Report R-233

THE M.I.T. SYSTEMS OF AUTOMATIC CODING:

COMPREHENSIVE, SUMMER SESSION, AND ALGEBRAIC

Talk delivered by

C.W. Adams

at the Symposium on Automatic Programming
for Digital Computers
sponsored by
the Navy Mathematical Computing Advisory Panel
May 12, 1954

DIGITAL COMPUTER LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Cambridge 39, Massachusetts

July 12, 1954

The M.I.T. Systems of Automatic Coding: Comprehensive, Summer Session, and Algebraic

by Charles W. Adams

(Text of a paper presented May 12, 1954, at the Symposium on Automatic Programming for Digital Computers sponsored by the Navy Mathematical Computing Advisory Panel.)

Any implication in the title that the automatic coding systems described here represent all of the M.I.T. activity in the field is unintentional. The work of Dr. F.M. Verzuh using a CPC is not described, nor are the several rather specialized techniques developed by various users of the Whirlwind computer. The paper concludes with a description of the algebraic system developed by Dr. J.H. Laning, Jr., and Mr. N. Zierler of the M.I.T. Instrumentation Laboratory, an experimental system not widely used but pointed toward a goal talked about by many but realized by few. Primarily, however, this is a report on the two systems of coding used in most of the scientific and engineering computation and in the academic program, respectively, at the M.I.T. Digital Computer Laboratory.

The Problem

The ultimate goal of any automatic coding technique is presumably to maximize the simplicity and efficiency of solving a problem on a digital computer. Improvements can be made in:

a) Coding, in which technical personnel are required to

1. learn -- i.e., comprehend and remember a computer code,
2. write -- i.e., select and record a sequence of instructions,
3. debug -- i.e., locate and correct any mistakes which occur.

b) Clerical operations, in which trained personnel

1. prepare cards or tapes with programs and data,
2. operate the computer -- loading and unloading readers and recorders, pushing buttons, logging, recording data, etc.,
3. make corrections -- by changing computer switches or punching new cards.

c) Computer operation, including

1. debugging time,

2. input-output time,

3. computation time.

Clearly, to make a system easy to learn, one must make it simple, natural, and mnemonic. Ease of use, both for the coders and the clerks, entails concise methods of representation which must be readily and unambiguously producible both in manuscript and in typescript. The conventions by which the input operates should require of the coder and typist alike a minimum of unnatural, extraneous, implicit, or easily derived information. Making the computer handle the records of its own operation automatically can further simplify operation. It is but one of the many ways of placing the burden of the clerical part of program preparation and operation onto the computer itself -- the world's cheapest and most reliable clerk.

Many of the conventions and facilities most useful to the coder and the typist could readily be built into the computer hardware if the need is recognized in time. Other facilities which are highly complex (such as floating-address assignments) or infrequently-needed (such as extreme precision) can in all cases best be programmed rather than built in. Thus, automatic coding attempts in part to overcome imperfections in computer design and in part to add more elaborate facilities than could economically be built into the computer.

Some of the goals listed for automatic coding are not necessarily compatible with one another. It would seem that, in general, reducing the effort made by the coder would be at the expense of the computer time. But while conversion, compilation, and especially interpretation require computer time, this may be more than compensated for by the reduction: (1) in debugging time (through reduced likelihood of mistakes and increased facility of debugging), (2) to a small extent in input time (through potentially more concise coding), and even (3) in some cases in computation time where minimal-latency coding is produced by the computer.

Nonetheless, a truly ideal system has not been hypothesized, to say nothing of being put into practise, as yet. Any existing system represents a compromise between what is desired and what is achievable with the time and talent available. This compromise is made, consciously or unconsciously, largely in terms of the intended uses and users and in terms of the terminal equipment (as well as the central computer) available, with special emphasis on the keyboard of the tape unit or key punch.

The Environment

Consequently, the description of the MIT systems starts logically with a summary of the equipment characteristics, the keyboard, and the intended uses and users.

As will be seen in figure 1 (below), the Whirlwind I computer can be essentially characterized as having a short binary word-length, a high speed of operation, a storage hierarchy of ample sizes, speeds, and versatility, and relatively slow output. Two special features of minor importance, which will be alluded to later but which are not listed, are a real-time clock (a 19-bit counter, manually resetable, which counts 15 cps pulses) and a manual input

TERMINAL EQUIPMENT CHARACTERISTICS OF THE MIT WHIRLWIND I COMPUTER

CENTRAL - COMPUTER	Single-address, parallel, binary computer - magnetic core memory = 2048 16-bit words add = 24μsec, multiply = 40μsec, control transfer = 16μsec	35,000 ops/sec.
SECONDARY-STORAGE	ERA Magnetic DRUM, 16 x 2048 words, 60 rps. - Raytheon Magnetic TAPE, 4+1 units at 125,000 words each -	31,000 words/sec. 390 words/sec.
INPUT -	Ferranti photoelectric 7-hole READER, 2 units - Flexowriter mechanical 7-hole READER -	205 char/sec. 10 char/sec.
OUTPUT -	Raytheon magnetic TAPE for later printing or punching 2+1 units at 53,000 characters each -	133 char/sec.
	Flexowriter printer, 1 direct, 2 from tape - Flexowriter punch, 1 direct, 2 from tape -	8 char/sec. 11 char/sec.
	16" SCOPE with visible face, and graphical points - 16" SCOPE with computer-controlled digits (point-by-point) - Fairchild CAMERA digits (special generator) -	6200 points/sec. 200 digits/sec. 1200 digits/sec.

WORD = 5-bit Operation + 11-bit Address = Sign + 15-bit Fraction

CHARACTER = 6-bit representation of the 50 keys on a Flexowriter, in arbitrary, teletype-like code

FIG.1

of 34 bits, 2 of which involve pushbutton-fired gas tubes which may be reset by the computer. The MIT version of a standard Flexowriter, shown in figure 2 (page 4), involves a full 50-key typewriter with upper and lower case. A few small but useful keyboard modifications have been made, notably the inclusion of 10 decimal exponents together with a negative sign, and a vertical bar intended to permit

FIG. 2

GENERAL USERS OF WHIRLWIND I FOR SCIENTIFIC AND ENGINEERING COMPUTATION			HOURS USED PER WEEK 1953-1954
ONR STAFF: 6 numerical analyst-advisor-instructors			.5
4 automatic-coding programmers			7.4
MIT STUDENTS: classes - senior	25	exercises	1.5
graduate	30		
summer	100		
graduate	10	study problems	.2
theses - SB	4	parts of theses	6.6
SM	13		
PhD	22		
MIT PROJECTS: ONR and other sponsored projects			7.6

FIG. 3

vertical rulings as well as magnitude signs. The reader associated with the standard Flexowriter permits duplication of tape, insertion of subroutines, etc. The code-delete button permits easy correction of typographical mistakes. The Flexowriter is usually attached to an M.I.T.-designed combination verifier (using the built-in reader) and manual binary-tape-punching unit.

The users of the Whirlwind computer summarized in figure 3 (page 4) cover only that half of the usable time in which the machine is available for unclassified work, under the sponsorship of the Office of Naval Research. Much of the work is academic in nature, serving more to train potential users than to produce results, and all of it is done on a self-service or open-shop basis in which the person with the problem prepares his own program.

The Systems in General

The three systems to be described here are considered in order of decreasing efficiency computer-wise and, hopefully, increasing ease of use for the coder. The first is an integrated Comprehensive System of Service Routines, abbreviated CS, now in use by all scientific and engineering computation users. This has been developed by the combined efforts and contributions of the staff and guests of the Laboratory, starting in June 1952, at which time a conversion routine and a floating-point interpretive routine were already in general use. Those who have contributed ideas or working programs or both to CS include (besides the author) Messrs. J.M. Frankovich (conversion), F.C. Helwig (floating-point), J.D. Porter, E.S. Kopley, M.S. Demurjian (output), I. Hazel, H.H. Denman, D.N. Arden (post-mortem), S. Best (testing), and J.T. Gilmore, D.J. Wheeler, and D. Combelic. The second system was designed and developed during the summer of 1953 for use in a special summer course on programming in which 105 students from industry had a chance to put several programs on the Whirlwind computer during a two-week period. Major contributions were made by many of those named above and by Dr. Stanley Gill, but most of the work was done with great haste and proficiency by Mr. Manuel Rotenberg and Dr. David Finkelstein. The third system is, as mentioned earlier, an algebraic notation developed entirely by Dr. J.H. Laning, Jr., and Mr. Neal Zierler.

It will be readily appreciated that the binary nature of the computer requires conversion during input, which permits the use of mnemonic symbols without much extra cost. Further, and more important, the unique brevity of word length, admirably suitable for control applications, requires double precision on all but the most casual scientific problems. This requirement in turn implies interpretive techniques and permits a floating-point system to be incorporated along with the double-precision without much extra cost. Figure 4 (page 6) shows the structure of three of the possible double-length number systems. Each of these is described by a pair of numbers:

the number of digits exclusive of sign in the fraction and the number of digits exclusive of sign in the exponent. In the double-length case these two numbers total 30, since there are 32 bits available and two are required for the signs. The systems shown are then 30,0, 15,15, and 24,6 respectively. CS allows any system 30-j,j to be specified with j between 0 and 15 inclusive, but 24,6 is the most common by far. The SS system uses 4 bits for tagging and permits operations on both 27,0 integers and 20,6 floating-point numbers. The algebraic system actually makes use of the CS routines and uses 24,6.

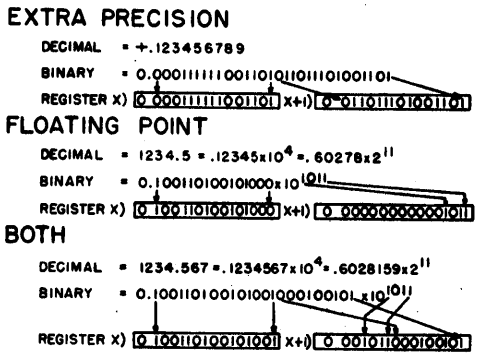


FIG. 4

ADDRESS SYSTEMS

	ABSOLUTE	RELATIVE	FLOATING
270	ta 289	0 ta 19+	ta k13
271	sr 5	1 sr 5	sr 5
272	ad 275	2 ad 5+	ad f7
273	td 277	3 td 7+	td f6
274	sl 15	4 sl 15	sl 15
275	mr 400	5 mr 400	f7 mr c3
276	ts 539	6 ts 3t	ts 3t
277	ad 0	7 ad 0	f6 ad

FIG. 5

Other features common to all the systems described here are the use of floating (or symbolic) addresses, cycle counter address-modification facilities, and automatically available output conversion and editing routines. In CS and SS, a floating address is a tag consisting of any lower-case letter, except o and l which are too readily confused with 0 and 1 on a typewriter, followed by a decimal number of 1, 2, or 3 digits (there is in CS an arbitrary limitation on the sum of the maxima of the numbers used with each letter). As will be seen in the example, figure 5 (above), the values of floating

addresses f 7 and f 6 are not necessarily determined at the time the instructions referring to them appear in the program. Unlike other sorts of preset parameters, each floating address has a single value in any given program, and the input program must first assign these values, then make another "pass" over the program to fill in these values where used. The time and extra storage required in processing floating addresses seems more than justified by the great convenience which floating addresses offer in single-address coding.

First and foremost, then, both CS and SS feature:

mnemonic alphadecimal input

floating addresses

floating-point arithmetic

cycle-counting address modification facilities

The Comprehensive System

In addition, CS permits the intermingling of machine code with interpreted code, yielding either 35,000 4.5-decimal-digit fixed-point operations per second or 800 7-decimal-digit floating-point operations per second. All Whirlwind input, output, and secondary storage units are available through automatically-selected subroutines or direct machine coding as desired. Changing from interpreted to machine code and vice versa is accomplished by jump instructions with special addresses, said instructions being designated by the words IN and OUT, for into and out of interpreted code. Use of special words to be handled by the conversion process is merely an extension of the general mnemonic system. Just as all machine instructions are denoted by pairs of letters, interpreted instructions are denoted by triples beginning always with "i". Since there are no unused bits in in the Whirlwind word, machine code and interpreted code are indestinguishable from one another once converted.

In addition to being easy to code, programs prepared in the Comprehensive System are intended to be easy to punch on tape, easy to operate, and easy to debug. Some of the incidental features aimed toward this goal are illustrated with varying effectiveness in figures 6, 7, and 8.

To facilitate tape preparation, considerable effort was devoted to eliminating ambiguities, unnecessary typing, and other manual operations. Thus, except in deference to the mnemonic notation, all typewritten characters convey some essential information and each word is as short as possible. Absolute addresses (written as integers followed by a vertical bar) may be put in anywhere for convenience in the following written program, but are required only when two consecutive words are to occupy non-consecutive registers. Non-significant zeros are never required (except in octal numbers which are always

written with a 0 or 1, a point, and 5 digits). Except for a few cases which are only likely to arise maliciously, a correct typescript implies a correct tape. For example, spaces at the end of words, tabulations at the end of lines, extra shifts to upper or lower case, and other Flexo characters which punch tape but do not appear on the typescript are all dealt with properly by the conversion program. The letter l and the digit 1 differ on the M.I.T. Flexowriter, but to simplify the job for an inexperienced tape preparer, the conversion program interprets the two characters identically so that either may be used at any time. Incidentally, the same is true of the letter o and the figure 0, even though they do not look alike, because the keys on the Flexowriter are so marked that confusion is easy for the hunt-and-peck typist.

The subroutines required for a given program are usually selected manually from a cabinet full of punched tapes and copied into the program where needed. CS, of course, has normal relative-address facilities and fairly elaborate preset-parameter facilities. Both output and floating-point routines are automatically compiled, and the same facility will soon be extended to other commonly-needed routines. The interpretive routine is included if the desired number-system designation, e.g., (24,6), is typed on the tape somewhere. Other parts of the interpretive routine which are relatively long and not always used (e.g., cycle counting) are automatically included or omitted depending on whether or not they are needed anywhere in the program.

Output routines (actually routines involving any terminal equipment) are called for by writing synthetic instructions consisting of three capital letters followed by any necessary parametric information. The first letter designates the equipment (M for magnetic tape, T for typewriter, D for drum, S for scope, etc.), the second letter specifies input (I) or output (O), and the third letter is A for alphadecimal, B for binary, or C for curve plotting. Thus MOA indicates alphadecimal output on the magnetic tape, presumably for later printing. The synthetic instruction is converted to a control transfer to a closed subroutine which is compiled and stored elsewhere automatically. In the case of printed output (MOA, TOA, or SOA), the form in which the printing is to occur is indicated by an example immediately following the three capital letters. Location of decimal point, zero suppression, number of digits, and other format information are all readily indicated in a fairly obvious notation. For example, MOA +1.234ss will record whatever is in the accumulator onto magnetic tape in Flexo code preceded by a plus or minus sign with one digit to the left of the decimal point and three digits to the right. The power of ten (with appropriate sign) by which the printed number must be multiplied to yield the number which was in the accumulator at the time of printing, follows the digits requested. This exponent is separated from the 4-digit mantissa by a vertical bar, and is followed by 2 spaces as requested by the ss terminating the sample number.

Simplification of computer operation hinges on reducing the possibilities of making mistakes and of wasting time by properly identifying everything, eliminating all switch settings and unnecessary pushbuttons, keeping the log automatically, and recording all trouble-indicating information automatically. CS does all of this except that, when trouble occurs in machine-coded sections, a manual notation must be made of the contents of the program counter, accumulator, etc., read in octal from indicator lights. This situation will be alleviated only by a sizable change in computer hardware which is being seriously considered.

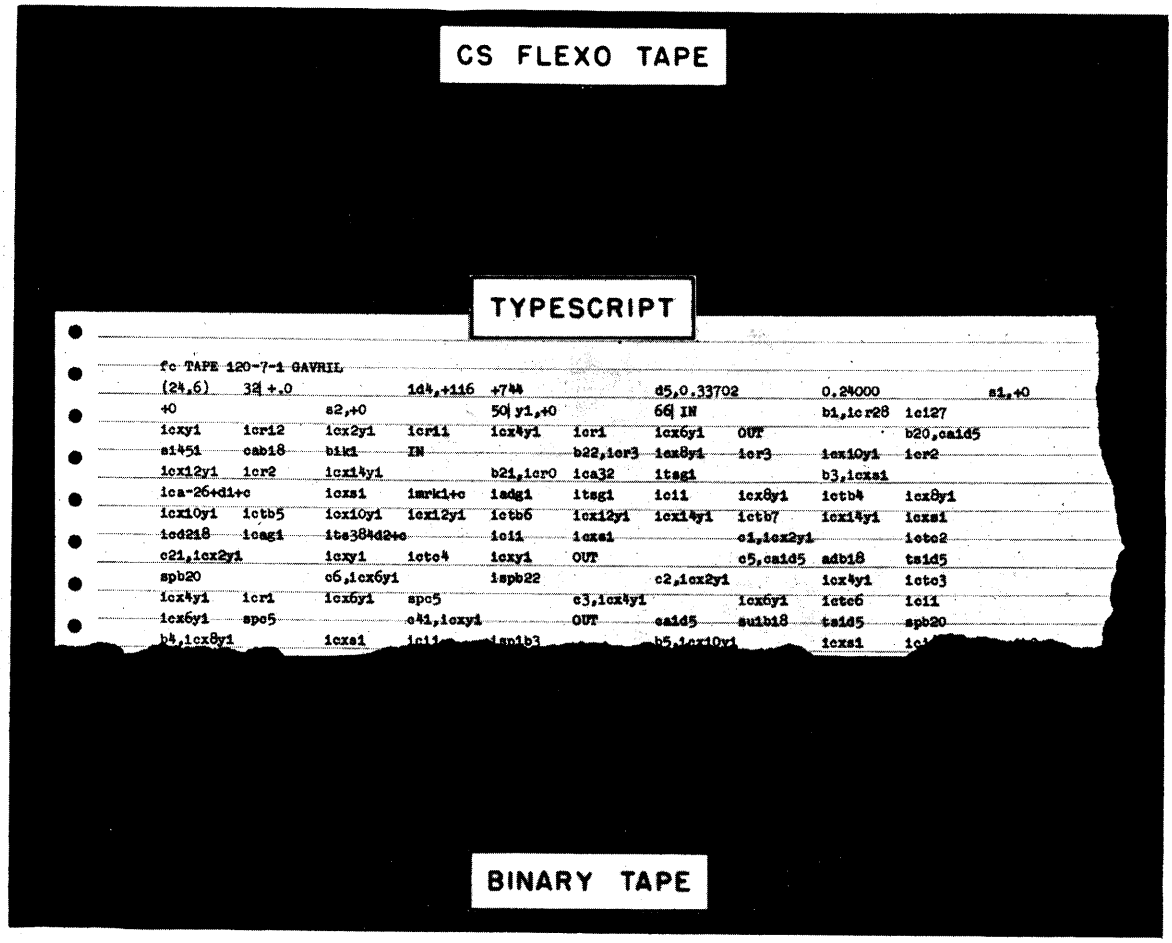


FIG. 6

The sample CS typescript and the Flexowriter tape which corresponds to it, shown in figure 6 (above), start with a typical identification line put on all tapes. The letters fc (flexo conversion) cause the input program to treat the tape as a CS tape whenever it is read in, without any manual switches being set to distinguish this from other types of tape. The tape number is ordinarily made up of three integers: a problem number assigned to each new project chronologically, a programmer number assigned uniquely to each person

entitled to submit programs, and a tape number assigned at will by the programmer for his own convenience. The name of the programmer, and/or the problem, and the date (not shown in this example) may follow the tape number to any length desired and will be reproduced as a heading for any diagnostic data printed by the computer. The tape number required on all tapes, is used for logging as well as for general identification. If a binary tape is prepared by the computer for repeated insertion of the program (eliminating the need for repeated conversion and compilations), that tape is identified by visible numbers (as shown) and by an identifying block at the start for use in logging the read-in of the binary tape. The visible tape number is not read in since it contains no holes in the "seventh" level, which to Whirlwind serve as second sprocket holes without which the other six holes are disregarded.

One frequently wants to know what absolute values were given during read-in to the floating addresses that were used. This information, useful in debugging, is recorded on a routine basis (unless suppressed) in the form indicated in the upper right of figure 7 (page 11). The word DECIMAL indicates of course that decimal addresses are used. The programmer is able to specify OCTAL if he so desires. The numerical results shown are unedited floating-point numbers with the decimal-point location indicated by the two-digit integer in each number.

There are a number of mistakes which the machine or the CS routines will detect and indicate. If the mistake is detected during conversion, a conversion post-mortem is printed automatically on the output Flexowriter in the form shown at the bottom of figure 7. The example given shows what happens when a careless programmer tags words in two different locations with the same floating address (flad) d37.

Mistakes occurring during computation can give rise to three different sets of data: manual records, interpretive routine control information (PA PM), and storage printouts. As has been mentioned, manual recording of octal data is the only alternative when a stop occurs in machine-coded programs.

The data recorded in the top five lines of the bottom form in figure 8 (page 11) illustrates a programmed arithmetic (interpretive) post-mortem, abbreviated PA PM. The PA PM shown gives, after suitable identification, the address (81) at which trouble occurred, the contents of that register and of the register to which that contents refers, and finally the contents of the multiple-register (pseudo) accumulator the (MRA). The following line appears only if cycle counters are used and records their contents at the time the program stopped (the pair of numbers in each are respectively the index and the criterion, since the M.I.T. philosophy has the counters count up to a preset criterion rather than count from a preset value up or down to zero). The absolute address 1734 is the starting address of the block of

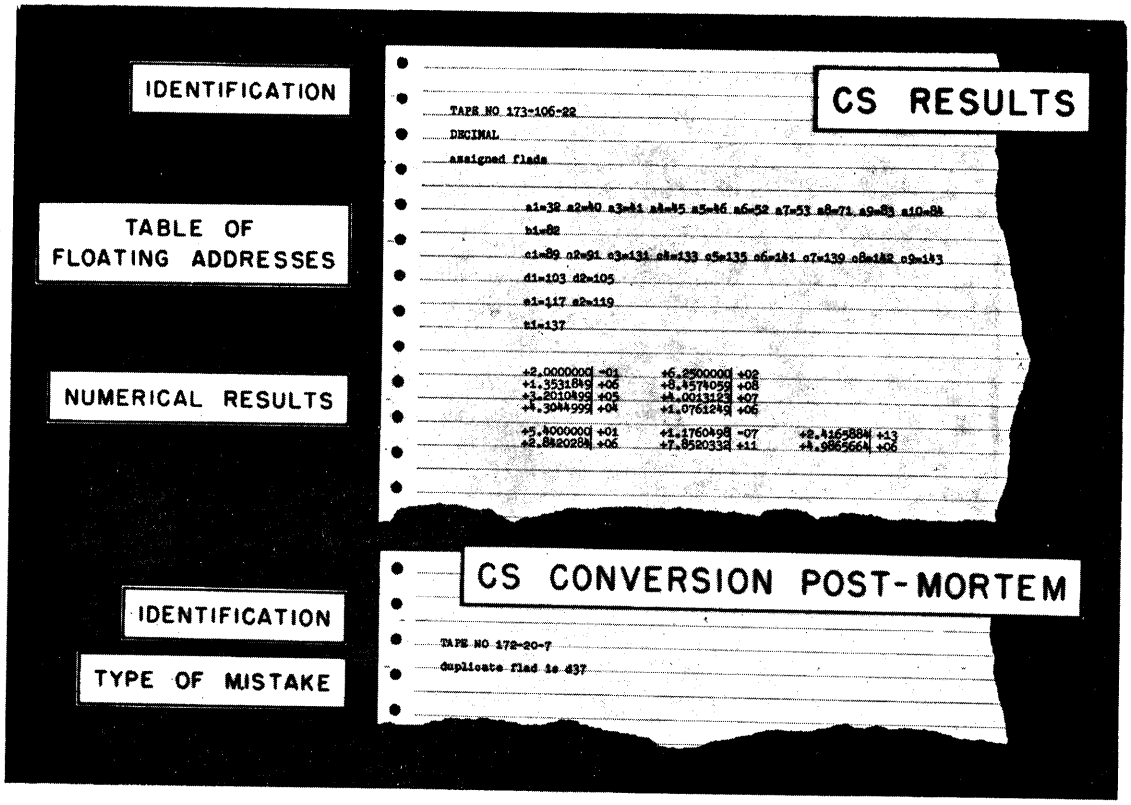

IDENTIFICATION
TABLE OF FLOATING ADDRESSES
NUMERICAL RESULTS
CS RESULTS
TAPE NO 173-106-22
DECIMAL
assigned flads
a1=38 a2=40 a3=41 a4=45 a5=46 a6=52 a7=53 a8=71 a9=83 a10=88
b1=82
c1=89 c2=91 c3=131 c4=133 c5=135 c6=181 c7=139 c8=182 c9=183
d1=103 d2=105
e1=117 e2=119
f1=137
+2.0000000 -01 +6.2500000 +02
+1.3531849 +06 +8.4574059 +08
+3.2010895 +05 +4.0013123 +07
+4.3044999 +04 +1.0761249 +06
+5.4000000 +01 +1.1760498 -07 +2.4165884 +13
+2.8820284 +06 +7.8520332 +11 +4.9865664 +06
IDENTIFICATION
TYPE OF MISTAKE
CS CONVERSION POST-MORTEM
TAPE NO 172-20-7
duplicate flad is d37

FIG. 7

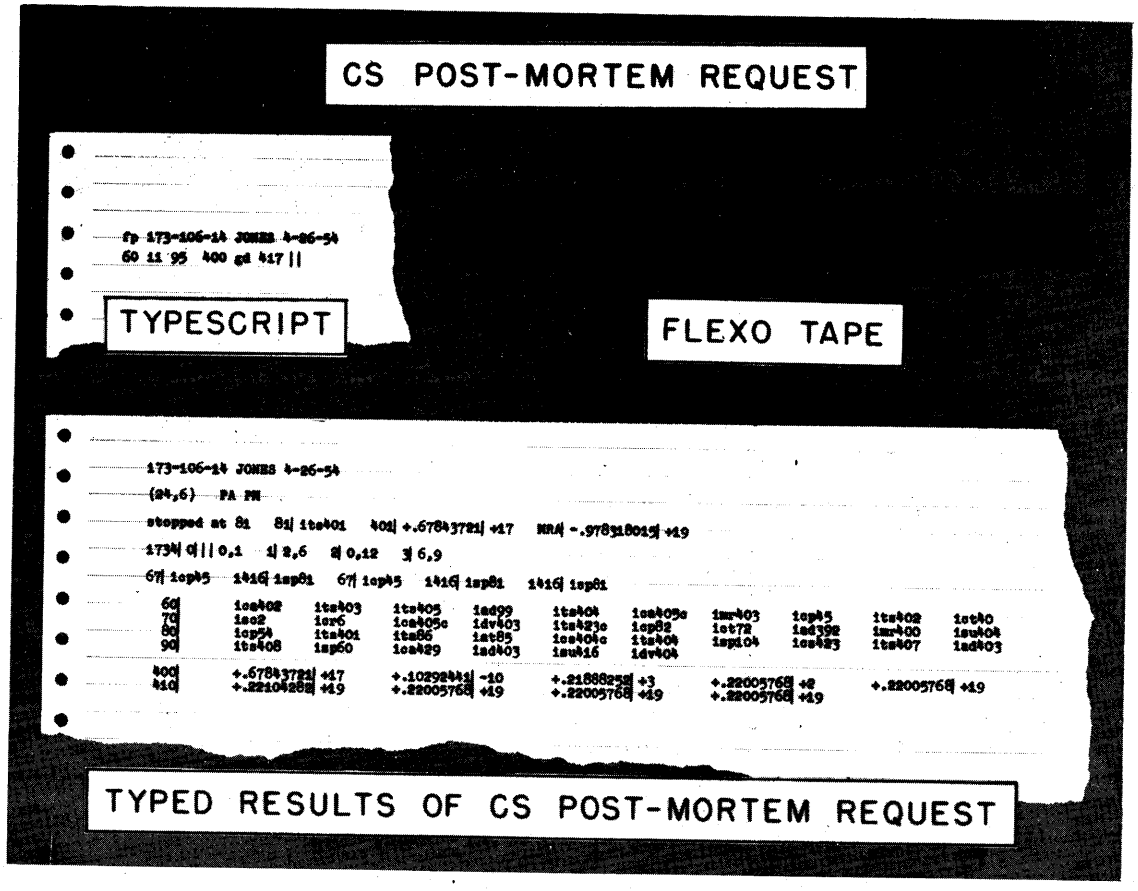

CS POST-MORTEM REQUEST
fp 173-106-14 JONES 4-26-54
60 11 95 400 gd 417 ||
TYPESCRIPT
FLEXO TAPE
173-106-14 JONES 4-26-54
(24,6) PA PM
stopped at 81 81| its401 401| +.67843721| +17 NRA| -.97831801| +19
173| 0|| 0,1 1| 2,6 3| 0,12 3| 6,9
67| icp45 1416| isp81 67| icp45 1416| isp81 1416| isp81
60| isa402 its403 its405 isd99 its404 isa405c imr403 icp45 its402 ist40
70| isc2 isr6 isa405c idv403 its423c icp82 iot72 isd398 imr400 isu404
80| icp5A its401 its86 ist85 isa404c its404 icp82 isp104 isa423 its407 isd403
90| its408 isp60 isa429 isd403 isu416 idv404
400| +.67843721| +17 +.10298411| -10 +.21888258| +3 +.22005768| +2 +.22005768| +19
410| +.22104288| +19 +.22005768| +19 +.22005768| +19 +.22005768| +19
TYPED RESULTS OF CS POST-MORTEM REQUEST

FIG. 8

storage in which the counter information is stored (this varies from program to program according to the length of interpretive routine required and is information which is sometimes wanted). The two extra vertical bars after the 0 indicate that counter 0 was the last one to be referred to. The fifth line shows the addresses from which the five most recent interpreted transfers of control occurred and the instructions which caused them. This "jump table" is kept up to date throughout the computation by the interpretive routine and is a dynamic tracing rather than a static post-mortem process. For a storage printout, a CS post-mortem request tape is typed by or for the programmer to specify what registers are to be printed and in what form. In the request typescript shown in figure 8, after the usual identification (fp for flexo post-mortem request), the programmer has requested a printout of the contents of registers 60 through 95 as interpreted instructions and of registers 400 through 417 in general decimal form, where each floating-point number occupies two consecutive storage registers. The resulting printout is shown on the bottom six lines of the figure 8. To obtain the entire post-mortem shown, the operator had merely to read in the request tape by pushing the read-in button, the same as for any other tape.

The Comprehensive System of Service Routines has been developed over a period of two years, during which time the computer has been considerably changed by the addition of a second bank of core storage, 16 drum groups, extra tape units, manual insertion registers, a more effective scope camera and Flexowriter printer, etc. The system has, furthermore, evolved and changed while always in constant use. Consequently, many of its features are compromises based on easy adaptation from one stage of development to another. Changing conventions so as to improve the system can readily invalidate the training and the programs of current users unless the changes are carefully restricted. The version described here, known as CS II, has a number of well-recognized shortcomings, many of which are hard to vindicate except in the light of the evolutionary process. Current plans for improvement center on a CS III which will incorporate all known improvements in a new system with conventions differing from CS II wherever a considerable gain results.

The Summer Session Computer

The Summer Session computer, on the other hand, was developed without regard for previous conventions or for computer efficiency. It is intended entirely for classroom use. For this it should be easy to learn, simple yet versatile to use, and sufficiently typical of available computers to be an effective example without being enough like any one to be offensively specialized. Like CS, it incorporates interpretive floating-point arithmetic and cycle-counting facilities, but unlike CS it is an exclusively interpretive system (with machine code not available) in which all words (floating, fixed, and instructions),

occupy two registers, with 4 bits used for tagging the words as to type. Consequently, to the user, the SS computer appears as a 28-bit-word, single-address computer capable of handling fixed-point integers (which are conceptually easier to teach than fractions) and floating-point numbers. The instructions are abbreviated mnemonically with 3 letters, and each performs only one useful, clean-cut function to avoid any confusion in learning or applying it. The arithmetic instructions apply to fixed- or floating-point numbers interchangeably, according to the tag on each word which designates its type. Almost all possible mistakes (and tricks) of coding are made illegal in an effort to make the computer stop as soon as it encounters any situation likely to have resulted from a coding mistake. The input and output are alphadecimal. There are no shift instructions (multiplication and division must be used instead), with the result that the binary nature of the computer is in no way apparent except for the powers of two which define register capacity. Similarly, since conditional jumps are so defined that they discriminate between strictly positive, strictly negative, and zero, there is no way to detect what negative representation is being used (actually a ones complement is employed).

The typescript example in figure 9 (page 14) starts with fs (for flexo SS) followed by any arbitrary title. The words and floating addresses are written in an obvious way. Absolute addresses can be indicated if desired. Floating-point numbers are distinguished from fixed-point integers during input by the presence of a decimal point in the former and not in the latter. As with CS, the address of the first instruction to be executed is indicated at the end of the tape, since it is not always convenient to start all programs at the same point and it is unthinkable to have to select starting points manually. It will be noted in the example that the program has been typed with eleven words to a line, except for a few blank spaces. In both CS and SS, words may be typed as many to a line as desired for appearance and readability. Each word ends with one or more carriage returns or tabs. The Flexowriter tab stops are set eleven spaces apart. The blanks in the example result from the tab skipping the first stop because the tab mechanism moves a minimum of two spaces.

Perhaps the most satisfactory features of all in the Summer Session computer are its diagnostic facilities. Almost all likely and unlikely coding mistakes cause the program to run afoul of the numerous restrictions which are not only clearly defined (as they are in all computers) but are detected by the computer. For example, any mistakes detectable during conversion, even the omission of a counter designation in an instruction where one is required or the omission of a floating address (two things which are easy to omit since they are often left out to be filled in later), are detected and recorded succinctly, as can be seen in the bottom example in figure 10 (page 14).

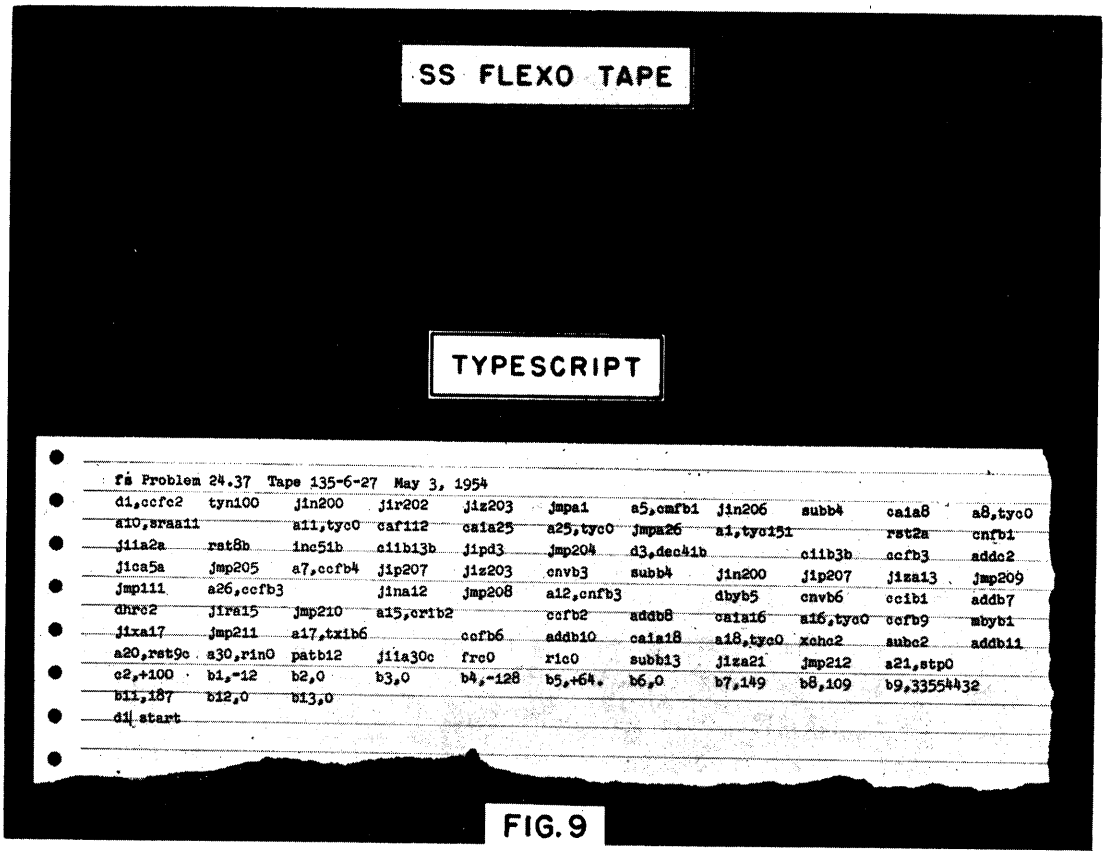

SS FLEXO TAPE
TYPESCRIPT
fs Problem 24.37 Tape 135-6-27 May 3, 1954
d1,ccfc2 tyn100 jin200 jir202 jiz203 jmpa1 a5,cmfb1 jin206 subb4 caia8 a8,tyc0
a10,sraa11 a11,tyc0 caf112 caia25 a25,tyc0 jmpa26 a1,tyc151 rst2a cnfb1
jiia2a rst8b inc51b ciib13b jipd3 jmp204 d3,dec41b ciib3b ccfb3 addc2
jica5a jmp205 a7,ccfb4 jip207 jiz203 cnvb3 subb4 jin200 jip207 jiza13 jmp209
jmp111 a26,ccfb3 jina12 jmp208 a12,cnfb3 dbyb5 cnvb6 ccib1 addb7
dhrc2 jira15 jmp210 a15,crib2 ccfb2 addb8 caia16 a16,tyc0 ccfb9 mbyb1
jixa17 jmp211 a17,txib6 ccfb6 addb10 caia18 a18,tyc0 xchc2 subc2 addb11
a20,rst9c a30,rin0 patb12 jiia30c frc0 ric0 subb13 jiza21 jmp212 a21,stp0
c2,+100 b1,-12 b2,0 b3,0 b4,-128 b5,+64. b6,0 b7,149 b8,109 b9,33554432
b11,187 b12,0 b13,0
d1 start

FIG. 9

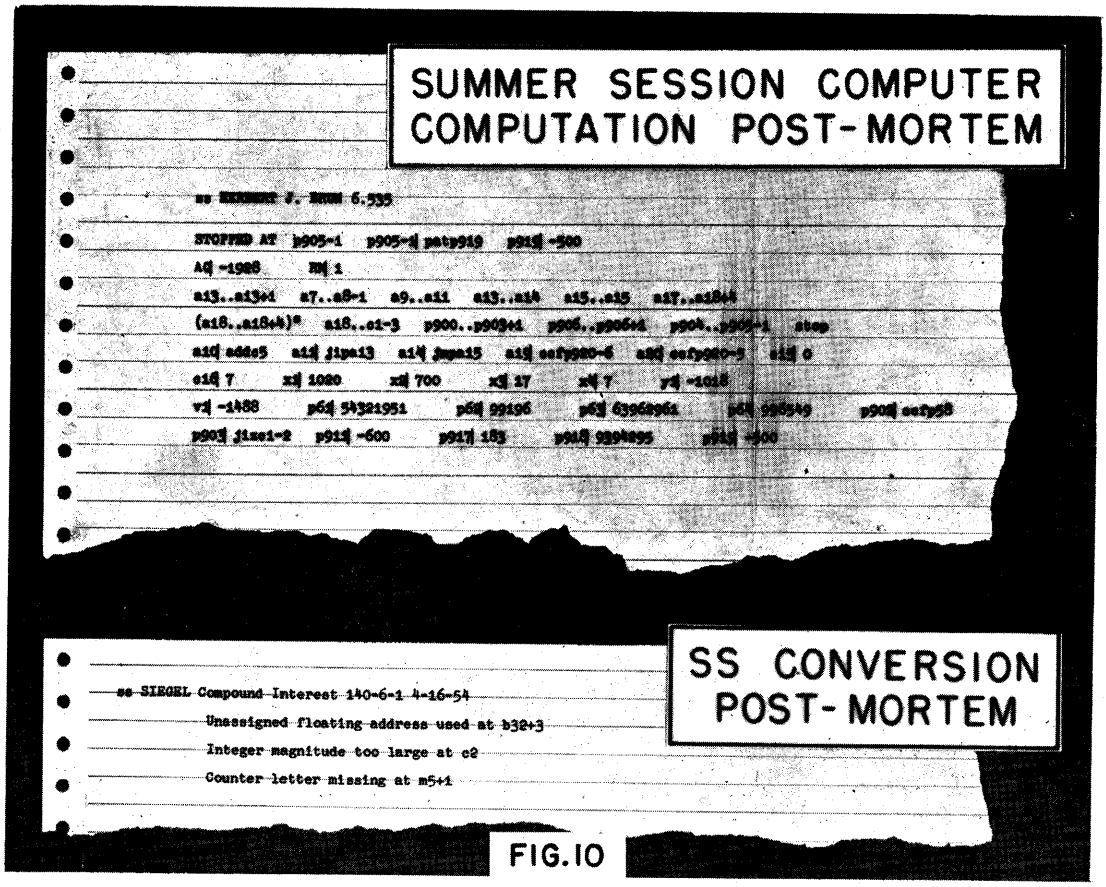

SUMMER SESSION COMPUTER COMPUTATION POST-MORTEM
ss HERBERT J. BRUN 6.535
STOPPED AT p905-1 p905-1 patp919 p919 -500
A0 -1928 RM 1
a13..a13+1 a7..a8-1 a9..a11 a13..a14 a15..a15 a17..a18+4
(a18..a18+4)* a18..c1-3 p900..p903+1 p906..p906+1 p904..p905-1 stop
a10 addc5 a11 jipa13 a14 jmpa15 a19 ccfp920-6 a20 ccfp920-5 a15 0
c16 7 x1 1020 x2 700 x3 17 x4 7 y1 -1018
v1 -1488 p61 54321951 p62 99196 p63 63962961 p64 996549 p902 ccfp58
p903 jizc1-2 p911 -600 p917 183 p918 9394295 p919 -500
SS CONVERSION POST-MORTEM
ss SIEGEL Compound Interest 140-6-1 4-16-54
Unassigned floating address used at b32+3
Integer magnitude too large at c2
Counter letter missing at m5+1

FIG. 10

It is worth noting that the original version of SS stopped everything as soon as one mistake was detected. Suprisingly, many programs proceed to have several of the small mistakes detectable during conversion. It is important, therefore, for the conversion program to detect as many mistakes as possible at once, as the present version does. When mistakes are thus readily detectable, the coder tends to use the computer to check his program rather than to bother even rereading it himself. There is a good deal to be said in favor of this, where often a minute of computer time can save hours of coder time. It is not apparent that checks of the sophistication needed can be performed on simpler and cheaper machines as effectively or inexpensively as they can be in the digital computer, any more than a complicated conversion and compilation can be.

Once the conversion is accomplished with no mistakes being found, the SS interpretive routine starts looking for mistakes during computation, such as numbers being used as instructions, arithmetic overflows, and improper combinations of numbers with instructions. Any of these will result in a full post-mortem such as the example at the top of figure 10 shows. The information is essentially the same as in a CS post-mortem: (1) identification, (2) where and why the stop occurred, (3) the contents of the accumulator (AC) and remainder register (RM), (4) the addresses to and from which took place each of the last ten jumps before the stop (printed on two lines), (5) the contents of any cycle counters used (omitted to avoid confusion if none were used, as in the example shown), and finally (6) a storage printout of every word which has been changed. The three major differences from CS are as follows:

1. All addresses are printed directly in floating form. i.e., the computer performs the inverse table look-up to convert absolute addresses back to floating, rather than presenting the table to the coder and letting him look them up, as in CS.

2. The storage printout occurs automatically; no request is needed. i.e., the registers to be printed are determined by a comparison with the original input (actually the SS uses a dynamic technique, tagging each word that changes as it changes) and the contents of only those that have changed are printed out, in the form in which they were originally written.

3. The jump table contains the ten most recent jumps and deals with one-jump loops by printing parentheses and an exponent to indicate how many repetitions occurred.

None of these things is in principle difficult to mechanize. The fact that they are not included in CS is due to the historical development which makes these features difficult to incorporate in the existing system. It is also not particularly difficult to deal

with multiple-jump and loops-within-loops in the parenthetic notation. This was in fact programmed for the SS computer but not incorporated because of the large amount of time consumed with a fairly small return in prospect. The changed-register storage printout is not only efficient from a printing-time point of view, but is extremely powerful in pin-pointing the mistakes. No coding mistakes have been made which the student programmers have not readily found merely by checking the SS post-mortem item by item, usually in a few minutes or even seconds.

Logging

The real-time clock in the Whirlwind computer together with the identification at the start of each tape permits the computer to be programmed to produce an accurate log of program operations. Debugging scientific programs for self-service users by means of the diagnostic features built into CS and SS means many very short runs. Sometimes 20 programs are tried in an hour. The film in figure 11 (below) shows the identification numbers plotted on the scope to identify any plotted results or scope-post-mortem information. Since the 35-mm film resulting from one day's work may run to several hundred frames, visible identification of the sort shown are necessary

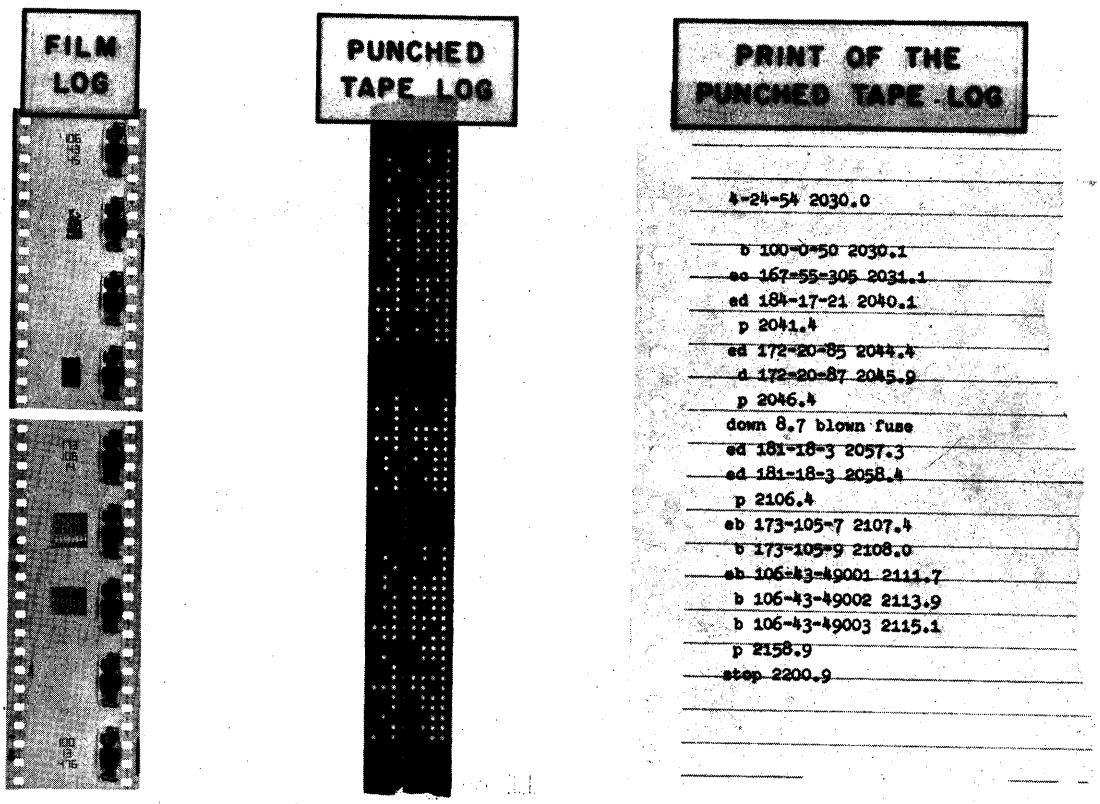


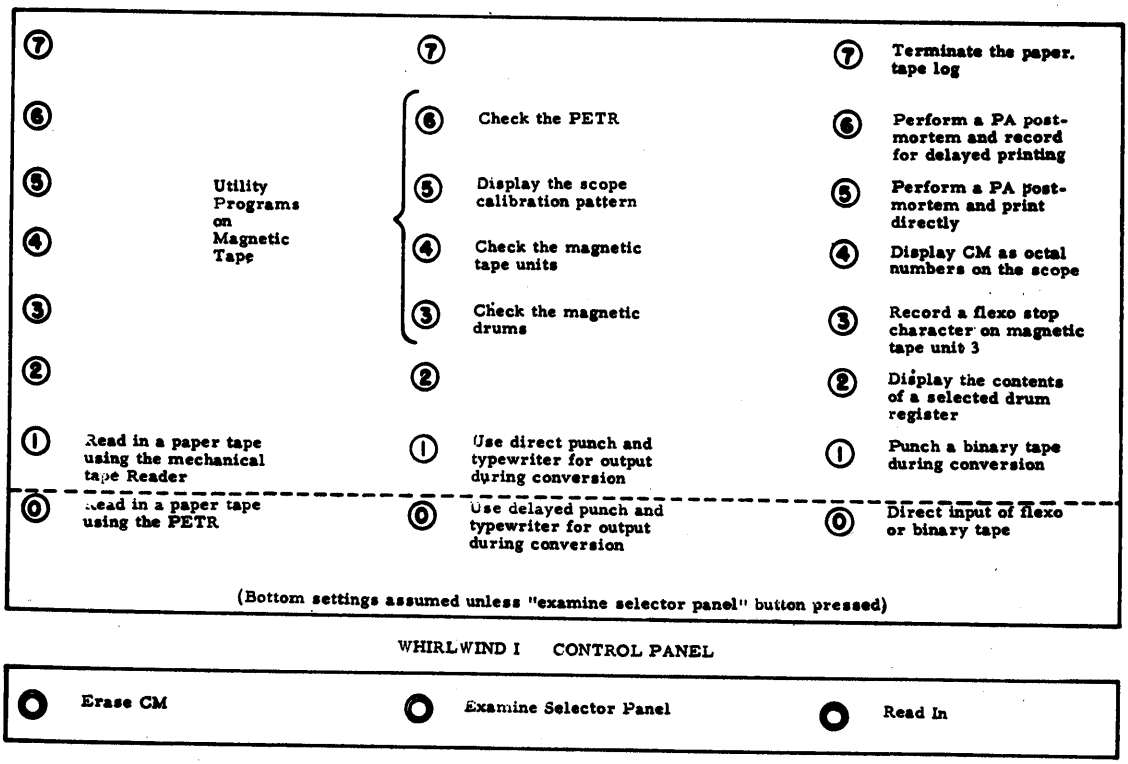
FIG. 11

to facilitate cutting and filing. Numerical information appears considerably smaller, as in the middle of the bottom strip of film where each frame can record the contents (in 6 octal digits) of 288 storage registers - a total of about 1750 decimal digits plotted in less than 10 seconds.

The log itself is a punched tape, prepared on a tape punch which is not normally used for any other purpose. Each time a tape beginning with f is read into the computer, an entry is punched on the log tape. When the log tape is printed on a Flexowriter later, each entry appears as a line indicating whether the tape was converted to binary (c), converted and performed directly (d), a post-mortem request (p), or a binary tape (b). The letter e indicates that the previous contents of the core storage were erased before the tape was read in. The hyphenated numbers are of course the tape numbers. The time is recorded on a standard 2400 hour basis to the nearest tenth-minute. The date and time at the start are set manually (the computer clock counter is reset to zero and refers to any absolute starting time set on switches). Any special information, such as machine malfunctioning, is recorded manually onto the tape by the computer operator by typing on the Flexowriter keyboard. The stopping time is recorded by setting a switch and pushing a button on the computer control panel. All the rest is done without any action required by the operator. A program to process a sequence of computer log tapes to produce biweekly and quarterly-report statistics automatically is underway.

From time to time a disapproval of pushbuttons and especially of switches has been indicated in this paper. It becomes necessary to admit that there are some switches used in operating the Whirlwind computer. These are shown in figure 12 (page 18). The read-in button in the lower right is the key to everyting else. Nothing ever happens until it is pushed and, when it is, the computer either (1) reads in tape from the photoelectric tape reader (PETR) and deals with it in accordance with the first two characters (e.g., fp, fc, etc.) or (2) if the "examine selector panel" button has been pushed since the last read-in, the read-in button causes the computer to do whatever is indicated on the switches on the selector panel above (these switches behave like adding-machine keys, only one of which is depressed in any one column at any one time). There are, in addition to the settings shown, other combination settings for infrequently needed operations (e.g., initiating the tape log, which requires setting switches in both the second and third columns). The read-in operation will erase core memory if the "Erase CM" button has been pressed since the last read-in and if the operation called for is not a post-mortem or other operation in which erasing would be undesirable. It should be noted that there are very few switches, that in normal operation none of these need be set, and that unless another button is pushed, none of them have any effect anyway.

WHIRLWIND I UTILITY ROUTINE SELECTOR PANEL



WHIRLWIND I CONTROL PANEL

FIG. 12

Mechanization

This is not the place to describe the mechanization of the CS and SS in any detail. The diagram in figure 13 (page 19) shows in a general way the manner in which the terminal equipment enters into the mechanization. Whirlwind registers 0 through 7 are permanent non-erasable storage for a program which the read-in button initiates. This copies all of core memory onto drum group 0, then copies all of group 11 into core memory. Drum group 11 contains facilities for examining the start of the tape or the selector panel to initiate any required conversion or service routine. Group 11 is permanently recorded (with the recording tubes disabled). Similarly, tape unit 0 is permanently recorded with the main body of the CS and SS conversion, compilation, interpretive, output, and post-mortem routines. The various parts of CS, except output, are normally left in the four groups of drum #2. These programs are not protected from destruction, accidental or intentional, except by social pressure. Consequently, the group 11 program checks, when CS is needed, the sum-mod-one of groups 12-15 to ascertain whether the CS routines are in place; and, if not, the proper routines are searched for in tape unit 0 and copied into groups 12-15 via core memory.

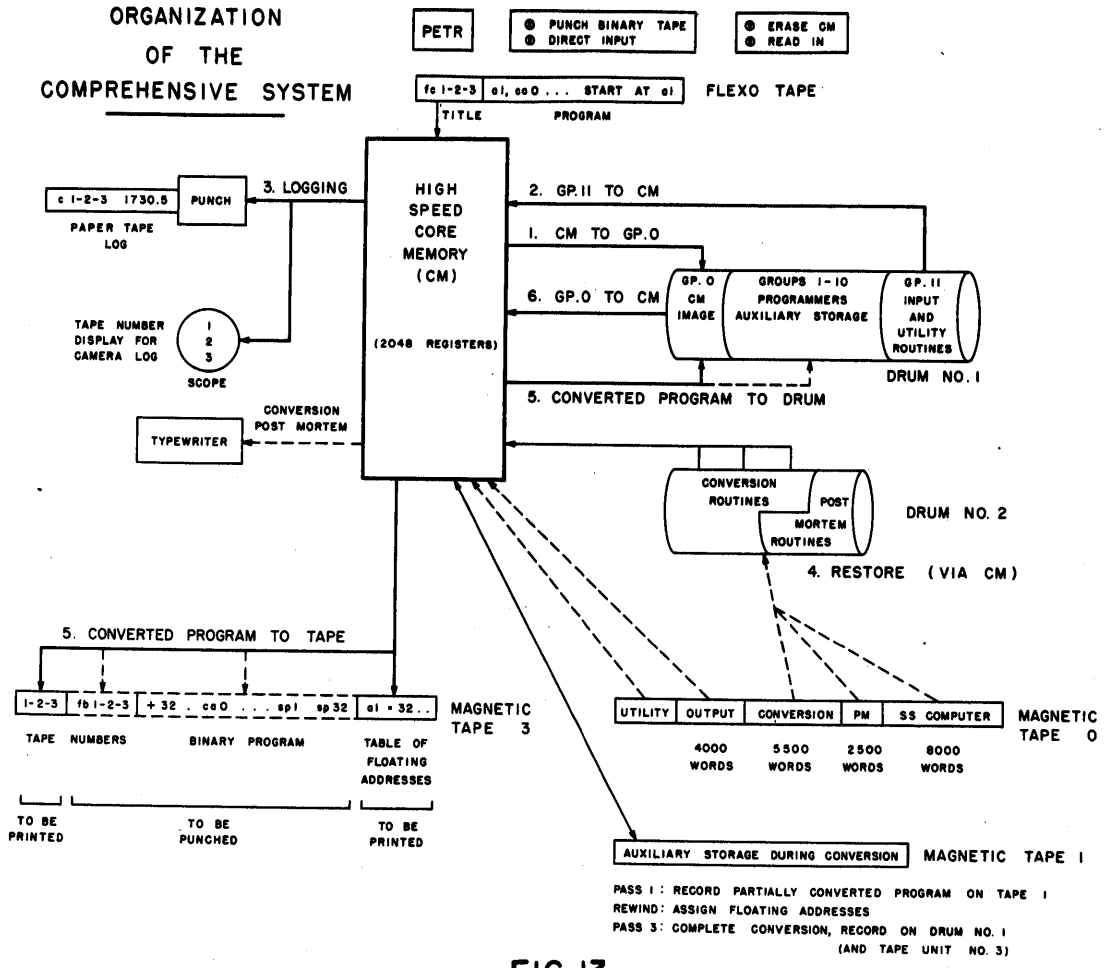


FIG. 13

As the Flexowriter tape is read-in, it is partially converted and stored on tape unit 1. The floating addresses require a second pass over this tape to fill in the assigned values. For historical reasons (when storage was more limited), the floating addresses are assembled and assigned on a second pass (done in the reverse direction), and the final assignment is done on a third (forward) pass. The resulting converted program is stored on drum groups 0 to 10 as required by the program itself, with group 0 serving to store what will ultimately occupy core memory after the CS process is complete. If a binary tape is desired, the same program is also recorded on tape unit 3 in a form suitable for later punching. The interpretive and output routines needed, according to information gathered on the second pass, are assembled from drum #2 and tape #0 respectively before the third pass and stored in group 0 (and on tape 3) as part of the program being converted. Finally, group 0 (including the proper interpretive and output routines) is copied into core memory and the program is ready to be performed. Meanwhile, if wanted, tape unit 3 has the complete program ready to be punched as a binary tape.

Algebraic System

An algebraic system is any system of notation which approximates standard mathematical notation. Any such system is faced with three major difficulties. These are that the conventions adopted, while being comprehensive and unrestrictive, must be

1. not misinterpretable--and some mathematical notation is suprisingly ambiguous (e.g., a/b c).
2. producible on a typewriter--and here not much can be done until a typewriter is built with a two-directional, half-space line-feed and a few extra double-height symbols.
3. realizable by converters, compilers, and/or interpreters produced in a reasonable time.

Even without regard for computational efficiency, these difficulties are of major proportions. Since the third is a matter of economics and ambition rather than principle, perhaps only the first two are of fundamental interest; but the third will always have its effect.

The system of Laning and Zieler shows much of the power that the algebraic technique promises, and at the same time illustrates some of the compromises one may be forced to make. The system is mechanized by a compilation of closed subroutines entered from blocks of words, each block representing one equation. The sequence of equations is stored on the drum, and each is called in separately every time it is used. The compiled routine is then performed interpretively using the CS routine.

The use of a small number of standard closed subroutines has certain advantages of logical simplicity; however, it also often results in the execution of numerous unnecessary operations. This fact, plus the frequent reference to the drum required in calling in equations, results in a reduction of computing speed of the order of magnitude of ten to one from an efficient computer program. It should be pointed out, however, that the algebraic system was started at a time when computer storage consisted solely of 1024 registers. Many of the reasons for the present slow speed can be traced directly to this fact and the subsequent steps through which the computer evolved. There appears to be no intrinsic reason for believing that a speed ratio of two to one or considerably better cannot easily be obtained.

The notation used is for the most part simple, natural, and powerful. A complete description of all the conventions is, of course, somewhat too long to present here. The examples in figure 14 (page 21) show most of the high and low points of the system.

Basically, one writes a series of equations expressing a variable, represented by any lower case letter, in terms of constants and other previously evaluated variables. In fact, as the fifth line

EXAMPLES OF ALGEBRAIC NOTATION used by LANING and ZIERLER

Information to be coded	Program as typed, ready for compiler-interpreter
Program to evaluate an arbitrary function z(x, y) as shown, for x = 173.972 y = 15, 16, . . . , 30	x=173.972, y=15, 1 z=x+y|2/(7x-y(x|3+7+3x|-3)|15), PRINT y,z, y=y+1, c=y-30.5, CP1, STOP,
to copy an equation at another location:	SR3
variable with a subscript:	u| 3=x|2+y|2-6.3,
a running index:	v| j=x+j|2, j=j+1,
a stored table of values:	g| N=1,1.2,1.4,1.6,1.8,2,3,4,5,
or:	g| N=1(.2)2(1)5,
evaluate $q = \sqrt{\lvert y+3-\tan(y+\cos^{-1}z)\rvert}$:	q=F1(y+F11(3-F4(y+F6(z)))),
one time step h in $dy_3/dt = t+y_2$ by the method of Gill:	Dy| 3=t+y| 2,

FIG. 14

illustrates, the equality sign is not read as "equals" but as "replaces". The system works ideally for any normal algebraic equation using plus, minus, times (implied or explicit), divided by (using a shilling fraction), and integer exponents, together with any necessary associativity relationships indicated by parentheses.

Since the alphabet does not provide enough symbols for the variables in many problems, some sort of subscript notation is required. The choice made, since actual subscripts are not available on M.I.T. Flexowriters, was to use superscripts (exponents) preceded by a vertical bar. The vertical bar is not used as a magnitude sign because it would not be unambiguous (e.g., |a+b| cc |d+e|). Common functions (magnitude, roots, log, exponential, trigonometric, inverse trig, and hyperbolic) are easily expressed, as in the next-to-bottom example, by writing a capital F with a superscript chosen from a catalog (obviously, a several-letter mnemonic abbreviation could have been used but did not seem worth the trouble, especially in view of the increased

difficulty entailed thereby in any adding of new functions). Differential equations of any order can be written as a series of first-order equations in the form of the bottom example. The series of equations is automatically integrated by a modified Runge-Kutta method. Printing is done in a standard floating-point format with no editing possible.

Repeating a calculation with new parameter values or performing an iterative process, is made possible by the introduction of a conditional, and an unconditional, transfer of control. Any equation can be numbered with any number from 1 to 100, without regard for sequence, in the way that the third equation in the examples is numbered with a 1. Then a CP followed by a number will transfer control to the correspondingly numbered equation if the previously computed quantity was negative. An SP is an unconditional transfer. The symbols CR (or SR) are used for interrupting the sequence by the conditional (or unconditional) insertion of a single numbered equation, after which the equation following the CR or SR, not that following the inserted equation, is executed. In the example program shown, a criterion c=y-30.5 is formed. The reason for using 30.5 rather than 30 or 31 is, of course, to avoid any possible difficulty with zero. No difficulty is likely in this case, involving integers, but 24,6 floating binary will round any decimal fractions to the nearest 24-bit binary fraction, so that in CS, for example, the quantity 30 - 300(0.1) will not equal zero, and its sign cannot a priori be determined.

Cyclic processes are facilitated and tabulated-value processes are made possible by a variable index notation in which any of the single letter variables can be used as an index. Thus, $v|j$ corresponds to $v|^6$ if $j = 6$, or 6.49 or 5.51, and to $v|^7$ if $j = 6.5 \leq j < 7.5$, etc. The same indexing procedure applies for SPj, etc. To simplify the storing of a table of values initially, the notation $g|N$ may be used for any lower case variable. As many values as are wanted can be written either in a sequence separated by commas or in standard table-maker notation for equal increments. Consequently, the twelfth line in the example would serve the same purpose as the 9 equations $g|^1 = 1$, $g|^2 = 1.2, \ldots\ldots, g|^9 = 5$. The line below gives the same result using the table notation (which reads: 1 to 2 in steps of 0.2, 2 to 5 in steps of 1.)

There are in the algebraic system no very adequate facilities for mistake diagnosis, other than to determine the cause of any arithmetic (floating-point) overflows. Dr. Laning has not felt a great need for powerful diagnostic techniques since very few mistakes occur.

Conclusion

The three systems of semi-automatic coding in use in the M.I.T. Digital Computer Laboratory have been superficially described. Each of them serves a useful purpose and demonstrates some interesting facts.

For reasons of efficiency and versatility, the Comprehensive System of Service Routines is the system ordinarily used. Much remains to be done to make it more useful and more fool proof. The Summer Session computer, though inefficient, is practically fool proof, versatile, and typical of trends in available computers, and is therefore useful for classroom work. Mistakes made in using it are fewer and easier to find than mistakes made in CS, where much machine-code is used as well. The algebraic system has been used for several problems, and has set new records for the small number of mistakes made in using it. Making it sufficiently efficient in terms of computer operations and sufficiently versatile in its logical operations will require considerably more work. All of these systems have the same final goal, and differ only in the emphasis on efficiency, etc. At present no one of them can be called the best in general, since each has its own special advantages.

It seems apparent that improved versions of all machine-like systems will grow to look more and more alike, and that they will ultimately give way to algebraic notation, at least in the calculational phases of a problem solution. Whether machine-like or man-like terminology will triumph in the logical aspects (cycling, selecting, filing, editing, etc.) remains to be seen. At present, since there really is no satisfactory man-like terminology for the most ticklish (and most important) logical sequences, it seems possible that here the man may wish to adopt the machine language as his own, if the machine language can be simplified and standardized enough. There is, in other words, no real conflict here between man and machine--merely the problem of choosing a consistent notation convenient for the man and acceptable to the machine. The present symposium itself, and all the papers and discussions in it, attest to the significance of the problem and indicate that better and better solutions will rapidly be reached.

Signed C.W. Adams
C.W. Adams

Approved J.W. Forrester
J.W. Forrester

CWA:cf