 1, Operation of SEAC Control Panel as of Oct., 1951.

   9.2 Tech. Memo. No. 2, Non-Mathematical Routines.

   9.3 Tech. Memo. No. 3, Conversion Subroutines for Integers.

   9.4 Tech. Memo. No. 4, Decimal to Binary and Binary to Decimal Subroutines.

   9.5 Tech. Memo. No. 5, Subroutines for 2^x and e^x.

   9.6 Tech. Memo. No. 6, Subroutines for $\sqrt{N}$, $\sqrt[3]{N}$, $\sqrt[n]{N}$.

   9.7 Tech. Memo. No. 7, Subroutines for Sin x and Cos x.

10. Computation Laboratory, National Bureau of Standards, SEAC Operating and Programming Notes, II, (July, 1952); includes following:

   10.1 Tech. Memo. No. 8, Subroutine for Tan^{-1} a/b

   10.2 Tech. Memo. No. 9, Hexadecimal-Decimal Converter Table

   10.3 Tech. Memo. No. 10, Four-Hexi Converter Table.

10.4 Tech. Memo. No. 12, Subroutine for Decimal to Binary Conversion of a Double Precision Number with Fixed Binary Point.

10.5 Tech. Memo. No. 13, Subroutine for i^{th} Differences, $i=1,2,\ldots,6$

10.6 Tech. Memo. No. 14, Three-Address System and Base Order.

10.7 Tech. Memo. No. 15, Preparation of Pure Hexadecimal Coding for Punched Card Composition.

11. Computation Laboratory, National Bureau of Standards, SEAC Operating and Programming Notes, III, (Aug., 1952); includes following:

11.1 Tech. Memo. No. 16, Magnetic Tape to Wire Transfer and Check Routine.

11.2 Tech. Memo. No. 17, Subroutines for Basic Arithmetic Operations and Square Root in Floating Decimal Form.

11.3 Tech. Memo. No. 18, General Dump Routine (512 or 1024 Word Memory)

11.4 Tech. Memo. No. 19, Memory Decomposition Routine (512 or 1024 Words)

11.5 Tech. Memo. No. 20, Flow Charts for Magnetic Tape Checking Routines.

12. Computation Laboratory, National Bureau of Standards, SEAC Operating and Programming Notes, IV, (Sept., 1952); includes following:

12.1 Tech. Memo. No. 21, Composition Routine (SEBBE).

12.2 Tech. Memo. No. 22, Basic Arithmetic Operations I, Floating Binary Point, Single Precision Numbers.

12.3 Tech. Memo. No. 23, Subroutine for Cube Root, Single Precision, Floating Binary Point.

12.4 Tech. Memo. No. 24, Subroutine for sin z, cos z, z=x+iy

12.5 Tech. Memo. No. 25, Corrections and Changes to Technical Memoranda Numbers 1-24.

13. Computation Laboratory, National Bureau of Standards, SEAC Operating and Programming Notes, V, (Sept., 1952); includes following:

13.1 Tech. Memo. No. 26, Interpretive Subroutine for Operations on Complex Numbers.

13.2 Tech. Memo. No. 27, Subroutine for Square Root of a Single Precision Number with Floating Binary Point.

13.3 Tech. Memo. No. 28, Subroutine for Sinh y, Cosh y, $|y| \leq 2.0634$.

13.4 Tech. Memo. No. 29, Code Checking.

13.5 Tech. Memo. No. 30, Basic Arithmetic Operations for Double Precision Numbers with Fixed Binary Point.

13.6 Tech. Memo. No. 31, Interpretive Subroutine for Operations on Double Precision Numbers.

THE NATIONAL BUREAU OF STANDARDS

Functions and Activities

The functions of the National Bureau of Standards are set forth in the Act of Congress, March 3, 1901, as amended by Congress in Public Law 619, 1950. These include the development and maintenance of the national standards of measurement and the provision of means and methods for making measurements consistent with these standards; the determination of physical constants and properties of materials; the development of methods and instruments for testing materials, devices, and structures; advisory services to Government Agencies on scientific and technical problems; invention and development of devices to serve special needs of the Government; and the development of standard practices, codes, and specifications. The work includes basic and applied research, development, engineering, instrumentation, testing, evaluation, calibration services, and various consultation and information services. A major portion of the Bureau's work is performed for other Government Agencies, particularly the Department of Defense and the Atomic Energy Commission. The scope of activities is suggested by the listing of divisions and sections on the inside of the front cover.

Reports and Publications

The results of the Bureau's work take the form of either actual equipment and devices or published papers and reports. Reports are issued to the sponsoring agency of a particular project or program. Published papers appear either in the Bureau's own series of publications or in the journals of professional and scientific societies. The Bureau itself publishes three monthly periodicals, available from the Government Printing Office: The Journal of Research, which presents complete papers reporting technical investigations; the Technical News Bulletin, which presents summary and preliminary reports on work in progress; and Basic Radio Propagation Predictions, which provides data for determining the best frequencies to use for radio communications throughout the world. There are also five series of nonperiodical publications: The Applied Mathematics Series, Circulars, Handbooks, Building Materials and Structures Reports, and Miscellaneous Publications.

Information on the Bureau's publications can be found in NBS Circular 460, Publications of the National Bureau of Standards ($1.25) and its Supplement ($0.75), available from the Superintendent of Documents, Government Printing Office. Inquiries regarding the Bureau's reports and publications should be addressed to the Office of Scientific Publications, National Bureau of Standards, Washington 25, D. C.

NBS