Massachusetts Institute of Technology

Summer Session 1954

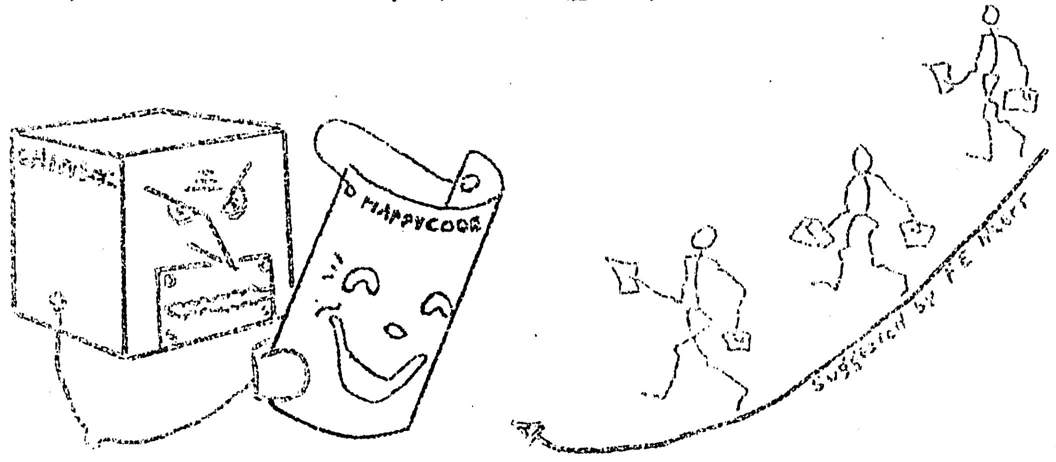
DIGITAL COMPUTERS

ADVANCED CODING TECHNIQUES

Massachusetts Institute of Technology
Summer Session 1954

Notes from a special
summer program in

DIGITAL COMPUTERS



ADVANCED CODING TECHNIQUES

prepared under the direction of
C.W. Adams, S. Gill, and D. Combelic

with the cooperation of the
Digital Computer Laboratory of the
Department of Electrical Engineering

sponsored in part by the
Office of Naval Research

Digital Computers: Advanced Coding Techniques

Table of Contents

* No notes preseantly available

1. Introduction

The title of this week-long session has been listed sometimes as Advanced Coding Techniques, sometimes as Automatic Coding Techniques. It seems obvious that any automatic technique is, in a general sense, advanced. The converse is not true. Consequently, "advanced" is more general than "automatic", and these notes are described as Advanced Coding Techniques; but they will nonetheless in large part describe automatic coding techniques. Automatic coding, according to the ACM glossary, is "any technique in which a computer is used to help bridge the gap between some 'easiest' form, intellectually and manually, of describing the steps to be followed in solving a given problem and some 'most efficient' final coding of the same problem for a given computer".

The processes which are required in preparing a program for a computer involve

1. analyzing problem
2. planning
3. coding
4. typing (or keypunching)
5. trying
6. debugging
7. running
8. analyzing results.

Of these, items 3, 4 and 6 involve essentially routine operations, capable a priori of mechanization on a computer. Not only may these clerical operations be made easier to do, but they may be made easier to learn as well. Furthermore, the running may be made more efficient by careful coding, (in some machines this is especially so because of the opportunity for minimal latency coding). Thus reducing the computer time as well as simplifying the learning and the doing of coding, typing, and debugging are various aspects of automatic coding. Simplifying or mechanizing the planning is, on the other hand, an aspect of automatic programming.

The interest in automatic coding evidenced by the number of applicants for this session at M.I.T. has grown rapidly in the past year or so. Ostensibly this enthusiasm is due to the need for simplification of coding to accomodate the new and the non-professional programmers -- the amateurs who regard programming merely as a necessary evil. The professionals of course benefit from any reduction of routine effort. The origin of automatic coding stems from work done by the early professionals -- the first for whom the novelty of computer coding wore off. Tiring of the tedium of everyday coding, looking in their spare time for new fields to conquer, they solved both problems at once by undertaking truly herculean coding projects whose aim was the simplification of coding.

The general plan of attack for these sessions is to spend most of the first two days in establishing a common language and a common frame of reference. The terms we will need we will develop gradually as we go along - but a few basic terms are needed at the onset. Most important to us are coding, programming, routine, and their derivatives - terms which have been tentatively defined in the ACM First Glossary of Programming Terminology which is included in these notes.

By description of existing systems - M.I.T.'s hypothetical Summer Session computer, Remington Rand's A-2 Compiler, and IBM's Speedcode - we hope to establish a common frame of reference. The lectures and discussions which occupy the last three days will touch on many other aspects and will permit, if not the reaching of any conclusions, at least the dissemination and perhaps the codification of the ideas of many of those most familiar with the field.

2. History of Automatic Coding: its aims and difficulties

Logical Development of a Routine

Coding methods have already come a long way since the first electronic digital computers were built. The first sophistication was the writing of addresses in decimal form for binary machines, thus making the written form of an instruction appreciably different from its form inside the machine.

Research went in two directions: towards systematizing the way a routine operates within the machine, and towards greater automatic processing of the routine before its execution. D.J. Wheeler (E7, D1) injected into an early input scheme for the Edsac some basic ideas (e.g. relative addresses) for pre-execution processes. However, for some years the greatest effort went into studying the behaviour of a routine during its execution.

Slowly it was realised that many logical changes could be made at various stages in the life history of a routine. One typical logical change is the "unwinding" of a cycle; this may be performed while the machine is executing the program, or before the program is executed. In the latter case it may be performed by the coder, by the typist, or by the machine itself. The choice must be made on the basis of over all efficiency; in general this means mechanizing as much of the work as possible, in one way or another.

The logical changes may be roughly classified as follows:

(1) Replacing human terms by terms suitable for a machine.

(2) Replacing general statements by particular statements.

(3) Expansion of detail.

The first may be subdivided thus:

(1a) Replacing terms which are intelligible <u>only</u> to a human, by terms which can be handled by a machine.

(1b) Forming terms that are especially convenient for the machine from terms that, although meaningful to the machine, are more convenient for the human coder.

The only type of change that cannot be mechanized is 1a. Type 3 involves a great increase in the amount of information to be stored and handled, but it is not usually an involved process. Types 1 and 2 involve considerably more computing, with little change in the amount of information. Hence, wherever possible, changes of types 1 and 2 should precede those of type 3. However, changes are often of mixed types, and also some changes must necessarily follow certain others, so that the general pattern is not simple.

Kinds of Techniques Available

The following are typical of the ways in which logical changes can be made by a machine.

Mnemonic coding: it is convenient for humans to use suggestive letter pairs or triples to denote the operation sections of instructions, or to identify subroutines. Machines usually ultimately require a digital representation. This is a simple change of type 1b. Whirlwind I has always performed this change on the operation sections of instructions.

Identification of words: the control unit of a machine can operate on words only if it is given their absolute addresses. Humans find it more convenient when coding to use relative or symbolic addresses. The necessary change is of type 2 and can be performed by the machine, usually before execution of the routine.

Representation of numbers: these may originate as, say, .023, 47.10^{-5} or 7/45. The conversion to the standard machine form can be performed by the machine, though the routine required to do this is often lengthy. Type 1b.

Interpretive routines: these form an extremely powerful technique for carrying out all kinds of logical change during the execution of a routine; however, their use is inefficient unless accompanied by considerable expansion of information (type 3).

Library of Subroutines: the incorporation of a library subroutine is largely a type 3 change, hence it is best left until a late stage. However, it can only be made fully automatic if the machine has a large store; many users of small machines have had to include subroutines in the input tape. Furthermore there are often several type 2 changes to be made to the subroutine after it has been copied, so that usually at least one copy of the subroutine is incorporated in the routine before execution. This is also more economical in machine time if the auxiliary store is of slow access. The Programming Research Section of Remington Rand, Inc., led by Dr. Grace Hopper, has done a great deal of work on the automatic insertion of subroutines; this is one of the chief functions of the Univac "compiler" routines (F2, F3).

Subroutines

Basically a subroutine is just a list of instructions to be obeyed by the machine. However, the library form of the subroutine frequently undergoes some development before it is executed, so that one must distinguish the initial and final forms of the subroutine. The first step in this direction was Wheeler's introduction of relative addresses and preset parameters (E7, D1). Later it was found that many subroutines required special changes that could only be made by special bits of coding ('interludes') attached to the library copy (D1). These interludes appear in the initial but not in the final form of the subroutine.

Finally, Dr. Grace Hopper's group have developed subroutines whose initial forms are entirely "interlude"; these are called 'generators'.

<u>Language Aspect</u>

The fact that changes are made to a routine after it is written and before it is executed means that the code in which it is written differs from the instruction code of the machine. Such codes are often called 'pseudo-codes'.

Our aim is to enable the coder to write his programs in a pseudo-code which is as convenient for him to use as possible. For most purposes the best pseudo-code would be a very free mixture of English and mathematical symbols, but there are difficulties in providing such a pseudo-code:

(1) existing transcription machines (e.g. Flexowriters), although capable of most of the actions of typewriters, do not readily handle all mathematical formulae.

(2) the meanings of words often depend on the context in a way which is known intuitively to humans but cannot be defined, and therefore cannot be coded for a machine.

(3) even where meanings can be defined, the routine required to perform the translation may be extremely cumbersome.

<u>Exposition of Pseudo-Code</u>

Whatever pseudo-code is used it must be accurately described for the coder. If it were possible to use English with no limitations the description would be simple. If we choose to provide no facilities for the coder, so that he must use the machine code itself, this again can be comparatively simply described. If however, we just provide the best pseudo-code we can, then the description entails a considerable amount of carefully written exposition. Without such a description the whole scheme is useless, and it obviously pays to spend a great deal of effort in preparing it.

Even the best description must necessarily demand some study by the user. We face the dilemma that although we want to encourage and to assist the newcomer to coding, we can only do this by making him study a formidable handbook of coding.

These considerations react on the design of pseudo-codes. Unfortunately a code which is easy to use is not always easy to describe, so that some compromises must be made.

RESUMÉ OF DISCUSSION - SESSION 2

D. Combelic asked for a further clarification of the distinction between the conversion function of changing general statements into particular statements and the function of detail expansion.

S. Gill gave the setting of parameters as an example of the former function and the calling of a subroutine identified by catalogue number as an example of the second.

A comment was made to the effect that a flexible code would be necessary to implement a complex automatic conversion routine. Such a routine might be difficult to construct for a computer which could not completely modify its own program (e.g., Mark III).

D. Arden suggested that the classification of conversion functions be modified to include specification of variables instead of particularization of general statements.

AN EXAMPLE OF AN AUTOMATIC CODING SYSTEM

MIT SUMMER SESSION COMPUTER

The MIT Summer Session Computer, hereafter referred to as the SS Computer, does not exist as an assembly of electronic apparatus; rather its realization is achieved by appropriate conversion, assembly, interpretive routines operating in the Whirlwind I Computer.

In order to allow students to write and operate programs within only a few days after their introduction to the basic concepts of digital computers, programming had to be easy to learn and teach. In addition, it was necessary to provide means for finding mistakes in programs, means which were simple to use and the results of which were easy to interpret, so that, within the very short time available, students would write one or more programs and run them successfully on the computer. The SS Computer, the development of which required about 12 man-months of work by experienced programmers, is an attempt to achieve these goals.

Description of the SS Computer

The SS Computer is a single-address, medium-speed (about 600 operations per second) digital computer with a basic word length of 28 binary digits and a 4-binary-digit tag. Each word is stored in two consecutive 16-binary-digit Whirlwind I registers. There are three kinds of words: (1) fixed-point integers, (2) floating-point numbers, and (3) instructions. Input to the computer is by means of punched Flexowriter paper tape; output equipment includes a direct printer, a "delayed" printer for later printing of information recorded at high speed on magnetic tape, and an oscilloscope, on which individual points may be plotted by the computer and the result observed and/or simultaneously automatically photographed.

Integers and Numbers

Fixed-point integers are represented by 28 bits, the first being the sign, the last 27 bits representing the magnitude of the integer.* Since 2^{27}=134,217,728 is approximately equal to 10^8, integers may be thought of as roughly equivalent to 8 decimal digits.

An integer is written in an SS program with a + sign (may be omitted) or - sign, followed by 1 to 8 decimal digits, and is terminated by either a tab or carriage return.

Floating-point <u>numbers</u> are numbers in the form A . 2^B, where B is an integer and A is a fraction with $\frac{1}{2} \leq |A| < 1$. In the SS Computer the mantissa A is represented by a sign and 20 bits, the exponent B by a sign and 6 bits. Since 2^{20}=1,048,576 is approximately equal to 10^6, numbers in the SS Computer have 6 decimal digits of precision. The 6 binary digits available for the magnitude of the exponents allow non-zero numbers to range in magnitude from 2^{-64} to 2^{63} or approximately from 10^{-19} to 10^{+19}. Zero has the special representation 0 x 2^{-63}, i.e., a zero mantissa, and a negative exponent of the largest permissible magnitude. A number written with a decimal point is treated as a floating-point number. Alternatively, or in addition to the decimal point, a number intended as a floating-point number may be followed by the letter x and 10 to some power; thus any of the following are treated as floating-point numbers:

-12.73   +.0063 x 10^{-2} +.0   97.6 x 10^4

<u>Arithmetic Element</u>

The Arithmetic Element consists of principally an Accumulator (AC), which deals with either integers or floating-point numbers as the situation demands.

When integers are involved, the AC contains a sign and 54 bits, i.e., is double-length. Another register, known as the Remainder Register (RR)

* The remaining 4 bits of the 32 available comprise a so-called logical information tag. This tag contains information about the kind of word, i.e., integer, number, instruction, or undefined, and also whether the word has been altered from its original form during the operation of a program.

may be thought of as a kind of right-hand extension of the AC -- the RR holds the remainder after unrounded division of one integer by another.

Sums, differences, products, or quotients may be as large as 2^{32767} without exceeding the capacity of AC, but only numbers less than $2^{63} \approx 10^{19}$ in magnitude may be copied from AC into storage. Numbers which become smaller than 10^{-19} are automatically set to zero when copied into storage.

To discourage little tricks and to help isolate real mistakes, one special restriction is that integers and numbers may not be mixed in the AC; e.g., it is not permitted to add an integer to a number. If such mixed operation is attempted the computer stops and prints out information likely to be useful to the programmer in diagnosing the mistake. This automatic print-out when a programming mistake is made is called a "computation post-mortem", and more will be said of it later.

Words in the SS Computer

Instructions are represented in the SS Computer by an operation section, an address section, and an additional "counter letter" to select one of 7 counters, or no counter at all. The counters are used for cycle-counting and address modification (like the Manchester B-box), as will be explained.

There are 35 operations, including: arithmetic operations (most of which apply equally well to either numbers or integers), operations which copy words from one place to another, "jump" operations for interrupting the normally consecutive carrying out of instructions, operations for changing the contents of the counters, and operations for controlling the in-out equipment. Operations are specified by the programmer as a mnemonic combination of three lower case letters, a tabulation of which is given on the last two pages with the meanings of the three letters, the definition of each associated instruction, and information about what may cause a post-mortem when performing each instruction.

Addresses may be written in absolute or floating form. An absolute address is any positive integer 0 through 299. (This limitation thus restricts SS programs to maximum length of 300 words.) A floating address is a single lower case letter (except o or l) followed by not more than 3 digits.

Floating addresses are used as part of an instruction by writing, for example:

ccf b3

The word referred to by the instruction ccf b3 must have the floating address b3 assigned to it. This is done by using a comma; thus

b3, 750

will tag the register contining the integer 750 with the floating address b3 so that all instructions in the program with b3 as their address sections will refer to this same register.

For corrections purposes only, words may be assigned to registers by writing, for example,

b3| -750

Following the floating address by the vertical bar instead of a comma results in the previous contents of the register b3 being replaced, in this case, by the integer -750.*

A counter letter (a, b, c, d, e, f, or g) may be appended to the address section of an instruction, and the contents of the specified index will be added to the address section of any such instruction before the instruction is executed without changing the original form of the instruction in storage. Each of the 7 counters consists of an index and a criterion, i_a and n_a, i_b and n_b, etc., respectively. In an ordinary cyclic process i_b, for example, is set to 0 and the criterion is set to some value n_b. Then, for each step in the cycle, i_b is increased by 1 until $i_b = n_b$, at which time the cycle is complete. The counters are designed primarily for counting and for modifying addresses, although

* Words to which no floating address has been assigned may also be modified. For example, if instead of the word in b3, the fifth word after the one in b3 were to be changed to +625, the word assignment would be

b3+5| +625

other applications are possible.

Programmed Output

There are three output devices: (1) a scope on which discrete points may be plotted. A camera is attached to the scope so that a display may be photographed if a permanent record is desired. The operation pat and frc are used in controlling the scope. (2) A "direct" typewriter on which integers and arbitrary characters may be recorded at the rate of 10 per second. (3) A "delayed" typewriter on which the same sort of characters may be recorded in Flexowriter-coded form on magnetic tape at a rate of 125 per second and later typed out at 10 per second while the computer is doing something else. The instruction tyc m will type the Flexo character whose octal equivalent is equal to the address section m. The instruction tyn m will type out the contents of AC as a series of decimal digits, the particular form of the print and the number of digits to be printed being specified by the address section m. Details of these very useful instructions are given in the tabulation of the operation code.

Programmed Input

Once a program has been read into the computer more data can be supplied to it only by using the operations ric or rin. Both these operations control the Mechanical Tape Reader (MTR) into which is inserted a punched Flexo tape. The operation ric is used to read in individual characters punched in the tape; rin is used to read in an entire integer or number, the termination of which is indicated by a tab or carriage return character punched on the tape.

Program Preparation

Programs are prepared for input to the computer by typing them on a Flexowriter tape perforator. The sequence is as follows.

1. The first line of the typed program consists of at least one lower case letter s followed by an identifying program title followed by a carriage return.

2. 25 or more equal signs followed by a carriage return.

3. The program itself, consisting of integers and/or numbers, and instructions; each such word must be terminated by one or more tabs or carriage returns.

4. Any word assignments (e.g., b3| -750) which are necessary, each terminated as in 3, above.

5. The address at which the program is to start operating followed by a vertical bar, followed by the word start in the lower case. Examples are:

a7| start   127| start   g6+3| start

Post-Mortems

When an error has been made by the programmer, and that error is detected either during input or operation of the program, the computer stops and prints out information about that error. Such a print-out is called a post-mortem. There are two types of post-mortems in the SS computer.

Conversion Post-Mortem

The punched program tape is read into the computer through the photoelectric tape reader. After the tape has been read in and the binary equivalents of the characters stored, the SS conversion program processes the stored data and eventually produces a sequence of words in binary form which will be correctly interpreted as a program by the SS computer interpretive routine. If, however, certain logical or typographical errors have been made which are detectable during the conversion process, the computer will stop the conversion and print on the direct typewriter the title of the program tape followed by a description of the mistake and its location in floating address form.

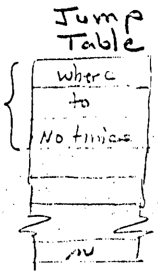
The mistakes detected by the conversion program are:

1. unassigned floating address
2. undefined instruction
3. floating address assigned to two or more registers
4. absolute address too large
5. program longer than 300 words
6. integer or number too large
7. numerical part of floating address too large
8. no counter letter specified in rst, jii, jic, inc, dec, or cii instruction.

Computation Post-Mortem

If, during the operation of a correctly converted program, a situation arises which is defined as a programming mistake (see list page 9), the computer automatically records on magnetic tape certain information and then stops. The recorded information is known as Computation Post-Mortem, and is subsequently typed out on the delayed typewriter. A typical Computation Post-Mortem might appear as follows:

```
            ss program 27, John Smith, Sept. 23, 1953     Identification
            STOPPED AT  dl+6   dl+6|patal7   al7|-981
Accumulator→AC|1034   RR|5
                         ←Remainder Register
            h4..j8   d2..dl+8   (dl..dl+8)21   dl..a8   z7..p97+3  q6..q9
            p97+4..a5-1   h6..h7+4   r4..r6-2   t8..y37-1   d2..dl+6  stop
     altered COUNTERS  a|2,5  b|6,6  c|23,23  d|12,12  e|0,0  f|0,0  g|0,0
     during il|19   -601   al7|-981   p902|ccfp920d   q9|jmpp97+4  etc.
                  i1+1
```



The information given is as follows:

Line 1: the program title for identification purposes.

Line 2: the computer stopped while performing the patal7 instruction in dl+6. al7 contained the integer -981.

Line 3: The AC contained 1034, which is why the computer stopped on the patal7 instruction. (See programming mistake 6A in the operation code.) The Remainder Register (RR) contained 5 -- had the RR not been used, no information about it would have been printed.

Lines 4 and 5: traces the path the computer followed over the ten most recent jump instructions--only those jumps which actually were effective are included. The example given shows that the computer performed each instruction from h4 to j8; then some kind of a jump instruction in j8 took it to d2. Each instruction from d2 to d1+8 was carried out; then the sequence d1 to d1+8 was repeated 21 times. The next time through this sequence it went right on through to 28, whence some kind of jump took it to z7, etc. Each address is given in terms of the nearest floating address.

Line 6: gives the index and criterion, in that order, for each of the 7 counters--had no counters been used this item would not appear.

Line 7, etc.: lists the address, in terms of the nearest floating address, and the contents of every register whose contents have been altered during the program. If the contents of several consecutive registers have been changed, an address is given for the first and for each one to which a floating address has been assigned.

INSTRUCTION CODE OF THE MIT SUMMER SESSION COMPUTER

INSTRUCTION	MEANING	DEFINITION	POST-MORTEM*if
ccf al b	copy contents from	$C(al+i_b) \longrightarrow AC$	L14
cci al b	copy contents into	$C(AC) \longrightarrow al+i_b$	A4, A5, U9
cnf al b	copy negative from	$-C(al+i_b) \longrightarrow AC$	L14, L15
cmf al b	copy magnitude from	$\lvert C(al+i_b)\rvert \longrightarrow AC$	L14, L15
cri al b	copy remainder into	$C(RR) \longrightarrow al+i_b$	
xch al b	exchange	$C(AC) \longrightarrow al+i_b$, $C(al+i_b) \longrightarrow AC$	A4,A5,L14,U9
add al b	add	$C(AC)+C(al+i_b) \longrightarrow AC$	A1, L12, U3
sub al b	subtract	$C(AC)-C(al+i_b) \longrightarrow AC$	A1, L12, U3
mby al b	multiply by	$C(AC) \times C(al+i_b) \longrightarrow AC$	A1, L12, U3
dby al b	divide by	divide $C(AC)$ by $C(al+i_b)$ rounded quotient $\longrightarrow AC$	A11, L12, U3
dhr al b	divide holding remainder	divide $C(AC)$ by $C(al+i_b)$ quotient $\longrightarrow AC$, remainder $\longrightarrow RR$	A11, L13
txi al b	transfer excess into	divide $C(AC)$ by 2^{27} quotient $\longrightarrow al+i_b$, remainder $\longrightarrow AC$	U10
jmp al b	jump	take next instr. from $al+i_b$	L17
jip al b	jump if positive	ditto, if $C(AC) > 0$	L8, L17, U9
jin al b	jump if negative	ditto, if $C(AC) < 0$	L8, L17, U9
jiz al b	jump if zero	ditto, if $C(AC) = 0$	L8, L17, U9
jir al b	jump if remainder	ditto, if $C(RR) \neq 0$	L17
jix al b	jump if excess	ditto, if $\lvert C(AC)\rvert \geq 2^{27}$	L17
sra al b	set return address**	replace address section of $C(al+i_b)$ with 1+the address of the register containing the most recent jmp or conditional jump which took effect	L16
caf al b	copy address from**	address section only (as an integer) of $C(al+i_b) \longrightarrow AC$	L16
cai al b	copy address into**	$C(AC)$ becomes the new address section of $C(al+i_b)$	A7, L16
rst m b	reset (counter b)	set $i_b=0$, $n_b=n$	U2, U19
jii al b	jump if incomplete	increase i_b by 1, then jump to al if $i_b < n_b$	A7,L3,A18,U19
jic al b	jump if complete	increase i_b by 1, then jump to al if $i_b \geq n_b$	A18, U19
inc m b	increase (counter b)	increase both i_b and n_b by m	A18, U19
dec m b	decrease (counter b)	decrease both i_b and n_b by m	A18, U19
cii al b	copy index into	i_b as an integer $\longrightarrow$ al	
cnv al b	convert	$C(AC)$ as an integer $\longrightarrow AC$ $C(AC)$ as a number $\longrightarrow al+i_b$	A1, L8, U9
stp 0	stop	stop the computation	

* The programming mistakes which result in a post-mortem are listed on the next page. A post-mortem results while performing an instruction if any of the programming mistakes listed with that instruction are made. A post-mortem will always occur also if $(al+i_b) \geq 300$ or if $(al+i_b) < 0$.

** When executing this instruction, a counter letter, if any, is not considered part of the address section of the instruction in register $al+i_b$.

INSTRUCTION	MEANING	DEFINITION	POST-MORTEM*if
pat al b	plot at	plot a point on the scope at $x = C(al + i_b)$ and $y=C(AC)$	A6, L8, L14 L15, U9
frc 0	frame(scope)camera	move the next film frame into place and open the camera shutter if it was closed	
ric 0	read in character	read the next char. via the MTR into AC as a pos. integer ≤ 77	(Comp. stops if no tape in MTR)
rin 0	read in numerically	read the next complete integer or number via the MTR into AC	A1, A2, (also see ric above)
tyc m tyc m+100	type character	record on delayed printer (m), or on direct printer (m+100), the Flexo. char. specified by m	L20
tyn m tyn m+100	type numerical value	record on delayed printer (m), or on direct printer (m+100), C(AC) as specified by m (See table below)	A4, A5, L8 U9, U21

Tabulation of m values for use with tyn

m	initial zeros	no. of digits printed = d	total space pos.	total space neg.	zero prints as	
0	ignored	$1 \leq d \leq 9$	d	d+1	0	
1-9	printed	d=m	m	m+1	m 0's	$C(AC) \geq 10^{m}$
11-19	spaced over	$1 \leq d \leq (m-10)$	m-10	m-9	see examples	$C(AC) \geq 10^{m-10}$
21-29	ignored	d=n-20	m-11	m-11	0	

Examples	C(AC)=1234	C(AC) = -789	C(AC) = .004786	C(AC)=13.57	Direct/Delayed
m=0	1234	-789	0	14	Delayed
m=103	Post-Mortem	-789	000	014	Direct
m=5	01234	-00789	00000	00014	Delayed
m=116	**1234	**-789	*****0	****14	Direct
m=23	$*1.23x10^{3}$	$-7.89x10^{2}$***	$*4.79x10^{-3}$**	*1.36x10	Delayed

*represents a space on the printed copy

PROGRAMMING MISTAKES which cause a POST-MORTEM

A 1 - Result is an integer of magnitude $\geq 2^{54}$
A 2 - Result is a number of magnitude $\geq 2^{63}$
U 3 - Result is a number of magnitude $\geq 2^{32767}$
A 4 - C(AC) is an integer of magnitude $\geq 2^{27}$
A 5 - C(AC) is a number of magnitude $\geq 2^{63}$
A 6 - $|C(AC)| \geq 1024$ or $|C(al+i_b)| \geq 1024$
A 7 - C(AC) is not a positive integer < 300
L 8 - C(AC) is an instruction
U 9 - C(AC) is undefined
U10 - C(AC) is not an integer
A11 - $C(al+i_b) = 0$
L12 - C(AC) and $C(al+i_b)$ are not either both integers or both numbers

L13 - C(AC) and $C(al+i_b)$ are not both integers
L14 - $C(al+i_b)$ is undefined
L15 - $C(al+i_b)$ is an instruction
L16 - $C(al+i_b)$ is not an instruction
L17 - If $C(al+i_b)$ is not an instruction and the jump takes effect, the Post-Mortem will occur after the jump is executed
A18 - Resulting magnitude of $i_b \geq 512$
U19 - $m \geq 512$
L20 - $m > 77$ or m corresponds to an illegal Flexo character
U21 - m=10 or m=20 or $m \geq 30$

A - Arithmetic overflow L - Logical mistake U - Unlikely mistake

DEFINITIONS OF SYMBOLS

$\longrightarrow$	becomes the new contents of
AC	Accumulator
C(al)	Contents of register al. al represents any floating address, i.e., any letter except o or l, followed by any positive decimal integer < 1000
$C(al+i_b)$	Contents of the register whose address is obtained by adding to al the value of i_b.
i_b	The index associated with counter b, where b represents any of the 7 counters a, b, c, d, e, f or g. Except for the 6 instructions rst, jii, jic, inc, dec, cii, a counter letter need not be specified at all.
n_b	The criterion associated with counter b.
RR	Remainder Register, which holds the remainder after dhr and is not changed by any other instruction.
MTR	Mechanical Tape Reader into which is inserted a punched Flexo tape to be read in under the control of the computer.

Discussion

Session 3

In reply to questions, Donn Combelic stated that the number of Whirlwind I instructions required to execute one Summer Session instruction varies from 30 to 900, and the average time required to execute one Summer Session instruction is about 100 times the time taken by one Whirlwind instruction. However, it must be borne in mind that a Summer Session instruction performs much more work than a Whirlwind instruction.

Mr. G. Clotar asked why the input conversion routine could not be made to accept, for example, cfc as being equivalent to ccf, instead of reporting this as a mistake. Prof. Adams said this was an important point. It was thought better not to allow freedom where there was nothing to be gained by it, and it was felt desirable to detect as many as possible of the mistakes that might commonly be made. Mr. Charles G. Lincoln, who had used the Summer Session computer, agreed with this view.

Stanley Gill described a technique used on the Illiac which makes it unnecessary to list constants separately and refer to them in the program; instead, the constants may be written directly in the instructions which use them; thus, instead of writing

ccf al and al, +123

one might write simply

ccf +123.

Donn Combelic gave, as an example of mnemonic coding, the way in which input and output editing is requested in the M.I.T. Comprehensive System. Three letters specify respectively the medium, whether input or output, and the notational form; this may be followed by a sample number. Thus MOA +1.2345 calls for output to the magnetic tape in alphanumeric form, numbers consisting of 5 decimal digits with a decimal point after the first digit and preceded by a sign. DIB calls for input to the high-speed store from the drum in binary form.

In answer to a further question, Donn Combelic stated that no official means for reverting from Summer Session instructions to Whirlwind instructions during the run of a problem had been provided. The Summer Session computer was designed for the one-shot programmer rather than for the one-shot program, and was designed to be easy to learn rather than efficient to use.

Session 4

J.W. Backus asked whether summer session programmers modified instructions by any other means than the B-box instructions since these were so comprehensive. Stanley Gill replied that the other operations designed for instruction modification were used. One probably could restrict instruction modification to B-box operation, but it seems that in some situations the lack of flexibility is undesirable.

In reply to a question, Prof. Adams stated that internal operation was binary rather than decimal. Conversion took place during input and output. This was a sore point since some students attempted to count in floating point arithmetic. Because of the inexact binary representation of such numbers as .01 etc., successive additions did not yield integral multiples, e.g., a student might print the first digit of 1.999 ... instead of the expected 2.0000 ... Donn Combelic pointed out that this could be avoided by using the auxiliary counters. When asked whether there was a demand for more than 7 B-boxes in the summer session computer, he replied that no one has yet required more. Mr. Walter A. Ramshaw commented that experience at United Aircraft where 3 are available indicates that 30 might be useful. Donn Combelic pointed out that in the case of the summer session computer the fact that only 300 memory registers were available undoubtedly limited the demand for B-boxes.

Session 5

In answer to questions from R.E. Porter and J.W. Backus, Donn Combelic stated that different floating address tags are independent; i1 and i2 do not necessarily label consecutive words. Words not tagged by the coder have no tags, but may still be operated upon, e.g., by instructions which depend on counters. Space can be allocated for vectors by writing several zeros; only the first need be tagged. Replying to further remarks by J.H. Brown and J.D. Porter, he described more convenient ways of doing this in the Comprehensive System.

L. Rosenthal and J.H. Brown asked what methods were available for finding mistakes not found by the automatic checks. Donn Combelic agreed that mistakes of planning could not be detected by the system. He also agreed that dynamic diagnosis routines would be valuable, but said that they had been able to manage without them. He did think that dynamic diagnosis routines were apt to be overrated. D.L. Shell remarked that they were often valuable in giving a customer confidence in the machine's work.

J.W. Backus asked whether, since the mistake detection occupied 1/3 to 1/2 of the interpretation time, it could be switched off when not required. Donn Combelic replied that it could not, although experience with the Comprehensive System showed this to be a useful feature in that case.

Donn Combelic described a Whirlwind post mortem routine for displaying the entire contents of the high-speed store on the oscilloscope, and said that it was more useful than it was often thought to be. J.H. Brown said that Midac has a post mortem routine which (like the Summer Session post mortem) indicates only those words that have been changed during execution. Stanley Gill said that such a post mortem had been in use at the University of Illinois for at least two years, and described whole-store post mortems as archaic.

Dr. Hopper said that routines for Univac can be put through an "analyzer" which detects very many coding errors before execution, including, for example, certain arithmetic operations on instructions or on unplaced store contents. L. Rosenthal said that a list of all locations can be printed, showing all references to each location; this also helps when modifying a routine.

6. The Operation of a Computing Center

Introduction. The purpose of this chapter is to indicate how advanced coding techniques can affect the operation of a computing center. The topics to be described have been selected from observations made on the operation of the Digital Computer Laboratory at M.I.T. For the sake of discussion the operation of this laboratory will be divided into six activities.

1. Problem consultation. When a problem is proposed for solution at the D.C.L. the following steps are usually taken. The over-all problem is discussed and a procedure is selected (or the problem may be rejected), the problem is then coded, the resulting routines are debugged and finally run, and the results are analyzed and described in suitable reports.

The selection of a procedure involves not only the determination of a suitable numerical method, but also the selection of the computer to be used. By the coding processes being discussed in this course, it is possible to transform a center possessing just one computer into a multi-machine project. For example, at the D.C.L., without altering the hardware, we have increased the number of computers available for the solution of problems from one to four. Each of these computers has its own advantages and disadvantages. The factors involved in choosing among them include: ease of coding (as measured by the time it takes a programmer, who may be untrained, to code his problem), available storage, computing speed, computer reliability (for a simulated computer, this will depend upon the degree of testing), ease of error detection and tape correction, available precision, and available subroutine library.

The coding of the problem can be greatly simplified by the use of such techniques as floating-point representation, symbolic and relative addresses, and counting facilities. Moreover, the use of mnemonic instruction codes, compiling routines, subroutine libraries, etc. abbreviates the training period of a new programmer. Of course, the availability of more than one computer (real or simulated) does increase the number of conventions that a programmer may have to learn. Also the slowing down of the machine by interpretive routines does place a certain responsibility on the programmer to make more efficient use of such routines.

Debugging is facilitated by routines that detect and describe coding errors. It is usually simple to find the source of such errors since the run is stopped as soon as the error is detected. Many errors can be found while the problem routine is still being read into the computer. Once an error has been detected, the use of service routines permits the printing out of pertinent information in a palatable form. It should be mentioned that the printing out of post-mortem information does consume machine time. Consequently a compromise must be reached as to just how much information should be printed out each time. Also the inclusion of error detection in interpretive routines slows the machine down all the more. This might be particularly objectionable in production runs so that it might be desirable to make such features optional.

Unfortunately there will always be a few cases where the source of error is difficult to locate. The use of executive routines introduces in itself a very troublesome source. However, in practice, errors arising from an actual mistake in an executive routine or from a transient malfunction are not as difficult to localize as one might expect. Of course, debugging the executive routine itself depends on the degree of familiarity of staff members with the routine in question.

The running of the problem on the machine will be discussed in the section below on "performance". It should be mentioned that many computing centers have found it very convenient (and at times even necessary) to make use of rerun routines. At the D.C.L. the nature of the problems we have run and the reliability of WWI have made it unnecessary for us to include rerun routines as a regular feature. [illegible]

Error analysis can sometimes be effected by an interpretive routine that carries out a parallel computation on the error estimates. Such routines can be used to alter the program if certain bounds are exceeded.

2. Development of utility programs. A computing center serves as a fertile source of suggestions for new automatic routines. However there are many pitfalls awaiting the introduction of any new routines. At the D.C.L. a revised version of the comprehensive system of service routines was recently introduced in the following way. First the revisions were described and criticized at group conferences. When the changes were finally agreed upon, they were coded and debugged as separate problems.

A new copy of the comprehensive system (called CS II) was then prepared incorporating the changes. CS II was then tested by the members of the staff who were responsible for the changes. When they felt that the system was sufficiently debugged, they supervised its use by a few of the more experienced student programmers. In the meantime separate detailed memos were prepared describing how CS II would affect tape room procedures, computer operation, and the programmers themselves. Finally the new system was adopted for every day use, but special care was taken so that all existing routines could still be handled properly.

3. Clerical. Since each executive routine that may be used by a programmer will have certain conventions of its own, the use of such routines will complicate the rules for preparing tapes (or punched cards) for reading routines into the computer. However errors can be minimized by suitable supervision, dissemination of information, and tape preparation request forms. Primarily it is the responsibility of the programmer to comply with the conventions of the automatic system he is using. Inconsistencies are often detected by a staff consultant, the tape room supervisor, or the typist. Other checks can be incorporated in the read-in routine.

It should be noted that in many ways the typing of read-in tapes can be simplified (and even made elegant) by the use of executive routines. The use of pseudo-codes and library subroutines reduce the time of tape preparation. Correction of the tapes is simplified if the read-in routine can be made to ignore certain characters.

The processing of results can be facilitated by coding techniques. All computer output, whether it be typed out directly, recorded on magnetic tape or on film, can be automatically tagged with such pertinent information as the problem number, programmer's name, date, etc.

4. Performance. It is possible to automatize the running of problems on a computer so that a chosen sequence of problems can be run with no manual intervention beyond the pushing of a starting button. Many elements of such a system are already in use at the D.C.L. By the use of special symbols on the tape for a given problem, the read-in routine for the simulated computer desired will automatically be selected.. In turn, the read-in routine will provide the appropriate routines for carrying out the desired arithmetic, cycle counting, etc. For each routine

that is read into the computer, a log is punched out on paper tape of the tape number, kind of tape, the time, and whether memory was erased. (In case computer operation is interrupted for any reason this is recorded directly on to the log tape by the operator in attendance.) If a problem that makes use of an interpretive routine fails, appropriate information will be automatically printed out. Programmers who desire particular post-mortems can request these in advance by having a suitable tape prepared. Thus by splicing together a sequence of tapes and by making use of a special routine to serve as director, we expect to have a large portion of our computing periods run automatically.

5. Reports. The logging tapes produced during a computing period can be used with suitable routines to compile records summarizing computer operation over any desired period. For example, the machine time used by each problem and programmer, the amount of "down" time, percentages, etc. can all be computed and printed out for direct inclusion in reports.

6. Maintenance. Special routines have proved very useful in the testing of computers. The extent to which such routines can be used will depend, of course, upon the ingenuity of the programmers and the nature of the computer. At the D.C.L. two sets of routines have come into use. The first is used with marginal checking for the routine testing of the various computer sections. The second set is used for diagnostic purposes to locate actual failures in the auxiliary drum system and terminal equipment. In the future it is planned to combine some of the features of both sets of routines.

In answer to H. Brown's question, J. Porter stated the responsibility for having proper identifying information and properly following conventions rested with the programmer at the Whirlwind installation. The typists are not expected to find errors, but are encouraged to report any observed to their supervisors. H. Brown mentioned that at MIDAC, typists had been very useful in this respect. In answer to G. Clotar's question, J. Porter commented that under this system there is little advantage in having typists with a knowledge of computing. In answer to E.A. Voorhees' question, it was observed that the Whirlwind installation does set aside a specified period for maintenance and testing.

S. Gill questioned the necessity for duplication of labor in having programs checked by typist, programmer and machine. C.W. Adams observed that as much inexpensive checking as possible seems desirable and that the programmer's check is often not thorough.

D. Williams asked to what extent mathematical formulation of problems was checked at Whirlwind and J. Porter replied that staff limitation made such checking very difficult. J.C.P. Miller mentioned that at Cambridge a priority committee had been very successful in screening both programs and programmers and that in many cases a formulation had been changed.

W. Ramshaw inquired as to whether checks or hand solutions are required at Whirlwind and was assured that these were usually required.

The question was raised as to whether M.I.T.'s policy of having programmers do their own coding was a matter of preference or due to lack of personnel. J. Porter replied that lack of personnel and MIT's policy of training programmers were the primary factors.

P. Bremer wondered what was done at Whirlwind about machine malfunctions during computation. J. Porter and C.W. Adams replied that these were rare as marginal checking generally anticipated them. However, if a result was not repeatable or could be reasonably attributed to machine malfunctions, the engineers are given as much information as possible and generally correct the difficulty very quickly.

In answer to a question about error estimates, it developed that some centers actually used parallel computation to keep track of error accumulation.

In the course of further discussion, it appeared that Whirlwind's operating time could be broken down to approximately 65 - 75% yielding results 20 - 30% for debugging, 3 - 5% malfunction. Scheduling is done on a day-to-day basis except for large production runs.

S. Gill and L. Rosenthal discussed the fequency with which a programmer should be allowed on the machine. The consensus seemed to indicate that two or three times a day gave good results.

To allow for the unpredictability of the duration of debugging runs, a list of standbys or trading machine time were advocated.

a 16 better than 11's etc.

THE A-2 COMPILER

The A-2 Compiler is an organic executive routine which produces a running program for a specific mathematical problem. It is organic in the sense that it is an out-growth of compilers A-0 and A-1 and essentially contains them within itself and in time, will itself become a part of A-3. It is a prototype of A-3 as well.

Compiler A-2 draws on a library of subroutines of two basic types:

A. Static subroutines which need only be transformed from relative coding to specific coding and entered in the running program. These static subroutines fall into three classes:

1. Stored subroutines, including the elementary arithmetic operations. These subroutines are stored during the entire running of the problem. The running program indicates only the arguments, results and jumps necessary to use them.
2. Tape-stored subroutines. These subroutines are entered in the running program, and thus are read from the tape as required and repeated in the program as needed.
3. "Own-Coding" tape-stored subroutines. Subroutines peculiar to the specific problem, either extremely specific or of rare incidence and hence not normally included in the subroutine library.

B. Dynamic subroutines are generative routines. These fall into two classes:

1. Computational - subroutines in which one or more parameters such as exponents, decimal points, or units are defined by the input information. The library then contains a routine which processes a skeletonized relative-coded subroutine contained within it to produce a static routine.

2. Data-handling - all data-handling subroutines are generative in nature. Included in the subroutine library are generators which yield static subroutines when supplied with such information as item size, type of transfer, contraction or expansion specifications, etc.

The compiler acting on suitable information defining the problem controls the generation and transformation of subroutines of all types and their integration into a running program for the specific problem under consideration.

The information defining a problem is submitted to the compiler in a pseudo-code. Four phases can be distinguished in the operation of Compiler A-2.

Phase I - The compiler expands the information defining the problem into more readily digestible form, "Information A", and supplies certain added data such as complete call-numbers and operation numbers. In future compilers, this "translation phase" will also include the translation from functional notation or English words to a computer notation and it will be integrated into the compiling process.

Phase II - Information A is processed to "segment" the problem. Since ample provision is made for working storage, and since the arithmetic and frequently used subroutines are stored for RU reference, the running program will usually extend beyond one storage load. Hence it is necessary to subdivide the running program into segments. Each segment is so defined as to constitute a storage load or less and to contain an integral number of subroutines. In order to achieve this segmentation, during Phase II, "own coding" subroutines are transferred to the generated library tape, as are all other generated subroutines defined for the problem. During this phase, a reference record is created in which is entered for each subroutine in serial order of its appearance in the running program, its call-number, the number of the segment in which it appears, the operation number, and the location of its first line when its segment is in use. The output of Phase II includes the generated library, the record, and Information B with the added segmenting definitions.

Phase III - The record now contains all the data required to define the jumps ordered by Information B. These may be indicated by the original definition of the problem or by the segmenting sentinels. Thus Phase III creates the necessary jump instructions. Its output is "Information C" which now completely describes the required program.

Phase IV - is the main compilation. The subroutines from the main library tape and generated library tape are read as called for by "Information C", are transformed from relative to specific coding, and, together with the required jumps, read, and write, instructions, are entered in the running program. This running program is a complete and specific checked program for the specific problem.

29 October 1953

Revised 19 July 1954

7. A-2 Compiler System

I Purpose: conservation of time

1. Classes of effort contained in elapsed time per problem
   a. Analysis
   b. Programming
   c. Coding
   d. Debugging
   e. Running
   f. Re-running
2. "One-shot" and repetitive problems
3. Minimal latency coding

II Logic of computer as determining factor

1. Input-output
2. Auxiliary storage
3. Alpha-numeric
4. Checking

III Method of attack

1. Proof of feasibility
2. Prototype
3. Production model
4. Tests to be applied to any method
   a. Feasibility: is it possible, practical
   b. Suitability: does it accomplish the purpose
   c. Acceptability: is it satisfactory time-wise, economically; does it fully exploit the computer

IV Development of pseudo-code

1. A-0 code
2. A-2 code
3. Translators
4. Basic code
5. Contemplated pseudo-codes

V Subroutines < mathematical / logical Add to a limit

1. Static Sin etc!
2. Generative < sort— print operating instructions and make short test pgm
3. Multiple libraries , complex, floating decimal, fixed decimal
4. Processing

VI Allocation of storage similar to SS computer, Sub&div — not used too much Cos

1. In running tape
   a. Program
   b. Stored arithmetic
   c. Working storage
   d. Input-output
2. Segmentation
3. Auxiliary storage
4. Unwinding

VII Compiler A-2

1. Translation
2. Record
3. Segmentation
4. Jumps
   a. "Neutral Corner"
   b. Extra sweep
5. Generation
6. Compiling

VIII Results

1. Optical Ray
2. Carne Problem
3. Function Evaluation
4. Method of Relaxation
5. Commercial Applications
   a. Need of definitions
   b. Specification technique

IX Future Developments

1. Pseudo-codes
2. Translators
3. Operators
4. Generators

7. Resumé of Discussion

Replying to D. L. Shell, Dr. Hopper said that applications of the sorting generator were chiefly commercial. L. Rosenthal said it was particularly useful for sorting long items and had been used with items of 63 digits.

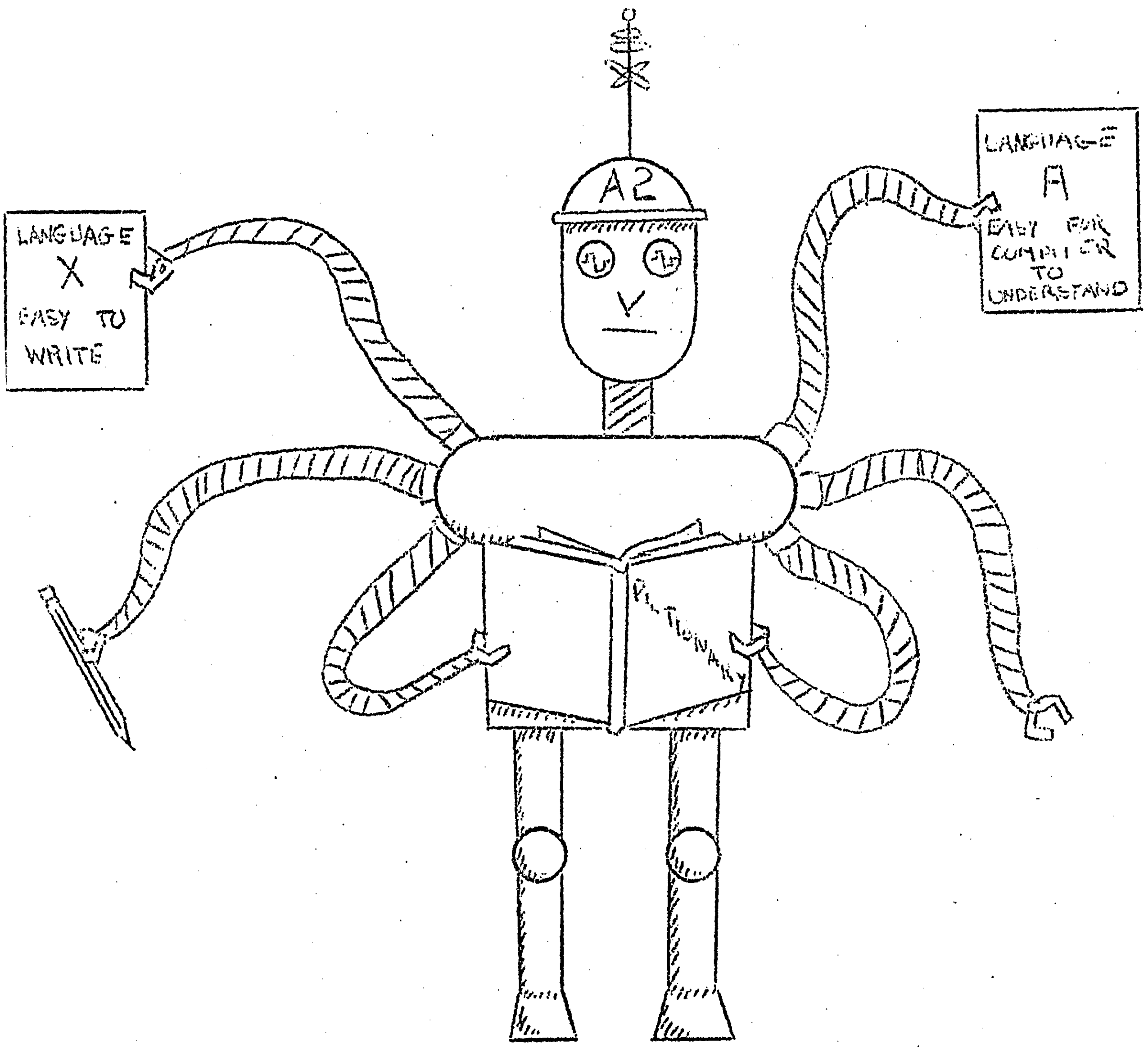
In answer to questions from J. W. Backus, Dr. Hopper showed how to code the evaluation of the scalar product of two vectors. She said that matrix operations were coded using a special matrix library.

E. A. Voorhees asked whether the compiler was used because people were dissatisfied with the machine. Dr. Hopper replied that the reason was solely to simplify and shorten coding. A 3-address code was used merely because it fitted a Univac word. She would not advocate building a machine on the lines of the pseudo-code. In reply to K. F. Powell, she said that the only machine feature associated with the compiler technique was a moderately large store, such as magnetic tape or a large drum. She thought the A-2 library could be stored on most existing drums.

Replying to W. A. Ramshaw, she said that the space allocation of large masses of data involved storing it in batches on the tape, and could be handled by the data handling generator developed by the Army Map Service. Univac coding facilities were developed by various installations, particularly the six government ones, and were circulated frequently to the others. The Compiler is thus constantly growing, but no changes are made which invalidate the former arrangements.

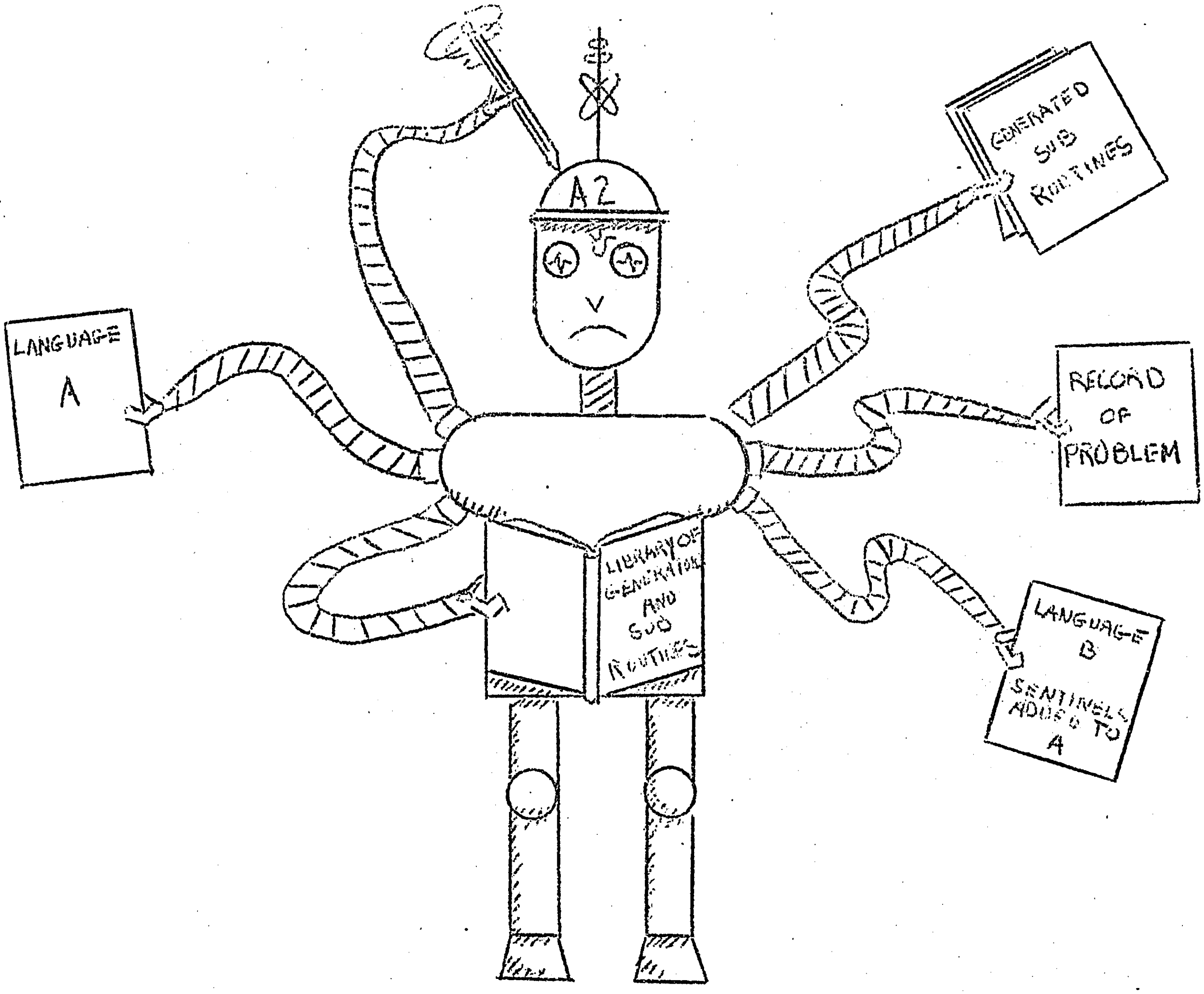
Answering L. Rosenthal, Dr. Hopper said that one of the difficulties of commercial applications was that of defining the meanings of various subroutines. Some (such as income tax and social security deductions) were defined legally, but others had widely different meanings to different people. In answer to T. M. Hurewitz, she thought that the future emphasis would be on generative subroutines.

COMPILER A-2

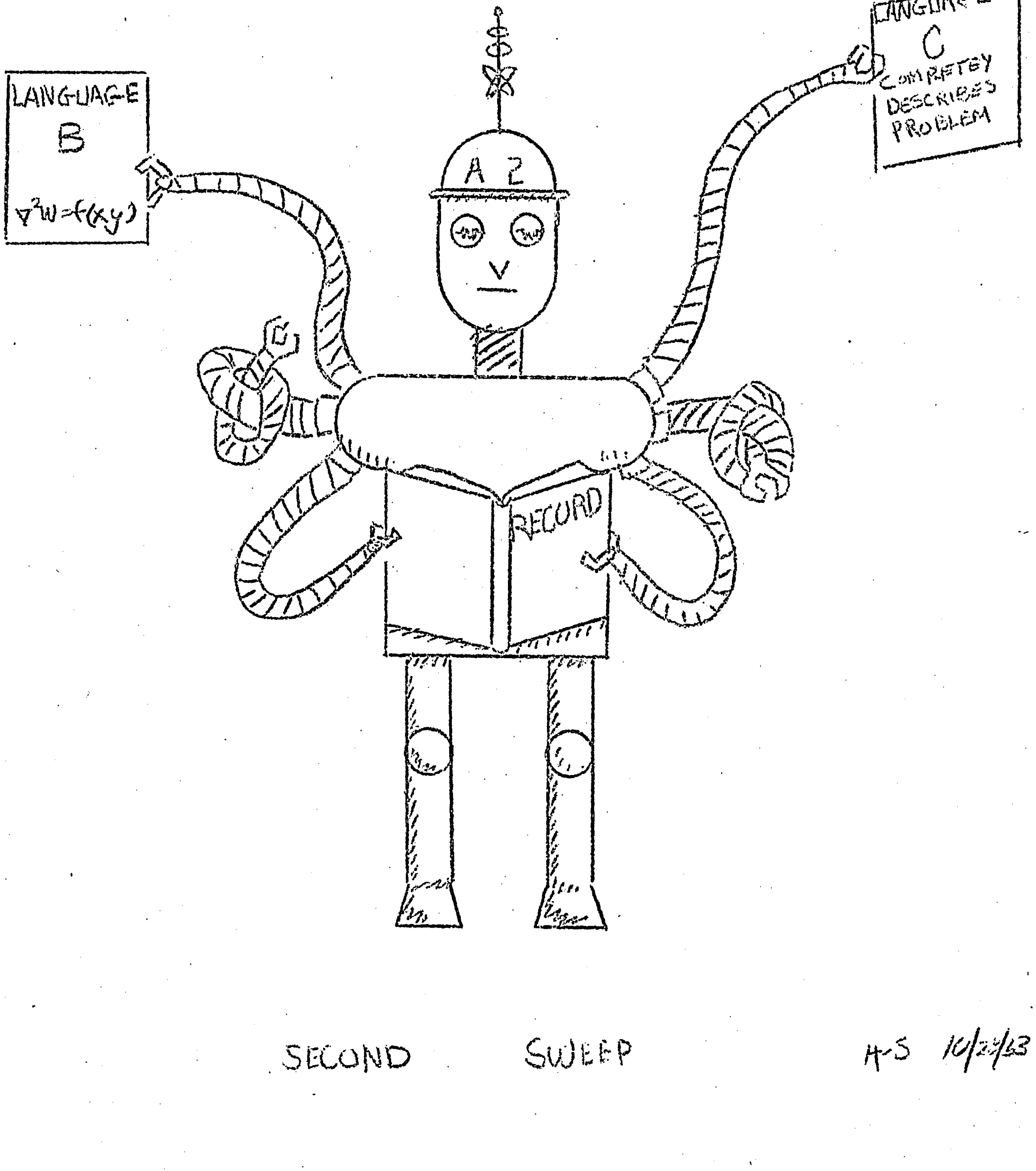


TRANSLATION PHASE

HS 10/23/53

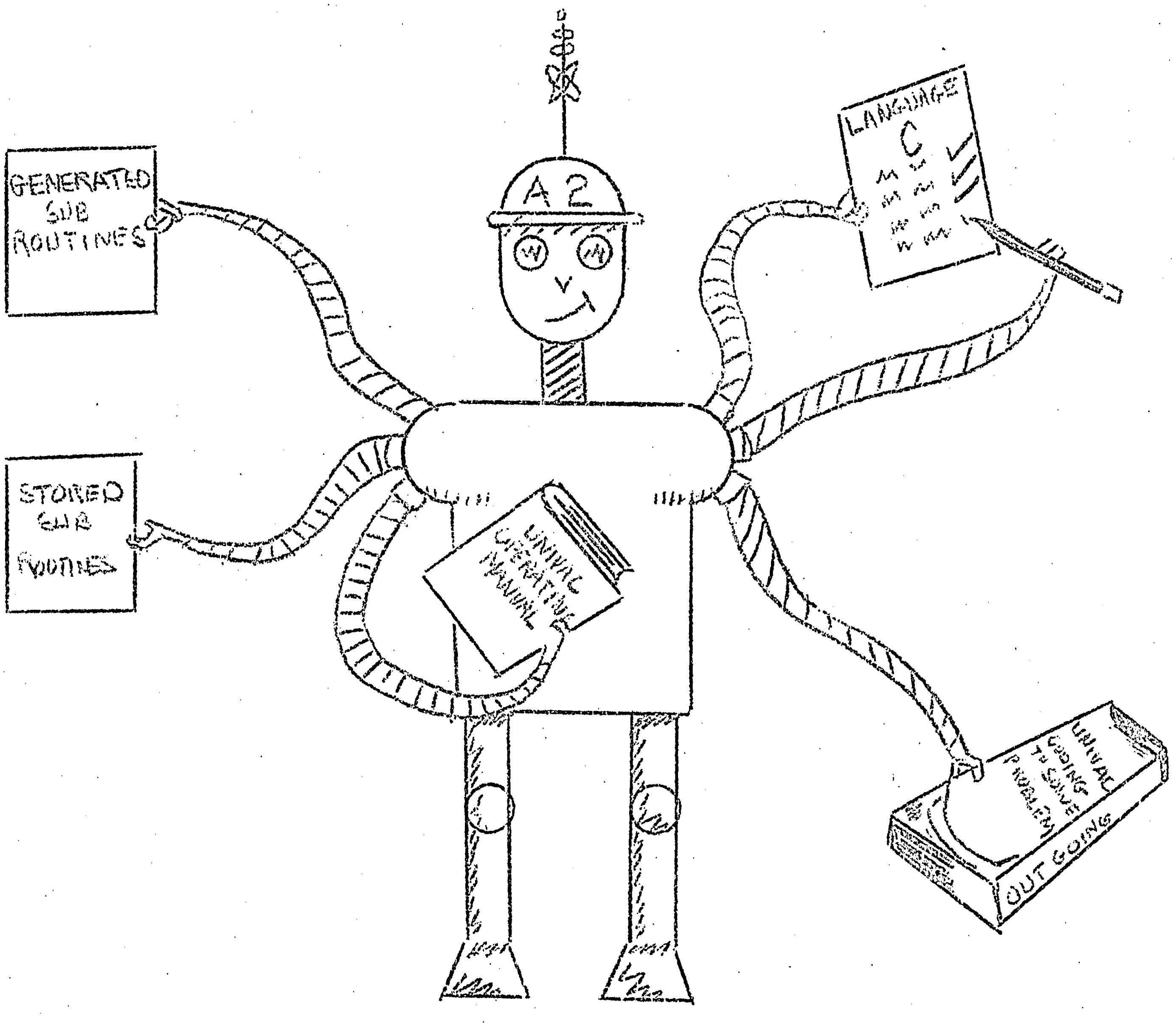

A2
LANGUAGE A
GENERATED SUB ROUTINES
RECORD OF PROBLEM
LIBRARY OF GENERATORS AND SUB ROUTINES
LANGUAGE B SENTINELS ADDED TO A

FIRST SWEEP H-S 10/8/53



SECOND SWEEP

A-S 10/23/63



MAIN COMPILATION

H.S 10/2/53

8. The I.B.M. 701 Speedcoding System

by J.W. Backus

reprinted from Journal of A.C.M., vol. 1, no. 1, (January 1954)

The IBM 701 Speedcoding System is a set of instructions which causes the 701 to behave like a three-address floating point calculator. Let us call this the Speedcoding calculator. In addition to operating in floating point, this Speedcoding calculator has extremely convenient means for getting information into the machine and for printing results; it has an extensive set of operations to make the job of programming as easy as possible. Speedcoding also provides automatic address modification, flexible tracing, convenient use of auxiliary storage, and built-in checking. The system was developed by IBM's New York Scientific Computing Service.

Since this floating point Speedcoding calculator is slower than the 701, despite the conveniences that it offers, you might ask: Why go to a lot of trouble to make an extremely fast calculator like the 701 behave like a slower one? There are two principal reasons. The first is that some computing groups are working against time, and the elapsed time for solving a problem may often be reduced by minimizing the time for programming and checking out the problem even though the running time is thereby increased.

The second and most important reason for having a Speedcoding calculator, in addition to the 701, is a matter of economy. Often, the expense of operating a computing installation is almost equally divided between machine costs and personnel cost. Furthermore, machine time spent checking out problems is frequently a very appreciable percentage of the total machine time.

It is clear, therefore, that programming and testing cost often comprise between 50 and 75% of the total cost of operating a computing installation. Since Speedcoding reduces coding and testing time considerably, and is fairly fast in operation, it will often be the more economical way of solving a problem.

Speedcoding is an interpretive system. I have implied that Speedcoding is a three-address system. Actually this is not quite the case. In a floating point system data and instructions have completely different forms and are treated differently. Therefore, it was thought desirable to have separate methods of dealing with each of these two types of information. Thus, each Speedcoding instruction has two operation codes in it called OP_1 and OP_2. OP_1 has three addresses A, B, and C, associated with it and is always an arithmetic or an input-output operation. OP_2 has one address, D, associated with it and is always a logical operation. OP_1 deals with floating point numbers, OP_2 deals with instructions. This arrangement was also adopted because it makes efficient use of the space available for an instruction and because it often speeds up operation by reducing the number of instructions which must be interpreted.

OP_1 operations consist of the usual arithmetic operations plus square root, sine, arc tangent, exponential, and logarithm. There are also orders for transferring arbitrary blocks of information between electrostatic storage and tapes, drums or printer. These input-output orders have built-in automatic checks for correct transmission. Accompanying the OP_1 operation code is a code to specify that any or all of the three addresses, A, B, C, should be modified during interpretation by the contents of three associated special registers (B tubes) labeled R_A, R_B, R_C. This feature often enables one to reduce the number of instructions in a loop by a factor of 1/2.

The OP_2 operation in an instruction is executed after the OP_1. By means of this operation one can obtain conditional or unconditional transfer of control. One can initialize the contents of any of the R-registers or one can, in one operation, increment any or all of the R-registers <u>and</u> transfer control. Another OP_2 operation allows one to compare the contents of an R-register with the given D-address and skip the next instruction if they are equal. OP_2 also provides a set of operations for using a fixed point accumulator for computations with addresses and for comparing the contents of this accumulator with the D address. Finally, OP_2 provides a convenient means of incorporating checking in a problem if desired. This feature consists mainly of two operations, START CHECK, and END CHECK; all instructions between these two orders may be automatically repeated as a block and at the end of the second repetition two separate check sums which have been accumulated during the two cycles are compared and the instruction following the END CHECK skipped if they agree.

Instructions or data may be stored anywhere in electrostatic or auxiliary storage as single Speedcoding words. Average execution times for various Speedcoding operations are as follows

Add:	4.2 milliseconds
Multiply:	3.5 milliseconds
Read tape:	14 milliseconds access plus 1.6 milliseconds per word
Transfer Control:	.77 milliseconds

Electostatic storage space available is about 700 words.

Let us follow a problem from its coded form on programming sheets and data sheets until it is checked out and ready to run. First the instructions and data are punched on decimal cards whose formats are identical to those of the sheets. If there are any data or instructions which the program requires from tapes or drums, loading control cards are punched (one for each block of information) which will cause the loading system to put this information in the proper places in auxiliary storage. The deck of binary cards for Speedcoding is placed in front of this decimal deck consisting of instruction cards, data cards, and, possibly, loading control cards, and the entire deck is put in the 701 card reader. When the load button is pressed, the information will be stored in electrostatic storage, on tapes or on drums as indicated by locations on the cards. When the last card is read, execution of the program will begin automatically.

In checking out the program, use will be made of a feature of Speedcoding which has not been mentioned yet. Each Speedcoding instruction includes a list code which may be assigned one of three possible values. Associated with each of these values is a switch on the operator's panel of the 701. During execution of a program all instructions will be printed which have list codes corresponding to switches which are on.

If one has properly assigned list codes, one may then check out a problem in the following way: One begins execution of the program with all three switches on, after seeing the most repetitive portions of the program printed once or twice, one of the switches is turned off, after which only moderately repetitive parts of the program are listed. Finally, the second switch is turned off and only the least repetitive instructions are seen. If trouble is encountered in the last cycle of a much repeated loop, one can approach this point rapidly with a minimum of printing and just before reaching it one can turn on all three switches and see all details of the program. Each instruction is printed with alphabetic operation codes just as it was originally written on the programming sheet. The floating point numbers at A, B, and C, the contents of the R-registers and the address accumulator, are also printed with each instruction.

Now that we have briefly seen how Speedcoding works, you might be interested to know what our experience has been with it. At present, I believe that five 701 installations are using or plan to use Speedcoding. Although many improvements and extensions to Speedcoding are possible, those who have used it report that it is actually easy to use. Just this week, in fact, one of our customers arrived at our computing center in New York with a Speedcoding problem all punched and ready to test. He had programmed the entire problem at his own office using only the Speedcoding manual. He got his results with a minimum of help from us.

We have done problems with Speedcoding which ran for 100 hours, and problems which took three minutes.

Experience has shown that many problems which might require two weeks or more to program in 701 language can be programmed in Speedcoding in a few hours. One reason for this is the small number of instructions required. For example, a matrix multiplication program requires about twelve instructions. There are only two instructions in the principal loop.

Once a problem is coded one can often have it punched, checked out on the 701, and ready to run inside of an hour or two.

The speed of operation of Speedcoding makes it an economical system to use. We have solved some problems in Speedcoding for a fraction of the cost on other equipment. Speedcoding is, of course, particularly valuable for small problems and for such problems is often cheaper than 701 language programs, since it reduces the costs of programming and testing.

One other interesting fact which I learned yesterday was that one 701 installation has developed a mechanical prodedure for translating their standard CPC routines into Speedcoding programs and have already used these programs quite a lot.

To summarize:

Speedcoding is a floating point three-address system which greatly simplifies programming, and checking out a program. Speedcoding provides convenient input-output operations, built-in checking, easy loading and printing. Therefore, Speedcoding reduces programming and testing expenses considerably. These expenses are often a large part of the cost of operating a computing installation. Thus Speedcoding is economical as well as convenient to use.

RESUMÉ OF SESSION 8

In answer to a series of questions, John Backus stated that one SpeedCo word consists of 72 bits; available SpeedCo storage is about 700 registers (equivalent to 1400 701 registers); provision is not included for handling symbolic addresses although United Aircraft has added such a provision; OP_1 is carried out first, then OP_2 can be omitted, but something must be written for OP_1 - e.g. NOOP gives no operation. Mr. Ramshaw indicated that SpeedCo would be about twenty times slower than a directly-coded scale-factored routine.

Mr. Backus indicated that he would now prefer a one-address code in SpeedCo. A three-address code was initially adopted since it seemed to correspond to what people were using at the time. However, in practice he has observed that about half of the instructions in SpeedCo programs are effectively one- or two-address instructions. Also it is more difficult to unpackage the three-address pseudo-instructions. W.A. Ramshaw pointed out that in a SpeedCo instruction OP_1 is a three-address but OP_2 is a one-address operation. Mr. Ramshaw also pointed out that by providing the option (by use of a special bit) of allowing the result of an arithmetic operation to be added to instead of replacing the contents of the accumulator, a 20% saving in time could be obtained.

D.L. Shell stated that at G.E. they found it easier to write the "interpreter" or "dispatcher" in the pseudo-code. On the average it takes about 2.5 milliseconds per interpreted instruction - and this includes instructions for square root, transcendentals, etc.

John Backus indicated that a three-address code facilitates the specification of input-output routines. Also the use of R - quantities (B-box) can modify all three addresses at once. He stated that in some cases a set-up combining computing and subroutines may be slower than an interpreter. Among other things, this depends on having input-output speeds that are much slower than the computer operation. This sort of situation usually results in the condition that "saving space saves time". G.M. Hopper stated that at Univac they compile because they can read in as fast as they can execute instructions. Mr. Backus then stated in answering a question asked by D. Combelic that it would be desirable to design a machine to do compiling - but this also depends upon whether it has built-in floating-point arithmetic. D.L. Shell expressed the opinion that this discussion pointed out what he feels to be the chief weakness of the 701 and 1103; viz. that the input-output speeds do not match the machine speeds.

John Backus then described the algebraic coding scheme being developed at I.B.M. He also discussed a logical procedure for assigning storage space to sections of a large routine.

D.L. Shell emphasized the point that in any algebraic coding system, it is desirable to be able to get out a record of what was programmed. This would be difficult in the system suggested by Mr. Backus since many of the characters used do not appear on I.B.M. equipment. Mr. Backus indicated that many of these difficulties can be overcome by using combinations of available characters or by changing some of the keys (e.g. $). Mr. Shell remarked that his group had problems with traditional notation which is not like any of the auto-codes proposed. G.M. Hopper suggested that suitable labels could be placed over the typewriter keys. Jack Jones noted that this would make reading and checking rather difficult.

Dean Arden suggested that the problem of finding the index i for which a sequence $\{x_i\}$ assumes its maximum value would be as difficult to code in an algebraic code as in the more commonly used codes.

9. CHOICE OF NUMERICAL METHOD

The object of this discussion is to consider how processes of automa-coding fit into the general field of calculation by automatic computers, and I shall indicate some cases where efficiency of the coded program is more important than the saving of time spent on coding.

1. An important subdivision of problems is into two categories

(I) Much input-output —— little computing.

(II) Little imput-output —— much computing.

This subdivision (artificial except as an indication of extremes) has considerable effect on the design of machines.

2. I am interested in another subdivision of similar importance (and of similar artificiality) in the design of programs.

(i) Much coding —— little running-time on the machine.

(ii) Little coding —— much running-time on the machine.

3. This whole course has been mainly concerned with methods for simplifying coding - by trying to make it as automatic as possible. There may thus be a loss of efficiency in the final program, to a greater or lesser extent, since the possibilities for human ingenuity are reduced. These methods are thús primarily applicable to problems of type (i), for which little machine time is needed. They may, of course, be applied to problems of the type (ii), but it is more important, particularly with inner loops, to be used very oftem, to get the utmost machine efficiency, even at the expense of considerable time spent on coding (the words 'much' and 'little' are, of course, relative to total problem time).

4. The price for simplicity in coding is very often greater machine time, sometimes by a considerable factor, on the problem concerned. I am thinking mainly of the effect of choice of method, i.e., whether this is simple or sophisticated, but the same applies, with less force, to methods of coding after the method of computation has been chosen.

The desire to reduce coding effort and to produce programs rapidly for machines has led to the saying that,

"Automatic machines prefer simple repetitive methods, in which cycles of orders are used many times."

In this we should replace "automatic machines" by "coders," and it is then clear that this principle enables a few coders to deal with a larger number of problems, and to make good use of machines which might otherwise have been idle.

5. Now, many machines are overloaded and it becomes important to use efficient methods, even though these may involve much more time spent on coding, if machines are to play their proper part. A machine may easily be kept working full time, on a useful problem, but taking 10 or 20 times as long as it might with a better program.

As an example of this we may consider the use of iterative methods of computation. I have seen cases of iterations taking perhaps 15-30 seconds each, repeated over 100 times. This means, say, a half-hour machine run, which may seem reasonable. There are however two points.

(i) A slow convergence, approaching a limit as $\frac{1}{n}$, say, where n is the number of iterations, will ease when the alteration vanishes -- this alteration varies as $\frac{1}{n}^2$ normally, and so vanishing to 10^{-p} means an accuracy of only $10^{-\frac{1}{2}p}$ in the result. Slow iterations can settle down very far from the limit point.

(ii) Standard methods of extrapolating results from 3 or more successive iterates (e.g., the method of Aitken) can reduce time substantially, besides avoiding the difficulties in (i). 100 or 10 Aitken 10 Aitken etc also for divergence

An iterative cycle should be used several times (short ones perhaps more often than long ones). Then the last 3 or 4 iterates should be used to extrapolate a new start. Thus instead of 100 iterates, one might have 10, extrapolate, 10 more, extrapolate - result, saving 75% of the time for a single set of data.

6. We may say, then, that the choice of method now plays a more important part in dealing with a problem on an automatic machine than was formerly the case. The use of library subroutines greatly assists in the use of more sophisticated problems, and expansion of a library is helpful. I cannot catalogue suggested methods, but shall give one or two illustrative problems and describe how they have been tackled.

7. I shall first consider ways of dealing with solutions of differential equations.

First: possible needs

I. A tabulated solution at fairly close intervals.

II. A brief survey of a solution as a whole, or a need for end-values only, without intermediate results.

Some possible methods (there are others) are:

(a) Runge-Kutta type, involving only values at one argument x, and the evaluation of first derivatives only. These usually need relatively small steps h.

(b) Methods involving differences, or function values at several neighbouring points. These can be used with small steps, or with medium steps h somewhat larger than are possible by Runge-Kutta methods.

(c) Methods involving derivatives. These can often be used with very large steps depending on the radius of convergence of Taylor expansions. The difficulty here is in the computation of derivatives; this is, however, clearly possible by suitable recurrence formulae in many cases familiar to mathematicians and physicists, and can, by ingenuity, be made possible in a considerable number of other cases.

Starting the integration is easy in (a) and (c) but may be troublesome in (b). Large steps, as a rule, reduce possible trouble from rounding errors.

8. It is desirable, as a rule, to use an interval in the calculation which shall be equal to, or not much smaller than, the interval of tabulation desired. Thus, generally, the order of preference of method is (c), (b), (a). However, if (c) is impossible or if a small interval is needed in the final table as in case I above and (b) is to be used, then (a) may be used to give the necessary starting values. Thus methods (a) and (b) should both be provided in a library of subroutines.

If case II holds, as large an interval as possible is desirable, and (c) should be used if it can be made to work. It is more difficult to code up (c), as it is much more dependent on the particular equation to be solved.

Recently in Cambridge, England, M. L. Albasiny has coded up a program to deal with Homogeneous Linear Differential Equations of the Second Order with Quadratic Coefficients. This covers a very large number of the functions arising in mathematics and mathematical physics - too many to list now.

9. The method used is to obtain from the differential equation

$$p(x)y'' + g(x)y' + r(x)y = 0$$

by differentiating n times, a 5-term recurrence relation between successive terms $T_n = h^n y^{(n)}/n\,!$ are connected by

$$(n+1)(n+2)p_1 T_{n+2} = -(n+1)(np'+q)hT_{n+1} - (\tfrac{1}{2}n(n-1)p'' + nq'+r)h^2 T_n$$
$$-(\tfrac{1}{2}(n-1)q''+r')h^3 T_{n-1} - r''h^4 T_{n-2}$$

Starting from $y(x_0)$ and $y'(x_0)$ at $x = x_0$, we compute

$$y(x_0+h) = y(x_0) + T_1 + T_2 + T_3 + \text{------}$$

$$T_1(x_0+h) = hy'(x_0+h) = \qquad T_1+2T_2+3T_3+4T_4+\text{---}$$

obtaining y and y' at the next point, and repeat the process for the next step. The cycle for computing T_n is carried on until T_n and T_{n+1} both vanish.

10. The program is not in strict floating binary or floating decimal form, but has automatic scale changes coded in, both binary for computing and decimal for printing. It is slow in action on individual steps, but this is unimportant in view of the size of step that is possible. It has been used to tabulate e^{x^2} at interval 0.5 in x, and retained about seven-figure accuracy to x = 12 to 15; at the end, the function was increasing a million-fold at each step. Steps needing 50 derivatives were successfully used.

I have incidentally used this method successfully on desk machines (hand machines in fact) to tabulate Bessel functions at unit interval in x to 25 decimals, needing 20 to 25 terms. The resutls will appear in due course in one of the Royal Society Mathematical Tables.

Programs for Computing Residue Indices

11. My next example is in Number Theory. It is over 300 years old, and still of interest. It concerns the problem of finding "residue-indices." Consider a prime number P, and a small integer a (the base) It is required to find the least number e such that

$$a^e - 1 \text{ is a multiple of } P$$

This number exists (except when a is a multiple of P) by Fermat's theorem, which states that, for such an a

$$a^{P-1} - 1 \text{ is divisible by } P.$$

The exponent e is obviously not greater than P-1 and it can be shown that $a^e - 1$ is a factor of $a^{P-1} - 1$, and hence that P-1 is a multiple of e, say $P-1 = \nu e$. Then $(P-1)/e = \nu$ is the residue-index we seek.

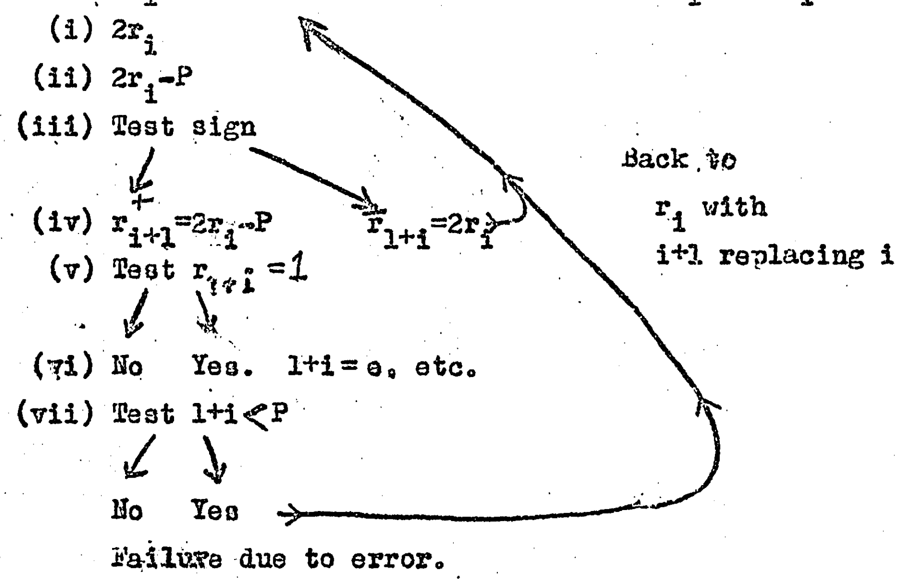
12. We can confine attention to the case a=2, and consider methods that have been used.

A simple repetitive method has been used by Lehmer on ENIAC, by myself on EDSAC, on SEAC, on MOSAIC and by C. Strachey on the Elliott 401 computer. (I also coded it for SWAC, and suggested it in other cases.) It has been used by Gruenberger on a CPC.

Originally intended as a test problem for comparing speeds (it is important to emphasize that one needs several test problems for fair comparison; this problem and method suits some machines well - others very badly - for yet others it is fairly 'neutral.') and one may mention some times.

The method is basically as follows.

Suppose r_i is known, where $a^i = \lambda p + r_i$, $0 < r_i < P$,



Test (vii) may perhaps be omitted - failure is usually clear because of lack of result in an appropriate time. One can cut short at $\frac{1}{2}(P-1)$, since this must be ± 1.

13. Time is evidently basically proportional to $\underline{e}$ and hence to $\underline{P}$, since $\underline{\nu}$ is most often 1 or 2, and rarely large. For P= 9000, times for $\underline{e} \doteq \underline{P}-1$ are approximately:

1949	ENIAC	5 sec.	
1950	SEAC	20 sec.	
1949	EDSAC	3 min.	
	SWAC	$2\frac{1}{2}$ sec.	(estimated)
1953	MOSAIC	$7\frac{1}{2}$ sec.	
1953	"401"	4 sec.	

The interesting values of P only start at 100000 so that it was only when the last one came along that actual production (there are thousands of values of P to be tried) seemed worthwhile and this was set in train. Going to $\frac{1}{2}(P-1)$, for P up to 250,000, the average time was 20-30 seconds; 50-80 hours production was run.

These programmes were all fairly easily made; the last and fastest of them tried involved optimum programming and was some trouble.

14. There is, however, a completely different approach. That is to aim at $\underline{\nu}$ directly:

(a) ν must be a factor of (P-1)

(b) every ~~ν~~ μe which is also a factor of (P-1) must also give $\underline{a}^{\mu e}-1$ a multiple of P. ← $a^{\mu e}-1$ a multiple of P

We have therefore to factorize P-1 and eliminate it from those factors which are not factors of 2ν. This is done by removing them one at a time.

(i) There are comparatively few factors of P-1(order $\log_e P$ or less).

(ii) The trial of separate factors is the basic cycle.

(iii) This cycle is very considerably longer than the previous one, and itself includes a sub-cycle for each binary digit of $\mu\nu$.

(iv) Time is virtually independent of P in ranges of interest 10^5 to 10^6. It is proportional to $(\log_e P)^2$ approximately, and $\log_e P$ varies from 11.6 to about 12.3 .

(v) Time is almost independent of $\underline{\underline{a}}$ (except that a=2 can be specially coded at about twice the speed for $a \geq 3$).

15. This method is one of considerable sophistication and was coded for the ELLIOT 401 by D.B. Gillies. It took about 150 hours of highest grade optimum coding, with intermingling of computing and printing.

The result is, as usual, adoption of a larger production program than was originally considered possible, revision of previous runs in about 10 hours, and a total program covering 300-400 hours at the new speed. This gives masses of results to be regarded as data for further study rather than as material for publication.

J.C.P. MILLER

9. Resumé of Discussion

Since, in solving partial differential equations, a small mesh-size usually slows down convergence as well as increasing the computation per iteration, D. L. Shell reports that G.E. has experimented with extrapolation through several mesh sizes and with a fixed number of iterations. The final number was generally more satisfactory.

Use of very high differences is recommeded by Miller as one approach. He has gone to higher than 20th differences. Machines cannot usually use all the tricks which an experienced human computer can apply, but the machine does not usually get into the same kind of messes that a human computer can get into.

In reply to a question by Arden, Miller admitted that relatively few large-step problems have been done. Highly non-linear problems are of course especially intractable.

A solution at only 3 points that agreed nicely with a more elaborate computation was reported by Shell, but Arden and Miller felt that this was an unusual case, especially since 3 points cannot yield high-order differences.

W. C. Carter made the point that often when a few points are sufficient, a good analytical approximation can also be found. Miller brought out the possibility of polynomial approximations as being frequently practical.

L. Rosenthal commented that comparisons of solutions obtained by several methods may be a practical way of getting results, but that no study seems to have been made of any organized approach along these lines.

Miller replied that there is much we do not know. Treating ordinary differential equations by matrix method has some promise. The converse was suggested, but Miller feels that a nice matrix often corresponds to a horrible differential equations, so that the method only works one way.

The machines have surpassed man in most cases, but not all. Comrie solved 157 simultaneous equations fifteen years ago. However, in the calculation of constants, machines have made man's efforts seem trivial - e.g., Shank's 707 places of π, of which 529 were correct, and the 1000 or more calculated by Wrench have been surpassed by a Fourth of July weekend's work on ENIAC, while Wheeler has calculated e to 60000 places.

J. F. Kelley replied to a question about linear programming by saying that the trend, by people like Hoffman and Charnes, is toward special methods. A modified simplex methods still semms the best general way, excelling iteration techniques except possibly that of Thompkins.

After a long, contemplative silence, the session adjourned.

ME Nov 53
Feed Water
Heating
Card Techique
[illegible] Codes

Babcock and Wilcox
No programming by engineers
form sheets - fixed problems

Check Sum each w/ 407 on
TTY/mic

10. The Effectiveness of Automatic Coding Systems Currently in Use

The design of an automatic coding system itself presents a formidable task of programming and coding, and the completed system represents a considerable capital investment. It is therefore of some importance to weigh carefully the advantages to be gained from such a system before embarking upon its formation. There is little doubt that the majority of existing systems have proved, or at least will prove, to be well worth the coding effort that went into them. On the other hand this has not been true of all the systems. This short note is intended simply to stimulate a discussion of the lessons that have so far been learnt from practical experience, so that we may avoid the pitfalls in future and concentrate on those features that appear to offer the greatest reward.

The design of an automatic coding system differs from other programming tasks in one important respect. The success of most programs may be judged objectively, from such things as the freedom from error and the machine time required. These factors enter into an automatic coding system also, but in this case the ultimate test is a subjective one: is the system useful to the people for whom it is intended? The designer must therefore be not only something of a mathematician but also something of a psychologist, and his success cannot be judged from a few machine tests but only after the system has been used by several people over a period of some months.

Here are some of the questions that can only be answered on the basis of experience:

How long does it take a coder to become familiar with the pseudo-code?

How important is the provision of exhaustive mistake diagnostic facilities?

How easy is it to design a system so that future additions may be made without affecting the coder who does not wish to use them?

How important is a convenient notation, compared with the provision of more fundamental things such as symbolic addresses?

How useful is a mnemonic notation?

Ramshaw 100 people
- [illegible]
[illegible]
Seven Lectures 15 hrs.
double precision Speedco
Relative addresses

ERA 1103 2 Addr System
[illegible]
int OP
[illegible] Normal

GE Add GEPURS
Sub
Mul
DIV
IN-OUT
Take a function
Stop

Dual
85% other [illegible]
15% Dual [illegible] 25%
5% Shared [illegible] 10%

The LMO Edit Compiler

Introduction

The LMO Edit Compiler is a routine which edits raw data tapes into any form. The process of editing a unit of raw data is reduced to making up the format of the printed page using Uniprinter symbols and specifying by "pseudoinstructions" which digits of the raw data are to be transferred to which positions in the page format, called the "matrix."

From the pseudoinstructions, the LMO Edit Compiler forms a set of instructions which will edit as many units of the raw data as desired. This set of instructions is called the "running instructions," since it is read consecutively a block at a time. It and the matrix are then used for the actual editing of the raw data.

This arrangement saves programming time since making up pseudoinstructions is a relatively simple process compared to coding C-10 Univac instructions. It also minimizes bugshooting time. Errors are easily detected, either through the self-checking features of the routine, or by a quick glance at the output, and it is a simple matter to make the needed corrections in the pseudoinstructions.

The Matrix

The first step in the use of the Edit Compiler is to decide what the printed page is to look like and to make up the matrix with this in mind. The matrix will consist of 600 or fewer Univac words containing:

1. The title and other words to be printed on every page.
2. The commands to the Uniprinter, such as carriage returns, tabs, shifts, ignores, spaces, etc. (It is usually a good idea to begin each page with a printer breakpoint followed by a carriage return so that manual adjustments can be made before the printing begins.)
3. Places for the words and digits from the raw data to be inserted. There are a number of considerations governing the choice of these "holes in the matrix, two of which are:
   (a) Since one pseudoinstruction affects just one matrix word, it is usually wise to restrict a hole in the matrix to a single matrix word.
   (b) If a decimal point is desired, it should be put in the matrix in the correct position.

4. The word "Page" followed by a space and a four-consecutive-digit place for the number in the appropriate place in the matrix if the pages are to numbered.
5. Own special subroutines which are to be used. These should be made up at this time in accordance with the section Matrix Subroutines.

Note: The matrix should be thoroughly checked before the pseudoinstructions are made up, so that errors discovered in the matrix will not necessitate changes in the pseudoinstructions.

<u>The Pseudoinstructions</u>

After the matrix has been made up, the next step is to specify by pseudoinstructions the digits with which the "holes" in the matrix are to be filled.

Writing the pseudoinstructions is simply a matter of translating the wishes of the programmer into the forms described below and putting them down on paper. The General Pseudoinstruction specifies the word and digits of the raw data in the left half, and the word and digit location in the matrix in the right half. Certain other types of pseudoinstructions are used to permit the routine greater flexibility.

<u>The General Pseudoinstruction</u> is represented by XnLLDD Qllldd, which says, "Take n digits, beginning with digit DD, in word LL of the block being edited, act on them with subroutine Q, and put them in line lll of the matrix, beginning with digit dd." One block of raw data at a time is edited and changed by the indicator "X", which says either "Read in next block of raw data", or "Do not read", depending on whether X is R or 0 (zero). Does Not Apply to EL110- Repeater Routine

<u>Form of Instructions</u>

The pseudoinstructions can be introduced into the UNIVAC manually (Breakpoint 6) or by the use of a control tape (Servo No. 3). The instruction word is of the form

XXXYYYiiiCnn

<u>XXX</u>	is the first word location of an item to be modified or copied
<u>YYY</u>	is the last word location of an item to be modified or copied
<u>iii</u>	is the location of the one word in an item which is to be modified. Note the control letters D and E.
<u>nn</u>	is the number of times an item is to be copied or modified
<u>C</u>	is the control letter, it can be: (A) This will copy an item on the output tape nn times in succession. iii is not used in this instruction.

(B) This will add a constant to each word in an item and copy the altered item nn times on the output tape. <u>NOTE</u>: The word following a B, C, D, or E, command must be the constant which is to be added or accumulated. iii is not used in this instruction.

(C) This will transfer an item to the output tape nn times, adding accumulated constant to all words each time. iii is not used in this instruction.

(D) This will add a constant to iii and copy the item nn times on output tape.

(E) This will transfer an item to the output tape nn times, adding an accumulated constant to iii each time.

10. Resume of Discussion

Effectiveness of Automatic Coding Systems Currently In Use

The first comment was in the form of a question by K. F. Powell: What is compiling anyway? Is it interpreting, calling in subroutines--is buffer storage needed--is it for slow machines only, etc? Seems to be a matter of definition. One aspect: Output of a compiler is a sequence of instructions, not numerical results (by H. F. Hunter). Further clarification of the definition deferred until the discussion of the ACM glossary.

Comments by W. Ramshaw, United Aircraft:

SpeedCo has been used at United Aircraft since last October by a group of about 100 "outsiders."

An outsider with no previous experience on digital computers can usually learn the system and carry through his first problem (if it isn't too big) in about 15 hours, of which about 12 hours is devoted to a series of lectures on SpeedCo - the remaining 3 being spent in debugging his first problem.

Exhaustive mistake diagnosis of the SS type is not provided by SpeedCo. This omission does not appear to be too serious, however, since common coding errors such as multiplying by an instruction usually produce characteristic SpeedCo malfunctions, which the operator can identify merely by reading the indicator lights on the panel.

SpeedCo turns out to be so written that it is very easy to modify without invalidating previous programs. We feel that this is one of its strong points.

Relative addressing is used. Admittedly this system is not as attractive as floating addresssing but in view of the small number of words of storage available to the programmer, it appears to be an adequate means of avoiding the pitfalls of using absolute addresses.

The operation codes now in use are alphabetic and mnemonic. This appears to be a very worthwhile feature. The older numeric operation codes were evidentally hard to remember and this frequently leads to wasted 701 time.

The professional coders use SpeedCo on an IN-OUT basis. Including such problems, SpeedCo usage constitutes about 95% of all useful machine time.

Comments by Bryan D. Smith, ERA Div., Rem.Rand:

A system of coding intermediate between coding in machine instructions and completely interpretive coding is available for use on the ERA 1103. It was felt that completely interpretive systems such as SpeedCode were primarily for convenience in coding floating point.

A new instruction, Interpret, has been incorporated into the 1103 instruction repetoire. The machine utilitzes only the left six bits of the instruction word and upon the occurence of the Interpret Code, stores the contents of the program address counter (return address) at storage address zero. It then jumps to address one for its next instruction. By this mechanism the Interpret Code causes control to be transferred to an interpretive mode of operation. The remaining 30 bits of the Interpret instruction are available for interpretive coding.

For floating point operations, the format of the interpretive coding is the same as the machine instructions with the restriction that operand addresses refer only to the high speed store (12 bits). The floating point operations are functionally the same as the machine fixed point operations.

Incorporated with the Interpretive floating point operations is an interpretive Repeat instruction so that the advantages of repeated arithmetic operations used in fixed point are carried over to the floating point operations.

This system appears to be quite feasible on a machine such as the 1103 which has an extensive instruction repertoire (43 two address instructions including seven jump instructions). Logical Operations, address modifications, etc., are carried out at machine speed by machine instructions, and those operations that need to be interpretive are called in automatically.

One additional feature is a routine which scans the coding to be interpreted and then brings into high-speed store only those sections of the interpreter needed to interpret the routine just scanned.

<u>Comments by D. L. Shell</u>, Interpretive Routines at G.E. (AGT)

I. <u>GEPURS</u>.

This is a completely interpretive three address, floating decimal system. It was designed to be used by a novice on a more or less casual basis. No provision was made for any sort of floating or regional addressing or for mnemonic coding. Facility was provided for complete tracing of the coded operations as they are executed by the machine. The GEPURS instruction code comprises seven 3-address instructions: add, subtract, multiply, divide, stop, input-output, take a function.

The system has worked successfully for its intended purpose. However, people tried to use it for problems much larger than it was designed for and naturally found it too rigid to be convenient. It is now used for about 10% of our production.

II. SEESAW (Originated at Los Alamos)

This is a single address, floating decimal interpretive routine with a word structure identical to that of the basic machine (701). It provides a convenient method of switching control back and forth between the real and "abstract" machines, i.e., going IN and OUT.

Relative addressing is used in practically all the coding for this system. The pre-execution assembly techniques used for straight machine code are also applicable here. Seesaw interpreter uses about 1200 701 half-words, about half of which are required for tracing.

Provisions are made for either continuous or selective tracing of program execution, which, together with post-mortem storage printouts, constitute the major mistake diagnosis procedures.

The arithmetical operation code is identical to straight machine code, making for easy use by the "professional" group.

This system is now used for about 75% of our production. However we expect that the advent of a built-in floating point machine will practically obviate the need for this type of system.

Comments by H. A. Voorhees, Los Alamos:

The first 701 interpretive system developed at Los Alamos was called SHACO. It was essentially a simulation of a CPC on a 701. This was done primarily to mollify and coddle the hard-to-hire physicists who were used to using the 6 CPC's at Los Alamos. SHACO was pretty much a flop: the physicists by-passed it for at least 90% of their coding, preferring to use machine code even with the attendant scale-factoring difficulties, and now SHACO is not used at all.

SHACO was too slow-- each interpretive instruction was done in tandem for checking purposes --the speed factor being about 40 to 1.

Another system DUAL, has been developed. Its use increased to a maximum of 23% and has now leveled off to about 15%. It is used for difficult, unscalable problems and for one-shot, lazy-coded routines. DUAL emphasizes the IN-OUT feature found so useful in other interpretive systems.

DUAL coding is not mnemonic, so that coding is not fast; in fact, an expert scaler seems to code almost as fast using machine code. Los Alamos runs a wide open shop, which yields a varied cross-section of coders.

Comments from James H. Brown, Univ. of Michigan:

The MIDAC computer has the following characteristics:

3-address

512 words high-speed storage

6144 words magnetic drum storage

44 bits per word plus sign

Base counter related to instruction counter

MIDAC is quite short on high-speed storage, so that interpretive techniques are not suitable at present. Instead we use translation routines which translate directly into machine coding on a 1 to 1 correspondence basis. The coder uses:

Floating addresses, 2 letter

Mnemonic operation symbols

3address instructions

Numbers are written in standardized decimal notation including any binary scale factor. Numbers may be translated to equivalent binary or into standardized floating binary. There is also provision for inserting words in pure binary.

Both passes are made from paper tape via Ferranti input. The output of the translator is a punched paper tape ready for insertion for normal computation.

The translation routine is designed to permit programmers to code using full machine capabilities. They may also use a floating point interpretive routine, which is not available automatically. This routine permits IN-OUT type of operation. Coding appears the same as for normal code. There is also an interpretive complex number routine available.

Subroutines in the library have a standardized form in which the first word is exit, the second word is entrance, and arguments are stored in fixed location in storage. Subroutines are relative-address coded--relative to instruction counter. This permits orientation of subroutines in any part of storage without recoding.

Trouble-shooting facilities include a changed-word post-mortem and an automatic routine.

Plans for future expansion call for integration of many of these facilities into one overall system called MAGIC, Michigan Automatic General Integrated Computation. MAGIC will include an extensive translator and compiler, and extensive automonitor facilities.

The MASIAC computer has been developed for use during the Michigan Summer course on Digital Computers. MASIAC is a completely interpretive 3 address computer featuring floating point aritmetic, mnemonic instruction code, symbolic addresses, alpha-decimal input. It has 7 base counters available. Mistake diagnosis facilities are adequate but not elaborate.

<u>Comments by K. F. Powell, Babcock and Wilcox Co.:</u>

A compiler and assembler similar to one developed for a 701 by General Electric's Steam Turbine Group at Lynn, Mass., is being developed by Babcock and Wilcox. The object is to relieve all engineers of coding. All the engineer is asked to do is to fill out one or more of several forms. For example, one form is used for a "Tube Bank" Calculation. The engineer fills in the form with information which he would ordinarily have written down in any case. Some difficulty occurs when the fluid flow in the boiler being designed is interlocked in such a way that the "heat balance" at one point is affected by something which is occurring farther along the path of the fluid flow. This is solved by deferring the "heat balancing" and typing of results until all the feedback effects have been taken into account.

These forms are then typed and they comprise a routine for running on a 701. Keller, at Lynn, has gone somewhat further by then searching a certain tape unit for the subroutines needed for the indicated calculations and laying them all out serially on anohter tape unit ready for execution.

Babcock and Wilcox feel that this automatic coding system is justified for two reasons: 1) an engineer's time is too valuable to spend in coding, 2) their work is highly specialized (boiler design), and nearly all the 701 calculations are concerned with this particular problem.

Dr. Hopper commented that Mr Powell's remarks seem to indicate a trend toward the kind of coding approach which will become more common in commercial applications.

Mr. Powell described in some detail the cards (counterpart of Powell's forms) used by Keller. This technique is described in the copy of the Washington Navy Meeting in May, 1954 (see Bibliography ref. C13).

Ramshaw, Powell and Shell discussed possibilities of having a computer synthesize a design. Shell made the point that one difficult aspect is finding out just what thought processes go on in the engineer's mind when he designs anything. There seemed to be general agreement that extracting this information from the engineer may be nearly as difficult as it would be embarassing.

13. The Algebraic Coding System of Laning and Zierler

I. Introduction

The Automatic Coding System to be described was developed in 1952 and 1953 by J. H. Laning and W. Zierler of the M.I.T. Instrumentation Laboratory. The translation and interpretation of the algebraic coding is realized in the M.I.T. Whirlwind Computer.

The coder specifies his problem as a series of mathematical equations and other special symbols. From this manuscript a Flexo tape is punched, which constitutes the input to the computer. The Flexo characters are translated into Whirlwind instructions which are then carried out. The results requested by the programmer are presented in typewritten form. All arithmetic operations are carried out in floating point using a so-called (24, 6) system. Numbers may range up to 10^{19} with a precision of about 7.2 decimal digits.

II. Basic Operations

All of the lower case letters may be used as variables, and equations of the following form are used.

$a = 5,$

$y = -6.3a,$

$b = 0.0053(a - y)/2ay,$

$n = n + 2,$

$w = -w,$

$x = a(b + c(d - e)),$

$Z = r + 2s/t + u,$

A comma terminates all equations. Plus and minus signs, slashes and parentheses, have their usual mathematical significance. No more than 4 parentheses may be open at any one time. Plus and minus signs separate terms so that the last equation above is the same as

$Z = r + (2s/t) + u,$

Exponents: Upper case numbers on the MIT Flexowriters appear as exponents; there is also an upper case minus sign, but no upper case plus sign. The following are interpreted correctly.

$a = 5^2,$

$b = (a - 2)^{-2},$

$c = (a + b)^2/a^{-3},$ must be intergers

III. Output

The current value of any number of letter variables may be recorded in Flexo code on magnetic tape for later printing by inserting the word PRINT followed by the desired letters, followed by a period. The first and last characters recorded by each PRINT instruction are carriage returns.

IV. Jump Instructions

Equations are ordinarily carried out in the sequence in which they are written. This sequence may be interrupted by inserting one of four jump instructions. The address section of a jump instruction must be an integer less than 100. This integer is the number of the equation to which the jump is to be made. An equation is numbered by preceding it by its number, e.g., $15x = 3a$, assigns the number 15 to the equation $x = 3a$.

The instruction SP 15, inserted in a routine will cause equation 15 to be executed next, the normal sequence continuing from that new point.

The instruction SR 15, causes equation 15 to be executed next, but then control returns to the equation following the SR 15. SR evidently implies that a closed subroutine is to be executed.

The instructions CP and CR are obeyed only if the quantity most recently computed was negative; otherwise they are ignored.

V. Function Subroutines

23 Function Subroutines are now available in the Laning-Zierler system. Each is assigned a number (1-23) and is called for by writing the proper number as the exponent of an upper case F. For example:

$x = 2(F^1(y)F^2(Z) + F^{11}(Z))$,

sets $x = 2(\sqrt{y}\ \sin Z + |Z|)$, since

Subroutine 1 is the square root, 2 is the sine, and number 11 is a subroutine which produces the magnitude of the indicated argument. Other subroutines include inverse trigonometric functions, exponentials and hyperbolic functions, logs to the base 10, 2, and e, etc.

VI. Additional variable and variable indicis

Subscripts are not available on the MIT Flexowriters. Numerical subscripts are obtained by typing $x|^3$ for x_3, etc. Variable subscripts are obtained by using letters after the vertical bar, e.g., x/n is typed for x_n, and if n happens to equal 3, x/n is equivalent to $x/^3$.

VII. Auxiliary Storage

There is room in the Whirlwind high-speed storage for about 250 variables. Additional values, such as tables, must be assigned to the Whirlwind auxiliary magnetic drum. The word ASSIGN is used for this purpose. For example, the instruction,

ASSIGN $a/^4$

automatically reserves space on the drum for variables $a/^1$ through $a/^4$. If these variables have the values 2, 4, 6, 8, respectively, they may be assigned these values <u>and</u> have space reserved for them on the drum by writing only

a/N = 2, 4, 6, 8

Further, the same thing can be accomplished by writing

a/N = 2(2)8,

A more complicated example might be

g/N = 1(.5)2(.25)2.5(1)4.5,

VIII. Differential Equations

Provision has been made for the automatic solution of ordinary differential equations using Gills' variation of the 4th order Runge-Kutta Method. For this purpose

1. the letter t must be used for the independent variable,
2. h must be used for the increment in t,
3. D must be used to denote d/dt,
4. Any other variables and/or superscript variables may be used for the dependent and auxiliary variables.

Suppose we have the two equations

$$f_1'(t, y_1, y_2) = y_2 + 1$$

$$f_2'(t, y_1, y_2) = -y_1$$

Dr Lanning – Request posted limit.

let t = 2(0.5)10, that is, h = 0.5,

Our program might be:

$t = 2,$

$h = 0.5,$

$y/^1 = 0,$

$y/^2 = 0,$

1 $Dy/^1 = y/^2 + 1,$

$Dy/^2 = -y/^1,$

$k = t - 10.1$

CP 1

STOP

t is automatically increased by h upon completion of the last equation that starts with D. One important restriction is that all relevant auxiliary computation must be done between the first and last D equations.

IX. Post-Mortems

Automatic post-mortem features are still in the design stage. Features now available include:

1. If the program is too long the computer stops and types out information indicating where and how storage was exceeded.

2. If an alarm occurs the computer prints out the number of the equation in which the alarm occured and the number of the equation which preceded the alarm. (Equations not assigned numbers by the programmer are automatically assigned numbers from 101 to 200)

3. The programmer may obtain the values of any variables he desires after an alarm by writing the appropriate PRINT instruction as equation 100.

X. Conclusion

The system described is a working system and has been used by the Instrumentation Laboratory to solve several complex problems. One problem involved a set of six simultaneous differential equations. The equations involved extremely complicated algebraic and trigonometric calculations. Coding required only a few hours and the routine ran successfully the first time. Computing time was about six to eight times as long as it would have been using the single-address interpretive system ordinarily used at the Digital Computer Laboratory.

Much work remains to be done on the system particularly with repect to increasing the computing speed and improving post-mortem facilities. Heretofore there has been very little need for elaborate post-mortem facilities, because, with a single exception, all programs coded in the Laning-Zierler system have been completed successfully on their first run.

13. Resume of Discussion

K.F. Powell asked why functions such as cosine were not written in the usual form instead of using the letter F. Donn Combelic said that in simple cases this would be possible, but there might be snags in some cases. Prof. Adams said that Laning and Zierler were tired of adding ingenuities when they reached this point. H.F. Hunter pointed out that the simple standard notation used was more readily extended to include further functions.

W.A. Ramshaw asked whether implicit equations could be handled. Prof. Adams said that such equations requiring iteration must be avoided. D. Combelic explained that in Laning and Zierler's system an equation of the form y = f(y) caused y to be replaced by the function value.

Then ensued a discussion on the use of indices, multiple indices, and indices of indices. Donn Combelic said that the notation is limited by the capabilities of the Flexowriter, but that all these can be handled in some fashion. D.N1 Arden pointed out that the effect of multiple indices can be achieved by multiplication and addition to form a single index. Dr. Miller asked whether an index must be an exact integer; Combelic replied that indices are rounded off to the nearest integer, and that this is useful when one has to interpolate. Replying to H.F. Hunter, he said that interpolation must be programmed by the user. J.W. Backus emphasised that the rounding off of indices is done during execution, not during the reading of the program.

Backus stated that equations are stored individually on the drum in CS form. If post mortem print requirements are written as equation 100, it is merely necessary to call this equation when the program dies. Donn Combelic said that post mortems very rarely had been necessary.

Concerning the system as a whole, Combelic said it had taken about a year and a half to develop, and had not been finished thoroughly. There are no annotated copies of the conversion routine, and when the next algebraic coding scheme is developed it will be done from scratch. E.A. Voorhees asked whether this meant that good ideas would be wasted; this led to a discussion on the utility of one programmer's work to another. Dr. Hopper said that one could trade flow charts, but not coding; Prof. Adams said that a simple flow chart was obvious to anyone and a detailed flow chart too specific to be traded, and added that he didn't draw them anyway. H.F. Hunter asked what improvements might be made over the Laning and Zierler system; Donn Combelic replied that it would be better if more symbols were available, such as integration signs.

Automatization of Processes

1. This lecture concerns coding of some algebraic processes for automatic machines. This includes differentiation of elementary functions, where the change is essentially algebraic in character, after being defined for each function.

Usually included are x^n, e^x, $\ln x$, $\sin x$, $\cos x$ and sometimes $Ei(x)$, $Si(x)$, $Ci(x)$, Fresnel integrals, error integral, etc.

It is not essentially more difficult to include Bessel Functions, Parabolic cylinder functions, etc., satisfying 2nd order linear differential equations.

2. Kahrimanian (C_2, F_1) and Nolan have considered the evaluation of derivatives of combinations of elementary functions, and of functions of functions. Reports are available and will not now be described. They appear to aim at presentation of the derivatives in terms of elementary functions exclusively.

3. I would like to propose a different approach. This implies

(i) That the derivatives are needed mainly to compute numerical values

(ii) That <u>all</u> derivatives to a given order are needed, or may as well be found and <u>used</u>.

The essence is to express each derivative in terms of previously known functions, which may be elementary functions, or previously computed derivatives.

I find it easier to consider in terms of normalized power series

$$A = 1 + a_1 t + a_2 t^2 + a_3 t^3 + \ldots.$$

We need routines to find AB, A/B, A^n, $\sin A$, $\ln A$, e^A, etc.

Consider A^n as an example, with a_1, a_2, ... supposed known

Let $B \equiv A^n = 1 + b_1 t + b_2 t^2 + \ldots$

then we find $AB' = nBA'$

whence $\underline{rb_r = rna_r + (\overline{r-1}\,n-1)\,a_{r-1}b_1 + (\overline{r-2}\,n-2)\,a_{r-2}b_2 + \ldots + (n-\overline{r-1})a_1 b_{r-1}}$

ideal for a kind of 'scalar product" evaluation.

I could expand this technique considerably. It works well with $B = e^A$, $B = e^{iA}$ or $C + iD = \cos A + \sin A$, the latter with $2CD = \sin 2A = D(2A)$ as a check, $C^2 + D^2 = 1$.

4. J. L. Turner in Cambridge has designed and coded a program for differentiating a long series of terms each of one or the other of the forms

$$Ce^{-x}x^a y^b z^c u^d v^e \quad \text{or} \quad CEi(-x)\, x^a y^b z^c u^d v^e$$

in several variables

Differentiation involves

(i) recurrence relations to identify new terms for each old one

(ii) listing terms in order, and combining with previous similar terms

(iii) arrangements to allow for limited storage capacity

A large amount of work has been done with this program.

5. C. B. Haselgrove has made two or three programs for dealing with problems in Group Theory.

Given relations between operations, like

$$AAA \equiv I, \; BB \equiv I, \; ABAB \equiv I$$

the program first seeks a comprehensive list of similar relations such as

$$BABA \equiv I, \; AAB \equiv BA, \text{ etc.}$$

derived from the original set.

It then examines all one, two, three letter combinations to see if it can reduce them. It retains the irreducible ones, here 6. I, A, B, AA, AB, BA.

The program is slow, but it's as fast as an expert group-theorist on the job. It uses an interpretive routine, where each "instruction" involves a very large amount of work.

6. Dissatisfied with this program, ingenious as it is, Haselgrove has developed another, in which whole sets of elements of a group are treated as a unit. This is his 'method of cosets' and is much faster and should eventually yield interesting results.

J. C. P. Miller

Resume of Session #14

D. Arden commented that Haselgrove's work has practical applications in crystallography.

In answer to a question, G. Hopper commented that the differentiator developed by H. Kahrimanian for Univac gives only the derivatives requested and that intermediate derivatives are not necessarily calculated.

D. L. Rosenthal inquired about the convergence of power series obtained from differential equations and Dr. Miller replied that this is not likely to be a problem, and that the solution would get out of control if this was the case.

H. F. Hunter asked whether Kahrimanian's differentiator simplified its result algebraically by cancellation or collection of terms. G. Hopper replied that this was not attempted except for the most obvious cases, as the result was intended primarily for further calculation on Univac.

In connection with his comment concerning the development of a recurrence formula for the coefficients of the series representation of a function of another series from the differential equation satisfied by the function, Dr. Miller gave as an example

$$B = \sum_0^{\infty} b_i x^i = \tan A \quad \text{where } A = \sum_0^{\infty} a_i x^i$$

The b_i can be found by observing that

$$B' = \sec^2 A \cdot A' = (1 + \tan^2 A)A' = (1 + B^2)A'$$

Therefore

$$\sum_0^{\infty} (i+1) b_{i+1} x^i = \left(1 + \left[\sum_0^{\infty} b_i x^i\right]^2\right) \sum_0^{\infty} (i+1) a_{i+1} x^i$$

$$= \left[1 + \sum_{r=0}^{\infty} \sum_{i=0}^{r} \left(b_{r-i} b_i\right) x^r\right] \sum_0^{\infty} (i+1) a_{i+1} x^i$$

$$\sum_{0}^{\infty}(p+1)b_{p+1}x^{p} = \sum_{0}^{\infty}(p+1)a_{p+1}x^{p} + \sum_{p=0}^{\infty}\sum_{r=0}^{p}\sum_{i=0}^{r} b_{r-i}b_{i}(p-r+1)a_{p-r+1}x^{p}$$

and

$$(p+1)b_{p+1} = (p+1)a_{p+1} + \sum_{r=0}^{p}(p-r+1)a_{p-r+1}\sum_{i=0}^{r} b_{r-i}b_{i}$$

giving b_{p+1} in terms of a_i, b_i $i \leq p$.

Another example is

$$u = e^{-X^2}\sin\frac{X}{2}$$

for which a differential equation can be derived by defining

$$V = e^{-X^2}\cos\frac{X}{2} \text{ and therefore}$$

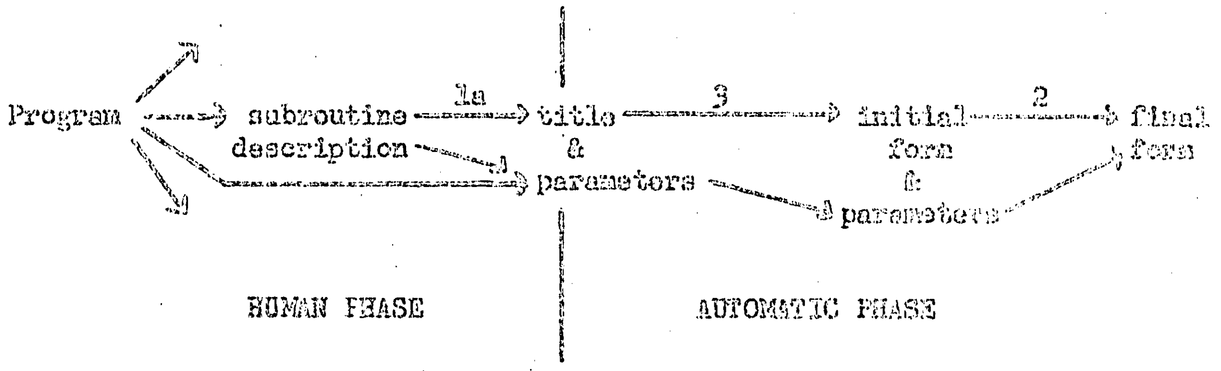
$$w = u+iV = e^{-X^2}\left(\cos\frac{X}{2} + i\sin\frac{X}{2}\right) = e^{-X^2}e^{i\frac{X}{2}} = e^{-X^2+i\frac{X}{2}}$$

$\frac{dw}{dx} = \left(-2X+\frac{i}{2}\right)w$. The real part of the resulting series would then represent u.

Dr. Miller and S. Gill mentioned that routines are being written at Edsac and elsewhere for economizing power series using Tschebycheff polynomials. That is, rewriting a partial sum of the series as a sum of tschebycheff polynomials and neglecting as many of the high order polynomials as is feasible.

15. The Library of Subroutines

The use of a library subroutine may be pictured thus:



The steps may be taken at various stages in the development of the whole routine depending on circumstances. The type 3 step may be made before the routine reaches the machine, in which case it is probably not fully automatic.

The library itself consists of two parts which are used in succession: the list of written descriptions, and the collection of sets of instructions for the machine. The net effect is primarily a change of type 1a. This is only effective if the descriptions are very clearly written; unfortunately only one coder in ten can do this.

It is important that the library catalog, or list of descriptions, be kept tidy and up-to-date. This is no small task. The Cambridge University catalog is in two parts, one giving only the information required by the regular users, the other giving the instructions in full and explaining the methods of operation of the various subroutines. Subroutines are denoted by a letter defining the class, and a number within that class. The classes fall into six main groups: arithmetic, simple functions, operations on functions, matrix operations, input and output, and mistake diagnosis.

It is questionable whether a library should be allowed to grow merely by including new subroutines as they happen to be written. The library should assist coders in preparing routines without delay; their needs should be anticipated at least to some degree. Subroutines written as they are required tend to be too specialized to be useful in a library, and do not cover the field efficiently.

There are many ways of writing a subroutine to do a given job, and in some cases the library should contain a selection. The following characteristics are desirable in a library subroutine:

(1) Compactness, particularly of the final form.

(2) Speed of execution.

(3) Numerical precision.

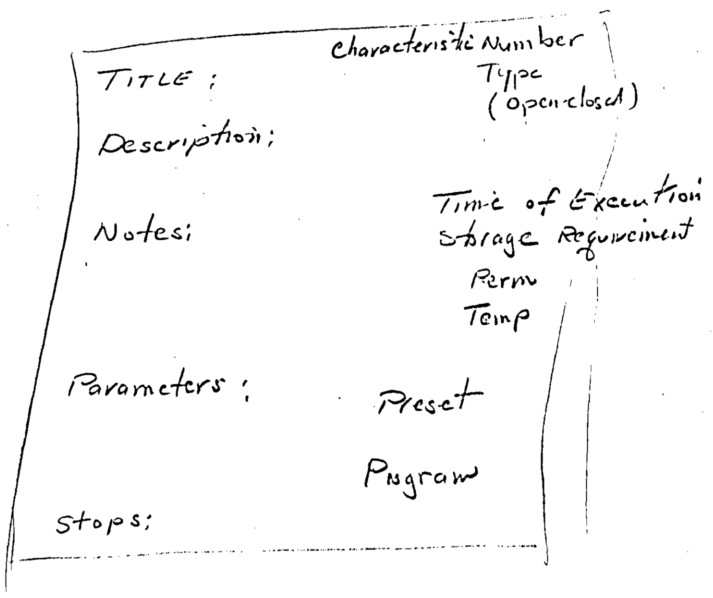
(4) Simplicity of use.

(5) Generality of application.

These are often in conflict, and a compromise has to be made. Different subroutines are arrived at by weighting these characteristics differently.

Two versions of all subroutines { minimum time / minimum storage

{ Page No __ of __ / Date
who is responsible for it



TITLE ; Characteristic Number
Type (open-closed)

Description ;

Notes; Time of Execution
Storage Requirement
Perm
Temp

Parameters ; Preset
Program

Stops;

important numbers on right hand side of page

Leave out unnecessary details for working book

if Necessary leave additional information for recording in a different place

characteristic letter and sequence number

Main - classes

A - Programmed Arithmetic

B - Simple Functions - Functions of 1 variable

C - operations on Functions - interpretative - tables
zero
Quadrature
Solution of Differential Equations

D - operations on matricies and vectors

F - Input and output routines

G - Mistake Diagnosis

E - Number Theory

Resume of Session #15

W. Ramshaw remarked that one difficulty with haphazard collection of subroutines is they are very often written in such a form as to make subsequent modification by someone other than the author difficult. J. Backus mentioned that use of an algebraic pseudocode had the advantage of requiring the use of standard notation. E. Voorhees emphasized that the user should be allowed to improve the subroutine and S. Gill mentioned that EDSAC's subroutines were circulated among staff in order to encourage comment.

C. Adams favored the development of subroutines only as the need for them occurred as it was difficult to predict just what mathematical routines will be needed. Subroutines written by users of Whirlwind are generally modified previous to inclusion in the library. The demand for input-output and arithmetic routines is usually predictable up to a point.

D. Shell mentioned that ad hoc subroutines could be written at customer expense by a skilled programmer in a form suitable for inclusion in a library. However, development of a certain type of subroutine can also be used to stimulate machine use in that area.

W. Ramshaw mentioned that getting rid of old subroutines is a problem. J.C.P. Miller said that in a forthcoming library description at Cambridge, the subroutines would be listed as current or obsolescent. He also advocated that deficiencies in a library be remedied by an expert who would fill as many gaps as possible, rather than by a programmer whose interests might be narrower. In general programmers seemed to like what was available and seldom criticized existing subroutines.

D. Combelic advised keeping a fast version and a short version of some subroutines.

G. Hopper commented that when attempting to penalize poor programming by requiring that the erring programmer write some required subroutines, it was found that some programmers used the compiler to write the subroutines and worried very little about the penalty.

J. Backus advocated letting programmer fill in some parts of subroutines which are subject to considerable variation in individual preference.

In response a question about the descriptions automatically written by compilers and generated, G. Hopper said that they usually included length and temporary storage required but not the operating time.

The notion of temporary storage was clarified by a short discussion.

C. Adams advocated that "unset" values of parameters by the ones most commonly used.

Resumé of Discussion

Mr. J. R. Belford opened the discussion by giving some facts about the IBM 704. Transfers between the drum and the high speed memory will be at 10,000 words per second as against 300 in the IBM 701. Floating point arithmetic will be built in, and there will be a means of automatically and rapidly searching a table in the store to find a given argument. There will be three index registers. Answering Donn Combelic, John Backus stated that the reason for not having more than three was that no room could be found for more; a point of diminishing returns was reached. Mr. W. A. Ramshaw expressed the opinion that three was much too few.

Mr. T. M. Hurewitz said that in the development of the BISMAC at RCA, the need for feedback of information from the programmers to the engineers appeared. Donn Combelic asked how many programmers' ideas were rejected by the engineers; Hurewitz replied that no ideas were rejected without conference between programmers and engineers. Unfortunately he was unable at this date to discuss the actual design. There ensued a heated discussion on the ethics of industrial secrecy.

Dr. Grace Hopper raised the possibility of using several small computers in parallel. The greatest demand was for small machines, and she hoped that each university would eventually have one. The size of a big machine was due to the number of operations it performed; this could be provided by automatic coding. She foresaw a mass produced small machine, delivered with a compiler and library appropriate to the customer's needs.

Mr. J. W. Backus disagreed with this philosophy on the gounds of computing speed; since increased speed costs little more, a large computer is cheaper to use than a small one. Such things as floating point require rapid computing. Dr. Hopper explained that she was thinking mainly of business rather than scientific applications, and of machines costing less than $50,000. Mr. G. E. Reynolds pointed out that the use of floating point could often be avoided by having long registers with the point in the middle. Dr. Hopper agreed that the costs of facilities would have to be balanced. Mr. Backus said that it was not the extra facilities that made a big machine cheaper to run, but merely the higher speed. Even using Speedcode, the IBM 701 was 10 times as fast as the CPC on one job.

Mr. G. G. Foster described the procedure employed at IBM(Endicott) in designing sample data systems, using a fast machine with only one program. A first estimate of the basic design parameters is made and the problem is coded for a machine with these characteristics. The engineers are then consulted on the cost and feasibility of the design, a revised design is produced, and the problem recorded for the revised design. A considerable amount of coding is involved, and this raises the question whether a translator could be produced; the idea does not seem impossible. A method of automatically coding for minimum latency would be required. Dr. Hopper offered the advice that this is best done by writing the routine in relative coding and applying the latency-minimization process starting at the end of the routine and working backwards.

Mr. J. H. Brown asked whether Mr. Foster had ever used an existing computer to simulate the hypothetical one. Mr. Foster replied that this would be done when the proposed design reached a sufficiently firm stage; so far no routine had been completed before the design was changed.

Returning to the topic of small versus large machines, L. Rosenthal said that each had its place. Donn Combelic interpreted Dr. Hopper's remarks as meaning that although big machines were used in scientific research, many businesses could use small machines. Dr. L. DeWitte said that the small machine was useful in experimental work too. Donn Combelic asked whether a problem tried on a small machine would eventually be put on a large one. Dr. DeWitte replied that this might be so; or a problem tried roughly on a large machine might be tailored to fit a small one.

Dr. J. C. P. Miller pursued the idea of a machine with a very simple instruction code. Most machine operations are made of a number of small units; these can each be made to correspond to a 'micro-instruction', and bigger instructions can result from the execution of a 'microprogram'. Such a scheme, using a microprogram of 256 micro-instructions, will be used in the new machine at Cambridge University. One might arrange to be able to change the microprogram for different applications, and even to program this change. Donn Combelic remarked that the latter idea had also been put forward by Dr. Perlis at Purdue University. Prof. Adams pointed out that the plugboard of the CPC could be regarded as a changeable microprogram. Mr. R. D. Smith said that a mchine with a drum-stored microprogram had been built for military applications.

Mr. P. F. Williams said that his firm is trying to decide on a computer to use. They want something intermediate between an IBM 650 and 704. The 704 seems too large: they would not be able to keep it busy. John Backus said that by time sharing, a big computer could be used as several small ones; there would need to be a reading station for each user. Mr. H. Freeman remarked that down time would be worse for one large machine than for several small ones. Stanley Gill said that the idea of a centralised computing bureau seemed a good one, except that if each subscriber were to have his own input-output equipment he would not be able to afford such flexible equipment. Prof. Adams pointed out that the idea was similar to that of a centralised stenographic bureau, which was not always successful. D. L. Shell said that Bell Laboratories had in fact used a central computer with remote input-output very successfully. He said that the cost per operation of a machine is roughly proportional to the square root of the time per operation; also the work load increases exponentially with the time. He felt that where elapsed time is vital in case of breakdown, a large computer is still preferable.

Mr. Hanshaw said that he would rather have one IBM 701 than eight CPC's; there is very much difference in the work capacity, price for price. They would, however, still keep spare small machines to handle important jobs. Dr. Hopper repeated that she was thinking of people who could only afford one small machine.

Resume of Session 17

The use of flow diagrams in coding was the main subject of discussion. In response to C. W. Adams's question as to how many have a higher level individual set up a flow diagram and persons at a lower level do the coding of each block, about 8 of the assembled 50 persons raised their hands. W. A. Ramshaw noted that at Douglas Aircraft (under John Lowe) this procedure is followed unequivocally. Rigid specifications have been set up for flow diagrams so that coding can be done by coders with high-school level training. Ramshaw added that flow diagrams are not used at United. C. W. Adams suggested that if the specifications are sufficiently rigid, the machine itself could handle the coding.

Dr. Hopper remarked that flow diagrams offer a potent means of communications since they are not tied to the computer. She has filled the squares with information varying from mathematical symbols to sentences. K. Powell reiterated the usefulness of flow diagrams as a communications medium and added that his group uses blocks, containing arbitrary amounts of information, that can later be isolated into subroutines. They have found that engineers can use these blocks to suggest decision points, etc. without being familiar with the coding.

G. E. Reynolds stated that his group uses a system that is half-way between flow diagram and basic machine coding. They make use of magnetized blocks that are easy to erase and to assemble. He added that this system is facilitated by the fact that they have a four-address machine.

There was general agreement that flow diagrams can be very useful to describe a routine after it has been written. Dr. Hopper remarked that flow diagrams have proved helpful in locating subroutines as common blocks. D. Shell noted that the two-dimensional appearance of a flow diagram is helpful and space-saving. However, S. Gill gave an example of integrating a differential equation that is easier to consider as subroutines rather than as a flow chart.

In replying to C. W. Adams's query as to the existence of mannuals on flow diagrams, Dr. Hopper indicated that Remington Rand has a mimeographed copy making use of Goldstine and von Neumann symbols.

In answer to a question, Dr. Hopper remarked that her group makes use of substitution blocks with numbered references to indicate how one block of a flow diagram may change another.

T. M. Hurewitz suggested that flow diagrams can serve as a check on the problem set-up. S. Gill remarked that he had found a coding error from a flow diagram rather than from the routine itself.

C. W. Adams noted that in teaching the use of flow diagrams in his course, he found that by introducing the concept too early, the problem to which it was applied was too simple. Consequently the students did not obtain a full appreciation of the use of flow diagrams.

In answer to a question, C. W. Adams stated that the Laning-Zierler system has proved very useful for a particular class of problems.

Resume of Session #13

Mr. Randall Porter, Boeing Aircraft, Seattle, Wash., described Boeing's experience with their 701. After waiting a year and a half, they received their 701 in Dec., 1953. About 6 months before this they put three people to work doing two important things: 1) Finding out what other people were doing with their 701's, particularly Los Alamos and United Aircraft, 2) Coding subroutines and assembly routines for their own use. After several months of this they spent a week on the New York 701. Their experience there convinced them they should completely revamp their plans by doing things much more automatically.

They now have an extensive Library of Subroutines, most of which are stored on a tape and transferred to the drum during assembly. Subroutine words are of two types: 1) normal words, 2) exceptions. The latter are words whose addresses are not to be oriented or treated in the normal manner during compilation. All normal words precede all exceptions in each subroutine and are thereby identified. Each subroutine has a call number which is specified by the coder; the assembly routine then calls the indicated subroutine from the tape and integrates it into the main routines.

The assembly routine occupies about 1/6 of high-speed storage; the remainder is available for the coders routines and for subroutines.

Although Boeing has both IBM Speedco and Los Alamos Dual system available, they are little used. About 70% of all their coding is done in straight machine code. Most of their engineers are willing and able to do the scale-factoring necessary when using fixed point machine code. Whenever the scale-factoring or the engineer becomes too difficult, Speedco or Dual is used.

In response to a question by P. Williams, Mr. Porter said that the 701 has gained acceptance among the engineers by word of mouth so well that it is kept very busy -- no proselyting has been necessary. W. Ramshaw of United Aircraft commented that his experience had been somewhat different. He found it best to have people at as high a level as possible be among the first to receive training in the use of the computer. Then, since the computer is definitely worthwhile, you have supervisors selling its use to subordinates -- a situation very different from subordinates trying to sell the computer to their supervisors.

Mr. Porter discussed the scheduled maintenance of Boeing's 701. Usually from 2 to 3 hours per 16-hour day. In response to a comment that this seemed unduly high, Mr. Porter replied that it compared favorably with Douglas, for instance. However, at Douglas they schedule maintenance for less than 2-3 hours per day, but their statistics show that maintenance requires at least 2-3 hours per day. Mr. Porter feels time scheduled for maintenance should be realistic.

Q. by W. Ramshaw: What is the output of the Boeing assembly routine? Answer: The routine on punched cards and/or on magnetic tape; the choice is made manually.

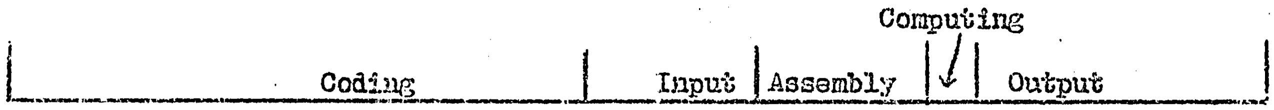
Q. by W. Hunter: Have you converted any CPC problems to 701? Answer by Mr. Porter: We still keep getting problems suitable for CPC, and we also keep CPC's on test run data reduction, for example, almost on a standby basis.

An exchange between R. Porter and S. Gill concerned subroutines. The originator of a Boeing subroutine checks it out and writes it up, occasionally getting help from staff members in the write-up. The machine operator, among his other duties, maintains the subroutine library.

D. Combelic queried R. Porter about how Boeing's plans to adopt an algebraic coding system would affect users of the present system there.

R. Porter believes that such changes can be made without serious disruption to existing routines, and that it would not be necessary to keep both systems in effect indefinitely. In fact, old-fashioned subroutines now are gradually removed from the tape library and made more difficult to use by being available only in card deck form.

Further comment by R. Porter. Typical division of time for problem solution is



Boeing's emphasis therefore is primarily on decreasing coding and output time.

19. Resumé of Discussion

Dr. G.Y. Cherlin outlined the needs of the Mutual Benefit Life Insurance Company. For arithmetic, fixed point numbers of 20 decimal digits would be useful; also there is a need to store alphabetical information. High speed tape searching and printing will be required. Quick access to any point in a file is wanted. The present plan is to review the whole file periodically. With regard to a basic library provided with the machine, there is a difficulty because needs vary with the company and with the time. The problem of transcribing the necessary records for automatic operation is a big one; this part of the switchover may take years. Variable length fields arise; premiums need only 6 or 8 digits, some sums of money may need 12. The company is not considering the electronic handling of investments; it is considering billing, accounting, file maintenance, etc. Reliability is important: there is interest in the "fail safe" type of check.

Mr. T.M. Hurewitz asked whether the acturial work would be mechanized. Dr. Cherlin replied that it would not. In reply to a further question, he said that dividends would be calculated individually, not by reference to a table. Operations on different files would be combined as this became possible. With the I.B.M. 650, it would be possible to combine first two and then three files. It may eventually be possible to combine 50 files into one for processing.

Prof. Adams raised the question of the extent to which automatic coding might be used in business. Would it be useful in optimizing procedures? Could a manufacturer usefully provide pseudo-codes?

Mr. C.G. Lincoln put forward the viewpoint of the non-life insurance business. The amount of computing is very small; it can be done simultaneously with input and output. However, a transcription problem arises. Competition forces the immediate furnishing of policies in the field; it is desirable to produce simulaneously a mechanical record. A machine is required that will print policies and also make a tape or card record. In the life insurance business, the Traveler's Insurance Company has no dividend calculation problem as it works on a guaranteed cost basis. Instead there is a more frequent and more intricate calculation of guaranteed premiums. Also the computation of forfeiture values is difficult as the rates are constantly changed. It is possible that a form of curve-fitting will speed up this computation.

Prof. Adams pointed out that in many cases if the management would agree to use a formal approximation to a curve, they might save more money in computing than they lost through errors in the curve. Payrolls are full of untidy exceptions, and this is one of the difficulties in automatic coding. Dr. Cherlin said that his company would handle payrolls first, and Mr. Hurewitz said that there are very few basic types of payroll computations.

Dr. Grace M. Hopper mentioned an existing machine used for essentially business computations, the I.B.M. 701 at the Aviation Supply Center. L. Rosenthal said that the Univac at the David Taylor Model Basin also is used for some business type calculations, namely the planning of supplies by the central supply office.

Résumé of Discussion

G. Hopper mentioned that for business problems it was often convenient to break data processing by automatic computers into four classifications - input, decisions, calculation, and output - and that in business iteration on fixed data plays a much smaller role than in mathematical programming.

A question arose concerning the difficulty of human access to information stored on magnetic tape.

Conway replied that printing out small files or only changed items in large files were two possibilities. The difficulty of convincing some people of the reliability of magnetically stored data was mentioned. In banking, a law requires checks to be sorted and returned and this operation is difficult to mechanize.

G. Hopper emphasized the importance of the development of a large scale random access file.

Carter emphasized cost and the ability to handle exceptional cases as important aspects of data-processing systems.

C.Adams mentioned that the ability of electronic computers to make it possible to handle many exceptions in a routine fashion, and in some cases was a strong point in favor of electronic equipment. The possibility of selling some inconvenience in return for faster processing might be considered and has been tried in some cases, i.e. punched card personal checks.

Hurewitz mentioned that new auditing procedures probably would have to be developed to handle automatic data-processing.

The possibility of an engineering quarantee of proper machine operation being considered a sufficient audit was conjectured by S. Gill.

Hurewitz added that a continuous audit would probably be more practicable with increased computing capacity.

Rosenthal inquired as to whether parallel checks and individual checks, e.g. overly large changes or item sizes, could be maintained.

Cherlin observed that in addition to this type of check, engineers could often suggest checks not obvious to programmers.

C.W. Adams emphasized that careful attention should be paid to checking the most unreliable parts of the equipment.

Carter noted that a careful probabilistic study of the economics of checking has been made and may be available on request.

Automatic Coding Techniques

List of Reference Material

compiled largely from lists prepared by Grace M. Hopper, Remington Rand Inc., and Frank E. Heart, M.I.T.

Conference Proceedings:

A. Proceedings of the Association for Computing Machinery, at the Mellon Institute, Pittsburgh, Pa., May 1952. Obtainable at $4.00 from the A.C.M., 2 East 63rd Street, New York 21, New York.

A1. p.99 "Small Problems on Large Computers"
C. W. Adams

A2. p.173 "Construction and Use of Subroutines for the Seac"
Joseph H. Levin

A3. p.231 "The Use of Subroutines on SWAC"
Roselyn Lipkis

A4. p.235 "The Use of Subroutines in Programmes"
David J. Wheeler

A5. p.237 "Progress of the Whirlwind Computer Towards an Automatic Programming Procedure"
John W. Carr

A6. p.243 "The Education of a Computer".
Grace M. Hopper

B. Proceedings of the Association for Computing Machinery, at the University of Toronto, Ont., September 1952. Obtainable at $3.00 from the A.C.M., 2 East 63rd Street, New York 21, New York.

B1. p.1 "Compiling Routines"
Richard K. Ridgway

B2. p.17 "Machine Aids to Coding"
E. J. Isaac

B3. p.19 "Computer Aids to Code-checking"
J. C. Diehm

B4. p.21 "Input Scaling and Output Scaling for a Binary Calculator"
E. F. Codd and H. L. Herrick

B5. p.46 "Logical or Non-mathematical Programmes"
C. S. Strachey

B6. p.55 "Simple Learning by a Digital Computer"
A. G. Oettinger

B7. p.81 "Interpretative Subroutines"
J. M. Bennett, D. G. Prinz, M. L. Woods

B8. p.121 "Pure and Applied Programming"
M. V. Wilkes

C. Proceedings of a Symposium on Automatic Programming for Digital Computers, sponsored by the Navy Mathematical Computing Advisory Panel, Washington, D.C., May 1954. Obtainable after October 1954 at about $3.00 from the Office of Technical Services, Department of Commerce.

C1. "Definitions"
Grace M. Hopper

C2. "Differentiators"
Harry Kahrimanian

C3. "Compiler Method of Automatic Programming"
Nora Moser

C4. "Editing Generators"
John Waite

C5. "New York University Compiler System"
Roy Goldfinger

C6. "Application of Automatic Coding to Logical Processes"
Betty Holberton

C7. "The M.I.T. Systems: Comprehensive, Summer Session, and Algebraic"
C. W. Adams and J. H. Laning, Jr.

C8. "Interpretive Routines in the Illiac Library"
David E. Muller

C9. "Planning Universal Semi-Automatic Coding"
Saul Gorn

C10. "Present Status of the Michigan Automatic General Integrated Computation(MAGIC)"
J. H. Brown and J. W. Carr III

C11. "Automatic Programming on the Burroughs Computer"
Hubert M. Livingston

C12. "IBM 701 Speed Coding System"
John W. Backus and Harlan Herrick

C13. "Programming for the IBM 701 with Repetitively Used Functions"
Allen Keller and R. A. Butterworth

C14. "Summary and Forecast"
Grace M. Hopper

Book

D1. M. V. Wilkes, D. J. Wheeler, and S. Gill, "The Preparation of Programs for an Electronic Digital Computer." Addison-Wesley Press, Cambridge, Mass., 1951.

Papers in Periodicals

E1. J. W. Backus, "The IBM 701 Speedcoding System," Journal of the Association for Computing Machinery, Vol. 1, No. 1, p. 4 January 1954.

E2. S. Gill, "The Diagnosis of Mistakes in Programmes on the EDSAC," Proceedings of the Royal Society (London), Section A, Vol. 206, p. 538, 1951.

E3. Margaret H. Harper, "Subroutines: Prefabricated Blocks for Building," Computers and Automation, Vol. 3, No. 3, p. 14, March 1954.

E4. Grace M. Hopper, "Compiling Routines," Computers and Automation, Vol. 2, No. 4, p. 1, May 1953.

E5. Grace M. Hopper and John W. Mauchly, "Influence of Programming Techniques on the Design of Computers," Proceedings of the Institute of Radio Engineers, Vol. 41, No. 10, p. 1250, October 1953.

E6. N.B.S. Machine Development Staff, "The Incorporation of Subroutines into a Complete Problem on the NBS Eastern Automatic Computer," Mathematical Tables and Aids to Computation, Vol. 4, p. 164, July 1950.

E7. D. J. Wheeler, "Programme Organization and Initial Orders for the EDSAC." Proceedings of the Royal Society (London) Section A, Vol. 202, p. 573, 1950.

E8. M. V. Wilkes, "The Use of a "Floating Address" System for Orders in an Automatic Digital Computer," Proceedings of the Cambridge Philosophical Society, Vol. 49, pt. I, p. 84, 1953.

Reports with Limited Circulation

Requests for copies should be addressed to the organizations shown.

Programming Research Section, Remington Rand Inc., 1624 Locust Street, Philadelphia 3, Penn.

F1. "Analytical Differentiation by a Digital Computer," by Harry G. Kahrimanian

F2. "A-2 Compiler Manual"

F3. "A-2 Compiler"

International Business Machine Corporation, 590 Madison Avenue, New York 22, New York

F4. "IBM Speedcoding System for the Type 701 Electronic Data Processing Machines."

Digital Computer Laboratory, Massachusetts Institute of Technology, Cambridge 39, Massachusetts

F5. "M-2539: Comprehensive Systems Manual: Part I, Introduction to Programming," by H. H. Denman, E. S. Kopley and J. D. Porter, December 1953.

Part II on the general principles of operation of the Comprehensive System, and Part III containing details of the executive routines are in preparation.

Instrumentation Laboratory, Massachusetts Institute of Technology, Cambridge 39, Massachusetts

F6. "E-364: A Program for the Translation of Mathematical Equations for Whirlwind I," by J. H. Laning and N. Zierler, January 1954.

University of California Radiation Laboratory, Livermore, California,

F7. "LMO Edit Compiler," UCRL-4286 (unclassified) by Merritt Elmore, February 1954.

A.E.C. Computing Facility, New York University, 253 Greene Street, New York 3, New York

F8. "New York University Compiler System," NYU-6478 (unclassified) by Roy Goldfinger, March 1954.

University Mathematical Laboratory, Free School Lane, Cambridge, England

F9. Programming notes and recent library routines. (1954; in preparation)

Digital Computer Laboratory, Graduate College, University of Illinois, Urbana, Illinois

F10. "Illiac Programming," April 1954