B 045

Digital Computer Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts

SUBJECT: **DIFFERENTIAL VIDEO PROBE**

To: Test Equipment Committee

From: Henry E. Zieman

Date: June 19, 1953

Abstract: The circuitry, use, and specifications for the differential video probe are discussed. The probe permits the viewing of small differential signals in the presence of large common signals. A bandwidth of 5 cps to 9 mc at a gain of 0.4 has been achieved for signals from 0 to 4 volts.

The differential video probe is an adapter for use with any scope to permit the viewing of small voltage variations across components which are not at any fixed potential. The circuit is shown in B-54224. Essentially it consists of a simple differential amplifier (V1) with a constant current load for the cathode resistor (V2). This gives a high common mode rejection. R1, R2, and R12 together with filter capacitors C3 and C4 form a voltage divider between 0 and -450 v. which sets the bias on the grids of V1 and V2. The signal is coupled to the amplifier through C1 and C2, and reappears at a terminating resistor at the scope. L1 provides the necessary d-c coupling of the probe chassis to the power supply ground, and yet isolates the chassis from power supply ground at RF signals to permit the output cable to be terminated and grounded at the scope.

In use the probe is placed conveniently near the circuit to be analyzed to maintain short connecting leads which are fastened to J1 and J2. A positive output occurs when J1 is driven positive. J3 is connected to a 93-ohm cable which is terminated (93-ohm termination) at the scope. The power supply requirements are 35 ma at -450 v. DC, and 1.8 amp at 6.3 v. AC at a d-c level of -150 v. Two means are available to provide this filament supply level. If a power supply jack is available which has no other equipment connected to its filament transformer, the power plug of the probe may be connected to this jack and S1 (on the probe) set to the "By-passed" position. If the filament transformer is being shared with other equipment, the isolation transformer T-1 must be plugged into X1 and S1 set to the "in" position.

The gain of the probe is 0.4 for input voltages less than 4 volts. SA-48444-G shows the variation in gain and output voltage as a function of input voltage for inputs up to 14 volts. The gain is also a function of frequency, being flat at 0.4 from 10 cps to 1 mc and down 3 db at 5 cps and 10 mc. SA-45443-G shows the upper end of this frequency response. The rise time has been measured at approximately .05 μsec. The common mode gain (both inputs moving equally with respect to ground) is 10^{-4}. The input impedance from J1 to J2 is .05 μfd in series with 2 megohms shunted by 12 pf. The input impedance from either J1, or J2, to ground is 0.1 μfd in series with 1 megohms shunted by 16 pf.

Signed Henry E. Zieman
Henry E. Zieman

HEZ/cs

Approved R L Best
R. L. Best, Section Leader

Approved N H Taylor
N. H. Taylor, Group Leader

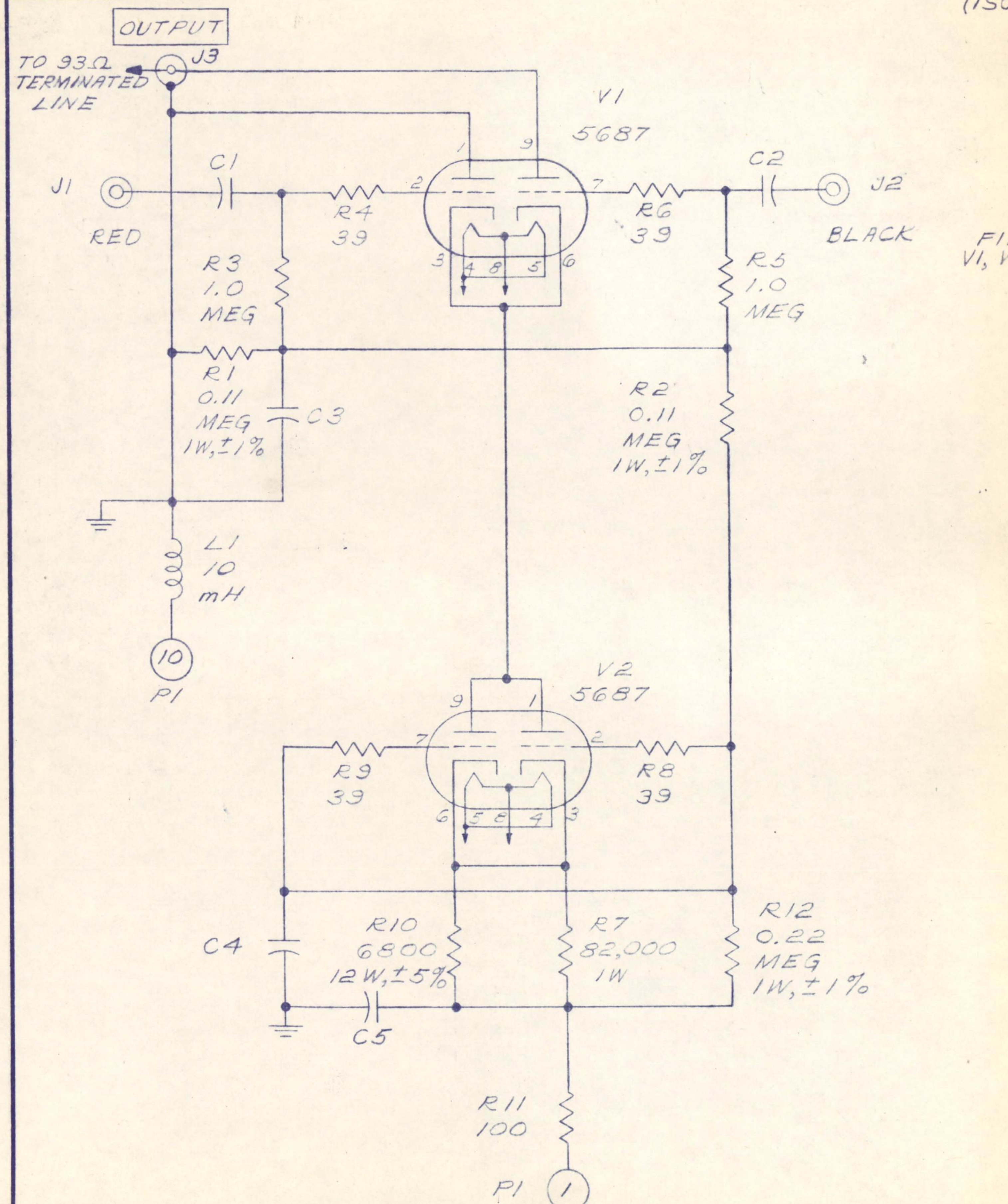
cc: MTC Section
Vacuum Tubes Circuit Section
Magnetic Memory Section

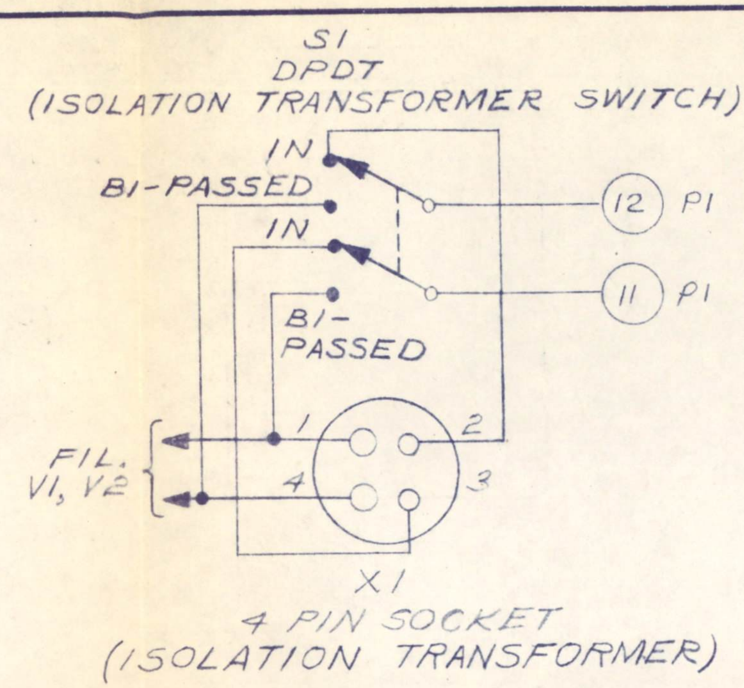
vellum copies:

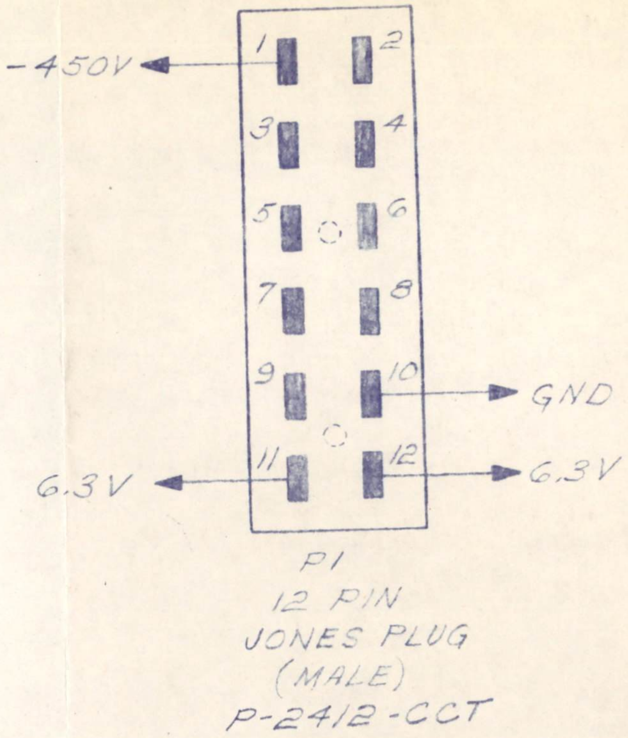
D.J. Crawford
Nate Edwards
R. Crago
} IBM - via Kromer

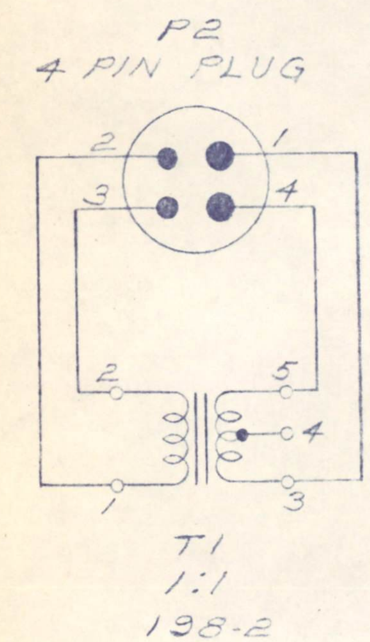
Drawings attached:

B-54224
SA-48444-G
SA-45443-G









NOTES:
1. UNLESS OTHERWISE SPECIFIED:
A. RESISTORS ARE IN OHMS, $\frac{1}{2}$ W, ±10%.
B. CAPACITORS ARE 0.1 MFD, 600 V.

DRAWING REFERENCES:
1. CHASSIS: SB-54252
2. TURRET ASSEMBLY: SA-54262

GRADED BY: DATE: THIS IS A GRADED DRAWING OF HIGHEST GRADE APPROVED BELOW:
GRADE I FOR REFERENCE ONLY
GRADE II PRELIMINARY DESIGN
DRB 6-19-53 GRADE III FINAL DESIGN

-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	CHG.
										CN#
										DATE
										APPD.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DIGITAL COMPUTER LABORATORY
DEPT. OF ELECTRICAL ENGINEERING – D. I. C. PROJECT NO. 6889

CIRCUIT SCHEMATIC, DIFFERENTIAL VIDEO PROBE, TE

SCALE: ______ DR. brm 5-14-53

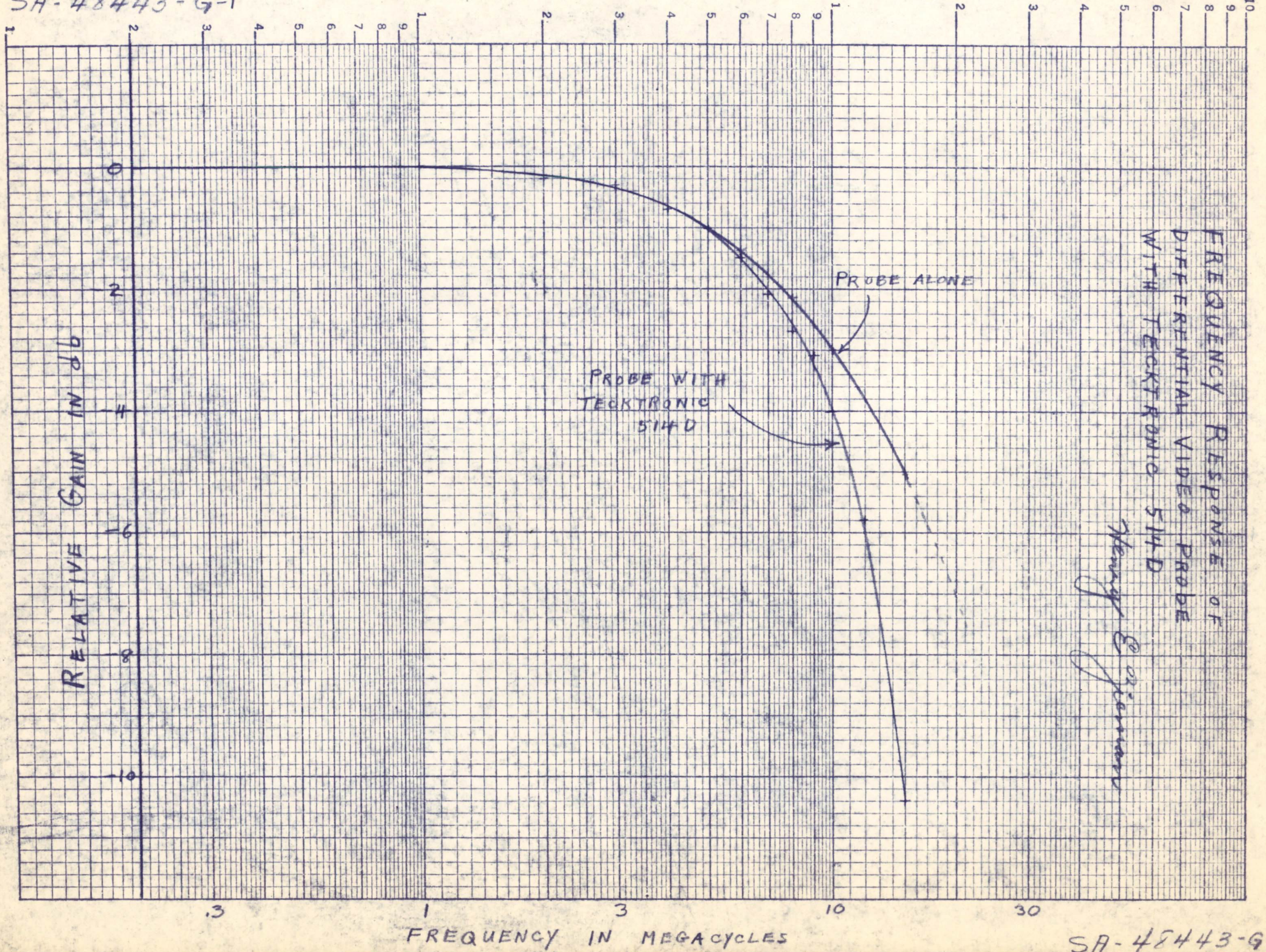
ENG. 6/18/53 H. Ziemann CK. 5/18/53 A. Amenta APPD. R L Best

B-54224

K+E ALBANENE 196L TRACING PAPER

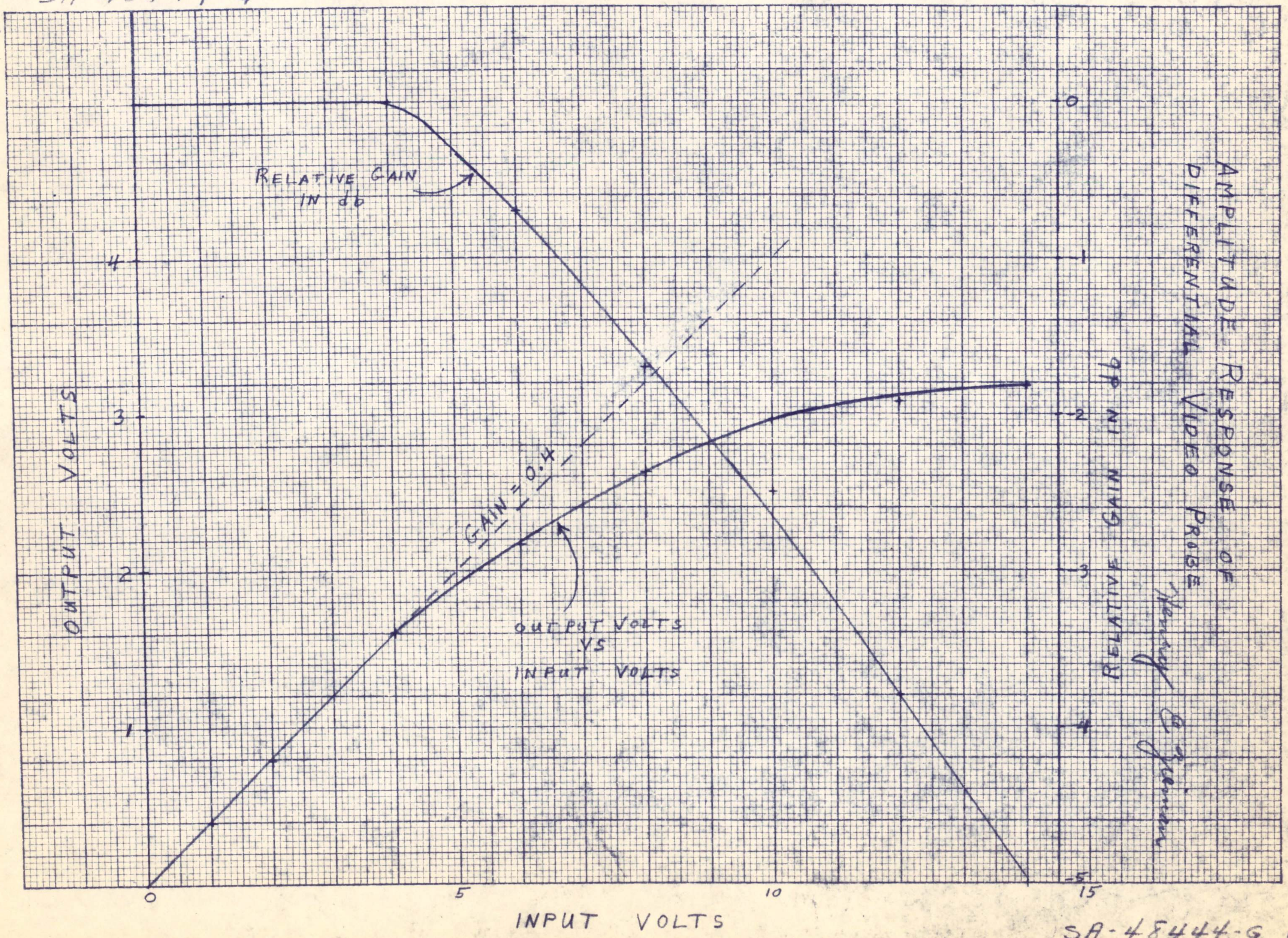
359-71 KEUFFEL & ESSER CO.
Semi-Logarithmic, 3 Cycles × 10 to the inch,
5th lines accented.
MADE IN U. S. A.

SA-48443-G-1



SA-48443-G-1

SA-48444-G



SA-48444-G

Digital Computer Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts

SUBJECT: SPECIFICATIONS FOR A FERRITE MEMORY CORE

To: Jay W. Forrester

From: David R. Brown

Date: June 30, 1953

1.0 GENERAL

These specifications are for a core to be used in a coincident-current magnetic-core memory with a two-to-one selection ratio.[1,2,3] They are based upon the pulse response of the core when driven by specified current-pulse sequences. The current-pulse sequences simulate critical conditions which will be encounted by the core in actual use. A post-write disturbing pulse is assumed.[4] The current pulses which drive the core being measured are carried by conductors which pass only once through the center of the core. The pulse response of the core being measured is the voltage observed across the terminals of a conductor passing only once through the center of the core, so arranged that mutual coupling due to leakage flux is zero.

---

1. Jay W. Forrester, "Digital Information Storage in Three Dimensions Using Magnetic Cores," Jour. App. Phys., vol. 22, pp. 44-48; Jan., 1951.

2. William N. Papian, "A Coincident-Current Magnetic Memory Cell for the Storage of Digital Information," Proc. IRE, vol. 40, pp. 475-478; Apr., 1952.

3. David R. Brown and Ernst Albers-Schoenberg, "Ferrites Speed Digital Computers," Electronics, vol. 26, No. 4, pp. 146-149; April, 1953.

4. P. K. Baltzer, Effect of Current Pulse Duration on the Pulse Response of M.T.C. Memory Cores, Digital Computer Laboratory Engineering Note E-533, March 10, 1953.

2.2 The Current Pulse Sequence (continued)

The basic prf is the reciprocal of the minimum time interval between reference time points of two consecutive pulses. All pulses have the same duration, rise time, and fall time. The peak currents of the pulses are related to the parameter I_m as follows:

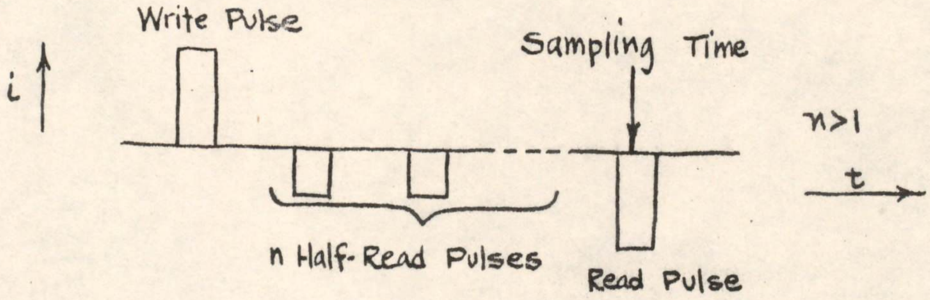
Write Pulse	$I_p = + I_m$
Half-write Pulse	$I_p = + I_m/2$
Read Pulse	$I_p = - I_m$
Half-read Pulse	$I_p = - I_m/2$

2.3 The Pulse Response

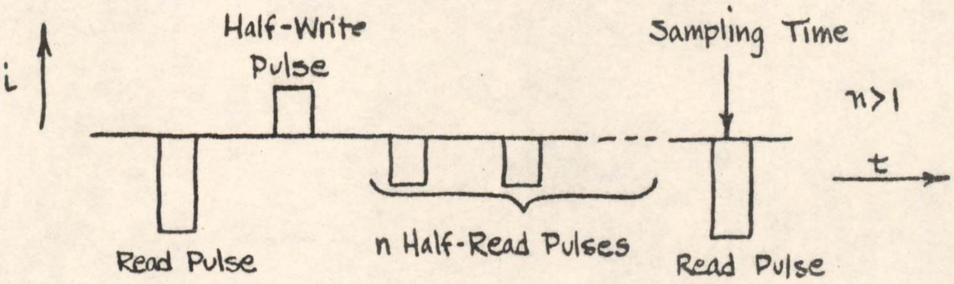
The pulse response is the voltage observed at the terminals of a conductor passing through the core when the core is driven by a read or half-read pulse. The voltage from a selected core is the voltage observed when the core is driven by a read pulse; the voltage from a half-selected core is the voltage observed when the core is driven by a half-read pulse. The response depends upon the state of the core before it is selected or half-selected: In each case the voltage to be measured is the instantaneous voltage measured at a specified sampling time T_m. The voltages of interest are:

${}_rV_1$	selected one, last disturbed by a half-read pulse
${}_rV_z$	selected zero, last disturbed by a half-read pulse
${}_rV_{h1}$	half-selected one, last disturbed by a half-read pulse
${}_rV_{hz}$	half-selected zero, last disturbed by a half-read pulse

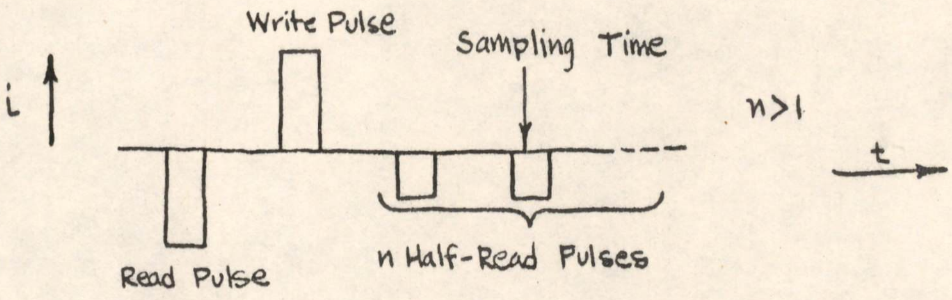
The current-pulse sequence for the determination of ${}_rV_1$ is:



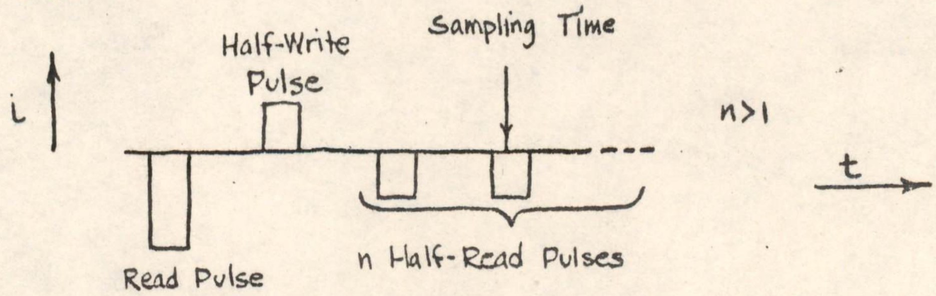
For ${}_rV_z$:



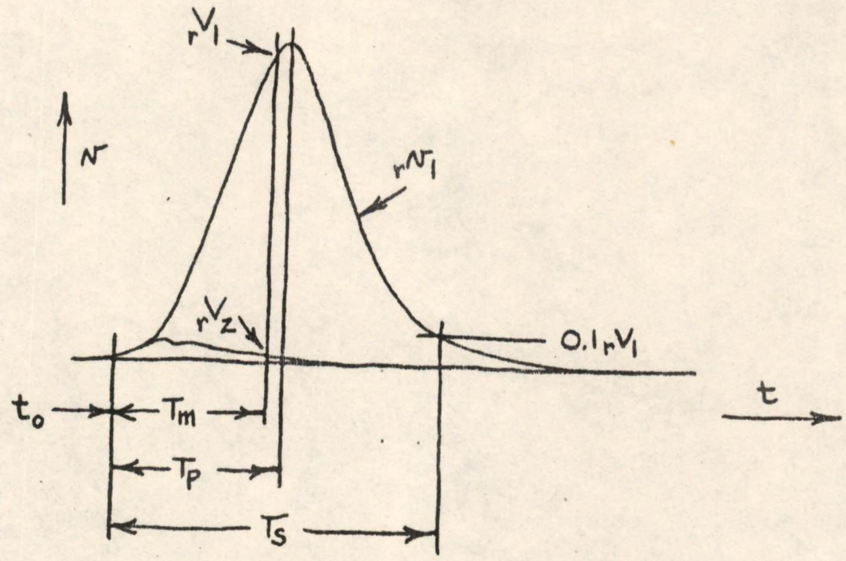
For ${}_rV_{h1}$:

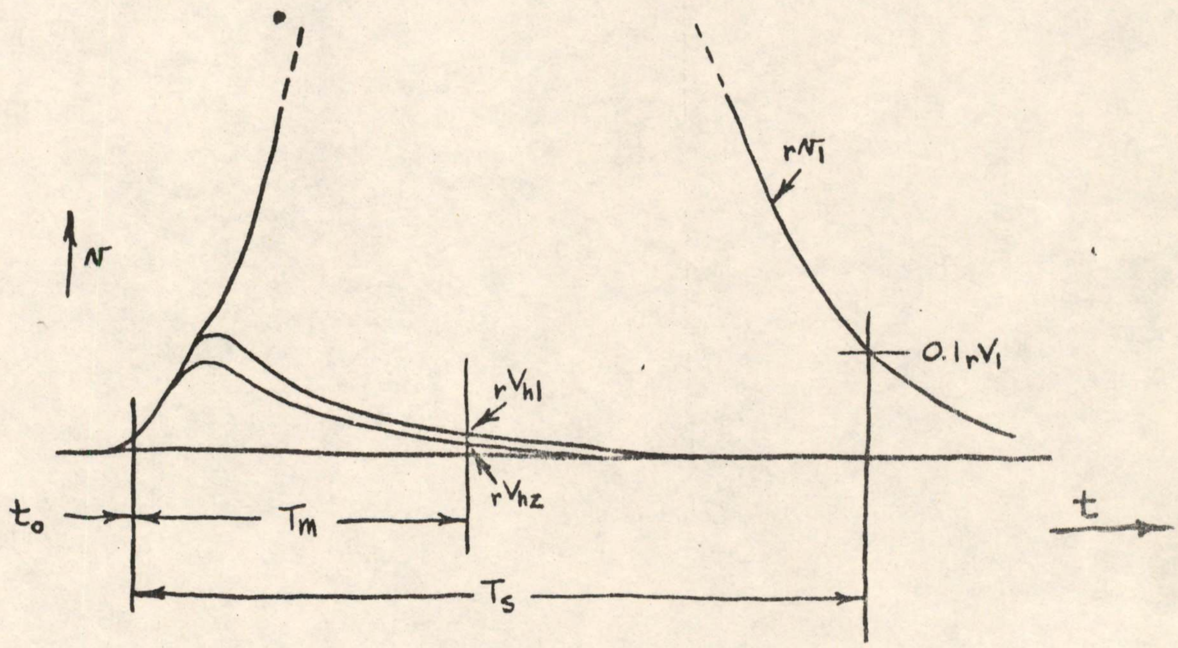


For ${}_rV_{hz}$:



Typical responses are illustrated.



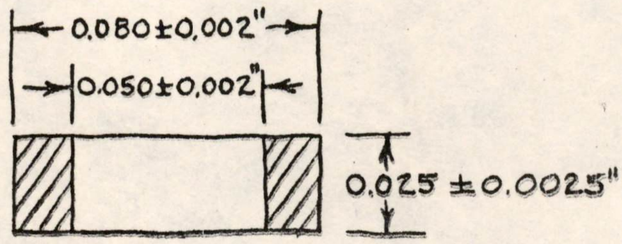


Symbol	Term
v	Instantaneous Voltage
T_s	Switching Time
T_p	Peaking Time
T_m	Sampling Time

3.0 PHYSICAL SPECIFICATIONS

3.1 Dimensions

The core is to be a toroid with rectangular cross sections.



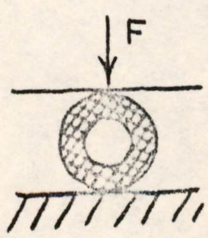
Fins left by the molding operation will not project more than 0.001 in.

Eccentricity (distance between the centers) of the inside and outside cylinders will not be more than 0.001 in.

Ellipticity (difference between the major axis and the minor axis) of the inside or outside cylinder will not be more than 0.001 in.

3.2 Fracture

The core will be placed between two parallel smooth surfaces and withstand a total compressive force of 500 grams before fracture.



4.0 ELECTRICAL SPECIFICATIONS

4.1 Test Conditions

4.11 The Current Pulse

Overshoot shall be less than 2 percent

Droop shall be less than 2 percent

$T_d = 2.0 \pm 0.1\ \mu s$

$T_r = 0.2 \pm 0.02\ \mu s$

$T_f = 0.3\ \mu s$

4.12 The Pulse Sequence

Basic prf shall be less than 2000 cps.

n = 8 (Except for determination of ${}_rV_1$ when n = 8 and n = 64. See below.)

4.2 The Pulse Response

The core shall meet the specifications listed in the table below when the core is in still air with an ambient temperature of 21° C.

I_m amp	T_m μs	T_s μs	T_p μs		${}_rV_1$ [1,2] v		${}_rV_z$ mv	${}_rV_{h1}$ mv	${}_rV_{h1} - {}_rV_{hz}$ mv
		max	min	max	min	max	max	max	max
0.900±0.009	0.60	---	---	---	0.130	---	0.4	0.2	0.1
0.820±0.008	0.64	1.25	0.55	0.70	0.095	0.115	---	---	---
0.740±0.007	0.65	---	---	---	0.070	---	0.4	0.2	0.1

1. When n is increased from 8 to 64, ${}_rV_1$ shall not change more than 2 percent.
2. The peak amplitude of ${}_rv_1$ shall fall within the limits specified for ${}_rV_1$.

4.2 <u>The Pulse Response</u> (continued)

At a temperature of 40° C, the core shall meet the specifications listed below. $T_m = 0.64\ \mu s$.

I_m amp	${}_rV_1$ v	
	min	max
0.820 ± 0.008	0.105	0.125

Signed David R. Brown
David R. Brown

DRB/jk

Air Traffic Control Project
Servomechanisms Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts

SUBJECT: INTRODUCTION TO CODING: PART I OF II PARTS

To: 6673 Project

From: David R. Israel

Date: September 29, 1949, Revised September 26, 1950
Pages 31, 42, 50, 67, 68, 69, 73.

Introduction

This report is based on a group of twelve lectures delivered by W. G. Welchman in the spring of 1949 as a part of a course in Machine Computation at the Massachusetts Institute of Technology. These lectures were given with the purpose of indicating the general nature of the sequences of operations used by high-speed digital computers in carrying out particular processes.

The accent in this report is on simplicity rather than on profundity; the object is to enable the reader to attain familiarity with the use of the various computer orders, this being done by the consideration of a number of simple codes rather than by a discussion of general theories of coding. Mathematical considerations of errors are not included and there is little attempt to discuss the relative merits of different methods of attack on a problem. The particular codes that are presented serve simply as illustrative examples and it is not suggested that these codes represent the best methods of handling the problems concerned.

In order to discuss coding it is necessary to specify the behavior of the computer for which the codes are to be written. This report is concerned with coding for Whirlwind I, a computer whose coding characteristics include the use of the binary number system, a fixed binary point, and a single address code. It is felt, however, that this report will provide a useful introduction to coding for computers with different characteristics.

Whirlwind I uses parallel operation in which all the digits of a number are handled simultaneously rather than in sequence. It seems reasonable to assume that an average speed of about 20 microseconds per operation will be achieved by this computer. With serial operation, a machine with similar characteristics could be expected to achieve an average speed of between 200 and 500 microseconds per operation.

In its present stage of development Whirlwind I is designed to handle binary numbers of a 15 digit length. This report will not consider the use of double-length (30 digit) numbers to increase the accuracy or the use of scale factors to increase the range of the machine. These matters, together with the control of input and output and certain special arrangements to facilitate the use of subprograms, will be discussed in a later report.

A computer program is defined as the sequence of operations by which a computer carries out a particular process. A computer code is the set of instructions which must be supplied the computer to enable it to execute a prescribed program. Broadly speaking, the instructions fall into two parts: the orders that tell the computer what operations to carry out, and the data upon which the computer is to perform these operations.

The principal internal elements of Whirlwind I with which we shall be concerned are the <u>storage</u>, the <u>arithmetic</u>, and the <u>control</u> elements. These three elements are connected by a communication system called the "bus". The storage element is used to hold both orders and numerical data, the arithmetic element carries out mathematical operations, and the control element insures that the correct operations are carried out according to the orders that are obtained in proper sequence from the storage.

Section I of this report contains a statement of the operational effects of the computer orders. The reader need not look at this section until it is referred to in the text but should proceed to Section II. The codes that will be described in the text are bound in a separate volume so that the reader may refer simultaneously to a particular code and to the descriptions of the orders used in that code.

TABLE OF CONTENTS - PART I

TABLE OF CONTENTS - PART I (continued)

Section I. Description of Orders

The operations which are described in this section are only those basic operations that are needed for this report. The description of the effects of these operations is incomplete, containing only what is needed for the present purposes.

For each order the following information is given:

(1) Descriptive name for the order

(2) Binary code for the order

(3) The operation to be performed

(4) The effect of the operation

(5) Comments, where necessary

Transfer operations

ca x
(1) Clear and add
(2) 10000
(3) Clear AC and add the contents of register x into it.
(4) AC - contents of register x.
BR - cleared.

cs x
(1) Clear and subtract
(2) 10001
(3) Clear AC and subtract the contents of register x from it.
(4) AC - complement of the contents of register x.
BR - cleared.

cm x
(1) Clear and add magnitude
(2) 101000
(3) Clear AC and add the absolute magnitude of the contents of register x into it.
(4) AC - positive absolute magnitude of the contents of register x.
BR - cleared.

ts x
(1) Transfer to storage
(2) 01000
(3) Transfer the contents of AC to register x.
(4) Register x contains the contents of AC, the previous contents of register x having been cleared (lost).

td x
(1) Transfer address digits
(2) 01001
(3) Transfer the right-hand 11 digits in AC to the address section of the order in register x.
(4) The right-hand 11 digits in register x are the same as the right-hand 11 digits in AC, the remaining digits of register x being undisturbed.

6673
Engineering Note E-2000-1

Arithmetic operations

ad x (1) Add
(2) 10010
(3) Add the contents of register x to whatever is in AC.
(4) AC - the arithmetic sum of the previous contents of AC and the contents of register x.
(5) An alarm signal will be given and the computer will be stopped if the magnitude of the sum (whether positive or negative) is greater than or equal to one.

su x (1) Subtract
(2) 10011
(3) Subtract the contents of register x from whatever is already in AC.
(4) AC - the arithmetic sum of the previous contents of AC and the negative of the contents of register x.
(5) The same provision for alarm and stop as in ad.

mr x (1) Multiply and round off
(2) 11000
(3) Multiply the contents of register x by whatever is in AC and round off the result to one register length.
(4) AC - the left-hand 15 digits of the product of the original contents of AC with the contents of register x, round-off having been performed (one being added to the right-most digit of AC if the sixteenth digit of the product was a one).
BR - cleared.

mh x (1) Multiply and hold full product
(2) 11001
(3) Multiply the contents of register x by whatever is in AC but do not round off.
(4) AC - the left-most 15 digits of the 30 digit product, not rounded-off, with the proper sign associated.
BR - the positive absolute value of the right-most 15 digits of the 30 digit product, followed by a zero in BR15.
(5) After the left-hand section has been stored, an sl 15 order can bring the right-hand section from BR into AC with the proper sign associated with it. The sl 15 must be performed before AC is cleared, otherwise the sign will be lost.

dv x (1) Divide
(2) 11010
(3) Divide the contents of AC by whatever is in register x.
(4) AC - +0 or -0 depending on whether the sign of the quotient is + or -.
BR - the positive absolute value of the quotient correct to 16 figures (i.e., there is no sign digit and all 16 digits are significant to allow correct round-off to 15digits by the subsequent sl order).
(5) After the dv operation the order sl 15, which must be the next order, brings the quotient into AC with the proper sign associated with it (see sl).
If the dividend is greater than the divisor, so that the quotient

dv x (5) -continued-
exceeds one, an alarm is given and the computer stopped. If the quotient equals one, the error will be detected in the subsequent sl order, for round off in the sl will cause an overflow. This is because the 16 digit quotient in BR will consist entirely of ones if and only if the divisor and the dividend are exactly equal.

Shift operations

sl n
(1) Shift left
(2) 11011
(3) Shift the contents of AC and BR n places to the left.
(4) AC - the sign digit remains unchanged. All other digits in AC and BR are shifted n places to the left and the result is rounded-off. Digits shifted left out of AC 1 are lost.
BR - cleared.
(5) As in sr the sign is sensed and remembered so that the shift and round-off can be performed using positive numbers. Digits that are shifted left out of AC 1 are lost and no alarm is given, but an overflow caused by the round-off performed after shifting is completed will give an alarm and stop the computer. The order sl 0 will be correctly interpreted, its only effect being a round-off.

sr n
(1) Shift right
(2) 11100
(3) Shift the contents of AC and BR n places to the right.
(4) AC - the former contents of AC shifted n places to the right and rounded-off.
BR - cleared.
(5) The sign is sensed and remembered and the number in AC is complemented if negative so that the shift and round off can be performed using positive numbers. The vacancies on the left-hand end are filled with zeros. After the shift and round-off the contents of AC are again complemented if the number was negative. The order sr 0 will be correctly interpreted, its only effect being a round-off. When and only when the digits in AC 1 to AC 15 and in BR 0 are all ones prior to the shift and round-off of an sr 0 order, the round-off will cause an overflow which will give an alarm and stop the computer.

Scale factor operation

sf x
(1) Scale factor
(2) 11101
(3) Shift the contents of AC and BR to the left until the first non-zero digit is in AC 1, and store the number of shifts in register x.

. Scale factor operation

sf x (cont.) (4) AC - the previous content of AC and BR so scaled that its magnitude is $\geq 1/2$ and < 1.
BR - the remaining digits of the previous contents of BR after the shift.
The address section of register x contains the number of shifts made (the scale factor). If the magnitude of the number in AC was already $\geq 1/2$ and < 1, no shift is made and zero is stored in register x. (If the number in AC and BR was identically zero, it remains zero and some indication will be given, perhaps by storing the scale factor 33)

(5) The sign is sensed and remembered and the number in AC is complemented if negative. After the scaling is completed, AC is again complemented if the number was negative. Consistent with other operations, the quantity in BR is always the positive magnitude.
The number stored in the address section of register x is a positive integer, as in the address section of an order. The left-hand five digits of register x are undisturbed, as in a td order.

Change of program

sp x (1) Subprogram
(2) 01111
(3) Transfer the register address x to the program counter.
(4) Program counter contains x.
(5) This operation does not involve the arithmetic element. The program counter determines the address of the storage register from which the next order is to be taken. After each operation the contents of the program counter are ordinarily increased by one. A subprogram order clears the program counter and substitutes the storage register address prescribed in the subprogram order itself. The next order is consequently taken from this new register address.

cp(-)x (1) Conditional program
(2) 01110
(3) Transfer the register address x to the program counter if the number in AC has a negative sign digit.
(4) Program counter contains x if the number in AC is negative. Nothing happens if the number in AC is positive.

(5) The minus sign is shown in brackets after the abbreviation cp to avoid possible confusion. It would be equally possible to have a similar order which would change the content of the program counter if the number in AC is positive, and this alternative order would be written cp(+).

Section II. Fundamental Concepts

A. Binary and Decimal Number Systems

1. A number in the decimal system is represented by a set of digits and a decimal point. The selection of these digits is restricted to the values 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The value and the position of each digit with respect to the decimal point specify a multiple of a power of 10, the multiple being equal to the value of the digit and the power being determined by its position. For example:

$$4025.809 = 4 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$$

$$+ 8 \times 10^{-1} + 0 \times 10^{-2} + 9 \times 10^{-3}$$

2. In a similar fashion a binary number is expressed by a set of digits and a binary point; in this case the digits are restricted to being either 0's or 1's. The position of each 1 with respect to the binary point specifies a power of 2. For example:

$$1010.011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1}$$

$$+ 1 \times 2^{-2} + 1 \times 2^{-3}$$

$$\text{or} \quad 2^3 + 2^1 + 2^{-2} + 2^{-3}$$

In the binary system, then, the numbers are made up of powers of 2; in the decimal system the numbers are made up of multiples of powers of 10.

3. The conversion of a number from <u>binary form to decimal form</u> may be performed by the direct (decimal system) addition of the specified powers of 2. Given the binary number 1010.011 the conversion proceeds as follows:

$$1010.011 = 2^3 + 2^1 + 2^{-2} + 2^{-3}$$

$$= 8 + 2 + 1/4 + 1/8$$

$$\text{or} \quad 8 + 2 + .25 + .125$$

$$= 10.375$$

4. The conversion of a number from <u>decimal form to binary form</u> may be accomplished by successive extractions of the proper (highest)

powers of 2 from the decimal form. Given the decimal number 10.375 this conversion proceeds as follows:

The highest power of 2 in 10.375 is 2^3 or 8:

$$10.375 - 8 = 2.375$$

The highest power of 2 in 2.375 is 2^1 or 2:

$$2.375 - 2 = .375$$

The highest power of 2 in .375 is 2^{-2} or .25:

$$.375 - .25 = .125$$

And .125 is 2^{-3}

Hence $10.375 = 2^3 + 2^1 + 2^{-2} + 2^{-3}$ or 1010.11

5. A portion of a table of binary-decimal equivalents is given below.

Decimal	Binary	Decimal	Binary
1	1	20	10100
2	10	30	11110
3	11	40	101000
4	100	50	110010
5	101	60	111100
6	110	70	1000110
7	111	80	1010000
8	1000	90	1011010
9	1001	100	1100100
10	1010	500	111110100
		1000	1111101000

6. Two rather important facts should be noted with respect to the binary and decimal representations of numbers:

(a) The representation of a number in the binary system will require approximately 3 1/3 times as many digit positions as the representation of the number in the decimal system.

(b) Multiplication of a binary number by a positive integral power of 2, say 2^k, is equivalent to shifting the binary point of the number k digits to the right; for multiplication by negative integral powers of 2, say 2^{-k} where k is positive, the binary point is shifted k digits to the left. Similar remarks are applicable to multiplication by integral powers of 10 in the decimal system.

(It should be obvious that in the above-described shifting the significant result is the relative shift between the digits and the binary (or decimal) point. This relative shift may be achieved with the digits held fixed and the binary point moved, or with the binary point fixed and the digits moved.)

7. The binary system of representation is particularly well suited for a digital computer since the machine need only distinguish and store two types of digits, 0's or 1's, instead of the ten different digits of the decimal system. The development of computers along logical lines similarly points to a "0 and 1" or "yes and no" system. Another desirable feature of the binary system is the relative ease encountered in performing arithmetic operations upon binary numbers.

8. Whirlwind I handles only information in binary form -- all numerical data is expressed as binary numbers, all computer orders are coded in the binary number system.

B. <u>Registers: Types and Identification</u>

1. The word <u>register</u> is used to denote a physical means of storing a set of binary digits. Each digit position within a register is represented by a toggle switch, relay, flip-flop, or spot position in an electrostatic storage tubes, etc. The particular digit (either 0 or 1) stored at any digit position is represented by the physical condition of the corresponding switch, relay, flip-flop, or spot.

2. The set of digits stored in a register is known as a <u>word</u>. The number of digits in a word is determined by the number of digit positions in a register or the <u>register length</u>. The register length of Whirlwind I is 16, hence all words are composed of 16 binary digits.

3. The <u>words</u> which may be stored in the registers of the computer are of two types:

(a) <u>Numbers</u> to be used by the arithmetic element.

(b) <u>Orders</u> to be used by the control element.

The representations of orders and numbers in the computer are discussed in parts C and D of this section.

4. The registers within the computer are of two types: the special-purpose registers and the storage registers. The special-purpose registers have been named in accordance with their operational uses and are referred to by those names or abbreviations thereof; the term register has been reserved, generally speaking, for use in referring to one of the storage registers of the computer.

5. Three of the special-purpose registers have particular importance in the arithmetic element of the machine. These are:

(a) the "A" register (abbreviation: AR)

(b) the "B" register (abbreviation: BR)

(c) the Accumulator (abbreviation: AC)

The 16 digit positions in the AR starting at the left are denoted as AR0, AR1, AR2 --- AR15. A similar notation is used for designating digit positions in the AC and the BR. This is shown below for the AC.

← 16 Digit Register Length →

AC0	AC1	AC2	- - - - - - - - - - -	AC13	AC14	AC15

The Digit Positions of the Accumulator

6. The BR is most conveniently thought of as the extension of the AC. In this capacity it is used to hold the second half of a product of two numbers or the quotient of a division. (In general the multiplication of an m digit number by another m digit number produces a number with 2m digits.) The effects of the various orders upon the AC and BR are described in Section I. For the present purposes it will not be necessary to consider the uses of the AR.

7. The storage registers of the computer are composed of spot positions in electrostatic storage tubes. Each register (storage register) is identified by an address in the form of a binary number. Eleven digit binary numbers are used for these addresses, permitting the identification of 2^{11} or 2048 distinct registers, the addresses running from 0 to 2047. These 2048 registers comprise the internal electrostatic storage or memory of the computer.

8. The computer identifies a register only by its address, hence in any discussion of coding it should be understood that the use of the word register implies an address.

C. Representation of Numbers

1. When registers are used to hold numerical quantities for use in calculation the content of the first (left hand) digit position indicates the sign of the number. A 0 indicates the storage of a positive number, a 1 the storage of a negative number. In writing out the contents of a register used to hold a number it is convenient to use an oblique stroke (/) to separate this sign digit from the remaining 15 numerical digits.

2. The representation of positive numbers is direct. If a register contains the digits

0/101100101000100

the number represented is the positive binary number

+ .101100101000100

where, since the sign digit is 0, the 15 numerical digits are obtained directly from the right hand 15 digits in the register. The binary point is placed at the left hand end. With this representation all positive multiples of 2^{-15} from

$$+ 2^{-15} = + .000000000000001$$

$$\text{to } 1 - 2^{-15} = + .111111111111111$$

can be stored in a register.

3. The representation of negative numbers is not direct. If a register contains the digits

1/100101000110110

the number represented is the negative binary number

\- .011010111001001

where, since the sign digit is 1, the 15 numerical digits are obtained by complementing (interchanging 0's and 1's) each of the right hand 15 digits in the register. The binary point is again placed at the left end.

4. Thus when the sign digit is 1 the numerical digits of a register must not be interpreted directly — their complements with the binary point at the left hand end and preceded by a negative sign form the desired (negative) number. With this representation all negative multiples of 2^{-15} from

$$-2^{-15} = 1/111111111111110$$

$$\text{to } -(1 - 2^{-15}) = 1/000000000000000$$

can be stored in a single register.

5. Zero has two representations, namely:

0/000000000000000 (this is called <u>positive zero</u>)

and 1/111111111111111 (this is called <u>negative zero</u>)

6. The above discussion should reveal that because of the choice of the position of the binary point we are limited in storage to numerical quantities which are multiples of 2^{-15} lying between $-(1 - 2^{-15})$ and $+(1 - 2^{-15})$.

7. A further discussion of the use of binary numbers in the computer is included in Section IV. We may remark here that it is possible to use two registers to represent a 30 digit number (two sign digits are used), and it is possible to deal with numbers outside the range $-(1 - 2^{-15})$ to $+(1 - 2^{-15})$ by the use of scale factors. The description of these techniques is left to a later report.

D. <u>Representation of Orders</u>

1. An <u>order</u> consists of two parts, of which the first specifies the <u>operation</u> to be performed and the second the <u>address</u> of the register with which the operation is concerned. (There is an exception in the case of the shift orders where the second part of the order does not specify an address but rather the extent of the shift. This matter is further discussed in Section IV.)

2. The first five (left hand) digits of an <u>order</u> determine the <u>operation</u> to be performed in accordance with the five-digit binary codes given in Section I. With five digits, 2^5 or 32 distinct operations can be specified. The remaining 11 digits of an order are regarded as forming a positive binary number with the binary point at the <u>right</u> hand end. This number thus specifies one of the 2048 storage registers.

6673

3. In the representation of an order, an oblique stroke is used to separate the operation and address sections. An example of an order would then be

3 2 1

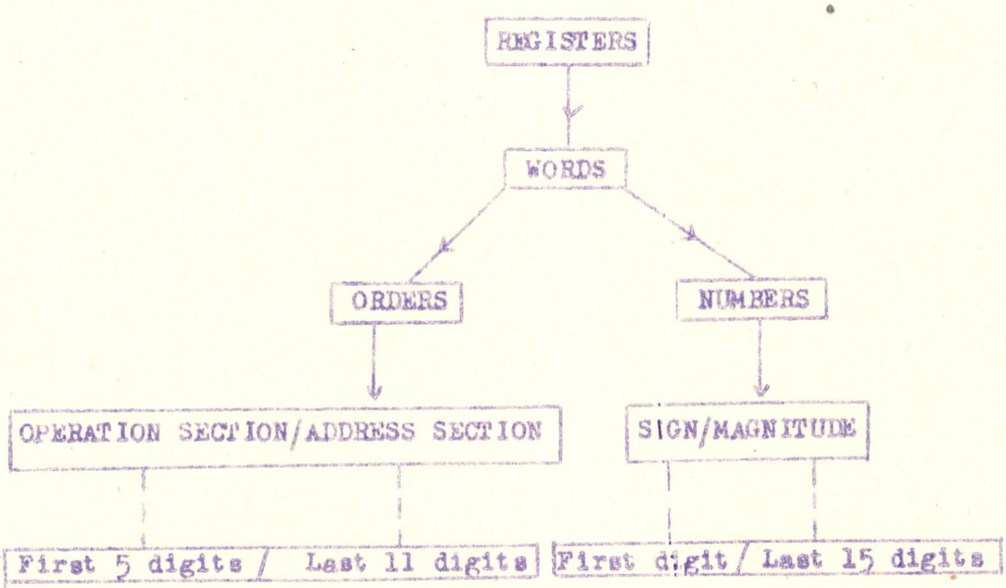
10010/00011010001

Since 10010 is the code for the ad operation and 00011010001 is the binary form of 209, the order is ad 209.

E. Orders and Numbers

1. The previous discussion of the use of registers for orders and numbers does not imply any permanent division of registers into the two types. A register which is used for an order in one program may be used for a number in another program.

2. The two possible uses of a register may be shown pictorially:



3. It should be noted that we shall be talking of two kinds of numbers. The addresses of registers are positive integers and are represented by 11 digit binary numbers with the binary point on the right, these numbers occupying only the 11 right hand digit positions of a register. On the other hand the numbers which are used in computation are multiples of 2^{-15} between $2^{-15}-1$ and $1-2^{-15}$. The representation of these numbers uses all 16 digit positions of a register, the left hand digit position being used to indicate the sign. In some cases it will be convenient to deal with an address as a numerical quantity and perform arithmetic operations on it. Due to the fact that addresses are positive integers, when an address n is being handled in the arithmetic element or being stored in a register, it should be referred to as $n \times 2^{-15}$.

F. General Operating Procedure

1. We are now in a position to discuss the general operating procedure of the computer. The control element contains a special eleven digit counter known as the program counter. This program counter holds the address of the register from which the next order is to be taken. The control element then proceeds to carry out the order whose address is in the program counter by performing the specified operation upon the quantity at the specified address section of the order.

2. For example, assume that register 973 contains the order (in binary code) ca 854, and that register 854 contains the quantity .5394 (also in binary form). Reference to the description of the operation "ca" in Section I shows that when the address 973 is placed in the program counter, the control carries out the order stored in register 973 and proceeds to place the quantity .5394 into the AC.

3. Following the completion of this order the number in the program counter is changed, thus bringing into effect a new address and a new order. In normal operation the number in the program counter is increased by one after each operation, with the result that the orders used are withdrawn from storage consecutively. Methods of changing the sequence of orders are described in Section III.

G. Written Form of a Code

1. The written form of a code requires the use of two columns, the first of which indicates the address of each register, and the second which indicates the content of that register. When the content is an order the second column splits into two parts, the first part containing the operation and the second the address; when the content of the register is a number, the second column splits into the sign and the numerical quantity (with the convention of representation of paragraphs D2 and D3 of this section).

2. Thus the entries in the code will be of the following two types:

Case	Column One (Address)	Column Two (Content)
a	Address	Operation / Address
b	Address	Sign / Numerical Quantity

3. An example of (a) would be (all quantities in decimal uncoded form for convenience):

1019 ca 346

which indicates that register 1019 contains the order "clear and add contents of register (address) 346".

4. An example of (b) would be:

1076 + .13149

which indicates that register 1076 contains the quantity + .13149.

Section III. Some Elementary Codes

We are now prepared to examine a few simple codes. Except in the case of Code I the actual codes themselves appear in Part II of the report. In accordance with a procedure to be demonstrated below the reader should follow the order-to-order progress of the codes, referring when necessary to the descriptions of the orders presented in Section I. The reader should attempt to determine the effect of each order of the codes, checking such a determination against the stated effects printed with the codes in Part II.

For convenience the codes of this section deal with literal rather than binary numerical quantities, it being understood that these quantities lie within the capacity of a register.

A. Code I: Sample Analysis of a Code

1. Code I:

1	ca	14
2	mr	14
3	mr	11
4	ts	15
5	ca	14
6	mr	12
7	ad	13
8	ad	15
9	ts	15
10	end of code	
11	a	
12	b	
13	c	
14	x	
15	--	(blank register)

2. The reader must assume, unless otherwise instructed, that the program counter initially contains the first address listed in the left or address column of a code. In this case the first address is 1. It should be noted that we are in no way restricted to beginning codes at address 1; however, we must use a group of consecutive registers for our orders. We may store orders in any group of consecutive registers provided that we make the appropriate changes in the address sections of these orders. Hence we could just as well have used registers 256 through 271 for Code I, with corresponding changes in all the addresses.

3. The effect of carrying out the order in register 1 is to put the content of register 14, $\underline{x}$, in the AC. The program counter then increases to read $\underline{2}$ and the order at that address is carried out. This order has the effect of multiplying the $\underline{x}$ in the AC by the content of register 14, or $\underline{x}$, leaving $\underline{x^2}$ in the AC. The next order, 3, multiplies the content of the AC, $\underline{x^2}$, by the content of register 11 or $\underline{a}$, leaving $\underline{ax^2}$ in the AC. The next order (that at address 4) transfers the $\underline{ax^2}$ to register 15.

4. The results thus far can be written as follows:

Address	Orders	Effects of the orders
1	ca 14	x in the AC
2	mr 14	x^2 in the AC
3	mr 11	ax^2 in the AC
4	ts 15	ax^2 in the AC ax^2 also in register 15

5. Before continuing several points must be stressed.

a) Transferring from one register to another (these may either be special-purpose or storage registers) does not alter the content of the register from which the transfer was made. In this connection special note should be made of the effect of the order at address 4 above.

b) In transferring a word into a register the transferred word is not affected by the original content of that register. (Hence the original content of register 15 is of no consequence.) In the same regard one must note the difference between the operations ca and ad.

6. The remainder of Code I can be written out as below:

Address	Orders	Effects of the Orders
5	ca 14	x in the AC
6	mr 12	bx in the AC
7	ad 13	(bx + c) in the AC
8	ad 15	$(ax^2 + bx + c)$ in the AC
9	ts 15	$(ax^2 + bx + c)$ in register 15 $(ax^2 + bx + c)$ in the AC
10	the computer is ready for another code.	

7. The action of the code has been, then, to form (evaluate) the expression $ax^2 + bx + c$, where a, b, and c are generally fixed constants and x is altered by changing the content of register 14.

B. Code II: Evaluating Polynomials

1. A similar order-by-order analysis may now be carried out for Code II in Part II. The effect of the code is seen to be the same as that of Code I, however Code II is more economical of orders (storage space) and

hence takes less time for completion.

2. The method used for forming the expression in Code II is quite applicable to general polynomial forming. Thus to form $ax^4 + bx^3 + cx^2 + dx + e$, the following order of processes should be used:

$a = (a)$	add
$(a) \cdot x = (ax)$	multiply
$(ax) + b = (ax + b)$	add
$(ax + b) \cdot x = (ax^2 + bx)$	multiply
$(ax^2 + bx) + c = (ax^2 + bx + c)$	etc.
$(ax^2 + bx + c)x = (ax^3 + bx^2 + cx)$	⋮
$(ax^3 + bx^2 + cx) + d = (ax^3 + bx^2 + cx + d)$	⋮
$(ax^3 + bx^2 + cx + d) \cdot x = (ax^4 + bx^3 + cx^2 + dx)$	⋮
$(ax^4 + bx^3 + cx^2 + dx) + e = ax^4 + bx^3 + cx^2 + dx + e$	⋮

C. Codes III and IV: Performing Division

1. It was previously mentioned (paragraph D1 of Section II) that the right hand eleven digits of an order are used to specify an address with one important exception. This occurs in the use of the shift orders (sr -- or sl --) where the address section specifies only the number of places the digits are to be shifted to the right or left. In this respect one must differentiate between the meaning of the 15 in the order ca 15 and the order sl 15. ca 15 orders the clearing of the AC and the addition of the contents of register 15 into it; sl 15 orders the shifting left by 15 places of the digits in the AC and the BR.

2. The BR is actually an extension of the AC, and the quotient is left there after a division has been performed. In a situation in which it is then desired to place the quotient of a division into the AC the dv order must be followed by an sl 15 order.

3. With the above remarks Codes III and IV should be examined. It will be observed from an examination of these codes that when a division is to be carried out the divisor should be formed first and stored. If the dividend is then formed in the AC, the division can immediately be performed by the dv order.

D. Codes V & VI: Conditional and Sub-Program Orders

1. Up to this point we have assumed that after the completion of each order the program counter is increased by 1, that is - control receives

its next order from the next (storage) register. This is not completely accurate, and when an order is completed one of three things may happen:

a) If the order was neither a sub-program order (sp) or a conditional sub-program order cp(-), the content of the program counter is increased by 1 so that the next order will be taken from the next address in storage. After the address 2047, the program counter reverts to address 0 and is ready to continue operation from there.

b) If the order was sp x, the address contained in the program counter is changed to read x so that the next order will be taken from register x.

It should be seen from this remark that after the completion of a code the next order should be sp x where x is the address of register containing the **first** order of the next code. (This will be superfluous if the two codes follow consecutively in the storage register). Codes I, II, III, and IV should be ended with <u>sp</u> orders.

c) If the order was cp(-)x the content of the program counter is changed to x if and only if the number in the AC is negative, that is-if the sign digit position of the AC contains a 1. If the sign digit in the AC is a 0 then the program continues to the next order.

2. In particular regard to (c) above it must be mentioned that the subtraction of two equal numbers gives rise to the negative zero, that is, a zero with a negative sign digit. The mathematical aspects of this are discussed in paragraph E1 of Section IV, but for the present it is important to realize that the negative zero is sufficient to cause the change in the program counter described in (c).

3. With the above discussion Codes V and VI can be inspected. These are codes which arrange numbers in a certain order. For convenience in understanding these codes one should initially assume a relationship among the numbers. That is, for Code V analyse the progress for $a > b$, $a = b$, and $a < b$. A similar analysis should be carried out for the possible relationships of a, b, and c of Code VI.

4. The analysis of Code V will reveal that the maximum of <u>a</u> and <u>b</u> is placed in register 10, the minimum in register 11. If $a = b$ the quantities are kept as originally stored.

5. The analysis of Code VI should indicate that <u>a</u>, <u>b</u>, and <u>c</u> are arranged in descending order in registers 26, 27, and 28.

Section IV. Binary Arithmetic and the Computer

A more detailed investigation will now be made of the manner in which arithmetic operations using binary numbers are performed by the computer. For convenience in representation we shall assume in this section that the available register length is only 8, with a sign digit and seven numerical digits. The binary point is assumed to be immediately to the left of the seven numerical digits, permitting the representation of numbers in the range $-(1-2^{-7})$ to $+(1-2^{-7})$. For the purposes of discussion we shall use two positive binary numbers:

$$x = +.0110100$$

$$\text{and } y = +.0011011$$

In accordance with the remarks in Section II concerning the representation of numerical quantities within registers, we see that the register representations of x and y are:

$$x = 0/0110110$$
$$y = 0/0011011$$

where, as was previously noted, the oblique stroke is used to separate the sign digit from the numerical digits.

A. Addition

1. Addition in the binary system is performed with the use of the familiar principle of the carry. We note that in the binary system 1 added to 1 gives 0 with a carry of 1, 0 added to 1 or 1 added to 0 gives 1 with no carry, and 0 added to 0 gives 0 with no carry In figures:

$$\begin{array}{cccc} 1 & 0 & 1 & 0 \\ \underline{+1} & \underline{+1} & \underline{+0} & \underline{+0} \\ 0 & 1 & 1 & 0 \end{array}$$

(carry) 1

2. The addition of x and y is as follows, two carry stages being necessary.

$$\begin{array}{rcl} x = .0110100 & = & 0/0110100 \\ y = .0011011 & = & \underline{0/0011011} \\ & & 0/0101111 \\ & & \underline{\quad 1 \quad\quad} \\ & & 0/0001111 \\ & & \underline{\quad 1 \quad\quad} \\ x + y = & & 0/1001111 \end{array}$$

3. The simultaneous addition of more than two numbers need not be considered since the computer adds only in pairs.

4. With a register length of 8 the computer is not equipped to handle numbers greater in magnitude than $1-2^{-7}$, hence arrangements must be made to stop the process of addition if at any stage the computer finds itself trying to add two positive numbers whose sum is greater than $1-2^{-7}$. This state of affairs would be indicated by a carry from the left hand numerical digit to the sign digit position. For an example, an attempt to form $2x + 2y$ would be as follows:

```
 2x = 0/1101000
+2y = 0/0110110
      0/1011110
       1
      0/0111110
      1            a carry into the sign digit position
```

This occurrence is called an <u>overflow</u> since the sum has overflowed and requires another <u>numerical</u> digit position. An alarm is sounded by the computer when such an overflow occurs.

B. <u>Subtraction and End-Around-Carry</u>

1. In accordance with the previous discussion of Section II the representations of $-x$ and $-y$ in the computer are

$-x$ = 1/1001011 , $-y$ = 1/1100100

where $+x$ = 0/0110100 , $+y$ = 0/0011011

It might be noted here that the negative of a number is represented in the computer by the complete complement of <u>all</u> digits, including the sign digit

2. Subtraction of a number is performed by adding the negative of the number. Thus the operation $a-b$ is replaced by $a + (-b)$. The addition of two numbers of which either or both are negative is performed by a process known as <u>end-around-carry</u>. The fact that this process gives the correct result will first be illustrated by esamples and then proved.

3. In the end-around-carry process the sign digits are treated exactly as if they were additional numerical digits, a carry from the left-hand numerical digit place being added in the sign digit place, but a carry from the sign digit place is taken around and added in at the other end in the right-hand numerical digit place.

4. <u>Examples of end-around-carry</u>:

Using the numbers x, y $(x > y)$ the various cases that may arise in the addition of two numbers are exemplified as follows:

(a) Both numbers positive: $x + y$
(b) One number negative, sum positive: $x-y = x + (-y)$
(c) One number negative, sum negative: $-(x - y) = y + (-x)$
(d) Both numbers negative: $-(x + y) = (-x) + (-y)$

The possibility of overflow in the addition of negative numbers, which does not occur in these examples, and the possibility of a zero sum will be considered later.

The computer's procedure in these four cases is as follows, where an end-around-carry is indicated by (1),

```
Case (a)      x = 0/0110100
              y = 0/0011011
                  0/0101111
                     1
                  0/0001111
                   1
          x + y = 0/1001111
```

```
Case (b)

         x   =   0/0110100
      (-y)   =   1/1100100
                 1/1010000
                   1 1
                 1/0011000
                 1
                 0/0011000
              (1)        1
        x-y  =   0/0011001
```

```
Case (c)

         y   =   0/0011011
        -x   =   1/1001011
                 1/1010000
                      1 11
                 1/1000110
                     1
        y-x  =   1/1100110
```

```
Case (d)

        -x   =   1/1001011
        -y   =   1/1100100
                 0/0101111
              (1)1       1
                 1/0101110
                          1
                 1/0101100
                         1
                 1/0101000
                        1
    -(x+y)   =   1/0110000
```

It is easily verified that the value obtained for $x - y$ is correct, and the values of $-(x - y)$ and $-(x + y)$ are the complements of those already found for $x - y$ and $x + y$.

5. To prove the validity of the end-around-carry procedure let z be any positive number less than 1 which is represented by seven binary digits immediately following the binary point. Denote by $\bar{z}$ the complementary positive number formed by complementing each of the seven digits of z. Then

$$\begin{aligned} z + \bar{z} &= 0.1111111 \\ &= 1.0000000 - 0.000001 \\ &= 1 - 2^{-7} \end{aligned}$$

This equation may be written in the form

$$z + (1 + \bar{z}) = 2 - 2^{-7} \quad \text{-----------} A$$

The number $1 + \bar{z}$, being greater than one, cannot be represented in the computer, but its digits are precisely those used in the computer to represent the negative number $-z$, if the sign digit is included.

6. Let x and y be two values of z, and take $x > y > 0$. Assume also that $(x + y) < 1$ to avoid overflow. The cases b, c, d of paragraph 4 will be considered in turn.

Case (b)

The computer's procedure in forming $x + (-y)$ with end-around-carry is equivalent to three successive operations

(a) adding the number $1 + \bar{y}$ to x

(b) subtracting $10.0000000 = 2$

(c) adding $0.0000001 = 2^{-7}$

the last two operations being equivalent to the end-around-carry. But from equation A for $z = y$

$$-y = (1 + \bar{y}) - 2 + 2^{-7}$$

so

$$x - y = x + (1 + \bar{y}) - 2 + 2^{-7} \quad \text{-----------} B$$

This proves that the computer process produces the correct result for $x - y$ provided that end-around-carry occurs. This must be so because from B

$$x + \bar{y} = (x - y) + 1 - 2^{-7} \geq 1 \text{ if } x > y,$$

from which it follows that the addition of x and $\bar{y}$ must produce a 1 in the sign digit place, which with the sign digit corresponding to -y must produce an end-around-carry.

Case (c)

Similarly the validity of the procedure for obtaining the negative sum y + (-x) = - (x - y) is proved as follows. Equation A applied to the positive number x - y gives

$$\begin{aligned} 1 + \overline{x - y} &= -(x - y) + 2 - 2^{-7} \\ &= y - x + 2 - 2^{-7} \\ &= y + (1 + \bar{x}) \text{ ----------------------C} \end{aligned}$$

The digits of the positive number $1 + \overline{x - y}$ are precisely the digits that the computer ought to obtain in order to represent the negative number -(x - y) and the digits of $y + (1 + \bar{x})$ are those that the computer actually does obtain, provided that there is no end around carry. In this case end-around-carry cannot occur because from equation C

$$y + \bar{x} = \overline{x - y} < 1.$$

Case (d)

Consider finally the negative sum of two negative numbers (-x) + (-y) = -(x + y). If $x + y \geq 1$ we have an overflow condition, which will be discussed in section C below. If $(x + y) < 1$ equation A gives

$$\begin{aligned} 1 + \overline{x + y} &= -(x + y) + 2 - 2^{-7} \\ &= (-x + 2 - 2^{-7}) + (-y + 2 - 2^{-7}) - 2 + 2^{-7} \\ &= (1 + \bar{x}) + (1 + \bar{y}) - 2 + 2^{-7} \end{aligned}$$

This shows that the computer will obtain the correct representation of the negative number -(x + y) because an end-around-carry is produced by the two negative sign digits of -x and -y.

C. Overflow.

1. It has already been explained that, if the computer is trying to add two positive numbers x, y whose sum $(x + y) \geq 1$, it must detect the overflow and stop the computation. In forming the sums x + (-y) and y + (-x) there is no danger of obtaining a sum whose magnitude is ≥ 1, but the computer, when trying to add two negative numbers (-x) and (-y), must guard against the possibility that the negative sum -(x + y) may be ≤ -1. When (-x) and (-y) are being added, with the end-around-carry, the sign digits are both 1, and their addition will leave a 0 in the sign digit place. There must therefore be a carry from the left-hand numerical digits to give the 1 in the sign digit place that will indicate that the sum is negative. The absence of this carry digit will indicate an overflow.

2. To prove this we notice that when the computer adds $(-x)$ and $(-y)$ an end-around-carry is immediately produced by the addition of the sign digits. The sum formed by adding the numerical digits is therefore

$$\bar{x} + \bar{y} + 2^{-7} = (1 - 2^{-7} - x) + (1 - 2^{-7} - y) + 2^{-7}$$
$$= 1 - 2^{-7} + 1 - (x + y)$$
$$< 1 \text{ if } (x + y) > 1$$

3. The overflow control that the computer must provide when it is adding two numbers is therefore as follows:

Both numbers positive: - Overflow signalled if carry occurs into the sign digit place.

One number negative : - No action.

Both numbers negative: - Overflow signalled if no carry occurs into the sign digit place.

D. Shifting and Roundoff

1. Multiplication and division will both be accomplished by a combination of additions, shifts and complementing operations. It will not be necessary to specify the methods used in detail, but it is important to stipulate that the actual operations of multiplication and division will be carried out with positive numbers, any necessary complementing being done at the beginning and end of the process.

2. Shift operations also will be performed on positive numbers. Thus if a negative number is to be shifted it is first complemented, then the complement is shifted and finally the result is complemented.

3. The operations of multiplication and shift right produce digits in the B register and the content of the accumulator may be rounded off by adding in a one in the right-hand place of the accumulator if the left-hand digit in the B register is a one. (This round off is optional in multiplication but will always be carried out in shifting right.)

4. The operations are so arranged that round-off is only carried out on positive numbers. For example, in forming the rounded off product $x \cdot (-y)$ the machine calculates the full product xy of the positive numbers x and y and rounds off this positive product before complementing to obtain the rounded off value of $-xy$.

5. In a shift right of a positive number the vacated digit places at the left of the accumulator are filled with zeros and, after the round-off has taken place, the B register is cleared. (Clearing means making all digits zero.) Thus, if

x = 0/0110100, the process of shifting right three places is as follows, the ∠ indicates the beginning of the BR:

Initial value of x	0/0110100
First stage of shift	0/0000110∠100
Round-off effect	1
Result	0/0000111

If -x = 1/1001011 is to be shifted right three places, the first step is to complement, giving x. The above procedure is then carried out, giving 0/0000111, which is complemented to give the result 1/1111000. Thus

-x	1/1001011
Complement	0/0110100
Shift right three places	0/0000110∠100
Round-off effect	1
Shifted complement	0/0000111
Complement back	1/1111000

6. As an example of a case in which round-off does not produce the addition of a one in the right-hand place of the accumulator, consider the process of shifting x four places to the right, which is as follows:

Initial value of x	0/0110100
Shift right four places	0/0000011∠0100
Round-off effect	0
Result	0/0000011

For the negative number -x the procedure would be

-x	1/1001011
Complement	0/0110100
Shift right four places	0/0000011∠0100
Round-off effect	0
Complement back	1/1111100

E. Zero.

1. In the above discussion we have assumed $0 < x < 1$, $0 < y < 1$, $(x - y) > 0$. We now consider what happens when $x - y = 0$, when $y = 0$ and when $x = y = 0$.

2. In the computer the number zero has two representations

(a) 0/0000000
and (b) 1/1111111

The sets of digits (a), (b) have the same meaning. However, as they look different on the computer it is reasonable to refer to them as positive zero and negative zero.

3. It will often be convenient in a computation to use the sign digit of some intermediate result to determine which of two alternative courses the computer shall follow. We must therefore consider carefully whether a zero occurring during the course of a computation will appear as a positive or a negative zero.

4. It has been remarked that subtractions are replaced by additions of complements, and that multiplications and divisions are performed by a series of additions, complementings and shifts. Now the sum of a number and its complement, i.e., $x + (-x)$, appears on the computer as the negative zero. For example

$$\begin{array}{rl} x = & 0/0110100 \\ -x = & \underline{1/1001011} \\ 0 = x-x = & 1/1111111 \end{array}$$

(Note that in the discussion of case (b) in section B, equation B gives $x + \overline{x} = 1 - 2^{-7}$ if $y = x$, showing that the addition of x and $\overline{x}$ gives no carry into the sign digit place, so that no end-around-carry takes place.) In particular the result of adding a positive zero to a negative zero is a negative zero.

5. The only other way in which zero can arise by addition is when two positive zeros or two negative ones are added together. These can give

$$\begin{array}{rl} 0 = & 0/0000000 \\ 0 = & \underline{0/0000000} \\ 0 = 0+0 = & 0/0000000 \end{array}$$

and

$$\begin{array}{rll} 0 = & 1/1111111 & \\ 0 = & \underline{1/1111111} & \\ & 0/0000000 & \\ & \underline{(1)1\ 1111111} & \text{end-around-carry} \\ 0 = 0+0 = & 1/1111111 & \end{array}$$

It appears therefore that a subtraction can only give rise to a positive zero when both numbers involved are zero. In fact, provided $x \neq 0$ we can be sure that the result of the subtraction $x - x$ will be a negative zero.

The occurence of positive and negative zeros in multiplication and division is not so important and will not be discussed. It is perhaps worth remarking that a shift right leaves a negative zero unchanged, because complementing occurs before and after the actual shift.

We have now examined what happens in the four cases of addition when $x - y = 0$ and when $x = y = 0$. Consider now $x \neq 0$ and $y = 0$. It follows from the investigations of cases (a) and (c) that, if y is represented by the positive zero, the addition of y to a non zero number has no effect on that number. Also, if in the investigations of cases (b) and (d) in section B we write

$$\underline{y} = .0000000$$
$$\overline{y} = .1111111$$

$$y + (1 + \overline{y}) = 2 - 2^{-7}$$

it follows that the addition of negative zero to a non zero number does not change that number.

Section V. Codes of a More Complex Nature

A. Codes VII and VIII: Further Illustrations of Shifting

1. The round-off effects of the sr order were illustrated in paragraph D of Section IV. A particular use of the sl order will now be demonstrated. (Refer to qualitative discussion of the order in Section I.) Assume the original contents of the AC were:

0/000000000100000

with the BR cleared (filled with zeros). If the first order is sl 10 the AC changes to:

0/000000000000000

where we have lost a 1 since any digits shifted out of AC1 are lost. (Note: because the BR was originally cleared we have no round-off at AC15.) If we now follow with a sl 10 order we are left in the AC with:

0/000000000000000

It is important to note that a sr 0 or sl 0 order on the original contents of the AC would not produce this result, but rather

0/000000000100000

2. Code VII, it will be noted, counts from 0 to 31×2^{-15} in steps of 2^{-15}, each count appearing in register 15. After 31×2^{-15} is reached the count begins again at 0.

3. Code VIII also performs this cycle count but does so using fewer orders and storage registers. This code employs the shifting scheme shown in (1) above so that when the count reaches 32×2^{-15} (0/000000000100000) the sl 10 and sr 10 orders clear the AC.

B. Modification of Orders

1. To facilitate the coding of more complex problems it becomes desirable to make use of several operations and modifications that may be performed upon orders. These operations and modifications are possible because of the fact that in the machine the orders have the same physical characteristics (16 digit binary numbers) as the numerical quantities. (Orders and numbers appear different when written on paper due to the oblique stroke used to separate the two parts of each.)

2. Suppose that the order under consideration is ca 183. This is represented in a register as

10000/00010110111

where 10000 is the binary coded form of ca and 00010110111 is the binary equivalent of 183. The result of adding $+ 2^{-15}$, represented by 0/000000000000001, to ca 183 gives (neglecting the oblique strokes)

$$
\begin{array}{rcl}
(\text{ca } 183) & = & 1000000010110111 \\
+ (+ 2^{-15}) & = & \underline{0000000000000001} \\
 & & 1000000010111000
\end{array}
$$

= 10000/00010111000 or ca 184

Thus the addition of 2^{-15} has increased the address section of the order by 1 without affecting the operation. The coding of this modification of the address section would be as follows, if it is assumed that register 205 contained the order ca 183 and register 958 contained $+ 2^{-15}$:

ca 205
ad 958
ts -- (to where the modified order is desired)

It is easily seen that an extension of the above method will permit the modification of address sections of orders by other values than 1.

3. As an other example of operations with orders consider the effect of the following sequence of orders:

ca 350
su 194

where register 350 contains the order ca 184 (10000/00010111000) and register 194 contains ca 183 (10000/00010110111):

$$
\begin{array}{lr}
\text{ca 350 puts into the AC} & 1000000010111000 \\
\text{su 194 adds } -(\text{ca } 183) \text{ or} & \underline{0111111101001000} \\
 & 1111111111110000 \\
\text{carry} & \underline{1} \\
 & 0000000000000001 \quad \text{or } + 2^{-15}
\end{array}
$$

4. In general the subtraction of two orders containing the same operation sections results only in the difference of the address sections of those orders multiplied by 2^{-15}. In view of the remarks in E of Section IV, if both orders are equal the result of the above operation is a negative zero.

5. The above mentioned operations and modifications of orders are used extensively in coding and will be illustrated in the codes which follow.

C. Codes IX and X: Cyclic Programs

1. One of the desirable aspects of Whirlwind I is the feature of modifications and operations which the computer can perform upon its orders. When this feature is coupled with the use of the cp (-) order the machine is able to perform cyclic programs in which the same orders or slight modifications thereof are used over and over.

2. The use of cyclic programs permits a good deal of saving in computer storage space, this being illustrated in a comparison of Codes IX and X. In these codes due to the large number of storage registers involved literal coefficients along with numbers are used for address designations. An A is used to refer to the registers containing orders whereas B is used for numerical data registers.

3. Code IX evaluates a polynomial in a linear fashion using the process indicated in Section III paragraph B2. The code uses $2n + 3$ orders. Code X also evaluates a polynomial using a cycle to repeat the similar sequence of orders. This necessary sequence in Code X, orders A3 through A6, have their address sections changed to permit handling the various a coefficients. The subtraction of two orders is used to give the necessary + or - quantity to permit regeneration of the cycle. Code X uses 15 orders and thus if $n \leq 6$ Code IX would be used, for $n > 6$ Code X would be more economical.

4. It should be noted that a cyclic code always requires a longer operating time (more orders are carried out) than a linear code due to the modification of orders during the cycle; in either type of code, however, the amount of useful arithmetic operation carried out is the same.

5. A particular advantage of the cyclic code in such a problem as polynomial evaluation is what might be termed the alteration possibility. By this term we refer to the ease by which a code is altered to extend its range of operation, whether it be the change of the degree of the polynomial formed or the extent of an iteration. In our example if the degree of the polynomial were increased from the value n, Code IX would require only the change in the address section of order A1; in both codes the same number of additional constants must be added.

6. A primary requirement of a cyclic code is that it be self-resetting. By this one means that if the address sections of orders are modified in the progress of a program these addresses must be restored to their initial values before the program is used again. This resetting can be done either as the program begins or as the program is completed. Code X is reset at the beginning of the program with orders A1 and A2.

D. Code XI: Scale Factoring and Overflow

1. Because of the restricted range of numerical quantities with which the computer can work $\left[-(1-2^{-15}) \text{ to } +(1-2^{-15})\right]$ we are faced with two requirements:

(a) Most numerical quantities before they can be introduced into the computer will have to be scale-factored, that is, multiplied by an appropriate number to bring them into the computer range.

(b) Particular care must be taken in a computer program to ensure that no overflow occurs as the result of arithmetic operations.

2. Code XI illustrates a simple example of addition in which the necessary scale factoring has been done before the quantities are inserted into the computer storage and in which the program is designed to prevent overflow. The problem is to add the n angles $\theta_1, \theta_2, \theta_3 ---- \theta_n$, where each angle is expressed in radian measure and each has a value such that $-2\pi < \theta_i < 2\pi$ for i = 1, 2, 3 ---- n.

3. Due to the stated range of θ_i, we are assured that $\frac{\theta_i}{4\pi}$ (for i = 1, 2 ---- n) lies within the accepted computer range and it is these values which are stored in registers 16 through 15 + n. Despite the initial scale factoring of these angles we forsee that the total sum of these scale factored angles may exceed the computer range unless we add the $\frac{\theta}{4\pi}$'s and cast out all multiples of 1/2, these corresponding to multiples of 2π in the θ's.

4. The procedure used in this code is to cast out a 1/2 each time it appears in the summation by the use of a sl 1 order followed by a sr 1 order. These sl and sr orders keep the summation below 1/2 at all times, and since all the remaining angular quantities to be added are less than 1/2 as a result of the scale factoring, we will not get an overflow.

5. The repeated summations and use of the sl 1 and sr 1 orders suggested the use of a cyclic code. The determination of the end of the summation is made by a subtraction of orders, the cycle ending when the difference of the two orders produces $+ 2^{-15}$. The final result of the summation appears in register 17 + n.

6. The reset of the address sections is carried out by orders 1, 2, 3, 4, and 5.

7. It is important to remark upon the previously mentioned point that shifting digits is similar to multiplication by powers of 2. Reference to the description of the orders in Section I should indicate an important difference between the shifting and multiplication orders as regards round-off considerations.

E. Code XII: Finding the Greatest of A Set of Numbers

1. Code XII provides another illustration of the manner by which the computer can change its own control instructions during the course of a program.

2. The numbers x_1, x_2 ---- x_n, all of which are positive and less than 1, are stored in consecutive registers C1, C2, ---- Cn. The code finds the greatest of these numbers, say, x_M, and stores it in B4. If two or more equal numbers are greater than all the other numbers of the set, the program stores the first of these in B4.

3. The program depends on changing the address sections of the orders A2.1 and A2.2 in such a manner that for the successive values 2, 3, 4 ---- n of m, the number x_m is compared with the number that has already been found to be the greatest of x_1, x_2 ---- x_{m-1}. This number will be called the temporary maximum.

4. The procedure of the computer is similar to that of a man running his eye down a set of numbers. The man would remember the first number until he reaches a greater number, which he would then remember until he reaches a still greater number, and so on. In fact the man compares each number in turn with the greatest of the numbers that he has previously examined. In Code XII the computer does the same thing except that it remembers the address at which a number is stored rather than the number itself.

5. The central part of the code is contained in Sections A2, A3, and A4. The first section, A1, resets the orders which may have been altered

during a previous application of the program. Section A5 determines if all the numbers have been dealt with. If not the computer returns to A2; if all numbers have been dealt with, section A6 puts the maximum number into B4.

6. The eight operations of sections A1 and A6 are used only once in a program; the eight operations of A2, A4, and A5 are used only once for each of the numbers x_2, x_3 ---- x_n. The 2 operations of A3 only occur when the number being examined is greater than all numbers previously examined. The total number of operations lies between 8n and 10n - 2. Assuming an average speed of 20 microseconds per operation and taking n = 1000, the time required to find the greatest set of 1000 numbers by this method is between 160 and 200 milliseconds.

Section VI. Coding Notation and Procedure

A. Notation for Coding

1. A program is a sequence of operation by which the computer carries out a particular process. The code for a program is the set of instructions that must be put into the computer's storage to enable it to carry out the program. Thus a code is essentially a statement of the initial content of the registers that are to be involved in the program, that is, the content immediately before the program starts. These registers are of two kinds:

(a) Action registers, from which the computer control obtains its instructions.

(b) Data registers, which are used to store other information that may be needed during the program.

(This distinction applies only to the way in which registers are selected for use in a particular program. Any register in the computer's storage can be assigned for use either as an action register or as a data register.)

2. The content of registers of both kinds may be changed during the program. For example, a particular data register may contain an order which may be transferred to an action register as the result of a comparison operation. The coder who is drawing up a code will have to keep track of the content of all the registers at all stages of the program, but the computer must be given the initial content, so the code must show the initial content only. (A distinction is drawn here between the code itself and any explanatory notes that may accompany a write up of the code.) In many cases part or all of the initial content of a register may be immaterial because the content is going to be supplied during the course of the program before that register is used. In writing out a code a dash is used to indicate this state of affairs. (Note the distinction between "ca--" and "ca zero".)

3. In Section II it was explained that in discussing coding the use of the word register implied an address. When we refer to the register containing a particular number or order we are usually, if not always, thinking of the address of the register though for the sake of brevity of language we do not mention the word address. Similarly when we talk about the content of a register, we are thinking of the register as identified by an address. This is reflected in the following abbreviations which are commonly used:

RC--- = (address of) Register Containing-----

CR--- = Content of Register (whose address is --)

4. The addresses that will actually be used when a code is put into the computer are not usually known when the code is being drawn up, so symbols are used to denote the actual addresses. To obtain a concise, written record of a code it is best to represent the addresses of the action and data registers by a set of consecutive serial numbers. This will be called the <u>serial notation</u>.

5. It would be natural to start these serial numbers at 1, and for the present this should be done. However, it may be decided to allot some of the registers of the computer to the storage of certain universal constants, and in particular registers with addresses 1 to 15 may be allotted to powers of 2, so that for $n = 1, 2, \ldots 15$

$$CR\ n = 2^{-(16-n)}$$

If this is done the serial numbers will have to be chosen so that they do not contain numbers that are addresses of registers allotted to special purposes. At the end of the action registers of a code a serial number should be reserved for an sp order that will switch the computer to its next job.

6. In what follows a standard notation is described which makes it easier to follow the execution of the program. This standard notation is also more convenient than the serial notation for use when a code is being prepared.

7. A <u>flow diagram</u> is a series of statements of what the computer has to do at various stages in the program. These statements are written in boxes and the boxes are joined by lines of flow which show how the computer passes from one stage of the program to another. When the procedure of the computer after a particular stage depends on a cp(-) order, the statement in the corresponding box is so worded that the lines of flow emerging from the box can be labelled "yes" and "no". When a code has been completely worked out the main object of the flow diagram is to make the main structure of the program clear. During the process of working out a code the flow diagram, by separating the program into stages, makes it easy to introduce alterations as coding proceeds.

8. In the process of solving a problem the first tentative step is to divide the program into a few main stages and to draw a flow diagram whose boxes will contain broad statements of what the computer must do. These main stages are denoted by A1, A2, A3, ... Each of these stages is then further analysed and, if necessary, is divided into substages, represented by boxes in a more detailed flow diagram. The substages into which A1 is subdivided are denoted by A1.1, A1.2, A1.3, ..., and similarly for A2, A3, ... If necessary, some of these substages are further subdivided into A1.1.1, A1.1.2, ..., A2.1.1, A2.1.2, ..., etc. Only experience will show how much subdivision is desirable, but it should be remembered that the two main objectives are to make the structure of the program clear and to make it easy to introduce alterations as coding proceeds.

9. It is often desirable to introduce more subdivisions in the working stage than will be used in the final write up of the solution. When a problem has been completely coded and the code is written out in serial form the serial numbers representing the action registers which correspond to the various boxes of the flow diagram should be written in the boxes.

10. An exception to the notation indicated above may be made in the case of auxiliary subprograms that are not to be written out in the solution. These subprograms can be denoted by Aa, Ab, ..., and their stages and substages by symbols such as Aa 5.2.

11. The action registers contain the program orders. In the standard notation they are grouped into blocks corresponding to the stages into which the program is divided in the flow diagram. The addresses of the registers that contain the successive program orders that are required for stage A3.2 will be denoted by A3.2.1, A3.2.2, A3.2.3, ..., which will be called <u>index numbers</u>. As was explained we shall refer to "the register A3.2.2", although strictly speaking the index number A3.2.2 represents the address of a register. No confusion will be caused if we refer to the order in register A3.2.2 simply as "the order A3.2.2".

12. A system of index numbers is also needed for data registers, but coding problems differ so much in their nature and complexity that it seems undesirable to lay down rigorous rules for the representation of the addresses of data registers. It seems reasonable to suggest that only the letters B and C should be used, and that any further subdivisions into blocks of registers of different types that may be necessary should be achieved by a notation similar to that used for action registers. In simple problems it may be desirable, at any rate at the working stage, to introduce no subdivisions but merely to allot temporary index numbers, B1, B2, B3, ... to each data register as the need for it arises. We shall refer to registers grouped under the letters B and C as B-class and C-class registers. Action registers may be called A-class registers. In subprograms denoted by Aa, Ab, ..., the data registers should be denoted by Ba, Bb, ..., and Ca, Cb, ...

13. One subdivision that will often be desirable is as follows:

<u>B-class registers</u>

Data that will differ in different applications of the program, possibly further subdivided into

B1. Input and output data.
B2. Data derived for use during the program.

<u>C-class registers</u>

Fixed data used for all applications of the program, possibly further subdivided into

C1. Universal constants stored in fixed registers.
C2. Other constants.

14. Other subdivisions that may sometimes be desirable are:

(a) The distinction between

(1) Data or registers used in some other program that is on the computer at the same time.

and

(2) Data or registers that are used only in this particular program.

(b) The distinction between

(1) Data registers whose addresses occur in the address sections of action registers in their initial state at the beginning of the program.

and

(2) Those whose addresses are derived during the program (as happens in the case of registers containing tabulated values of a function).

15. The form for writing out a code has been described. In the standard notation the addresses will be indicated by the index numbers of the action and data registers. For action registers the explanatory notes, which are to be given in a separate column on the right, contain statements of the following types:

(a) The content of AC, BR or some storage register resulting from the order in question. For a cycle the explanatory notes should refer to what happens when the cycle is being performed for the mth time. For a subcycle in a cycle the explanatory notes should refer to what happens when the subcycle is being performed for the k-th time as part of the m-th performance of the cycle.

(b) References to the origin of

(i) the address section of an order
(ii) the operation section of an order
(iii) the numerical quantity contained in a data register

when any one of these is changed during the program.

(c) References to sp or cp(-) orders that cause the control of the computer to jump to the order in question.

16. In the tabulation of the initial content of data registers explanatory notes should be added on the right in the case of any register whose content is changed during the program showing the various successive contents of the register and the orders from which they are derived.

17. The write-up of a program should include orders to cover any restoration that may be necessary to ensure that at the end of a particular application all registers are correctly set up for another application. (The need for such restoration orders at the end of a program will usually be avoided either by the insertion of suitable orders at the beginning or by including in the input data that is supposed to be supplied before the program starts the content of the registers that have to be dealt with.)

18. The write-up of a code should include statements of:

(a) The position in storage of input and output data.
(b) The total number of registers used.
(c) The total number of operations to be performed.

(Note that in calculating the total number of operations to be performed allowance must be made for the number of times that the computer has to go through any cycle that may occur in the program. In many cases it will not be possible to state a definite number, but only maximum and minimum numbers.)

19. The general form of the tabulation of a code and the index numbers used to represent the addresses of action and data registers have been discussed. In the standard notation, when an entry in the tabulation of the code represents an order, the second part of the second column contains the index number of the register to which that order applies. During the early stages of work on a problem, or for explanation of a code on a blackboard, it may often be more convenient to indicate in this place the content of the register referred to, rather than the index of that register. For this purpose the symbol RC, will be used. Thus if B17 contains 2^{-15} we may write

ad RC 2^{-15}

instead of ad B17.

20. In the standard notation the code itself is distinguished from the explanatory notes. The code shows only the inital content of the action and data registers, any indication of changes in content during the course of the program being confined to the explanatory notes. In the working notation the symbol RC, when it is used in the address section of a program order, indicates the (address of the) register whose content at that particular stage of the program is a certain quantity, although the initial content of that register may have been something else. In fact, this working notation allows explanatory matter to get into the columns which in the standard notation are strictly reserved for the code itself.

B. Example of Coding Procedure

1. This section starts with a provisional analysis of a problem leading to a first attempt to write out a code. This first attempt is then criticized and a number of improvements are made before the code is written out in a finished form.

2. We are given a set of n tabulated values of f(x) for the values $x_1, x_2, \dots x_n$ of the variable x. The differences between successive tabular values of x are all equal to a positive constant h, so

$$x_i = x_1 + (i-1)\ h.$$

Further

$$-1 < x_1 < x_n < 1.$$

3. It is supposed that during the course of computation two numbers a and b are derived by the computer, and a subprogram is wanted to find $\int_a^b f(x)\ dx$, using the trapezoidal rule which gives the formula

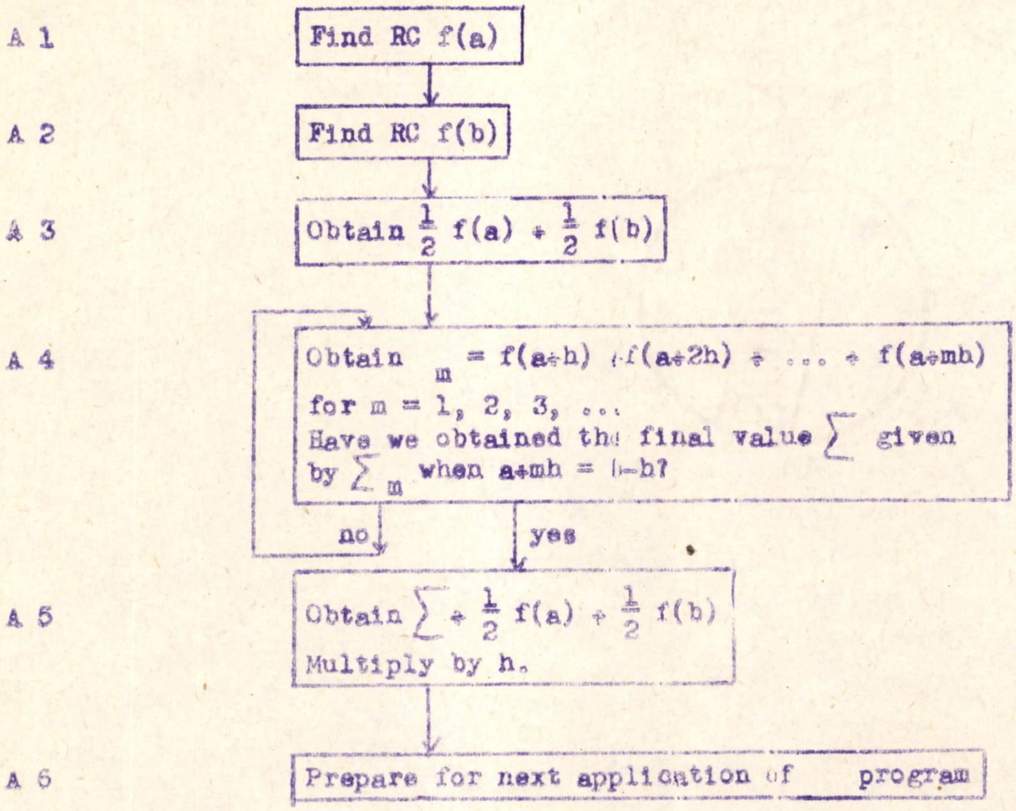
$$\int_a^b f(x)\ dx = h\left[\frac{1}{2} f(a) + f(a+h) + f(a+2h) + \dots + f(b-h) + \frac{1}{2} f(b)\right]$$

4. We can presume that a and b are known to be within the range convered by the tabular values $x_1, x_2, \dots x_n$ and further that $(b-a) \geq 2h$, so that there is at least one term inside the bracket of the formula in addition to $\frac{1}{2} f(a)$ and $\frac{1}{2} f(b)$. We are also told that $|f(x_i)| < \frac{1}{n}$ for all x_i, so there is no danger of overflow if we simply add the terms $\frac{1}{2} f(a), f(a+h), \dots, \frac{1}{2} f(b)$, since the total number of terms to be added together is less than n.

5. We shall actually calculate the integral from the tabular value nearest to a to the tabular value nearest to b, neglecting the errors thereby introduced at the ends of the range of integration. We shall not distinguish between these values and the exact values of a and b as derived by the computer.

6. We will first lay down a straightforward method of dealing with the problem, representing the method by a flow diagram and allocating blocks of A-class registers to the successive stages of the flow diagram. We shall then write down a code stage by stage, allocating B-class registers as the need for them arises. In this problem it is not necessary to subdivide the B-class registers into groups and there is no need to introduce a C-class. We shall need n consecutive B-class registers for $f(x_1), f(x_2) \dots, f(x_n)$, but we can add them at the end of the list of B-class registers required for other purposes. The allocation of B-class registers, which is built up during the process of coding, is shown in paragraph 15.

7.



8. <u>Stage One</u>. Suppose we have 2^{-15} RC f(x_1) stored in B 1. (This number may not be fixed until the program is actually put on the machine.) Then since the successive tabulated values of f(x) are stored in consecutive registers we have the equation

$$\text{RC } f(a) - \text{RC } f(x_1) = \frac{a-x_1}{h}$$

where the right-hand side must be rounded off to the nearest integer. This gives

$$2^{-15}\text{ RC } f(a) = 2^{-15}\text{ RC } f(x_1) + \frac{2^{-15}}{h}\ (a-x_1).$$

We shall have to make sure that the computer in calculating $\frac{2^{-15}}{h}$ $(a-x_1)$ produces the correct round off, but we shall leave this point for the present. It will be discussed in paragraphs 19 and 20. We store a, $\frac{2^{-15}}{h}$ and $\frac{2^{-15}x_1}{h}$ in B2, B3 and B4 and proceed as follows:

A 1.1	ca	RC a	AC: a
2	mr	RC $\frac{2^{-15}}{h}$	AC: $\frac{2^{-15}a}{h}$
3	su	RC $\frac{2^{-15}x_1}{h}$	AC: $\frac{2^{-15}}{h}\ (a-x_1)$
4	ad	B 1	AC: 2^{-15} RC f(a)
5	td	A 3.1	A 3.1: ca RC f(a)
6	ad	RC 2^{-15}	The need for these two orders arises in stage four
7	td	A 4.2	A 4.2: ad RC f(a + h)

9. Stage Two.

A 2.1	ca	RC b	AC: b
2	mr	RC $\frac{2^{-15}}{h}$	AC: $\frac{2^{-15}b}{h}$
3	su	RC $\frac{2^{-15}x_1}{h}$	AC: $\frac{2^{-15}}{h}\ (b-x_1)$
4	ad	B 1	AC: 2^{-15} RC f(b)
5	td	A 3.2	A 3.2: ad RC f(b)

10. Stage Three.

A 3.1	ca	RC f(a)	Initially ca-- Digits from A 1.5 AC: f(a)
2	ad	RC f(b)	Initially ad-- Digits from A 2.5 AC: f(a) + f(b)
3	sr	1	AC: $\frac{1}{2}$ f(a) + $\frac{1}{2}$ f(b)
4	ts	B 7	

11. Stage Four: In the cyclic process we want to add $f(a+h)$, $f(a+2h)$ ... successively, so we need an order ca RC $f(a + mh)$, or alternatively ad RC $f(a + mh)$, whose digits will be increased one in each cycle. We want to end the cyclic process after $f(b-h)$ has been added, when we shall have obtained the required sum

$$\sum = f(a + h) + f(a + 2h) + \ldots + f(b - h)$$

This suggests that we should do the increasing after the addition and use ad RC $f(b)$, which is in A 3.2, for the comparison which will end the cycles. Consequently we use ad RC $f(a + mh)$ in preference to ca RC $f(a + mh)$ as the order whose digits are to be increased. At the beginning of the first cycle this order must be ad RC $f(a + h)$, and the orders A 1.6 and A 1.7 have been introduced for this purpose. The successive partial sums $\sum_m$ and the final sum $\sum$ are put in B 8, whose initial content must be zero. In the following code the descriptions refer to what happens during cycle m, the cycle during which $f(a + mh)$ is added to the partial sum $\sum_{m-1}$ to give $\sum_m$. (In the first cycle $\sum_{-1}$ is the initial content of B 8, which is zero.

12. Stage Four

A 4.1	ca	B 8	AC: $\sum_{m-1}$
2	ad	RC $f(a + mh)$	Initially ad -- First digits from A 1.7, giving ad RC $f(a + h)$ Digits changed by A 4.6 AC: $\sum_m$
3	ts	B 8	B 8: $\sum_m$
4	ca	A 4.2	AC: ad RC $[f(a + mh)]$
5	ad	RC 2^{-15}	AC: ad RC $f[a + (m + 1)h]$
6	td	A 4.2	A 4.2: ad RC $f[a + (m + 1)h]$
7	ca	A 3.2	AC: ad RC $f(b)$
8	su	A 4.2	Content of A 4.2 was changed by A 4.6, and is now ad RC $f[a + (m + 1)h]$. AC: $2^{-15}[b-a-(m+1)h]$ Content of AC > 0 until last cycle when $a + (m + 1)h = b$ and -zero appears in AC.
9	cp	A 5.1	Another cycle until $\sum$ is obtained.
10	sp	A 4.1	

13. Stage Five.

A 5.1	ca B 7	AC: $\frac{1}{2}f(a) + \frac{1}{2}f(b)$
2	ad RC	AC: $\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum$
3	mr RC h	AC: $\int_a^b f(x)dx$
4	ts B 9	B 9: $\int_a^b f(x)dx$

14. Stage Six. For this final stage we look through the ts and the td orders and find that the content of registers A 3.1, A 3.2, A 4.2, B 7, B 8, B 9 have been altered during the program. Of these the only one that needs to be reset is B 8, which must be reset to zero. For this

A 6.1	ca RC zero	AC: zero
2	ts B 8	B 8: zero

15. Allocation of B-class Registers.

B 1	2^{-15} RC $f(x_1)$
2	a
3	$\frac{2^{-15}}{h}$
4	$\frac{2^{-15}x_1}{h}$
5	2^{-15}
6	b
7	$\frac{1}{2}f(a) + \frac{1}{2}f(b)$
8	Initially zero Then $\sum_1, \sum_2, \ldots$ to final $\sum$
9	$\int_a^b f(x)dx$
10	h
11	zero.
B12 to B 11+n.	$f(x_1)$ to $f(x_n)$

16. We first look at the B-class registers for chances of combinations and see that we can put B 7 = B 9, saving a register. (Alternatives of this type will involve renumbering the data registers before the final write up.)

17. Secondly, it appears that after obtaining $\frac{1}{2}$ f(a) + $\frac{1}{2}$ f(b) in AC by order A 3.3 we could use A 3.4 to transfer to B 8 instead of to B 7. This would mean that during the cyclic stage the content of B 8 would be $\sum_m + \frac{1}{2}\left[f(a) + f(b)\right]$ instead of $\sum_m$. The initial content of B 8 need no longer be zero, so stage six can be dropped, and one order can be saved in stage five. B 7 is no longer needed, but we were already proposing to combine this register with B 9. However, there is no longer any objection to combining B 8 with B 9. Formerly the restoration of B 8 to zero in A 6.2 prevented this combination.

18. It is worth pointing out that in general it will be better to do any restoration work at the beginning of a subprogram rather than at the end, because this allows the B-class registers which have to be restored in a particular subprogram to be used for other purposes in other parts of the whole program.

19. Another saving can be made by improving the technique for finding f(a), f(a+h), etc., in the tabulation of f(x). It will nearly always be the case that when a function f(x) is tabulated for equidistant values of x over a range including x = 0, one of the tabulated values of f(x) will be f(0). If this is the case, and if f(0) is stored in the register whose number is k, then f(a) will be in the register whose number is $k + \frac{a}{h}$, whether a is positive or negative. Even if the range of tabulation does not include zero (as would happen if both x_1 and x_n were > 0) we can assign a number k which would be the number of the register containing f(0) if the range of tabulation were extended to include zero, although in this case the register whose number is k would not actually be used to contain a tabular entry. With this assumption we can store

2^{-15} RC f(0) in B 1 and obtain

RC f(a) by the equation

$$\text{RC } f(a) - \text{RC } f(0) = \frac{a}{h}$$

instead of

$$\text{RC } f(a) - \text{RC } f(x_1) = \frac{a-x_1}{h}.$$

This saves orders A 1.3 and A 2.3 as well as B 4, as it is no longer necessary to store

$$\frac{2^{-15}x_1}{h}$$

20. Return for a moment to consider the round-off referred to in paragraph 8. As has just been said the range of tabulation will nearly always be such that for some integral value of λ

$$x_1 + \lambda h = 0$$

This means that $\frac{x_1}{h}$ is an integer so that the value of $\frac{2^{-15}x_1}{h}$ that is stored in B 4 is the exact value of this quantity. If a is not a tabular value the number $\frac{a}{h}$ is not an integer and the order A 1.2 produces a round-off to

$$2^{-15} \text{ (nearest integer to } \tfrac{a}{h})$$

In this case therefore the orders A 1.3, A 1.4 do lead to the register containing the value of f(x) for the tabular value nearest to a. However, if $\frac{x_1}{h}$ is not an integer, the combination of the round-off in $\frac{2^{-15}x_1}{h}$ and $\frac{2^{-15}a}{h}$ may lead to the wrong value of $\frac{2^{-15}(a-x_1)}{h}$. For example, if $\frac{x_1}{h} = 11.6$, $\frac{a}{h} = 14.4$, $\frac{a-x_1}{h} = 2.8$, the correct round-off for $\frac{2^{-15}(a-x_1)}{h}$ would be $(3)2^{-15}$, but the computer would obtain $(2)2^{-15}$. This error is caused by scaling down $\frac{a}{h}$ and $\frac{x_1}{h}$ before the subtraction. The error could be minimized as follows. Suppose p is the integer such that $2^{-(p-1)} \leq h < 2^{-(p-2)}$. We could then make the computer calculate

$$\frac{2^{-p}}{h} a - \frac{2^{-p}}{h} x_1$$

without danger of overflow, and could then shift right 15-p places to obtain $\frac{2^{-15}}{h}(a-x_1)$. This would involve storing $\frac{2^{-p}}{h}$ and $\frac{2^{-p}x_1}{h}$ instead of $\frac{2^{-15}}{h}$ and $\frac{2^{-15}x_1}{h}$ and would add one order to the program. However, as has been said above, $\frac{x_1}{h}$ will nearly always be an integer so that this complication will be unnecessary.

21. It would be possible to save the order A 1.6 by altering stage four so that the order ad RC f(a+mh) has its digits increased before it is used, going through the cycle once more to add f(b), and correcting this by calculating $\frac{1}{2}$ f(a) - $\frac{1}{2}$ f(b) in stage three. This, however, whole saving one order, would involve the additional operations of the extra passage through the cycle, which in almost all cases would more than counterbalance the saving.

22. A very desirable place to effect a saving of orders is within a frequently repeated cycle. The saving of a single order in the construction of a cycle actually results in a multiple saving of operations due to the repeated use of the cycle. Although the saving of one order in a cycle which is traversed n times appears first as a saving of one storage register, it manifests itself as a saving of n operations with the corresponding economy in time. Examining the end of stage four we notice that the sp order is the one which is used to produce a repetition of the cycle whereas the cp(-) order comes into use only when the cycle is to be discontinued. We can eliminate the sp order by changing A 4.7 to read cs A 3.2, A 4.8 to read ad A 4.2, and A 4.9 to read cp(-) A 4.1. The effect of these changes would be to produce the necessary negative quantity in the A6 to cause recycling by the cp(-) order. Actually, however, the negative quantity produced is -b + a + (m + 1)h which will produce recycling until f(b) itself has been added. Before further consideration of this difficulty (see paragraph 21) let us notice that order A 4.8 can also be deleted if we now change A 4.7 to read su A3.2.

23. In paragraph 22 we have described the changes necessary to produce a saving of two orders in the cycle, yet this saving also produces the superfluous calculation of f(b). We can eliminate the difficulty by changing A 4.7 to read su RC f(b-h). The determination of RC f(b-h) can be made by the addition of the following two orders to stage two:

A 2.6	su	RC 2^{-15}
A 2.7	td	td B8

Note that B 8 was available for use (see paragraph 17). Thus although two orders have been added to stage two, we have eliminated two orders in stage four and thus have made a saving in any repeated use of the cycle.

24. Making use of all these improvements the code is written out again in standard form. The B-class registers have been renumbered. The use of the RC symbol in the program orders has been dropped, but the explanatory notes have been retained. The code is written in serial notation on page 57.

25. Originally we used 11+n B-class registers and 31 A-class registers, of which 9 belong to a cycle which is performed $\frac{b-a}{h}-1$ times. This gives a total of 42+n storage registers and $13+\frac{9(b-a)}{h}$ operations. The final form of the code uses 35+n storage registers and $11+\frac{8(b-a)}{h}$ operations, a saving of 7 registers and $2+\frac{b-a}{h}$ operations.

26. STANDARD NOTATION

A 1.1 ca B5	AC: a
2 mr B2	AC: $\frac{2^{-15}a}{h}$
3 ad B1	AC: 2^{-15} RC f(a)
4 td A3.2	A3.2: ad RC f(a)
5 ad B4	AC: ad RC f(a+h)
6 td A4.2	A4.2: ad RC f(a+h)
A 2.1 ca B6	AC: b
2 mr B2	AC: $\frac{2^{-15}b}{h}$
3 ad B1	AC: 2^{-15} RC f(b)
4 td A3.1	A3.1: ca RC f(b)
5 su B4	AC: ca RC f(b-h)
6 td B8	B8: 2^{-15} RC f(b-h)
A 3.1 ca--(digits from A2.4)	AC: f(b)
2 ad--(digits from A1.4)	AC: f(a) + f(b)
3 sr 1	AC: $\frac{1}{2}$ f(a) + f(b)
4 ts B7	B7: " "
A 4.1 ca B7	start of cycle. (cp(-) in A4.8) AC: $\frac{1}{2}f(a)+\frac{1}{2}f(b)+\sum_{m-1}$
2 ad--	digits from A1.6 and A4.6 AC: $\frac{1}{2}f(a)+\frac{1}{2}f(b)+\sum$
3 ts B7	B7: " " "

A 4.4	ca A4.2	AC: ad RC f(a+mh)
5	ad B4	AC: ad RC f(a+(m+1)h)
6	td A4.2	A4.2: " " "
7	su B8	AC: 2^{-15}RC$\{a+(m+1)h - (b-h)\}$
8	cp(-) A4.1	
A 5.1	ca B7	AC: $\frac{1}{2} f(a) + \frac{1}{2} f(b) + \sum$
2	mr B3	AC: $\int_a^b f(x)dx$
3	ts B7	B7: $\int_a^b f(x)dx$

B 1	2^{-15} RC f(0)
2	$\frac{2^{-15}}{h}$
3	h
4	2^{-15}
5	a
6	b
7	A 3.4 gives 1/2 f(a) + 1/2 f(b) A 4.3 gives 1/2 f(a) + 1/2 f(b) + $\sum_m$ A 5.3 gives $\int_a^b f(x)\, dx$
8	ad-- A2.7 gives ad RC f(b-h)
9	$f(x_1)$
10	$f(x_2)$
....	
8 + n	$f(x_n)$

27. <u>FLOW DIAGRAM</u> (STANDARD NOTATION)

A 1 Find RC f(a)

A 2 Find RC f(b)

A 3 Obtain $\frac{1}{2}$ f(a) + $\frac{1}{2}$ f(b)

A 4 Obtain $\frac{1}{2}$ f(a) + $\frac{1}{2}$ f(b) + $\sum_m$ for m = 1, 2, 3, ...
where $\int_m$ = f(a+h) + f(a+2h) + ... + f(b-h).
Have we obtained the final value $\sum$ of $\sum_m$ given by a+h = b-mh?

no

yes

A 5 Multiply by h to give $\int_a^b f(x)\,dx$

28. SERIAL NOTATION

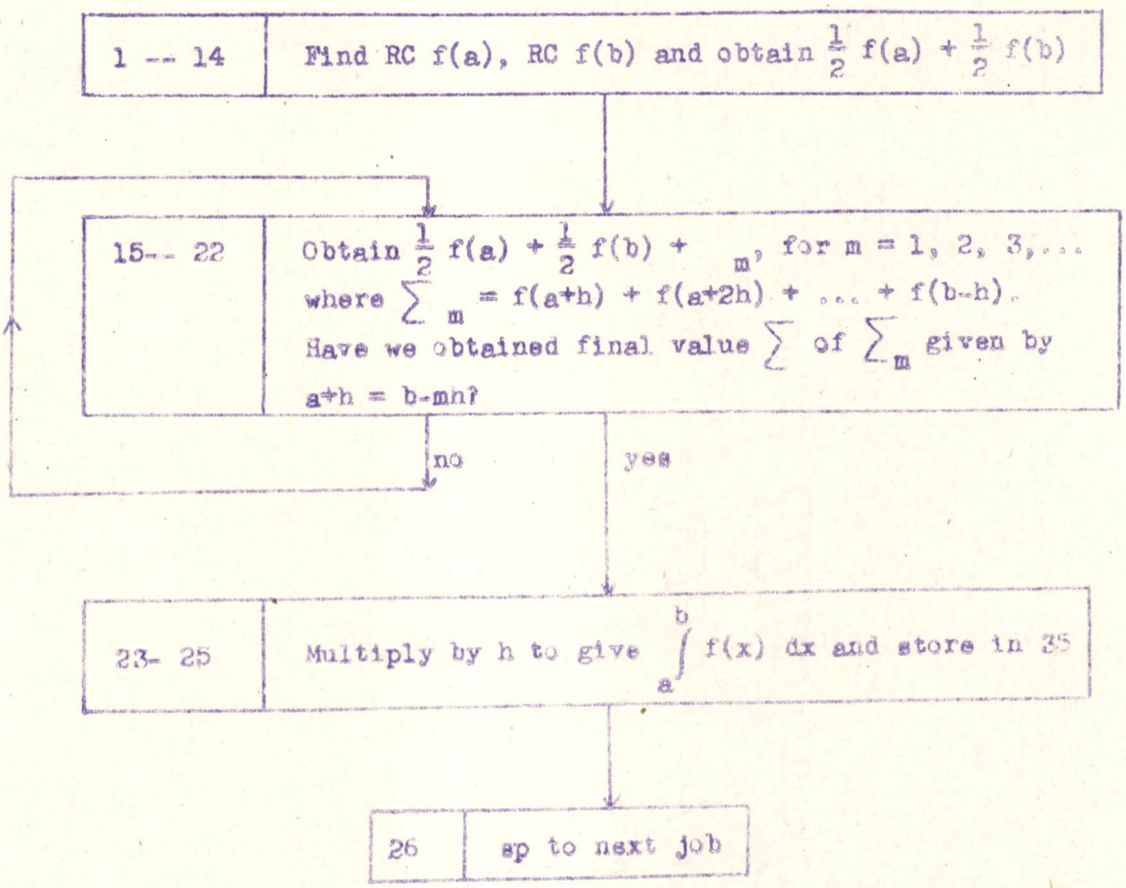
A

1	ca	33	12	td	18	21	su	32
2	mr	30	13	ca	--	22	td	18
3	ad	29	14	ad	--	23	su	14
4	td	14	15	sr	1	24	cp	17
5	ad	32	16	ts	35	25	ca	35
6	td	18	17	ca	35	26	mr	31
7	ca	34	18	ad	--	27	ts	35
8	mr	30	19	ts	35	28	(sp)	
9	ad	29	20	ca	18			
10	td	13						
11	su	32						

B

29	2^{-15} RC $f(0)$	33	a	37	$f(x_1)$
30	$\frac{2^{-15}}{h}$	34	b		. . .
31	h	35	--	36+n	$f(x_n)$
32	2^{-15}	36	--		

29. <u>FLOW DIAGRAM</u> (SERIAL NOTATION)



Number of operations 11 + $\frac{8(b-a)}{h}$

Input a, b in registers 33 and 34.

Output $\int_a^b f(x)\,dx$ in register 35.

Section VII. Iterative Processes

A. Iteration in the Computer

1. The mathematical process of iteration is available to the computer by the use of cyclic programs. In the general application of cyclic programs the decision to end the repetition of the cycle is made as the result of the comparison of a fixed quantity and a quantity derived during the course of the program; in iterative schemes the end of the cycle may either be determined in such a fashion or, when necessary, by a comparison of two quantities derived by the program. The former case is illustrated in Code XIII, the latter in Codes XIV and XV.

B. Code XIII: Summation of a Series

1. Consider the series

$$a_o + a_1 x + a_2 x^2 + \ldots + a_m x^m + \ldots$$

and assume that

$$\left|a_o\right| < 1, \quad \left|x\right| < 1, \quad \left|\frac{a_m}{a_{m-1}}\right| < 1 \text{ for } m = 1, 2, 3, \ldots$$

Let $\frac{a_m}{a_{m-1}} = r_m$, $a_m x^m = c_m$ and $\sum_m = c_o + c_1 + c_2 + \ldots + c_m$

Suppose that p is the first integer for which $\left|a_p\right| \leq 2^{-14}$. The object of the program is to obtain the sums $\sum_1, \sum_2, \sum_3, \ldots \sum_n$ successively, stopping the summation process as soon as the magnitude of the last term added, $\left|a_n x^n\right| = \left|c_n\right|$, is less than or equal to 2^{-14}. The sum $\sum_n$ so obtained is the required approximation to the sum of the series.

2. Since $\left|a_p x^p\right| < \left|a_p\right| \leq 2^{-14}$, the summation process is bound to stop when $n \leq p$. We therefore need only store the ratios $r_1, r_2, r_3 \ldots r_p$.

3. The central section of the program is A4 in which c_m is obtained from c_{m-1} by the equation

$$c_m = r_m c_{m-1} x$$

and adds c_m to $\sum_{m-1}$ to form $\sum_m$. Both c_m and $\sum_m$ are stored for further use. Section A3 arranges that another term of the series will be added and section A5 tests whether the magnitude of the last term added is less than or equal to 2^{-14}. These three sections A3, A4, and A5 form a cycle. The first two sections A1 and A2 make preparations for the first round of the cycle.

C. Code XIV: Linear Simultaneous Equations

1. An iteration scheme can be applied to the solution of linear simultaneous equations. In Code XIV this scheme is illustrated in the solution of two linear equations in two unknowns:

$$x_1 + a_1 x_2 = b_1$$

$$a_2 x_1 + x_2 = b_2$$

2. If we denote the order of the approximation by superscripts,

where $x_1^{(m+1)}$ is the $(m+1)$st approximation to x_1

and $x_1^{(m)}$ is the m^{th} approximation to x_1,

then the iteration formulae are given by

$$x_1^{(m+1)} = b_1 - a_1 x_2^{(m)}$$

$$x_2^{(m+1)} = b_2 - a_2 x_1^{(m+1)}$$

3. To avoid overflow we shall assume that

$$0 \leqq a_1 \leqq b_1 < 1$$

and $0 \leqq a_2 \leqq b_2 < 1$

from which it follows by induction that starting with $x_1^{(1)} = x_2^{(1)} = 0$,

we shall have

$$0 \leqq x_1^{(m)} < 1$$

$$0 \leqq x_2^{(m)} < 1$$

for all m.

4. The code which is presented is arranged to end the approximating process when both $x_1^{(m+1)} - x_1^{(m)}$ and $x_2^{(m+1)} - x_2^{(m)}$ are equal to or less than 2^{-10}.

5. The coefficients a_1, a_2, b_1, b_2 should be regarded as input data, which will be different for each application of the program but must always satisfy the inequalities given above. Unfortunately this problem is somewhat unrealistic, because simultaneous linear equations in two variables would not be dealt with by this method, while sets of linear equations in many variables involve scale factor problems which do not arise in this example.

D. Roots of Equations by Newton's Method

1. An iteration scheme is also applicable to solutions of roots of equations using Newton's Method. The formula in Newton's Method for finding a root of an equation $f(x) = 0$ is

$$x_{n+1} = x_n - \frac{f(x_n)}{f^1(x_n)}$$

where x_n is the n^{th} approximation, and f' indicates $\frac{df(x_n)}{dx}$

2. For example to find the square root of a number a we set $f(x) = x^2 - a$ and the formula for successive approximations to the positive root of the equation $x^2 - a = 0$ is

$$x_{n+1} = x_n + 1/2 \ \left(\frac{a}{x_n} - x_n \right)$$

3. Due to possible overflow, the above equation is not used in an actual square rooting program. The use of Newton's method in obtaining the square root of a number is dealt with in more detail in Section VIII.

Section VIII. Linear Interpolation and Finding the Square Root

A. Code XV: Linear Interpolation

1. Let us assume that values of $f(x)$, for $x = k \cdot 2^{-6}$, where $k = 0, 1, 2, 3, ---- 2^6$, are stored in 65 consecutive registers. The addresses of these 65 registers are given by the expression $\underline{K + k}$, in which K is the address of the register containing $f(0)$. The interval between the tabulated values of x, which shall be designated as $\underline{h}$, is 2^{-6}. It is assumed that $|f(hk)| < 1$ for all the k and further that if $k_2 = k_1 + 1$, then $|f(hk_2) - f(hk_1)| < 1$.

2. The problem is to obtain $f(x)$ for any value of x in the range $0 \leq x < 1$. The value of x is originally stored in B1 and $f(x)$ is to be put in B2.

3. It is first necessary to find two consecutive values k_1 and k_2 such that

$$hk_1 \leq x < hk_2, \quad k_2 = k_1 + 1$$

Then if $x - hk_1 = mh$ we shall have $0 \leq m < 1$ and the required value of $f(x)$ is given by

$$f(x) = f(hk_1) + m\left\{f(hk_2) - f(hk_1)\right\}$$

4. If the positive number x were brought into the AC by the order ca B1, the sign digit would be 0, the next six digits would determine k_1, and the remaining nine digits would determine m. To be more precise, the number obtained from x by replacing all the right hand nine digits by zeros would be hk_1, while the number obtained from x by replacing the first six digits after the sign digit by zeros would be mh.

5. For example, the positive number .001101101100000 is represented in the computer by 0/001101101100000. Since x lies between .001101 and .001110 we have

$$hk_1 = .001101$$

and $mh = x - hk_1 = .000000101100000$. These two numbers are represented in the AC by

0/001101000000000

and 0/000000101100000

It follows therefore that the order ca RCx followed by sl 6 would put the interpolation ratio m in the AC.

6. In Code XV m is first shifted into the BR by the order mh RC 2^{-9} and is brought into the AC at a later stage by the order sl 15. It should be noticed that the order mh RC 2^{-9}, which has the effect of shifting the content of AC nine spaces to the right, cannot be replaced by the order sr 9, since this latter order would give a round-off and clear the BR so that the interpolation ratio m would be lost.

B. Codes XVI and XVII: Finding the Square Root

1. The arithmetic operations of addition, subtraction, division, and multiplication are wired into the computer and are executed by single orders; however, other mathematical processes which may be used frequently in a particular computer application are available only in the form of codes. It appears likely that the determination of square roots is a process which will be needed quite often, and it is rather probable that some of the storage registers of a computer would permanently contain the orders for a square root code. The economy of orders and saving in operating time assume particular importance under such conditions. Although the square root can be determined using Newton's Formula as given in D of Section VII, a slight variation of the formula gives a shorter code, this being Code XVI. The square root is determined using a series expansion in Code XVIII.

2. The object of Codes XVI and XVIII is to find $\frac{\sqrt{a}}{2}$ when $0 < a < 1$. We find $\frac{\sqrt{a}}{2}$ rather than $\sqrt{a}$ in order to avoid the danger of overflow when a is nearly equal to 1. Both of the codes employ the scale factor operation to find the number of places, n, which a must be shifted to the left in the AC so that the first non-zero digit of a is put in AC1. This amounts to expressing a in the form

$$a = 2^{-n} \cdot b \quad \text{where } 1/2 \leq b < 1$$

which gives

$$\frac{\sqrt{a}}{2} = 2^{-n/2} \frac{\sqrt{b}}{2}$$

3. The scale factoring process shortens the program by reducing the number of approximations that are required to obtain the desired accuracy, but it also introduces the difficulty of dealing with the two cases which arise when n is odd or even. The difficulty is handled as follows:

$$\text{Let } \frac{n}{2} = m + k$$

where m is an integer and $k = 0$ or $1/2$. Then

$$\frac{\sqrt{a}}{2} = 2^{-m} \times 2^{-k} \times \frac{\sqrt{b}}{2}.$$

where the factor 2^{-k} is 1 or $\frac{1}{\sqrt{2}}$. The codes will actually calculate $\frac{\sqrt{b}}{2q}$, where $q = \frac{1}{\sqrt{2}}$ in code XVI and $q = 2^{3/4}$ in code XVII. The value $\frac{\sqrt{a}}{2}$ will therefore be calculated from the product

$$\frac{\sqrt{a}}{2} = 2^{-m} \times 2^{-k}q \times \frac{\sqrt{b}}{2q}$$

In order to obtain the required factor $2^{-k}q$, which takes the values q and $\frac{q}{\sqrt{2}}$ for $k = 0$ and $k = 1/2$, we evaluate the expression

$$q + (1 - \sqrt{2})\, kq\sqrt{2},$$

which is equal to q when $k = 0$ and equal to $\frac{q}{\sqrt{2}}$ when $k = 1/2$.

4. Code XVI uses the Newton method of successive approximations to the positive root of the equation

$$f(x) = x^2 - \frac{b}{8} = 0$$

starting with $x_0 = \sqrt{\frac{1}{8}} = \frac{\sqrt{2}}{4}$ as the first approximation. The formula for successive approximations is

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = \frac{1}{2}\left(x_i + \frac{b}{8x_i}\right)$$

The error in x_3 is less than 2^{-15} in the worst case when $b = \frac{1}{2}$, so the code uses three iterations, given by the formulae

$$x_1 = \frac{1}{2}(x_0 + \frac{b}{8x_0}) = \frac{\sqrt{2}}{8} + \frac{\sqrt{2}}{2}(\frac{b}{4})$$

$$2x_2 = x_1 + \frac{b}{8x_1}$$

$$2x_3 = x_2 + \frac{b}{8x_2}$$

The value of $2x_3$ obtained from the third step is the required approximation to $\frac{\sqrt{b}}{2q}$, when $q = \frac{1}{\sqrt{2}}$.

5. In order to obtain the expression

$$q + (1 - \sqrt{2})\, kq \sqrt{2}$$

for the final step we store $q = \frac{1}{\sqrt{2}}$ and $(1 - \sqrt{2})q\sqrt{2} = 1 - \sqrt{2}$.

Before the program starts the value of a is put in B1. At the end of the program the value of $\frac{\sqrt{a}}{2}$ is in B5.

It might appear more natural to use the approximation formulae for the positive root of the equation $x^2 - \frac{b}{4} = 0$. The equation $x^2 - \frac{b}{8} = 0$ is used to avoid a danger of overflow during the arithmetic operations involved in the successive approximations.

6. Code XVII uses a series expansion in powers of y, where

$$y = \frac{b - \frac{\sqrt{2}}{2}}{b + \frac{\sqrt{2}}{2}} \qquad \text{or} \qquad \frac{1+y}{1-y} = b\sqrt{2}$$

The object of introducing y is to obtain a series involving only even powers. We have

$$\sqrt{b\sqrt{2}} = \frac{1+y}{1-y} = (1+y)(1-y^2)^{-\frac{1}{2}}$$

$$2^{\frac{1}{4}}\sqrt{b} = (1+y)\left(1 + \frac{y^2}{2} + \frac{3y^4}{8} + \frac{5y^6}{6} + \ldots\right)$$

For $\frac{1}{2} \leq b < 1$ we have $-0.1715 \leq y < 0.1715$ so for the extreme values of y the value of $\frac{5y^6}{6}$ is about 2×10^{-5}. We therefore take

$$2^{-\frac{1}{4}}\sqrt{b} = (1+y)\left(\frac{\sqrt{2}}{2} + \frac{y^2\sqrt{2}}{4} + \frac{3y^4\sqrt{2}}{16}\right)$$

$$= (1+y)\, f(y^2)$$

7. Code XVII determines $\frac{\sqrt{b}}{2q}$, where $q = 2^{-\frac{3}{4}}$. In order to obtain the expression

$$q + (1-\sqrt{2})kq\sqrt{2}$$

for the final step we store $q = 2^{-\frac{3}{4}}$ and $(1-\sqrt{2})q\sqrt{2} = 2^{-\frac{1}{4}}(1-\sqrt{2})$. Before the program starts the value of a is put in B1. At the end of the program $\frac{\sqrt{a}}{2}$ is in B4.

8. Code XVI uses 38 storage registers and 29 operations, not counting the final sp order. Thus at 20 microseconds per operation the evaluation of the square root would take 580 microseconds. Code XVII uses 36 storage registers and 28 operations, requiring 560 microseconds.

Section IX. Codes for Sorting

A. Code XVIII: Rearrangement of a Set of Numbers in Ascending Order

1. As in Code XII the numbers x_1, x_2, ... x_n are stored in consecutive registers C1, C2, ... Cn. The problem is to rearrange the x's in ascending order. In the final arrangement equal numbers appear together, but not necessarily in their original order.

2. The program takes each register C_k in turn, starting with k = 1, finds the least number in the registers C_k to C_n, and interchanges this least number with the number in C_k. To simplify the explanatory notes it is assumed that after each interchange the numbers are renamed, so that at every stage of the program the number in Cm is called x_m.

3. The greater part of Code XII is used in section A2 of this code with the modifications needed to find the least number rather than the greatest one. In the explanatory notes the number m refers to the cycle that is performed inside section A2. This cycle is not shown on the flow diagram.

B. Code XIX: Sorting Sets of Numbers

1. It is supposed that n sets of numbers (x_1, y_1, z_1...), (x_2, y_2, z_2...), ... (x_n, y_n, z_n...) are stored in the machine and that a program is required that will enable the computer to deal with these sets in the order of ascending x_i, rearranging the associated y_i, z_i, etc., in accordance with the final order of the x_i.

2. The method of Code XVIII could be extended to provide for the rearrangement of the numbers y_i, z_i, etc., in accordance with the rearranged x_i, but the process of interchanging the positions of the sets of numbers in storage would be lengthy. It seems preferable to avoid the actual interchange of the numbers by dealing with the addresses at which the numbers can be found

3. As in Codes XII or XVIII we shall assume that the numbers x_1, x_2, ... x_n are stored in registers C1, C2, ... Cn, however in this code there will be no need to assume that these registers are consecutive. We assume that y_i is stored in the register whose address is u + C_i,

z_i is stored in the register $v + C_i$, etc., where u, v, ... have the same value for all i. The computer, having found the address of the register containing any x_i, can easily find the registers containing the associated y_i, z_i, ... by adding the numbers u, v, ...

4. The code uses a set of consecutive registers C(c + 1), C(c + 2), ... C(c + n) to contain the addresses C1, C2, ... Cn. The address contained in register C(c + t) will be denoted by C_{P_t}, and the number x_i stored at this address will be denoted by $x^{(t)}$. To be more exact, the quantity in C(c + t) is $2^{-15}xC_{P_t}$.

5. When the program starts the registers C(c + 1), C(c + 2), ... C(c + n) contain the addresses C1, C2, ... Cn in some order. The effect of the program is to rearrange the content of these registers so that the numbers x_i obtained successively from the addresses C(c + 1), C(c + 2), ... C(c + n) will be in ascending order. This is achieved by dealing in turn with the registers C(c + t) for t = 2, 3 .... n. For each t the content of the registers C(c + t - m), where m = 1, 2, 3, ... are examined in turn and moved to C(c + t - m + 1) until a register is reached which contains an address giving an x_i that is less than or equal to the x_i that was given by the address that was in C(c + t). The address that was then in C(c + t) is then put into the last of the registers previously examined. To stop the process a quantity q, known to be less than all the x_i, is put in C(0), and $2^{-15}xC(0)$ is put in C(c + 0).

6. The content of the register C(c + t) may be changed several times during the course of a program. In the explanatory notes C_{P_t} is taken to mean the address contained in C(c + t) at the stage at which the orders take effect.

Air Traffic Control Project
Servomechanisms Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts

SUBJECT: INTRODUCTION TO CODING, PART II

To: 6673 Project

From: David R. Israel

Date: September 29, 1949

Code I

Orders	Effects of Orders
ca 14	AC: x
2 mr 14	AC: x^2
3 mr 11	AC: ax^2
4 ts 15	register 15: ax^2
5 ca 14	AC: x
6 mr 12	AC: bx
7 ad 13	AC: $bx + c$
8 ad 15	AC: $ax^2 + bx + c$
9 ts 15	register 15: $ax^2 + bx + c$
10 end of code	
11 a	
12 b	
13 c	
14 x	
15 --	

Code II

Orders	Effects of Orders
1 ca 8	AC: a
2 mr 11	AC: ax
3 ad 9	AC: $ax+b$
4 mr 11	AC: $ax^2 + bx$
5 ad 10	AC: $ax^2 + bx + c$
6 ts 12	register 12: ax^2+bx+c
7 end of code	
8 a	
9 b	
10 c	
11 x	
12 --	

Code III

Orders		Effects of Orders
1	ca 17	AC: d
2	mr 19	AC: dx
3	ad 18	AC: dx + e
4	ts 20	register 20: dx + e
5	ca 14	AC: a
6	mr 19	AC: ax
7	ad 15	AC: ax + b
8	mr 19	AC: $ax^2 + bx$
9	ad 16	AC: $ax^2 + bx + c$
10	dv 20	BR: $\frac{ax^2 + bx + c}{dx + e}$
11	sl 15	AC: $\frac{ax^2 + bx + c}{dx + e}$
12	ts 20	register 20: $\frac{ax^2 + bx + c}{dx + e}$
13	end of code	
14	a	
15	b	
16	c	
17	d	
18	e	
19	x	
20	--	

Code IV

Orders		Effects of Orders
1	ca 14	AC: x
2	mr 14	AC: x^2
3	ts 16	register 16: x^2
4	ca 15	AC: y
5	mr 15	AC: y^2
6	ad 16	AC: $x^2 + y^2$
7	ts 16	register 16: $x^2 + y^2$
8	ca 14	AC: x
9	mr 15	AC: xy
10	dv 16	BR: $\frac{xy}{x^2 + y^2}$
11	sl 15	AC: $\frac{xy}{x^2 + y^2}$
12	ts 16	register 16: $\frac{xy}{x^2 + y^2}$
13	end of code	
14	x	
15	y	
16	--	

Code V

Orders	Effects of Orders (b > a)	Effects of Orders (b ≤ a)
1 ca 11	AC: b	AC: b
2 ts 12	AC: b register 12: b	AC: b register 12: b
3 su 10	AC: b-a (positive)	AC: b-a (negative)
4 cp(-)9	go to order at 5	go to order at 9
5 ca 10	AC: a	
6 ts 11	register 11: a	
7 ca 12	AC: b	
8 ts 10	register 10: b	
9 sp next job	(b is in register 10, a is in register 11)	(a is in register 10, b is in register 11)
10 a		
11 b		
12 --		

Code VI

(Analysed only for c > b > a)

Orders	Effects of Orders
1 ca 27	AC: b
2 ts 29	AC: b register 29: b
3 su 26	AC: b-a (positive)
4 cp(-)9	go to order at 5
5 ca 26	AC: a
6 ts 27	register 27: a
7 ca 29	AC: b
8 ts 26	register 26: b
9 ca 28	AC: c
10 ts 29	AC: c register 29: c
11 su 26	AC: c-b(positive)
12 cp(-)17	go to order at 13
13 ca 26	AC: b
14 ts 28	register 28: b
15 ca 29	AC: c
16 ts 26	register 26: c
17 ca 28	AC: b
18 ts 29	AC: b register 29: b
19 su 27	AC: b-a (positive)
20 cp(-)25	go to order at 21
21 ca 27	AC: a
22 ts 28	register 28: a
23 ca 29	AC: b
24 ts 27	register 27: b
25 sp next job	
26 a	
27 b	
28 c	
29 --	

Code VII

Orders	Effects of Orders (1st cycle)	Effects of Orders (2nd cycle)	Effects of Orders (32nd cycle)
1 ca 12	AC: 0		
2 ad 13	AC: 2^{-15}	AC: 2×2^{-15}	AC: 32×2^{-15}
3 su 14	AC: -30×2^{-15}	AC: -29×2^{-15}	AC: 1×2^{-15}
4 cp(-)6	go to order at 6	go to order at 6	go to order at 5
5 sp 9			go to order at 9
6 ad 14	AC: 2^{-15}	AC: 2×2^{-15}	
7 ts 15	AC: 2^{-15} register 15: 2^{-15}	AC: 2×2^{-15} register 15: 2×2^{-15}	
8 sp 2	go to order at 2	go to order at 2	
9 ca 12			AC: 0
10 ts 15			AC: 0 register 15: 0
11 sp 2			go to order at 2 (begin again)
12 +0			
13 2^{-15}			
14 31×2^{-15}			
15 --			

Code VIII

Orders	Effects of Orders (1st cycle)	Effects of Orders (2nd cycle)	Effects of Orders (32nd cycle)
1 ca 7	AC: 0		
2 ad 8	AC: 2^{-15}	AC: 2×2^{-15}	AC: 32×2^{-15}
3 sl 10	AC: 2^{-5}	AC: 2×2^{-5}	AC: 0
4 sr 10	AC: 2^{-15}	AC: 2×2^{-15}	AC: 0
5 ts 9	AC: 2^{-15} register 9: 2^{-15}	AC: 2×2^{-15} register 9: 2×2^{-15}	AC: 0 register 9: 0
6 sp 2	go to order at 2	go to order at 2	go to order at 2 (begin again)
7 +0			
8 2^{-15}			
9 --			

Code IX

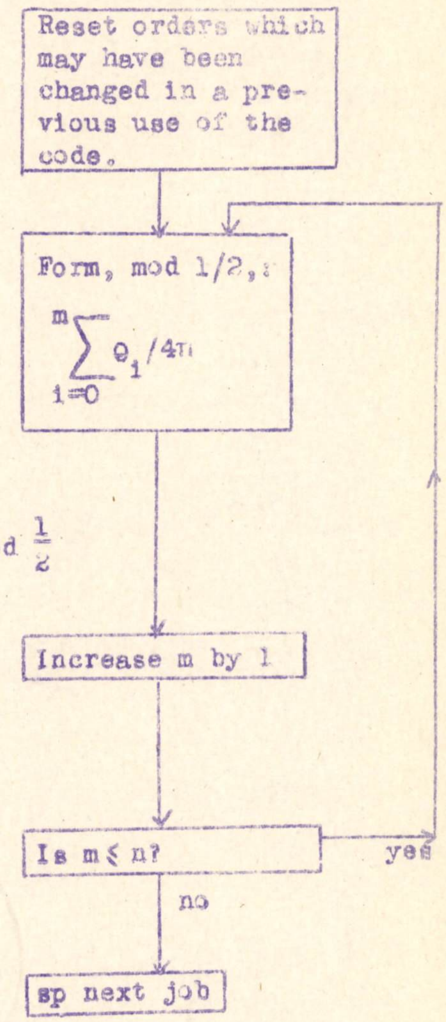
Orders	Effects of Orders	Storage
A 1 ca B n+3	AC: a_n	
2 mr B 1	AC: $a_n x$	B 1 x
3 ad B (n-1)+3	AC: $a_n x + a_{n-1}$	2 --
4 mr B 1	AC: $a_n x^2 + a_{n-1} x$	3 a_0
5 ad B (n-2) + 3	AC: $a_n x^2 + a_{n-1} x + a_{n-2}$	4 a_1
6 mr B 1	AC: $a_n x^3 + a_{n-1} x^2 + a_{n-2} x$	5 a_2
.	.	.
.	.	.
.	.	.
.	.	.
2n+1 ad B 3	AC: $a_n x^n + a_{n-1} x^{n-1} + \ldots\ldots + a_0$	n+3 a_n
2n+2 ts B 2	B2: $a_n x^n + a_{n-1} x^{n-1} + \ldots\ldots + a_0$	
2n+3 sp next job		

Code X

Orders	Effects of Orders (1st cycle)	Effects of Orders (2nd cycle)	Storage
A 1 ca B n+5	AC: a_n		B 1 2^{-15}
2 ts B 3	B3: a_n		2 x
3 ca B 3	AC: a_n	AC: $a_n x + a_{n-1}$	3 --
4 mr B 2	AC: $a_n x$	AC: $a_n x^2 + a_{n-1} x$	4 ad B4
5 ad B n+4	AC: $a_n x + a_{n-1}$	AC: $a_n x^2 + a_{n-1} x + a_{n-2}$	5 a_0
6 ts B 3	B3: $a_n x + a_{n-1}$	B3: $a_n x^2 + a_{n-1} x + a_{n-2}$	6 a_1
7 ca A 5	AC: ad B n+4	AC: ad B n+3	7 a_2
8 su B 1	AC: ad B n+3	AC: ad B n+2	.
9 ts A 5	AC: ad B n+3 A5: ad B n+3	AC: ad B n+2 A5: ad B n+2	.
			.
10 su B 4	AC: n+3-4	AC: n+2-4	.
11 cp(-) A 13			n+5 a_n
12 sp A 3	go to A3	go to A3	
13 ca A 1			
14 su B 1			
15 td A 5			
16 sp next job			

Orders		Effects of Orders
1	ca 19+n	AC: ad 16
2	td 6	register 6: ad 16
3	ca 16+n	AC: +0
4	ts 17+n	register 17+n: +0
5	ca 17+n	AC: $\sum_{i=0}^{m-1} \theta_i/4\pi \quad m=2,3\ldots$
6	ad --(see 2 and 12)	AC: $\sum_{i=0}^{m} \theta_i/4\pi$
7	sl 1	AC: $\sum_{i=0}^{m} \theta_i/4\pi$, mod $\frac{1}{2}$
8	sr 1	
9	ts 17+n	register 17+n: $\sum_{i=0}^{m} \theta_i/4\pi$, mod $\frac{1}{2}$
10	ca 6	AC: ad 15 + (m-1)
11	ad 18+n	AC: ad 15+m
12	td 6	AC: ad 15+m register 6: ad 15+m
13	su 20+n	AC: m-n
14	cp(-)5	If m≤n, go back to 5
15	sp next job	If m>n, have finished
16	$\theta_1/4\pi$	
17	$\theta_2/4\pi$	
.		
.		
.		
.		
15+n	$\theta_n/4\pi$	
16+n	+0	
17+n	--	
18+n	2^{-15}	
19+n	ad 16	
20+n	ad 15+n	

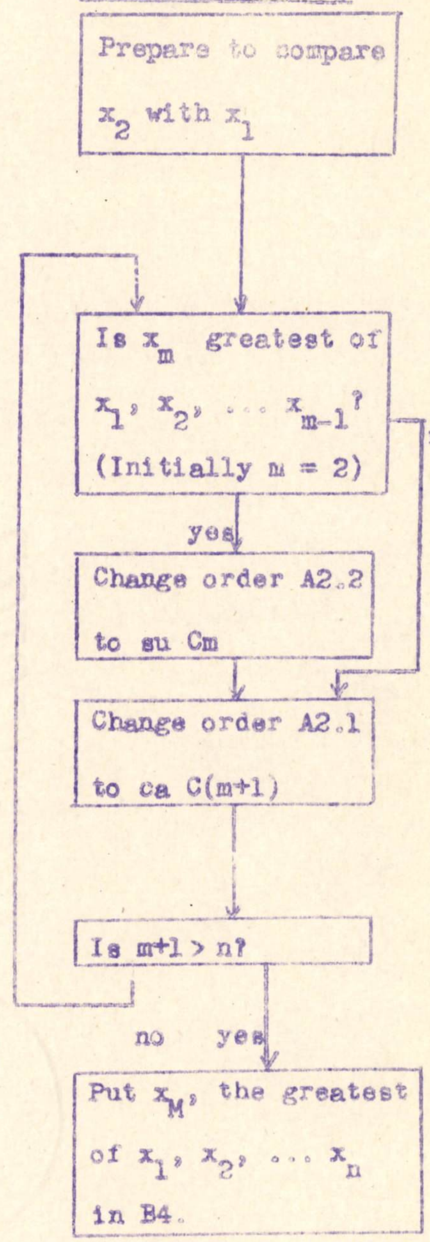
General Procedure



Code XII

Orders			Accumulator Contents
A1.1	ca	B2	
2	td	A2.2	
3	ad	B1	
4	td	A2.1	
A2.1	ca	--	x_m See A1.4, A4.3
2	su	--	x_m - (max of x_1 ... x_{m-1}) See A1.2, A3.2
3	cp(-)	A4.1	
A3.1	ca	A2.1	ca Cm
2	td	A2.2	
A4.1	ca	A2.1	ca Cm
2	ad	B1	ca C(m+1)
3	td	A2.1	
A5.1	su	B3	$(m+1-n) \times 2^{-15}$
2	cp(-)	A2.1	
A6.1	ca	A2.2	su C_M
2	td	A6.3	
3	ca	--	x_M See A6.2
4	ts	B4	

General Procedure

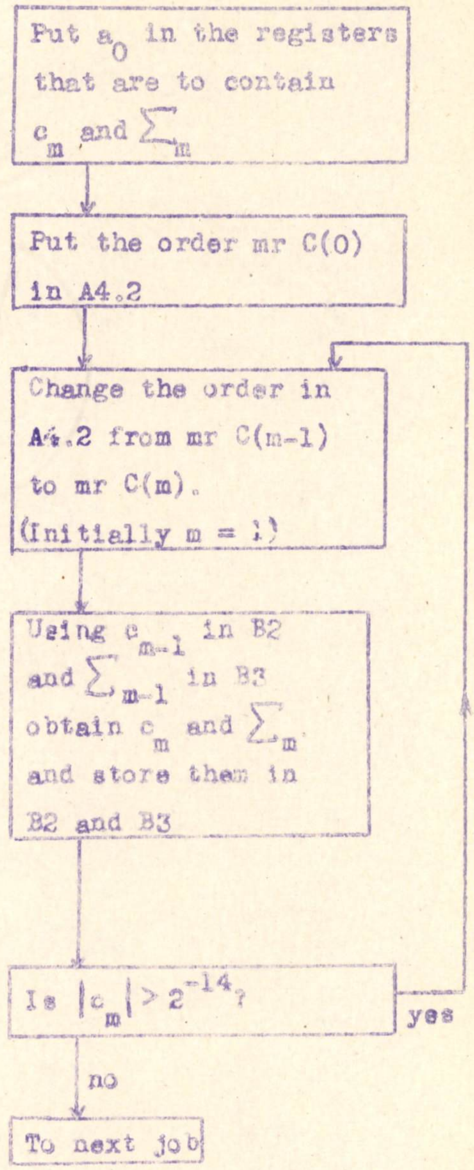


Data Storage for Code

B1	2^{-15}		C1	x_1
2	ca C1		C2	x_2
3	ca Cn		.	
4	-- Used for x_M		.	
			Cn	x_n

Code XIII

A1.1	ca	C(0)	$a_0 = c_0 = \sum_0$
2	ts	B2	
3	ts	B3	
A2.1	ca	A1.1	ca C(0)
2	td	A4.2	
A3.1	ca	A4.2	mr C(m-1) See A2.2, A3.3
2	ad	B5	
3	td	A4.2	
A4.1	ca	B2	$c_{m-1} = a_{m-1} x^{m-1}$ See A1.2, A4.4
2	mr	--	$a_m x^{m-1}$ See A2.2, A3.3
3	mr	B1	$c_m = a_m x^m$
4	ts	B2	
5	ad	B3	$\sum_m = \sum_{m-1} + c_m$ See A1.3, A4.6
6	ts	B3	
A5.1	cm	B2	c_m
2	su	B4	$c_m - 2^{-14}$
3	cp(-)next job		
4	sp	A3.1	



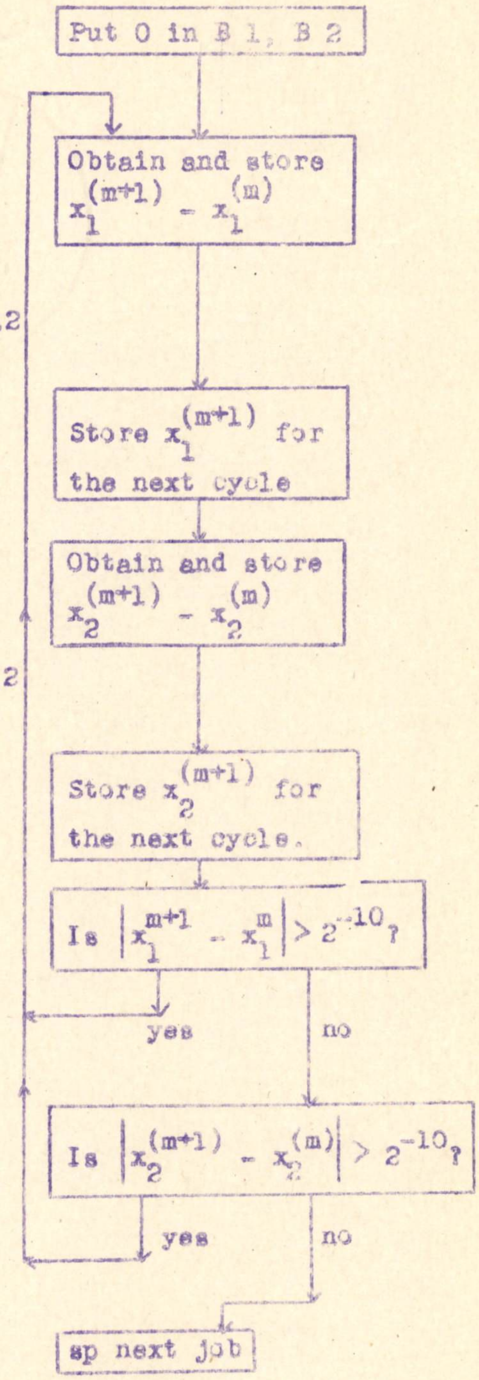
Data Storage for Code

B1	x
B2	-- used for c_m
B3	-- " " $\sum_m$
B4	2^{-14}
B5	2^{-15}

C(0)	a_0
C 1	r_1
C 2	r_2
.	.
.	.
.	.
C p	r_p

Code XIV

A1.1	ca B 10	0
2	ts B 1	
3	ts B 2	
A2.1	ca B 2	x_2^m See A1.3, A5.2
2	mr B 5	$-a_1 x_2^m$
3	ad B 7	$b_1 - a_1 x_2^{(m)} = x_1^{(m+1)}$
4	su B 1	$x_1^{(m+1)} - x_1^{(m)}$ See A1.2, A3.2
5	ts B 3	
A3.1	ad B 1	$x_1^{(m+1)}$
2	ts B 1	
A4.1	mr B 6	$-a_2 x_1^{(m+1)}$
2	ad B8	$b_2 - a_2 x_1^{(m+1)} = x_2^{(m+1)}$
3	su B 2	$x_2^{(m+1)} - x_2^{(m)}$ See A1.3, A5.2
4	ts B 4	
A5.1	ad B 2	$x_2^{(m+1)}$
2	ts B 2	
A6.1	cm B 3	$\lvert x_1^{(m+1)} - x_1^{(m)} \rvert$
2	su B 9	$\lvert x_1^{(m+1)} - x_1^{(m)} \rvert - 2^{-10}$
3	cp(-)A7.1	
4	sp A2.1	
A7.1	cm B 4	$\lvert x_2^{(m+1)} - x_2^{(m)} \rvert$
2	su B 9	$\lvert x_2^{(m+1)} - x_2^{(m)} \rvert - 2^{-10}$
3	cp(-) A8.1	
4	sp A2.1	
A8.1	sp next job	



B 1 -- Used for $x_1^{(m)}$, See A3.2, A1.2
B 2 -- Used for $x_2^{(m)}$ See A5.2, A1.3
B 3 -- Used for $x_1^{(m+1)} - x_1^{(m)}$, See A2.5
B 4 -- Used for $x_2^{(m+1)} - x_2^{(m)}$, See A4.4

B 5 $-a_1$
B 6 $-a_2$
B 7 b_1
B 8 b_2
B 9 2^{-10}
B10 0

Code XV

A1.1	ca	B1	x
2	mh	B5	$k_1 \times 2^{-15}$ (BR contains m)
3	ad	B3	RC $f(hk_1)$
4	td	A2.4	
5	td	A2.6	
6	ad	B4	RC $f(hk_2)$
7	td	A2.3	(BR still contains m)

Obtain RC $f(hk_1)$ and RC $f(hk_2)$ and transfer to section A2.
Retain interpolation ratio m in BR.

A2.1	sl	15	m
2	ts	B2	
3	ca	--	$f(hk_2)$ See A1.7
4	su	--	$f(hk_2) - f(hk_1)$ See A1.4
5	mr	B2	$m \quad f(hk_2) - f(hk_1)$
6	ad	--	$f(x)$ See A1.5
7	ts B2		

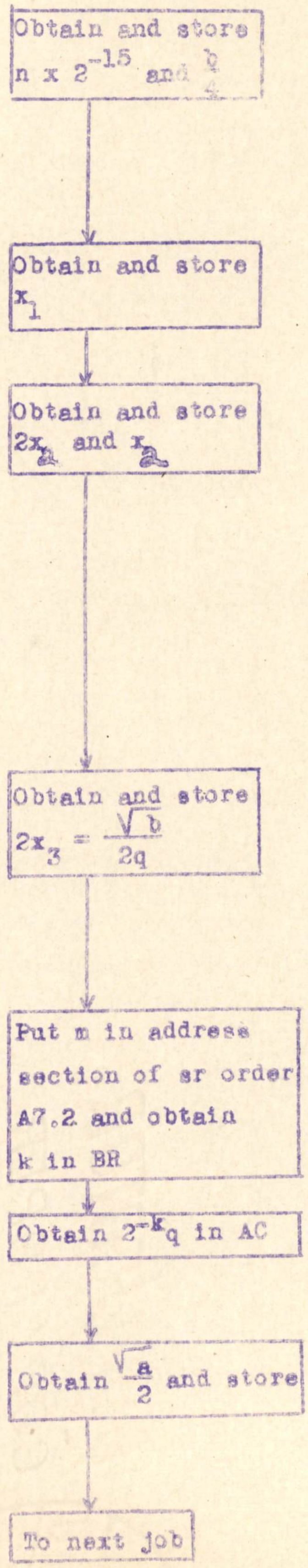
Put m in AC and apply linear interpolation formula to obtain $f(x)$.

<u>Data Storage for Code</u>

B1	x
2	-- Used for m
	Used for $f(x)$
3	$K \times 2^{-15}$
4	2^{-15}
5	2^{-9}

Code XVI

A1.1	ca	B1	a
2	sf	B2	AC contains b
			B2 " $n \times 2^{-15}$
3	sr	2	AC " $\frac{b}{4}$
4	ts	B3	
A2.1	mr	C2	$(\frac{b}{4})\frac{\sqrt{2}}{2}$
2	ad	C1	$x_1 = \frac{2\sqrt{2}}{8} + \frac{\sqrt{2}}{2}(\frac{b}{4})$
3	ts	B4	
A3.1	ca	B3	$\frac{b}{4}$ See A1.4
2	sr	1	$\frac{b}{8}$
3	dv	B4	
4	sl	15	$\frac{b}{8x_1}$ See A2.3
5	ad	B4	$2x_2$ " "
6	ts	B5	
7	sr	1	x_2
8	ts	B4	
A4.1	ca	B3	$\frac{b}{4}$ See A1.4
2	dv	B5	
3	sl	15	$\frac{b}{8x_2}$ See A3.6
4	ad	B4	$2x_3 = \frac{\sqrt{b}}{2q}$ See A3.8
5	ts	B5	
A5.1	ca	B2	$n \times 2^{-15}$ See A1.2
2	mh	C3	AC contains $m \times 2^{-15}$
			BR " k
3	td	A7.1	
A6.1	sl	15	k
2	mr	C4	$(1 - \sqrt{2})\, kq\sqrt{2}$
3	ad	C2	$2^{-k}q$
A7.1	sr	--	$2^{-m}2^{-k}q = 2^{-\frac{n}{2}}q$ See A5.3
2	mr	B5	$2^{-\frac{n}{2}}\frac{\sqrt{b}}{2} = \frac{\sqrt{a}}{2}$ See A4.5
3	ts	B5	
A8.1	sp	--	



Note: Data storage on following page.

Data Storage for Code XVI

B1. a

B2. -- Used for $n \times 2^{-15}$

B3. -- " " $\frac{b}{4}$

B4. -- " " x_1

" " x_2

B5. -- " " $2x_2$

" " $2x_3 = \frac{\sqrt{b}}{2q}$

" " $\frac{\sqrt{a}}{2}$

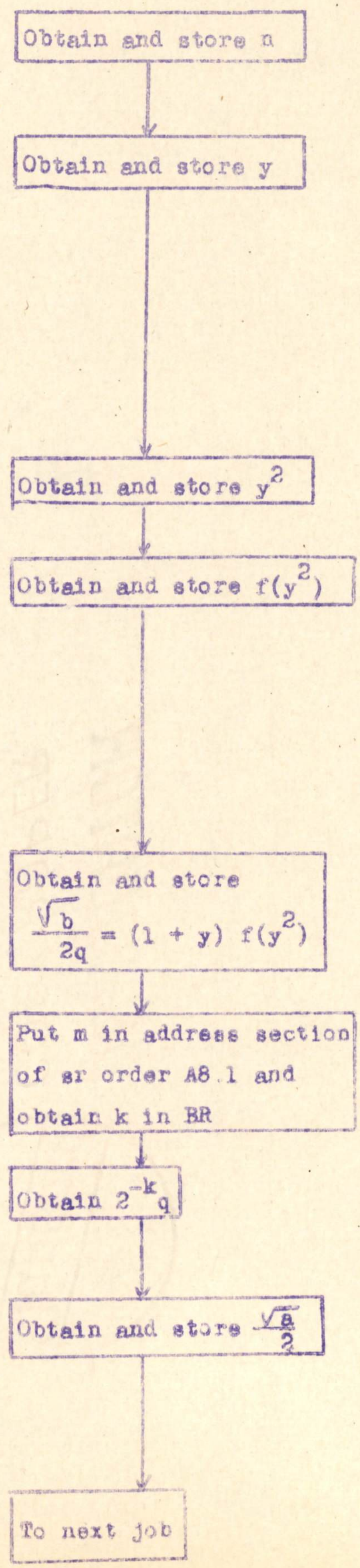
C1 $\frac{\sqrt{2}}{8}$

C2 $\frac{\sqrt{2}}{2} = q$

C3 $\frac{1}{2}$

C4 $1 - \sqrt{2} = (1 - \sqrt{2})q\sqrt{2}$

A1.1	ca	B1	a
2	sf	B2	AC contains b B2 contains $n \times 2^{-15}$
A2.1	sr	1	$\frac{b}{2}$
2	ad	C1	$\frac{b}{2} + \frac{\sqrt{2}}{4}$
3	ts	B3	
4	su	C2	$\frac{b}{2} - \frac{\sqrt{2}}{4}$
5	dv	B3	
6	sl	15	6 See A2.1
7	ts	B4	
A3.1	mr	B4	y^2 See A2.7
2	ts	B3	
A4.1	mr	C3	$\frac{3y^2\sqrt{2}}{16}$
2	ad	C1	$\frac{3y^2\sqrt{2}}{16} + \frac{\sqrt{2}}{4}$
3	mr	B3	$\frac{3y^4\sqrt{2}}{16} + \frac{y^2\sqrt{2}}{4}$ See A3.2
4	ad	C2	$f(y^2)$
5	ts	B3	
A5.1	mr	B4	$y\, f(y^2)$ See A2.6
2	ad	B3	$(1 + y)\, f(y^2) = \frac{\sqrt{b}}{2q}$ See A4.5
3	ts	B4	
A6.1	ca	B2	$n \times 2^{-15}$
2	mh	C4	AC contains $m \times 2^{-15}$ BR " k
3	td	A8.1	
A7.1	sl	15	k
2	mr	C5	$(1 - \sqrt{2})\, kq\sqrt{2}$
3	ad	C6	$2^{-k}q$
A8.1	sr	--	$2^{-m}2^{-k}q = 2^{-\frac{n}{2}}q$ See A6.3
2	mr	B4	$2^{-\frac{n}{2}}\frac{\sqrt{b}}{2} = \frac{\sqrt{a}}{2}$ See A5.3
3	ts	B4	
A9.1	sp	--	



Data Storage on following page

Data Storage for Code XVII

B1 a

B2 -- Used for $n \times 2^{-15}$

B3 -- " " $\frac{b}{2} + \frac{\sqrt{2}}{4}$

" " y^2

" " $f(y^2)$

B4 -- " " y

" " $(1 + y)\, f(y^2) = \frac{\sqrt{b}}{2q}$

" " $\frac{\sqrt{a}}{2}$

C1 $\frac{\sqrt{2}}{4}$

C2 $\frac{\sqrt{2}}{2}$

C3 $\frac{3\sqrt{2}}{16}$

C4 $\frac{1}{2}$

C5 $2^{-\frac{1}{4}}(1 - \sqrt{2}) = (1 - \sqrt{2})q\sqrt{2}$

C6 $2^{-\frac{3}{4}} = q$

Code XVIII

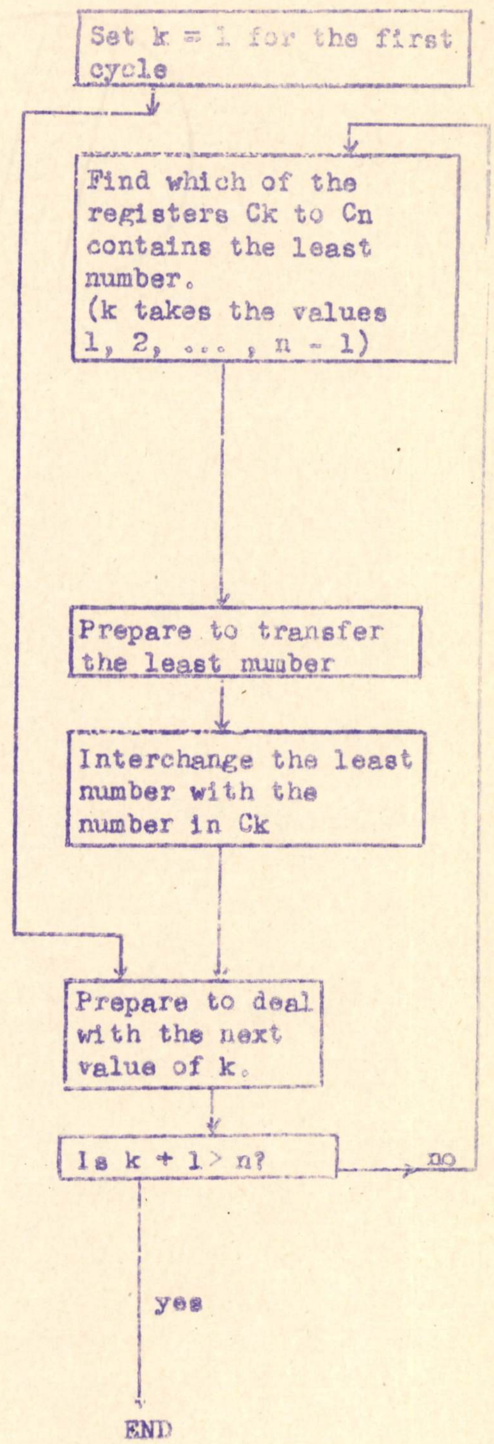
```
A1.1  ca  B2         cs C1
  2   ts  B5
  3   sp  A5.2

A2.1  cs  --         -x_m             See A5.7,A2.8
  2   ad  --         (min x_k ... x_(m-1)) - x_m
                                      See A5.2, A2.5
  3   cp(-)A2.6
  4   ca  A2.1       cs  RC x_m
  5   td  A2.2
  6   ca  A2.1       cs  RC x_m       See A2.3
  7   ad  B1         cs  RC x_(m+1)
  8   td  A2.1
  9   su  B3         (m+1-n) x 2^-15
 10   cp(-)A2.1

A3.1  ca  A2.2       ad  RC (min x_k ... x_n)
  2   td  A4.3
  3   td  A4.6

A4.1  ca  --         x_k              See A5.3
  2   ts  B4
  3   ca  --         min x_k ... x_n  See A3.2
  4   ts  --                          See A5.4
  5   ca  B4
  6   ts  --                          See A3.3

A5.1  ca. B5         cs Ck            See A1.2,A5.6
  2   td  A2.2
  3   td  A4.1
  4   td  A4.4
  5   ad  B1         cs C(k+1)
  6   ts  B5
  7   td  A2.1
  8   su  B3         (k+1) - n
  9   cp(-)A2.1
```



Data Storage for Code XVIII

B1 2^{-15}

2 cs C1

3 cs Cn

4 -- Used for x_k

5 -- " " cs Ck

C1 x_1

C2 x_2

.

.

.

(The numbers in C1 to Cn are renamed after each cycle)

Cn x_n

Data Storage for Code XIX

B1 -- $(2^{-15} \times Cp_t)$

B2 -- $x^{(t)}$

B3 2^{-15}

B4 ca C(c + 2)

B5 ca C(c + n)

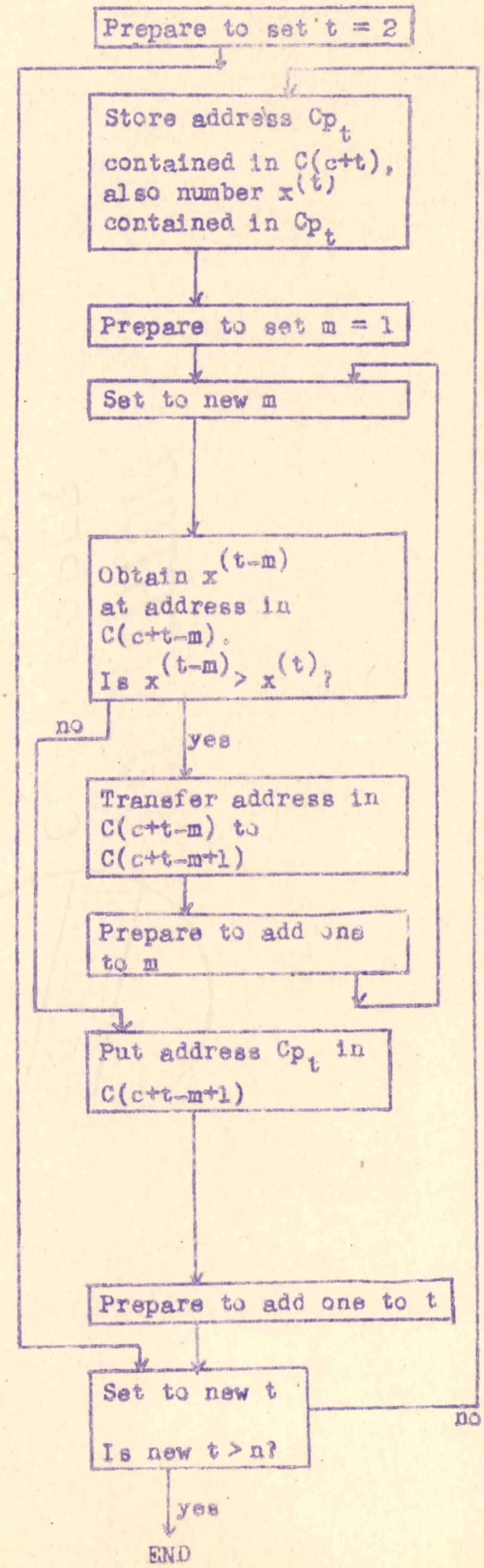
C(o) q ($<$ all x)

C1, C2, ..., Cn	$x_1, x_2, \ldots, x_n$
C(c + o)	$2^{-15} \times C(o)$
C(c + 1), C(c + 2), ..., C(c + n).	$2^{-15} \times Cp_t$

where Cp_t, t = 1, 2, ..., n, is some arrangement of C1, C2, ..., Cn.

Code XIX

A1.1	ca	B4	ca C(c+2)
2	sp	A10.1	
A2.1	ca	--	$2^{-15} \times Cp_t$ See A10.1
2	ts	B1	
3	td	A2.4	
4	ca	--	$x^{(t)}$ See A2.3
5	ts	B2	
A3.1	ca	A2.1	ca C(c+t)
A4.1	td	A6.2	ca C(c+t-m+1) See A3.1, A4.2
2	su	B3	ca C(c+t-m)
3	td	A5.1	
4	td	A6.1	
A5.1	ca	--	$2^{-15} \times Cp_{t-m}$ See A4.3
2	td	A5.3	
3	ca	--	$x^{(t-m)}$ See A5.2
4	su	B2	$x^{(t-m)} - x^{(t)}$ See A2.5
5	cp(-)	A8.1	
A6.1	ca	--	$2^{-15} \times Cp_{t-m}$ See A4.4
2	ts	--	See A4.1
A7.1	ca	A5.1	ca C(c+t-m)
2	sp	A4.1	
A8.1	ca	A6.2	ts C(c+t-m+1)
2	td	A8.4	
3	ca	B1	$2^{-15} \times Cp_t$ See A2.2
4	ts	--	See A8.2
A9.1	ca	A2.1	ca C(c+t)
2	ad	B3	ca C(c+t+1)
A10.1	td	A2.1	See A1.2, A9.2
2	su	B5	$(t+1-n) \times 2^{-15}$
3	cp(-)	A2.1	



Signed David R. Israel
David R. Israel

Approved W. Gordon Welchman
W. Gordon Welchman

DRI:lfu,aec