X-542-64-148

TM X-55074

CAMEO

COMPUTER-INDEPENDENT ABSTRACT MACHINE-LANGUAGE ENCODER AND OPERATING-SYSTEM

SYSTEM DESCRIPTION

JUNE 1964

NASA

GODDARD SPACE FLIGHT CENTER

GREENBELT, MARYLAND

CAMEO

Computer-independent Abstract

Machine-language Encoder and Operating-system

System Description

T. P. Gorman

CONTENTS

	Page
INTRODUCTION	1
NOMENCLATURE	6
THE ABSTRACT MACHINE	8
THE ENCODER	8
PROBLEM PREPARATION AND PROGRAM CHECKOUT	9
A TYPICAL PROGRAM	17
SYSTEM PERFORMANCE	29
CONCLUSION	34
ACKNOWLEDGEMENTS	34

INTRODUCTION

Computing machines cannot produce their own programs, nor can programmers perform the calculations they program. The problem can be posed and the results interpreted by neither, but rather by the problem sponsor who, on a problem of any size, is neither qualified to write the program nor able to do the calculation. In other words, a program is produced by a composite system made up of men and a machine. Since this paper deals with a technique for making this system function more effectively, it begins with a simplified analysis of the complete process.

In the original crude form of the programming process (Figure 1), the programmer is trained to read the problem language (PL) and write the machine language (ML). Only two disciplines are required, but the two are far removed from each other for all but a few types of problems. Furthermore, ML is often extremely awkward to use because of its necessary but unnatural rigor and complexity. Finally, since machine improvement and problem improvement proceed independently but more or less continuously, representation of a problem in ML frequently results in a choice between inevitable obsolescence and inopportune recording.

Early attempts to relieve some of the awkwardness of the usual ML produced exact simulations on the given computer of other more convenient "machines" in the form of interpretive programs. Such simulations were more than adequately effective in reducing both the cost and the delay involved in producing running programs. They sacrificed too much running speed, however, to win final acceptance and were finally abandoned.

Figure 2 shows a later, more sophisticated system. An assembly program has been introduced which accepts a computer oriented language (COL) and converts it to the required ML. Of course, COL is easier to use than ML and the above-mentioned awkwardness due to rigor has been removed. The complexity remains, however, and COL is no less remote from PL. Moreover, a compensatory disadvantage has appeared in the form of another system to master. The programmer, though he now writes in COL, still must know the ML to interpret the results of his program tests. Training now costs somewhat more because it involves more. A program will

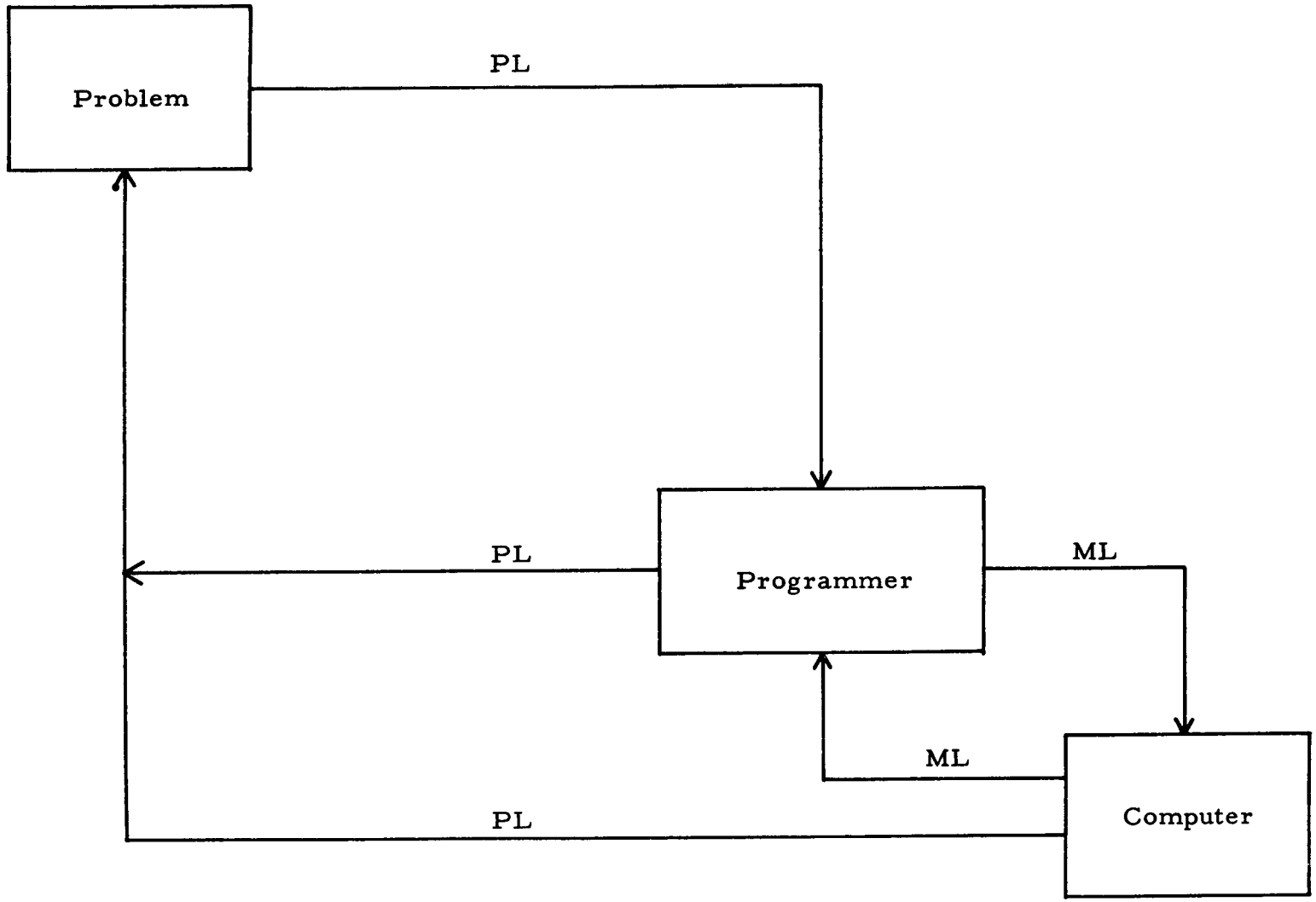

Figure 1

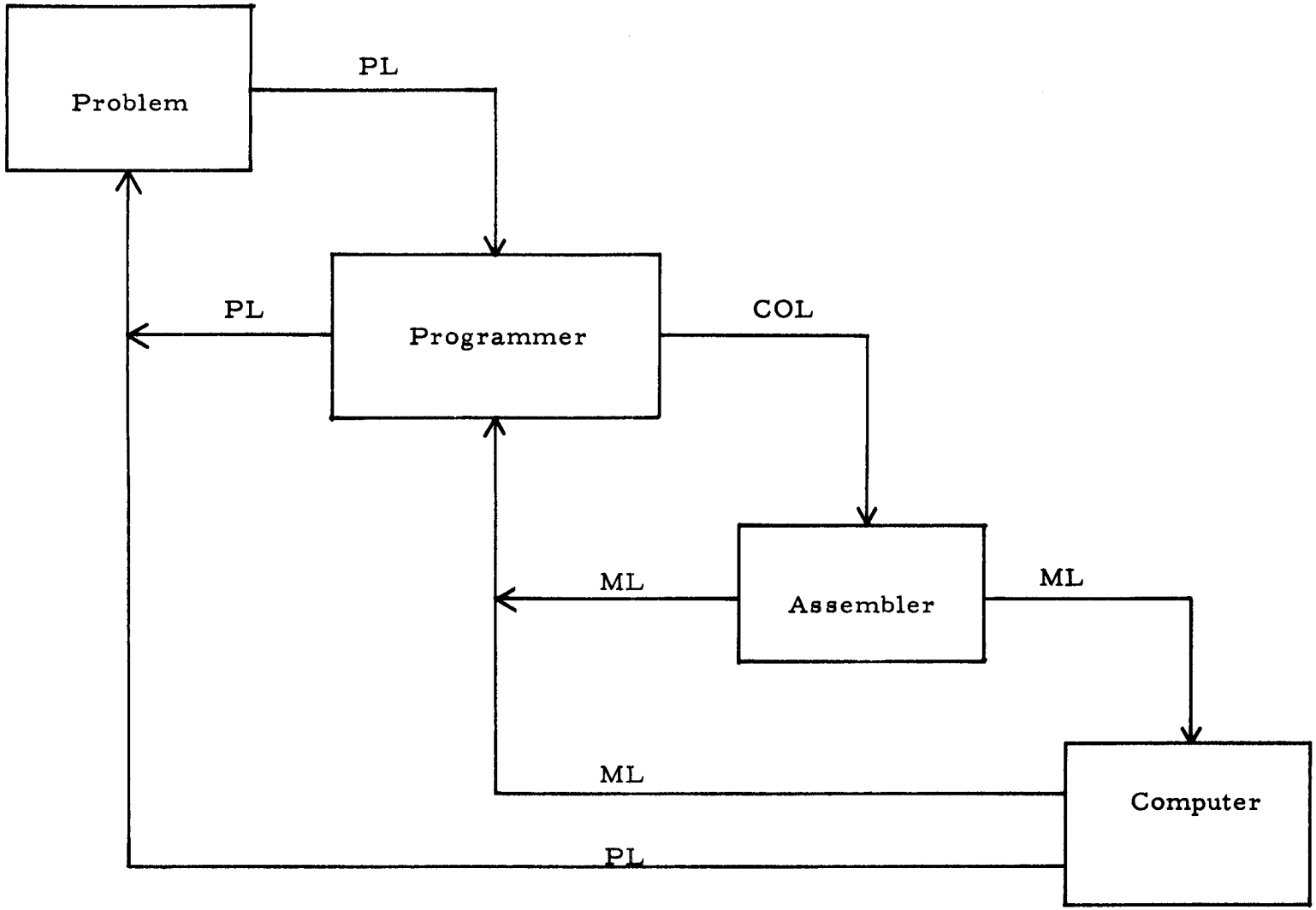

Figure 2

not work unless both the COL processor (a program) and the ML processor (a computer) are used properly, so that the incidence of programmer blunders can be expected to increase somewhat along with the average turn-around time. The programmer nonetheless reads and writes two closely related dialects of the same basic language and can reasonably be expected to become proficient at both, so that the net result is a state of relative balance with COL a little easier to write than ML is to read.

In an effort to remove the complexity still inherent in the COL, a second addition to the process was made as shown in Figure 3. A problem oriented language (POL) was introduced to produce COL to produce ML. This step, far from correcting the previous slight imbalance in favor of the less significant activity of program writing, only makes it worse. The programmer using such a system must sooner or later master four techniques: he must be able to read PL, write POL, read and write COL and read ML. The probability of programmer errors, spread now over three processors, increases significantly. Since the object of programmer training is to reduce this probability to a satisfactory level, there is a concomitant increase in its cost. Among even experienced programmers, fewer and fewer find it possible to become even moderately expert in the now fourfold complexities of their craft. Under such circumstances the net improvement in the productivity of the whole program production system is likely to be negligible.

It is suggested here, on the other hand, that the responsiveness of the program production man-machine system can be greatly improved and that those early attempts to achieve this objective by creating more convenient pseudo-computers were based on sound principle. They used a notation appropriate to the problem class of interest, and a straightforward operation-by-operation approach. They represented, in addition, a balanced aid to programming: the user learned the pseudo-machine in place of the true machine, not in addition to it.

The serious inefficiency in the execution of programs written for the early pseudo-machines was not the result of any essential quality of the programming language used. It was due rather to the fact that the simulation was unnecessarily exact, faithfully reproducing even such functions as instruction fetching and operation decoding. Since a

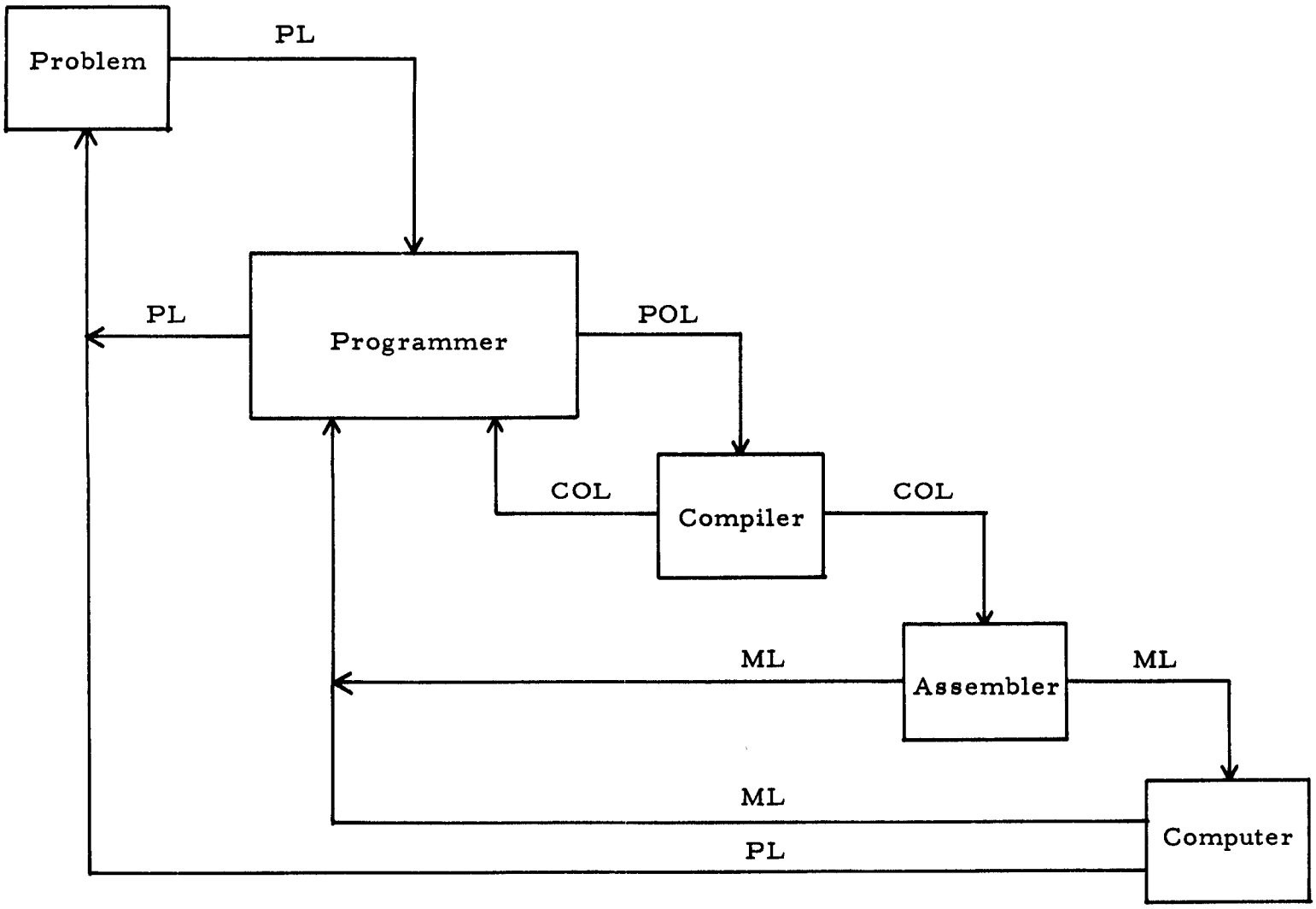
Problem
PL
PL
Programmer
POL
COL
Compiler
COL
ML
Assembler
ML
ML
Computer
PL

Figure 3

pseudo-machine is simply a convenient abstraction, it is by no means necessary to simulate it when it can easily be approximated by a program generator capable of producing more than adequately efficient code. If there is a need to abandon the elegant set theoretic notation of the classic computing machine for the chaos of pidgin-algebra, it does not arise from the requirement for efficient object codes.

This paper describes an abstract problem oriented machine (APOM) which restores to program production the balance and simplicity illustrated in Figure 4. As in Figure 1, the programmer once more need master but two disciplines: PL, as before, and abstract machine language (AML). Furthermore, the steady-state process of program production using an APOM leads naturally to a convenient division of labor, and thence to a remarkably flexible responsiveness. Groups or teams, each formed of experts in one of the three disciplines, PL, AML, ML, can be put to work in parallel on large problems with a consequent increase in productivity and reduction of lead time. More important is the fact that such a division of labor creates the practical capacity for asynchronous response to independent requirements, which include both changes in the problems presented and changes in the machines available. Almost four years of experience with the use of an APOM on a wide variety of problems will be cited to testify to these advantages.

NOMENCLATURE

For the sake of clarity several terms used later will be defined here. The APOM will be called the abstract machine while the machine on which it is approximated will be referred to as the underlying computer. The former will be said to execute commands, the latter instructions. A program which approximates commands by instructions is here called an encoder. The system to be described is called CAMEO, which is an acronym for "Computer-independent Abstract Machine-language Encoder and Operating-system." The abstract machine under discussion is called the Advanced Mystic, while the underlying computers are an IBM-7094 with two banks of core storage, and the Univac 1107.

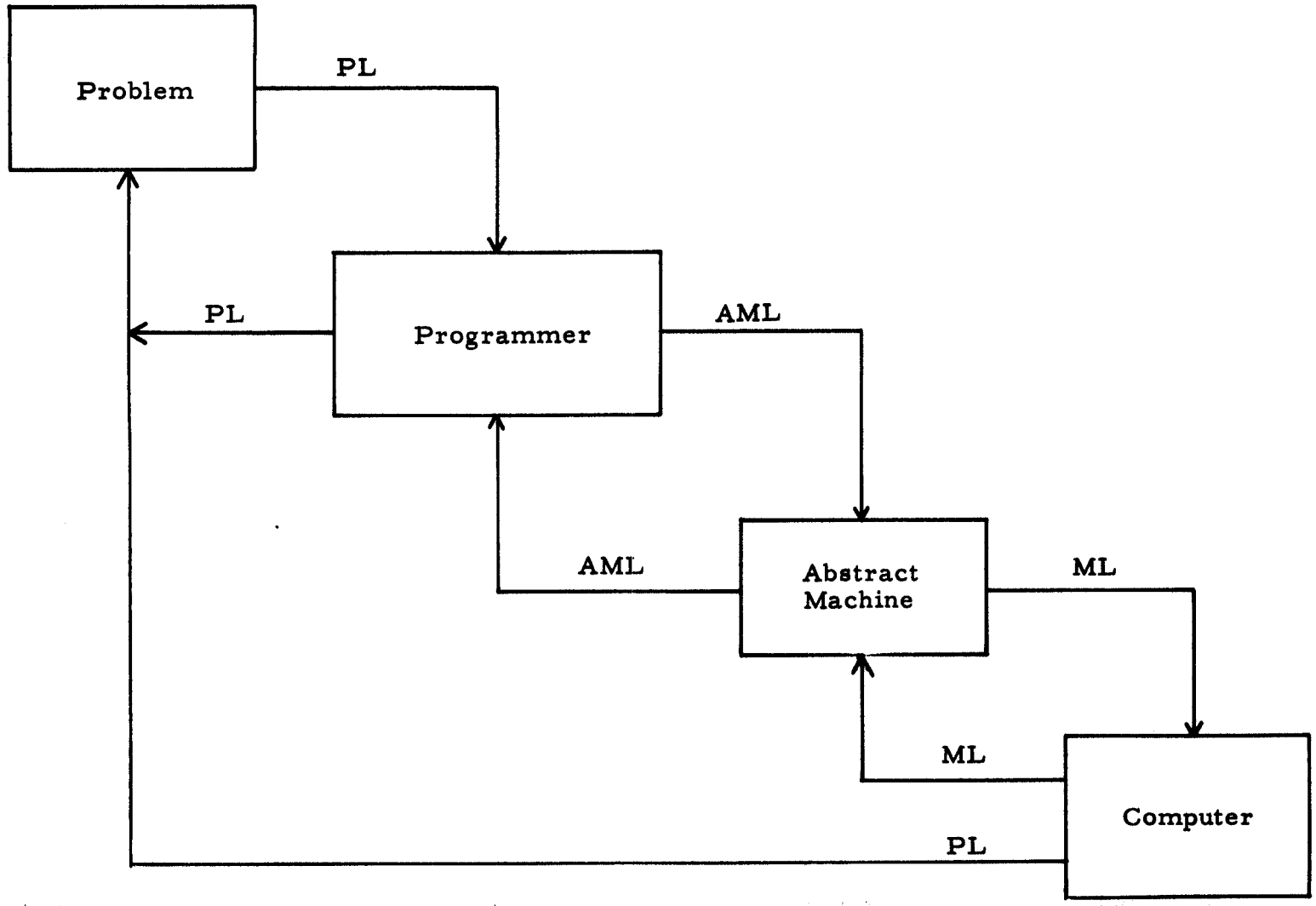
Problem
PL
Programmer
PL
AML
Abstract
Machine
AML
ML
Computer
ML
PL

Figure 4

THE ABSTRACT MACHINE

The Advanced Mystic is a multiple address, floating point, fixed program decimal computing "machine" recognizing eighteen separate commands. It utilizes two forms of memory: a fixed non-addressable operator memory for the program, and a modifiable addressable operand memory for all numbers, symbols and flow connectors. Any location in addressable memory can serve any of these functions and the distribution of underlying computer memory to these functions, while normally two-to-one operator-to-operand, can be reset to suit problem needs.

Input and output are buffered and handle integers only, so as to assure the possibility of exact conversion of base. Alphabetic characters are carried in the form of decimal integers as well, being represented in the base one-hundred system of Table 1.

No address modification is possible and individual commands are not externally associated with underlying or abstract machine locations. The Advanced Mystic is rather so designed that any cell in operand memory can serve as an index register for any other, and program logic can be altered dynamically by modifying the contents of those operand cells which serve as flow connectors. It is, in addition, equipped with a "pathfinder" cell which retains the address of the last used flow connector to aid in diagnosing unexpected halts.

THE ENCODER

Even though the Advanced Mystic is an abstract machine, the encoding process for it is little different from what a real computer would require. It amounts to readying the computer for solving the problem at hand by filling operator storage with the proper coding, and operand storage with the required constants, symbols and flow connectors.

The input to the CAMEO encoder takes the form of a 72 character operator record. The first (leftmost) character of this record represents the operator code. Each of the 48 recognizable symbols can serve as the operator code, and each of these can have associated with it a selection of arrangements of the remaining 71 characters as addresses and parameters of varying size, type, and function.

In all, 25 standard operation codes are recognized by CAMEO. These serve three purposes: four are used to control encoding, three to support encoding, and the remaining eighteen represent the commands of the Advanced Mystic machine. Two supplemental control operators, defined differently for different underlying computers, are described in the CAMEO manual for each.

The encoding process is controlled by three control registers: the current operator location register L, the key address register K, and the equivalent address table $Q_{1,2,\ldots 100}$. As each underlying computer instruction is generated to represent each operator, it is stored successively at the location identified by the contents of L, after which L is stepped by one to be ready for the next instruction. As each operator is encountered, each address is compared against the address equivalence pairs of the Q table register. If a match is found, the address is replaced by its equivalent. If no match is found, the address is increased by the current value of the K register. The encoder operations which affect the contents of these registers are described in Table 2.

Encoding receives necessary support from CAMEO operators which affect only operand storage. These operators serve to define logical flow-connectors and to set operand locations to initial or constant numeric or alphabetic values. The effect of these operators is described in Table 3.

When the operator is one of the eighteen Advanced Mystic machine commands, underlying computer instructions appropriate to the command are generated and stored in successive locations in operator memory. These operators are described in Table 4.

PROBLEM PREPARATION AND PROGRAM CHECKOUT

The preparation of a problem for solution is broken down into successive stages. The first step consists in reducing the problem to a series of procedural steps, usually by way of a program plan or flow chart. Instead, of course, a symbolic program may be written. This is a program in which at least some of the instructions have symbolic

Table 1

CAMEO Codes for Alphanumeric Characters

Character	Card Code	Octal Code	CAMEO Code
Blank	Blank	60	00
.	12-3-8	33	18
)	12-4-8	34	19
+	12	20	20
$	11-3-8	53	28
*	11-4-8	54	29
-	11	40	30
/	0-1	61	31
,	0-3-8	73	38
(	0-4-8	74	39
=	3-8	72	48
'	4-8	14	49
A	12-1	21	61
B	12-2	22	62
C	12-3	23	63
D	12-4	24	64
E	12-5	25	65
F	12-6	26	66
G	12-7	27	67
H	12-8	30	68
I	12-9	31	69
J	11-1	41	71
K	11-2	42	72
L	11-3	43	73
M	11-4	44	74
N	11-5	45	75
O	11-6	46	76
P	11-7	47	77
Q	11-8	50	78
R	11-9	51	79
S	0-2	62	82
T	0-3	63	83
U	0-4	64	84
V	0-5	65	85
W	0-6	66	86

Table 1 (Cont'd)

Character	Card Code	Octal Code	CAMEO Code
X	0-7	67	87
Y	0-8	70	88
Z	0-9	71	89
Zero	0	00	90
1	1	01	91
2	2	02	92
3	3	03	93
4	4	04	94
5	5	05	95
6	6	06	96
7	7	07	97
8	8	10	98
9	9	11	99

Table 2

CAMEO Operators to Control Encoding for the Advanced Mystic Machine

Name	Symbol	Effect on Encoding Control Registers
Key	K(0)	Clear the Q-table of all previous entries and set the K register to zero.
Key	K(p)	Add the number p to the K register.
Origin	O(p)	Set the L register to the number p.
Cue	Q(p, q)	Add the pair p, q to the Q-table of address equivalents.
Transfer	30(p)	Terminate encoding and begin execution of the compiled program at location p.

Table 3

CAMEO Operators to Support Encoding for the Advanced Mystic Machine

Name	Symbol	Function in Support of Encoding
Begin-point	B(p)	Make location p a logical flow connector for the commands which follow.
Value-given	V(p, m, n)	Record in location p the floating point number (m, n) for use as a given value in the object program.
Word-given	W(p, l)	Record in location p the symbol l as a coded floating point integer for use as a given word in the object program.

Table 4

CAMEO Operators Representing Advanced Mystic Commands

Name	Symbol	Advanced Mystic Command
Add	A (p, q, r)	Add the contents of locations q and r and place the sum into location p.
Compare_1	C (p, q, r, s)	Compare the contents of location p with the contents of location q. If contents-of-p exceeds contents-of-q transfer to location r, if contents-of-q exceeds contents-of-p transfer to location s, if contents-of-p equals contents-of-q continue with next instruction.

Table 4 (Cont'd)

Name	Symbol	Advanced Mystic Command
$Compare_2$	C (p, q, r)	Compare the contents of p and the contents of q. If contents-of-p exceeds contents-of-q, transfer to location r. Otherwise continue.
Divide	D (p, q, r)	Divide the contents of location q by the contents of location r and place the quotient into location p.
End	E (p)	Exit from the current instruction sequence by transferring to location p.
Function (Note 1)	F (p, q, r)	Store in location q + 1 the point-of-return, in cell q + 2 the number r - q, in cell q + 3 the number p - q, and transfer to location q.
Get	G (p, q, r)	Get into location p the contents of the location specified by the number q plus the contents of location r.
Hold	H (p, q, r)	Hold the contents of location r in the location specified by the number p plus the contents of location q.
Initialize	I (p, m, n)	Initialize location p to the value (m, n) a normalized floating point number.
Jump	J (p)	Jump to the program in system storage designated by the contents of location p.
$Load_1$ (Notes 2, 4)	L (p, q, a, b, $c_1 \ldots c_{18}$, $d_1 \ldots d_{18}$)	Load into successive locations beginning with p, contents-of-q records from the alphanumeric input medium indicated by a; where the i-th word in each record is the integer equivalent of an input field, c_i characters long, of type d_i, and b such sets of field descriptors follow in succeeding command records.

Table 4 (Cont'd)

Name	Symbol	Advanced Mystic Command
$Load_2$ (Notes 2, 4)	L (p,q,r)	Load into successive locations beginning with p, contents-of-q words from the machine-word input medium indicated by r.
Multiply	M (p,q,r)	Multiply the contents of locations q and r and place the product into location p.
Name	N (p)	Name location p a logical flow connector for the coding which follows.
$Print_1$ (Notes 3,4,5)	P (p,q,a,b, $c_1 \ldots c_{18}$, $d_1 \ldots d_{18}$)	Print from successive locations beginning with p, contents-of-q entitled records on the alphanumeric medium indicated by a; where the i-th field of each record, c_i characters long, of type d_i is determined from the i-th integer of the record, and b such sets of field descriptors follow in succeeding command records.
$Print_2$	P (p, q, r)	Print from successive locations beginning with p, contents-of-q words on the machine-word-output medium indicated by r.
Replace	R (p, q)	Replace the contents of location p by the contents of location q.
Subtract	S (p,q,r)	Subtract the contents of location r from the contents of location q and place remainder into location p.
Title	T ($t_1 \ldots t_{71}$)	Load the title register positions 1-71 with characters t_1 to t_{71}.
Unpack	U (p,q)	Unpack the integer portion of the floating point number stored in location q and store the integer in location p. The contents of location q remain unaltered.
Xtracode	X (p,$x_1 \ldots x_{13}$)	Transfer to the machine language subroutine located at p, with interface vector $x_1 \ldots x_{13}$.

Table 4 (Cont'd)

(Note 1)	This instruction makes it possible to transfer to a function and after its execution, continue to the next instruction. Normally r contains the input to the function and p is to contain the output.
(Note 2)	In case the input medium is tape, a special interpretation is placed on the contents of q as follows: if q is zero backspace one file, if q is the negative integer -n, backspace n records.
(Note 3)	In case the output medium is tape, a special interpretation is placed on the contents of q as follows: if a is zero, write end-of-file; if q is negative, rewind.
(Note 4)	The field a contains four characters. The leftmost is one of (C, P, T) for Card, Printer, Tape, respectively. The next is one of (A, B, C, D, E, F, G, H, I) for selection of units within the type. The next is either blank or B, for decimal or (Binary) machine-word, respectively. The field $c_1 \ldots c_{18}$ consists of eighteen two-digit numbers. The field $d_1 \ldots d_{18}$ consists of eighteen letters, where each is one of (A, N, F, S) for Alphabetic, Numeric, Full-numeric and Skipped. In case d_i is A, c_i must not exceed 4; in case d_i is N, c_i must not exceed 9; in case d_i is F, c_i must not exceed 8; when d_i is S, c_i may be as large as 15.
(Note 5)	An entitled record is the logical sum of the given record and the contents of the title register.

addresses. The variables of the problem are then, in any case, each assigned a box on a memory map; the form used for this purpose is shown in Table 5. Finally, the actual program is written on a suitable form and punched according to the directions in Table 6. Computer input in the form of a program deck is the final result and the program is ready for testing.

Programs to solve problems of any significance are generally decomposable into several sub-programs. In the CAMEO system, sub-programs are written as virtually independent programs to be assembled for final checkout. Each subroutine is mapped for the small number addresses, and addresses to be replaced at assembly are usually given illegal values to make possible automatic detection of overlooked assignments. Finally, an executive routine is written to bind all the

Table 5

ADVANCED MYSTIC STORAGE MAP

PAGE_____OF_____

PROGRAM: _____

_____

PROGRAMMER : _____

00	00	01	02	03	04
05	05	06	07	08	09
10	10	11	12	13	14
15	15	16	17	18	19
20	20	21	22	23	24
25	25	26	27	28	29
30	30	31	32	33	34
35	35	36	37	38	39
40	40	41	42	43	44
45	45	46	47	48	49
50	50	51	52	53	54
55	55	56	57	58	59
60	60	61	62	63	64
65	65	66	67	68	69
70	70	71	72	73	74
75	75	76	77	78	79
80	80	81	82	83	84
85	85	86	87	88	89
90	90	91	92	93	94
95	95	96	97	98	99

NOTES :

subroutines into the master problem solving routine. Each subroutine is first checked separately, then the whole routine is built up as it is subjected to successively more comprehensive systems checks. The K and Q commands of the CAMEO system make the entire process straightforward and trouble free.

When a program or sub-program is ready for testing, it is recorded on magnetic tape and, with suitable input, a test run is conducted. The program is encoded and executed and a print of memory is sent to magnetic tape. Since only operand storage is of any significance, only this is printed; and since all numbers are in floating point form, only floating point decimal output is obtained. This program also reads out the pathfinder cell to help locate unexpected troubles.

While the memory print is the basic checkout tool, other more sophisticated ones are available. These include an interval core dump in which specified memory locations are printed prior to the execution of specified logical connectors, and a tracing program in CAMEO language. Experienced use of these programs drastically reduces the number of records read out, and this reduction saves both computer time and man-hours with no loss of responsiveness or speed of program production.

A TYPICAL PROBLEM

The memory print routine used for checkout support in CAMEO is an Advanced Mystic program and a fairly representative one. It is written as a subroutine which calls for a subroutine, which in turn calls for a subroutine. Output of various kinds is produced and a certain amount of logical, as well as arithmetic manipulation is involved. Such a routine, of course, can not be called computer-independent since computer characteristics create its necessity and influence its logic. It is, however, much more nearly independent when written in a computer-independent language since it is applicable unchanged to computers similar to the IBM 7094 in number base and floating point conventions, and requires only trivial changes for application to computers dissimilar to the IBM 7094 in these respects.

The general objective of the program is to print a specified set of consecutive storage registers in the form of floating point decimal numbers. Each line is to consist of five such numbers and a line locator which identifies the storage register from which the leftmost number was taken. Other numbers in the line are understood to occupy

Table 6
CAMEO Punching Instructions

I. Field-type Index by Commands

A pqr	I pmn	P_2 pqr
B p	J p	Q pq
C_1 pqr	K p	R pq
C_2 pqrs	L_1 pqabcd	S pqr
D pqr	L_2 pqr	T t
E p	M pqr	U pq
F pqr	N p	V pmn
G pqr	O p	W pl
H pqr	P_1 pqabcd	X px

II. Field Descriptions by Field Types

Type	Length	Description
a	4	Alphabetic, left justified, blank filled
b	1	unsigned numeric
c	36	unsigned numeric, left justified, blank filled, representing eighteen two-digit numbers.
d	18	alphabetic, left justified, blank filled, representing eighteen one-character words.
l	4	alphabetic, left justified, blank filled.
m	9	signed numeric
n	3	signed numeric
p	5	unsigned numeric
q	5	unsigned numeric
r	5	unsigned numeric
s	5	unsigned numeric
t	71	alphanumeric, left justified, blank filled.
x	65	unsigned numeric, left justified, zero filled, representing thirteen five-digit numbers.

the following four consecutive registers. The printed set of registers is bounded from below and above by the integer contents of the two cells usually used to locate function argument and result.

The conversion process amounts to finding a pair of integers (D, d) such that for a given binary number B, we have

$$D \times 10^{-8} \times 10^{d} = B \qquad \text{where } 10^{7} \leq D < 10^{8}.$$

But this implies

$$D \times 10^{-8} = B \times 10^{-d} \qquad \text{and} \quad .1 \leq D \times 10^{-8},$$

so that

$$.1 \leq B \times 10^{-d} < 1 \qquad \text{and} \quad 10^{d-1} \leq B < 10^{d},$$

which defines d. Then

$$D = B \times 10^{8-d}.$$

In computers similar to the IBM 7094, the magnitude of B must be greater than 2 to the -129 but less than 2 to the 128. This in turn requires d to lie between -38 and +38, which means 8-d can be as large as 46. The inability of the computer to represent 10 to the (8-d) when (8-d) exceeds 38 is circumvented by carrying the powers of ten from 0 to 46 divided by 2 to the 27. This power of 2 is eliminated from D in the final step without, however, introducing any error since this operation effects an adjustment in the exponent alone, leaving the mantissa unchanged.

Table 7 represents the plan of the print routine. In step 2 a test is made to see whether the subroutine has been entered by a call from a function command or by an emergency manual jump. Initial and final values of the line counter are set in step 4 or 5. The loop begins at step 6 and runs to step 13. At step 9, printing is skipped if every word of the line is zero. Step 14 restores the argument cell to its nominal value, thereby anticipating the test of step 1 at the next entry.

The memory map with assignments for this subroutine appears in Table 8. Note that while the program is written as though it used locations 1-30, through the CAMEO key operator it can be assigned to occupy any 30 consecutive locations anywhere in operand storage.

Table 7

Program Plan for
Subroutine: Print-out Memory Control

Start Print:	1. Print Pathfinder cell and skip one line. 2. If the test word is not nominal, go to connector (1). 3. Set "End Print" to "Final Halt." 4. Set loop bounds to nominal values, go to connector (2).
Connector (1)	5. Set loop bounds from tag #1 and tag #2.
Connector (2)	6. Bump word counter by five. 7. If the next line is beyond the last line requested, go to connector (4). 8. Retrieve five consecutive words from storage. 9. If any of the words is not zero, go to connector (3). 10. Go to connector (2).
Connector (3)	11. Convert five words to integer pairs.* 12. Print word counter and five integer pairs. 13. Go to connector (2).
Connector (4)	14. Set the test word to its nominal value. 15. Go to "End print."

*Subroutine: Convert Binary to Decimal (see Table 10).

Table 8

ADVANCED MYSTIC STORAGE MAP

PAGE 1 OF 1

PROGRAM: Print-out Memory Control

PROGRAMMER: T. P. Gorman

00	00	Start Print 01	End Print 02	Tag #1 03	Tag #2 04
05	Connector (1) 05	Connector (2) 06	Connector (3) 07	Connector (4) 08	09
10	zero 10	one 11	12	13	14
15	five 15	16	Nominal Lo-bound 17	Nominal Hi-bound 18	Nominal Test-word 19
20	Word counter 20	1st word of line 21	2nd word of line 22	. . . 23	. . . 24
25	. . . 25	. . . 26	. . . 27	. . . 28	. . . 29
30	10th word of line 30	31	32	33	34
35	35	36	37	38	39

The code is shown in Table 9. The Q-operators allow necessary access to absolute locations 0-4 regardless of the area of storage assigned to the subprogram. The function commands refer to a location beyond the end of the program. This location is filled by the next subroutine to be discussed: the binary-to-decimal conversion subroutine.

The plan for this Binary to Decimal Conversion subroutine appears in Table 10, the assignments in Table 11, and the code in Table 12. Preceded by a K30 operator, this subroutine is made to begin just after the end of the printout memory subroutine.

Steps 1-10 of the plan generate the required table of powers of ten the first time the routine is entered. All subsequent entries are then made to occur at-step 11. Steps 11-14 split the argument, a binary number B, into sign and magnitude and store a zero result-pair for a zero input. If, at step 19, the magnitude of the input number exceeds

Table 9

CAMEO Code for
Subroutine: Print-out Memory

```
 1.  Q  90000        0
 2.  Q  90001        1
 3.  Q  90002        2
 4.  Q  90003        3
 5.  Q  90004        4
 6.  Q  90010    30000
 7.  B      1
 8.  T  PATHFINDER READS
 9.  P  90010       11     TI  150106
10.  T                        S S N
11.  P      0      11     TI
12.  C      3      19      5        5   Skip if specified print.
13.  R      2      10
14.  R      3      18                  Set up for nominal print.
15.  S     20      17     15
16.  E      6
17.  B      5
18.  S     20       4     15
19.  B      6
20.  A     20      20     15           Bump line counter
21.  C     20       3      8
22.  G     21   90000     20
23.  G     23   90001     20
24.  G     25   90002     20           Retrieve next line
25.  G     27   90003     20
26.  G     29   90004     20
27.  C     10      21      7        7
28.  C     10      23      7        7
29.  C     10      25      7        7   Test line for non-zero
30.  C     10      27      7        7   element
31.  C     10      29      7        7
32.  E      6
33.  B      7
34.  F     21      31     21
35.  F     23      31     23
36.  F     25      31     25           Convert Line to be printed
37.  F     27      31     27
38.  F     29      31     29
39.  P     20      11     TI  060209030109030109030109030109030
                              N S NNS NNS NNS NNS NNS NNS NNS NN
```

Table 9 (Cont'd)

40.	E	6		
41.	B	8		
42.	R	3	19	Restore test value of argument and return or halt.
43.	E	2		
44.	V	3	+ 12345678+08	
45.	V	10	+ 00000000+00	Necessary constants
46.	V	11	+ 10000000+01	
47.	V	15	+ 50000000+01	
48.	V	17	+ 00000000+00	
49.	V	18	+ 10000000+05	
50.	V	19	+ 12345678+08	

Table 10

Program Plan for
Subroutine: Convert Binary to Decimal

Start Convert	1. Form 2 to the 27th 2. Set powers of ten counter i to zero 3. Set power of ten P to 2 to the -27th 4. Set first item T_o in table T_i to P 5. Replace Start Convert by Start Convert**
Start Convert**	6. Bump i by one 7. Replace P by Px10 8. Replace T_i by P 9. If i is less than 45, go to Start Convert** 10. Replace Start Convert** by Start Convert*

Table 10 (Cont'd)

Start Convert*	11. Retrieve number to be converted, B using tag #1, and save in M(B) 12. Replace S by +1 13. If M(B) exceeds zero, go to connector (1) 14. Replace S by −1 and M(B) by −B 15. If M(B) exceeds zero, go to connector (1) 16. Store two zeros in cells indicated by tag #2 17. Go to End Convert
Connector (1)	18. Set shift direction j to +1 19. If unity exceeds M(B), go to connector (3) 20. Obtain d, the largest integer not greater than log M(B)* 21. If d exceeds seven, go to connector (2) 22. Retrieve $T_{(7-d)}$ from table T_i i = 0, 1, ... 45 23. Form M(D) = M(B) x $T_{(7-d)}$ x 2 to the 27th 24. Replace d by d + 1 25. Go to connector (4)
Connector (2)	26. Set shift direction j to −1 27. Retrieve $T_{(d-7)}$ from table T_i i = 0, 1, ..., 45 28. Form M(D) = M(B) ÷ ($T_{(d-7)}$ x 2 to the 27th) 29. Replace d by d + 1 30. Go to connector (4)
Connector (3)	31. Replace M(B) by 1 ÷ M(B) 32. Obtain d, the largest integer not greater than log M(B)* 33. Retrieve $T_{(d+8)}$ from table T_i i = 0, 1, ..., 45 34. Form M(D) = M(B) x $T_{(d+8)}$ x 2 to the 27th 35. Replace d by 1 − d
Connector (4)	36. Half adjust M(D) 37. If M(D) exceeds 99,999,999, go to connector (7) 38. If 10,000,000 exceeds M(D), go to connector (6)
Connector (5)	39. Form D = M(D) x S 40. Store D, d in cells indicated by tag #2 41. Go to End Convert
Connector (6)	42. Replace $T_{(d+8)}$ by $T_{(d+8+j)}$, and d by d-1 43. Go to connector (8)

Table 10 (Cont'd)

Connector (7)	44. Replace $T_{(d+8)}$ by $T_{(d+8+j)}$, and d by d +1
Connector (8)	45. If d exceeds seven go to connector (9) 46. Form M(D) = M(B) x $T_{(d+8)}$ x 2 to the 27th 47. Go to connector (5)
Connector (9)	48. Form M(D) = M(B) ÷ ($T_{(d+8)}$ x 2 to the 27th) 49. Go to connector (5)

*Subroutine: Table Search (see Table 13)

or equals unity the proper power of ten is determined by a table search (step 20), and the mantissa produced with a single non-trivial multiplication (steps 21-25) or division (steps 27-30) to minimize round-off error. In steps 31-35, the case where the magnitude of the input number is less than one is treated similarly. Steps 36-41 attach the proper sign to the output after half-adjustment and return to the calling routine.

This subroutine, too, calls for a subroutine to do the required table search. The program plan, memory assignment, and code for this subroutine appear in Tables 13, 14, and 15 respectively. Judicious location of the table to be used in the previous subroutine and to be searched in this one allows this subroutine to be adjoined to the others with a second K30 operator.

The Table Search to Bracket subroutine is a straightforward binary search which yields the largest power of ten still less than the number being matched. Step 2 of the plan scales down the argument to the table range. Steps 3, 4 initialize the search which is conducted in the loop described in steps 5-13. Step 14-15 handles the exact match case.

The printout memory subroutine is now complete. Memory assignments for it are given in Table 16 to show the effect of the two K30 operators used above. Prefixing a suitable K operator, such as K9700, to the entire Print-out Memory subroutine has made it occupy locations 9701-9829. Prefixing a different K operator, like K511, would on the other hand alone be sufficient to establish it in locations 512-640.

Table 11

ADVANCED MYSTIC STORAGE MAP

PROGRAM: Convert Binary to Decimal

PROGRAMMER: T. P. Gorman

00	00	Start Convert 01	End Convert 02	tag #1 03	tag #2 04
05	Connector(1) 05	Connector(2) 06	Connector (3) 07	Connector (4) 08	Connector(5) 09
10	Connector(6) 10	Connector (7) 11	Connector (8) 12	Connector (9) 13	99, 999, 999 14
15	2 to the 26th 15	forty-five 16	seven 17	18	2 to the 27th 19
20	zero 20	one 21	two 22	half 23	10,000,000 24
25	i and j 25	P 26	d-8 27	M(D) 28	1 ÷ M(B) 29
30	30	31	32	33	34
35	35	36	37	38	39
40	40	41	42	43	44
45	45	46	47	48	49
50	T_o 50	T_1 51	. . . 52	. . . 53	. . . 54
55	. . . 55	. . . 56	. . . 57	. . . 58	. . . 59
60	. . . 60	. . . 61	. . . 62	. . . 63	. . . 64
65	. . . 65	. . . 66	. . . 67	. . . 68	. . . 69
70	. . . 70	. . . 71	. . . 72	. . . 73	. . . 74
75	. . . 75	. . . 76	. . . 77	. . . 78	. . . 79
80	. . . 80	. . . 81	. . . 82	. . . 83	. . . 84
85	. . . 85	. . . 86	. . . 87	. . . 88	. . . 89
90	. . . 90	. . . 91	. . . 92	. . . 93	. . . 94
95	T_{45} 95	M(B) 96	S 97	D 98	d 99

NOTES:

Table 12

CAMEO Code for
Subroutine: Binary to Decimal Conversion

1.	K	30			Relocator
2.	B	1			Initial start of function
3.	A	19	15	15	
4.	R	25	20		
5.	D	26	21	19	
6.	R	50	26		
7.	N	1			
8.	A	25	25	21	Generate table of scaled-down powers
9.	M	26	26	22	of ten
10.	H	50	25	26	
11.	C	16	25	1	
12.	N	1			Steady-state start of function
13.	G	96	1	3	
14.	R	97	21		
15.	C	96	20	5	
16.	S	96	20	96	Test argument for zero, form absolute
17.	S	97	20	21	value and record sign.
18.	C	96	20	5	
19.	H	1	4	20	
20.	H	2	4	20	
21.	E	2			
22.	B	5			Argument is not zero
23.	I	25 + 10000000 + 01			
24.	C	21	96	7	Test for less than one
25.	F	99	31	96	Obtain nearest power of ten
26.	C	99	17	6	Test for greater than seven
27.	S	27	17	99	
28.	G	26	50	27	Form decimal equivalent number
29.	M	28	96	26	pair for number less than or equal
30.	M	28	28	19	to ten to the seventh.
31.	A	99	99	21	
32.	E	8			
33.	B	6			
34.	I	25 - 10000000 + 01			
35.	S	27	99	17	
36.	G	26	50	27	Form decimal equivalent number pair
37.	D	28	96	26	for number greater than ten to the
38.	D	28	28	19	seventh.
39.	A	99	99	21	
40.	E	8			
41.	B	7			
42.	D	29	21	96	
43.	F	99	31	29	

Table 12 (Cont'd)

44.	A	27	99	18	Form decimal number pair for binary
45.	G	26	50	27	number less than one.
46.	M	28	96	26	
47.	M	28	28	19	
48.	S	99	20	99	
49.	B	8			
50.	A	28	28	23	Half adjust and test output integer
51.	U	28	28		M(D) for normalization.
52.	C	28	14	11	
53.	C	24	28	10	
54.	B	9			
55.	M	98	28	97	Attach sign, store result and return to
56.	H	1	4	98	main program.
57.	H	2	4	99	
58.	E	2			
59.	B	10			M(D) is less than 10,000,000
60.	M	28	28	22	Shift left to normalize
62.	S	99	99	21	
63.	E	12			
64.	B	11			M(D) exceeds 99,999,999
65.	S	27	27	25	
66.	G	26	50	27	Shift right to normalize
67.	A	99	99	21	
68.	B	12			
69.	C	99	17	13	
70.	M	28	96	26	
71.	M	28	28	19	
72.	E	9			
73.	B	13			
74.	D	28	96	26	
75.	D	28	28	19	
76.	E	9			
77.	V	14 + 99999999 + 08			
78.	V	15 + 67108864 + 08			
79.	V	16 + 45000000 + 02			
80.	V	17 + 70000000 + 01			
81.	V	18 + 80000000 + 01			
82.	V	20 + 00000000 + 00			
83.	V	21 + 10000000 + 01			
84.	V	22 + 10000000 + 02			
85.	V	23 + 50000000 + 00			
86.	V	24 + 10000000 + 08			

Table 13

Program Plan for
Subroutine: Table Search

Start Search	1. Retrieve number to be bracketed M(B) using tag #1 2. Scale to table by multiplying by T_0 3. Set increment I to 2 to the 6th 4. Set counter C to zero
Connector (1)	5. Replace I by I ÷ 2 6. If 1 exceeds I, go to connector (2) 7. Replace trial counter C* by C + I 8. If C* exceeds table size, go to connector (1) 9. Replace C by C* 10. Retrieve C-th item from table: T 11. If M(B) exceeds T, go to connector (1) 12. Replace C by C-I 13. If T exceeds M(B), go to connector (1) 14. Replace C by C + I
Connector (2)	15. Store C using tag #2
End Search	

SYSTEM PERFORMANCE

The system described here has been in use in successively more powerful versions since mid-1955. A description of the first system was published in the Journal of the ACM in October of that year (see reference 1). Redesigned and reworked for the IBM-704 and IBM-650 and Univac 1103A, a new version was published through ASTIA in February of 1958 (see reference 2). Up until the end of 1959 the system was used by small groups of people at the Applied Physics Laboratory and at the RCA Patrick Air Force Base computation center. Several complex programs were written during this period including a six-degree-of-freedom guided missile simulation, a pseudo analogue computer, and a symbolic assembly program for the Univac 1103A. Reports on these programs were published internally at the Johns Hopkins Applied Physics Laboratory.

Table 14

ADVANCED MYSTIC STORAGE MAP

PAGE 1 OF 1

PROGRAM: Subroutine: Table Search

PROGRAMMER: T. P. Gorman

00	00	01 Start Search	02 End Search	03 tag #1	04 tag #2
05	05 counter	06 increment	07 trial counter	08 M(B)	09 Connector (2)
10	10 Connector (1)	11 one	12 two	13 forty-five	14 2 to the 6th
15	15 T_o	16 T_1	17 . . .	18 . . .	19 . . .
20	20	21	22	23	24
25	25	26	27	28	29
30	30	31	32	33	34
35	35	36	37	38	39
40	40	41	42	43	44
45	45	46	47	48	49
50	50	51	52	53	54
55	55 . . .	56 . . .	57 . . .	58 . . .	59 . . .
60	60 . . .	61 . . .	62 . . .	63 . . .	64 . . .
65	65 T_{45}	66	67	68	69
70	70	71	72	73	74
75	75	76	77	78	79
80	80	81	82	83	84
85	85	86	87	88	89
90	90	91	92	93	94
95	95	96	97	98	99

NOTES:

Table 15

CAMEO Code for

Subroutine: Table Search to Bracket

1.	K	30			Relocator
2.	B	1			
3.	G	8	1	3	Obtain and scale-down argument
4.	M	8	8	20	
5.	R	6	14		Set counter and increment
6.	I	5 + 00000000 + 00			
7.	B	9			Search loop
8.	D	6	6	12	Halve increment and exit on
9.	C	11	6	10	exhaustion of interval
10.	A	7	5	6	Augment counter and return on table
11.	C	7	13	9	overflow
12.	R	5	7		
13.	G	7	20	5	Obtain table entry and adjust counter
14.	C	8	7	9	up or down by comparison with
15.	S	5	5	6	argument.
16.	C	7	8	9	
17.	A	5	5	6	Argument equals entry. Restore counter.
18.	B	10			End of search. Store result and return
19.	H	1	4	5	to main program.
20.	E	2			
21.	V	11 + 10000000 + 01			Necessary constants
22.	V	12 + 20000000 + 01			
23.	V	13 + 45000000 + 02			
24.	V	14 + 64000000 + 02			

In December of 1959 this system was applied to the computational problems of the Data Systems Division of the Goddard Space Flight Center. This work consisted mainly in the production of a comprehensive spacetask computing system to support unmanned spacecraft orbit determination and general spacetask analysis. This IBM 7090 system (see reference 3) was produced in less than three years by nine people with no IBM 7090 experience and contains no machine language instructions

Table 16

ADVANCED MYSTIC STORAGE MAP

PROGRAM: Memory Print Routine

PROGRAMMER : T. P. Gorman

00	00	01 Start Print	02 End Print	03 Req. Lo Bound	04 Req. Hi Bound
05	05 Connector(1)	06 Connector(2)	07 Connector(3)	08 Connector (4)	09
10	10 zero	11 one	12	13	14 four
15	15 five	16	17 Nom. Lo Bound	18 Nom. Hi Bound	19 Nominal Test Word
20	20 Word counter	21 1st word of line	22 2nd word of line	23 . . .	24 . . .
25	25 . . .	26 . . .	27 . . .	28 . . .	29 . . .
30	30 10th word of line	31 Start Convert	32 End Convert	33 tag #1	34 tag #2
35	35 Connector(1)	36 Connector(2)	37 Connector(3)	38 Connector(4)	39 Connector(5)
40	40 Connector(6)	41 Connector(7)	42 Connector(8)	43 Connector(9)	44 99, 999, 999
45	45 2 to the 26th	46 forty-five	47 seven	48	49 2 to the 27th
50	50 zero	51 one	52 two	53 half	54 10,000,000
55	55 i and j	56 P	57 d - 8	58 M(D)	59 1 ÷ M(B)
60	60	61 Start Search	62 End Search	63 tag #1	64 tag #2
65	65 counter	66 increment	67 trial value	68 M(B)	69 Connector(1)
70	70 Connector(2)	71 one	72 two	73 forty-five	74 2 to the 6th
75	75	76	77	78	79
80	80 $10^{0} \times 2^{-27}$	81 $10^{1} \times 2^{-27}$	82 $10^{2} \times 2^{-27}$	83 . . .	84 . . .
85	85 $10^{5} \times 2^{-27}$	86 . . .	87 . . .	88 . . .	89 . . .
90	90 $10^{10} \times 2^{-27}$	91 . . .	92 . . .	93 . . .	94 . . .
95	95 $10^{15} \times 2^{-27}$	96 . . .	97 . . .	98 . . .	99 . . .

NOTES :

Table 16 (Cont'd)

ADVANCED MYSTIC STORAGE MAP

PAGE 2 OF 2

PROGRAM: Memory Print Routine

PROGRAMMER: T. P. Gorman

100	$10^{20} \times 2^{-27}$ 00	01	02	03	04
105	$10^{25} \times 2^{-27}$ 05	06	07	08	09
110	$10^{30} \times 2^{-27}$ 10	11	12	13	14
115	$10^{35} \times 2^{-27}$ 15	16	17	18	19
120	$10^{40} \times 2^{-27}$ 20	21	22	23	24
125	$10^{45} \times 2^{-27}$ 25	26 M(B)	27 S	28 D	29 d
30	30	31	32	33	34

at all. It represents more than half a million machine language instructions and is in current operational use, determining the orbits of all the unmanned scientific and applications spacecraft for which the Goddard Space Flight Center is responsible.

CAMEO has proved its capacity to enhance the productivity of professional programmers on more than one occasion. It was used to produce a two-phase data analysis and plotting program for an emergency trapped radiation experiment and for this task was modified to accept machine language programs in a standard form derived from FORTRAN output. Data analysis programs were written in CAMEO again for the launch of the S-6 Atmospheric Structure Satellite. This program required the extension of the system, with no external change, to the use of two-bank memory available on the IBM 7094. Most recently the use of the CAMEO system has permitted the simultaneous extension of the SPATS orbit determination system to complete 16 digit mode and the transfer of all eight digit programs to the Univac 1107 in a six-month period using two man-years of effort.

CONCLUSION

Experience has shown that when problems are programmed entirely in terms of commands for an APOM, all phases of the programming process are in practice as well as theory, greatly simplified. The practical capacity for extending routines as illustrated in the example has for instance significantly enhanced the efficiency of the program production process. The design of the input-output operations has straightened the way in and out of the machine for the programmer without restricting formats or erecting baroque report generators. The CAMEO programmer uses a strictly functional instrument to define the required algorithm, converts his subroutines quickly and surely, wastes no time with paraphernalia like binary cards and octal dumps and mysterious diagnostics, and has little reason to fret over anything but the problem to be solved. The result is not surprising: the CAMEO programmer solves more problems, solves them more rapidly and requires less time for training.

ACKNOWLEDGEMENTS

To Mr. R. G. Kelly, former Assistant Chief of the Data Systems Division, should go a substantial share of the credit for the production of CAMEO and Advanced Mystic. He presented the challenges, helped with the design, and was ready to do something a different way because it was a better way. Mr. J. J. Fleming, Chief of the Data Systems Division, and Mr. D. H. Gridley, Associate Chief, gave the project a fair evaluation based on its merits. Thanks are due to Mrs. Melba L. Roy, for her clear vision and firm leadership; and to Mrs. Joan T. Jones as well, by whose efforts the CAMEO system was finished and documented; and to Mrs. Sally A. Richmond who helped immeasurably with both activities. A final word of appreciation goes to those problem sponsors who had the insight to ask for results and not prescribe methods; in particular to Dr. Joseph W. Siry, Dr. Peter Musen and Dr. Wilmot N. Hess.

References

1. Gorman, T. P., Kelly R. G., and Reddy, R. B., Automatic Coding for the IBM-701, Journal of the ACM, Vol. 2 #4, pgs. 253-261, October 1955.

2. Gorman, T. P., and Kelly, R. G., Mystic: An Automatic Coder for the IBM 650, IBM 704 and ERA 1103 AF, Astia Document #134274 AFMTC-TN-58-2.

3. Gorman, T. P., Spacetask Planning Analysis and Tracking System Used at Goddard Space Flight Center for Scientific Spacecraft Orbits. Paper Presented at the Third Annual ACM Symposium on Computing in the Washington D. C. Area.