# THE SAC-1 INTEGER ARITHMETIC SYSTEM-VERSION III

by

George E. Collins

Technical Report #156

March, 1973

Received March 12, 1973

# THE SAC-1 INTEGER ARITHMETIC SYSTEM-VERSION III*

by

George E. Collins

Technical Report #156†

## ABSTRACT

This report documents a new version of the SAC-1 Integer Arithmetic System, a system for performing arithmetic operations and input-output on infinite-precision integers. The new version contains improved algorithms for base conversion and some new subprograms. However, the most important improvement is the documentation provided by this report, which includes, for each subprogram, a functional specification, a semi-formal algorithm description, both theoretical and empirical computing times, and an ANSI standard Fortran program listing which corresponds closely to the algorithm description.

*Research supported by NSF grant GJ-30125X.

†This report also appears as University of Wisconsin Madison Academic Computing Center Technical Report No. 31.

TABLE OF CONTENTS

1. Introduction ..... 1

2. Preliminaries ..... 3

3. The Algorithms ..... 6

3.1. The Primitives ..... 7

3.2. Addition and Subtraction ..... 11

3.3. Multiplication ..... 18

3.4. Division ..... 21

3.5. Greatest Common Divisors ..... 25

3.6. Input-Output and Conversion ..... 30

4. Fortran Program Listings ..... 38

5. A Test Program ..... 55

References ..... 61

Algorithm Index ..... 63

## 1. Introduction

The SAC-1 Integer Arithmetic System-Version III replaces the Revised SAC-1 Integer Arithmetic System [3]. The Version III System does not differ substantially from the Revised System. Apart from improvements in the input-output subprograms and the addition of three miscellaneous subprograms, the changes are primarily in the documentation. The present manual is written in the style of the more recently issued SAC-1 manuals, containing semi-formal algorithm descriptions which correspond closely to the Fortran listings, both theoretical and empirical computing times, and a test program.

In the input-output algorithms, conversion between base $\beta$ and decimal is now performed via a two-step process as recommended by Knuth in [13], Section 4.4.E. For output, numbers are first converted from base $\beta$ to base $\theta$, and then from base $\theta$ to decimal, and the reverse process is used for input. $\theta$ is a positive Fortran integer, and a power of ten. Like $\beta$ it is an implementation parameter which should be chosen as large as possible for a given computer; i.e., one should choose $\theta = 10^k$ where $k = [\log_{10} \beta]$. For example, in the UNIVAC 1108 implementation of SAC-1 at the University of Wisconsin, $\beta = 2^{33}$ and $\theta = 10^9$. $\theta$ is stored in labelled common block TR3 along with $\beta$ as follows:

```
COMMON /TR3/ BETA,THETA
```

Any main program for the use of SAC-1 must initially assign the appropriate values to BETA and THETA.

The new subprograms for conversion are IDTOH, IHTOD, IBTOH and IHTOD. But a user will seldom have occasion to use these except indirectly. The other new subprograms are IMADD, a special subprogram for multiplication and addition, IPOWER, for exponentiation of integers, and ILCM, for computing integer least common multiples. In addition, the specification of IQRS has been changed so that the divisor is now a Fortran integer. The only algorithm (other than IQR) in the SAC-1 system that calls IQRS directly is ELPOF2 in the Polynomial Real Zero System [8]. To use ELPOF2 with the Version III Integer Arithmetic System it will suffice to change line 12 on page 54 of [8] from

```
1       Z = IQRS(Q,TWO)
```

to

```
1       Z = IQR(Q,TWO) .
```

Also any user's programs that call IQRS directly must be altered.

Version 3 need not be substituted immediately for the revised system, but it is recommended since forthcoming new SAC-1 subsystems may assume the substitution has been made. The substitution entails only the keypunching and compilation of the Fortran subprograms since the integer arithmetic primitives remain unchanged. The ten currently available SAC-1 subsystems are listed near the end of this manual, under References.

## 2. Preliminaries

The SAC-1 Integer Arithmetic System-Version III performs operations, arithmetic and input-output, on arbitrarily large (infinite-precision) integers. Any such non-zero integer, $A$, is uniquely expressed, in $\beta$-radix representation, in the form

$$A = \sum_{i=0}^{m} a_i \beta^i ,$$

where each $a_i \geq 0$ if $A > 0$, each $a_i \leq 0$ if $A < 0$, and $a_m \neq 0$. $A$ is then represented by the first order list $(a_0,a_1,\ldots,a_m)$. The integer $0$ is represented by the null list ( ), whose location is $0$.

$\beta$ is a system parameter which may vary from one SAC-1 implementation to another, subject to the following conditions. For a given Fortran compiler, let $\gamma$ be the least positive integer such that if $a$ is an integer and $|a| < \gamma$, then $a$ is a Fortran integer. $\beta$ is then any number such that (1) $\beta < \gamma$, (2) $\beta \geq 64$, and (3) $\beta$ is even. Condition (1) ensures that $\beta$ and $-\beta$ are Fortran integers, and that $a$ is a Fortran integer for $|a| < \beta$. Condition (2) permits convenient representation of character strings as infinite-precision integers. Condition (3) helps to simplify algorithms for division and greatest

common divisor calculation. The parameter $\beta$ is stored in labelled common block TR3, as described in Section 1.

For convenience, infinite-precision integers will be referred to as L-integers, where the L may be thought of as standing for either "large" or "list". It should be noted that a non-zero Fortran integer, $a$, is not an L-integer and is generally not a valid input to the SAC-1 subprograms; rather one must construct the L-integer $A = (a)$ using, for example, PFA. A Fortran integer $a$ such that $|a| < \beta$ is called a <u>$\beta$-digit</u>.

In the following we will use the floor, ceiling and integer part functions. For any real number $x$, $\lfloor x \rfloor$, the <u>floor</u> of $x$, is the greatest integer $a$ such that $a \leq x$, $\lceil x \rceil$, the <u>ceiling</u> of $x$, is the least integer $a$ such that $a \geq x$, and $[x]$, the <u>integer part</u> of $x$, is defined by $[x] = \lfloor x \rfloor$ if $x \geq 0$ and $[x] = \lceil x \rceil$ if $x < 0$.

If $a$ and $b$ are arbitrary integers, we define quot $(a,b)$, the <u>quotient</u> of $a$ and $b$, to be $[a/b]$ if $b \neq 0$ and $0$ if $b = 0$. We also define rem $(a,b)$, the <u>remainder</u> of $a$ and $b$, to be $a - b \cdot \text{quot}\,(a,b)$.

For any integer $\beta \geq 2$, we define $L_\beta(a)$, the <u>$\beta$-length</u> of $a$, as $[\log_\beta(|a| + 1)]$ for $a \neq 0$, and

$L_\beta(0) = 1$ . Note that $L_\beta(a)$ is just the number of digits in the base $\beta$ radix representation of $a$ for $a \neq 0$ . Throughout the following $\beta$ is chosen to be the same as the implementation parameter discussed above, so we omit the subscript $\beta$ and refer to $L(a)$ merely as the <u>length</u> of $a$ .

## 3. The Algorithms

This section contains, for each subprogram of the system, a functional specification, an algorithm description, a theoretical computing time, and empirical computing time information, in this order.

The functional specification provides the necessary information for correct use of the subprogram, including a description of the set of valid inputs and a description of the output or outputs as a function of the inputs. Function subprograms are distinguished from subroutines by the presence of an equal sign in the caption containing the name of the subprogram and (arbitrary) symbols for the inputs and outputs. Unless explicit exception is made, function subprograms have only the function value as output, and do not modify the values of their arguments.

The algorithm description details explicitly the algorithm used to perform the operation implied by the functional specification. The Fortran subprograms correspond closely to these descriptions and in fact are obtained from them by a simple mechanical process.

The theoretical computing times are given in terms of the concepts and notations of dominance and codominance, and are independent of the computer of implementation.

For the definitions and some basic properties of dominance and codominance, see [12] and [14].

The empirical computing time information given is for the UNIVAC 1108 implementation of SAC-1 at the University of Wisconsin. For the simpler algorithms formulas are given which express the computing time as a function of some natural parameters. For the more complex algorithms, tables are given containing observed computing times for representative cases.

## 3.1. The Primitives

The following three subprograms are called primitives since they will usually be programmed in assembly language for any particular SAC-1 implementation. Although Fortran programs for these primitives are listed in Section 4 which are usable subject to the additional noted restrictions on $\beta$, the system will usually be speeded by a factor of at least 10 by rewriting these subprograms in assembly language.

### ADD3(A,B,C)

A, B and C are arbitrary $\beta$-digits. Let $A + B + C = D\beta + E$, where D and E are $\beta$-digits and $D \cdot E \geq 0$. D is returned as the new value of A, and E is returned as the new value of B. C retains its original value.

Algorithm

(1) $B \leftarrow A + B + C$ .

(2) $A \leftarrow [B/\beta]$ .

(3) $B \leftarrow B - A \cdot \beta$ ; return.

If $3(\beta-1) < \gamma$ , then this algorithm is directly translatable into a FORTRAN subroutine, which is listed in Section 4. Note that the parameter $\beta$ (BETA) is the first element of labelled common block TR3. However, this subroutine involves both a multiplication and a division, whereas if it is programmed in an assembly language and $\beta$ is a power of 2, for a binary computer, or a power of 10, for a decimal computer, then both the multiplication and division can be replaced by shifting. The resulting subroutine will generally be much faster and this is important since ADD3 is executed very frequently in performing all the arithmetic operations on L-integers. ADD3 is an assembly language subroutine in the UNIVAC 1108 SAC-1 implementation at the University of Wisconsin, with an execution time of approximately 13 microseconds. The execution time of the FORTRAN version on the 1108 is approximately 50 microseconds. On the 1108 the values of $\gamma$ and $\beta$ are $2^{35}$ and $2^{33}$ respectively.

MPY(A,B)

A and B are β-digits. Let C and D be the unique β-digits such that $A \cdot B = C \cdot \beta + D$ and $C \cdot D \geq 0$. The outputs C and D are returned as the new values of A and B, respectively.

The subroutine MPY can be realized in FORTRAN provided that the product $A \cdot B$ is always a FORTRAN integer, i.e., provided that $(\beta - 1)^2 < \gamma$. Following is an algorithm which can be used.

Algorithm

(1) $B \leftarrow A \cdot B$.

(2) $A \leftarrow [B/\beta]$.

(3) $B \leftarrow B - A \cdot \beta$ ; return.

Realization of this algorithm in FORTRAN is extremely inefficient, for two reasons. First, it specifies two multiplications and one division, whereas an assembly language subroutine will usually specify just one multiplication and no divisions. Second, the requirement $(\beta - 1)^2 < \gamma$ implies a choice of β containing only about half as many (binary or decimal) digits as would otherwise be possible. This results in each large integer requiring twice as many cells for its representation and each operation on large integers requiring at least twice as many primitive

list processing and arithmetic operations. In fact, multiplication and division will require four times as many applications of MPY, and each application will be much slower.

On the UNIVAC 1108, the computing time of the FORTRAN version of MPY is approximately 42 microseconds, whereas the computing time of an assembly language version is approximately 11 microseconds.

QR(A,B,C)

A, B and C are $\beta$-digits with $A \cdot B \geq 0$ and $|A| < |C|$. Let $D = \text{quot}\ (A\beta + B, C)$ and $E = \text{rem}\ (A\beta + B, C)$. D and E are returned as new values of A and B, respectively.

The following algorithm can be translated directly into FORTRAN provided $\beta^2 - \beta - 1 < \gamma$.

Algorithm

(1) $E \leftarrow A \cdot BETA + B$ .

(2) $D \leftarrow [E/C]$ .

(3) $E \leftarrow E - C \cdot D$ ; return.

As in the case of MPY, a great improvement will be achieved by programming QR in assembly language, with a value of $\beta$ nearly as large as $\gamma$. On the UNIVAC 1108,

the computing times of the FORTRAN and assembly language versions of QR are approximately 44 and 25 microseconds, respectively.

All subsequent UNIVAC 1108 computing times will be based on the use of the assembly language versions of the three primitives above.

## 3.2. Addition and Subtraction

COMPAT(S,A,B)

S is either +1 or -1 . A is either +1 , 0 , or -1 , and $S \cdot A \leq 0$ . B is a β-digit. Let C and D be the unique β-digits such that $A + B = C \cdot \beta + D$ and $S \cdot D \geq 0$ . C will be either +1 , 0 , or -1 , and $S \cdot C$ will be non-positive. C is returned as the new value of A and D is returned as the new value of B .

Algorithm

(1) $B \leftarrow A + B$ ; if $S < 0$ , go to (3).

(2) If $B \geq 0$ , go to (4); $B \leftarrow B + \beta$ ; $A \leftarrow -1$ ; return.

(3) If $B \leq 0$ , go to (4); $B \leftarrow B - \beta$ ; $A \leftarrow +1$ ; return.

(4) $A \leftarrow 0$ ; return.

Computing Time: $\sim 1$.

On the UNIVAC 1108, the average computing time of a FORTRAN subroutine for this algorithm is approximately

40 microseconds. Since COMPAT will be executed very frequently, it will usually be worthwhile to program COMPAT in assembly language. An assembly language version for the UNIVAC 1108 has an execution time varying between 11 and 15 microseconds, with an average of about 13. Execution times quoted for all future subprograms which directly or indirectly use COMPAT will be based on the assembly language version.

S=ISIGNL(A)

A is an L-integer. S is the sign of A , a FORTRAN integer. S = +1 if A > 0 , S = 0 if A = 0 , and S = -1 if A < 0 .

Algorithm

(1) S ← 0 ; if A = 0 , return; T ← A .

(2) ADV(B,T); if B = 0 , go to (2).

(3) S ← 1 ; if B < 0 , S ← -1 ; return.

Computing Time: The maximum computing time is $\sim L(A)$ ; the average and minimum computing times are $\sim 1$ .

On the UNIVAC 1108, the execution time of ISIGNL(A) is approximately 20 microseconds when A = 0 , and approximately 13k + 27 microseconds when the first non-zero $\beta$-digit of A is the $k^{\underline{th}}$, $k \geq 1$ .

B = INEG(A)

A is an L-integer. B is the L-integer -A .

Algorithm

(1) B ← 0 ; if A = 0 , return; U ← A .

(2) ADV(AI,U); B ← PFA(-AI,B); if U ≠ 0 , go to (2).

(3) B ← INV(B); return.

Computing Time: ∿ L(A) .

On the UNIVAC 1108 the computing time of INEG(A) is approximately 23 microseconds for A = 0 , approximately 36 L(A) + 40 microseconds for A ≠ 0 .

B = IABSL(A)

A is an L-integer. B is the L-integer |A| .

Algorithm

(1) S ← ISIGNL(A).

(2) If S ≥ 0 , (B ← BORROW(A); return).

(3) B ← INEG(A); return.

Computing Time: The maximum, average and minimum computing times of IABSL(A) for L(A) = m are respectively ∿m , ∿m , and ∿1 .

On the UNIVAC 1108, the computing time of IABSL(A) is approximately 50 microseconds when A = 0 . If

$L(A) = m$ and the first non-zero $\beta$-digit of $A$ is the $k^{\underline{th}}$, then the computing time is approximately $13k + 74$ microseconds if $A > 0$ and $36m + 13k + 85$ microseconds if $A < 0$ .

S = ICOMP(A,B)

A and B are L-integers. S is the sign of A - B , a FORTRAN integer.

Algorithm

(1) U ← ISIGNL(A); V ← ISIGNL(B); S ← U - V ; if U = 0 or V = 0 , return; if S ≠ 0 , go to (4); X ← A ; Y ← B .

(2) T ← FIRST(X) - FIRST(Y) ; if T ≠ 0 , S ← T ; X ← TAIL(X) ; Y ← TAIL(Y) ; if X = 0 , go to (3); if Y ≠ 0 , go to (2); S ← U ; return.

(3) If Y ≠ 0 , (S ← -U ; return).

(4) If S > 0 , S ← 1 ; if S < 0 , S ← -1 ; return.

Computing Time: The maximum, average, and minimum computing times of ICOMP(A,B) for L(A) = m and L(B) = n are $\sim m + n$ , $\sim \min(m,n)$ , and $\sim 1$ respectively.

On the UNIVAC 1108, the average computing time is approximately 65 microseconds if A = B = 0 ; if exactly one of A and B is 0 , the average computing time is

approximately 85 microseconds. If $A \cdot B < 0$, the average computing time is approximately 115 microseconds. If $A \cdot B > 0$, the average computing time is approximately $30k + 124$ microseconds, where $k = \min(L(A),L(B))$.

C=ISUM(A,B)

A and B are L-integers. C = A + B, an L-integer.

Algorithm

(1) If A = 0, (C ← BORROW(B); return); if B = 0, (C ← BORROW(A); return); C ← 0; U ← A; V ← B.

(2) S ← ISIGNL(A); T ← ISIGNL(B); if S + T = 0, go to (8); DI ← 0.

(3) ADV(AI,U); ADV(BI,V); ADD3(DI,AI,BI); C ← PFA(AI,C).

(4) If U = 0, (U ← V; go to (6)); if V ≠ 0, go to (3).

(5) ADV(AI,U); ADD3(DI,AI,0); C ← PFA(AI,C).

(6) If U ≠ 0, go to (5).

(7) If DI ≠ 0, C ← PFA(DI,C); C ← INV(C); return.

(8) F ← 0.

(9) ADV(AI,U); ADV(BI,V); DI = AI + BI; if DI ≠ 0, F ← DI; C ← PFA(DI,C).

(10) If U = 0, (U ← V; go to (12)); if V ≠ 0, go to (9).

(11) ADV(F,U); C ← PFA(F,C).

(12) If $U \neq 0$ , go to (11).

(13) If $F \neq 0$ , go to (14); ERLA(C); $C \leftarrow 0$ ; return.

(14) $C \leftarrow$ INV(C); $S \leftarrow 1$ ; if $F < 0$ , $S \leftarrow -1$ , $F \leftarrow 0$ ; $EI \leftarrow 0$ , $U \leftarrow C$ .

(15) $DI \leftarrow$ FIRST(U) ; COMPAT(S,EI,DI) ; if $DI \neq 0$ , $F \leftarrow U$ , ALTER(DI,U) ; $U \leftarrow$ TAIL(U) ; if $U \neq 0$ , go to (15).

(16) $G \leftarrow$ TAIL(F) ; if $G \neq 0$ , (SUCC(0,F) ; ERLA(G)) ; return.

Computing Time: $\sim 1$ for $A = 0$ or $B = 0$ . Otherwise the maximum, minimum and average computing times are $\sim m + n$ , where $m = L(A)$ and $n = L(B)$ .

The empirical computing time of ISUM(A,B) can be closely approximated by considering four cases. Let $m = L(A)$ and $n = L(B)$ , $h = \min(m,n)$ and $k = \max(m,n)$ . The approximate computing times on the UNIVAC 1108 are as follows:

Case 1: ($A = 0$ or $B = 0$) 37 microseconds.

Case 2: ($A \cdot B > 0$) $13h + 39k + 126$ microseconds.

Case 3: ($A \cdot B < 0$ and $A + B \neq 0$) $16h + 72k + 142$ microseconds.

Case 4: ($A \cdot B < 0$ and $A + B = 0$) $54h + 142$ microseconds.

C = IDIF(A,B)

A and B are L-integers. C is the L-integer A - B .

Algorithm

(1) If A = 0 , (C ← INEG(B) ; return); if B = 0 , (C ← BORROW(A) ; return); C ← 0 ; U ← A ; V ← B .

(2) S ← ISIGNL(A) ; T ← ISIGNL(B) ; if S + T ≠ 0 , go to (8); DI ← 0 .

(3) ADV(AI,U); ADV(BI,V); BI ← -BI ; ADD3(DI,AI,BI); C ← PFA(AI,C) .

(4) If U = 0 , go to (6); if V ≠ 0 , go to (3).

(5) ADV(AI,U); ADD3(DI,AI,0); C ← PFA(AI,C) ; if U ≠ 0 , go to (5); go to (7).

(6) If V = 0 , go to (7); ADV(BI,V); BI ← -BI ; ADD3(DI,BI,0); C ← PFA(BI,C) ; go to (6).

(7) If DI ≠ 0 , C ← PFA(DI,C) ; C ← INV(C) ; return.

(8) F ← 0 .

(9) ADV(AI,U); ADV(BI,V); DI = AI - BI ; if DI ≠ 0 , F ← DI ; C ← PFA(DI,C) .

(10) If U = 0 , go to (12); if V ≠ 0 , go to (9).

(11) ADV(DI,U); if DI ≠ 0 , F ← DI ; C ← PFA(DI,C) ; if U ≠ 0 , go to (11); go to (13).

(12) If V = 0 , go to (13); ADV(DI,V); DI ← -DI ; if DI ≠ 0 , F ← DI ; C ← PFA(DI,C) ; go to (12).

(13) If $F \neq 0$ , go to (14); ERLA(C); $C \leftarrow 0$ ; return.

(14) $C \leftarrow$ INV(C) ; $S \leftarrow 1$ ; if $F < 0$ , $S \leftarrow -1$ ; $F \leftarrow 0$ , $EI \leftarrow 0$ ; $U \leftarrow C$ .

(15) $DI \leftarrow$ FIRST(U) ; COMPAT(S,EI,DI); if $DI \neq 0$ , $F \leftarrow U$ ; ALTER(DI,U); $U \leftarrow$ TAIL(U) ; if $U \neq 0$ , go to (15).

(16) $G \leftarrow$ TAIL(F) ; SSUCC(0,F); ERLA(G); return.

Computing Time: $\sim 1$ for $B = 0$ , $\sim L(B)$ for $A = 0$ , $\sim L(A) + L(B)$ for $A \neq 0$ and $B \neq 0$ .

The UNIVAC 1108 computing times for IDIF(A,B) are approximately as follows, where $m = L(A)$ , $n = L(B)$ , $h = \min(m,n)$ and $k = \max(m,n)$ :

Case 1: $(A = B = 0)$ . 54 microseconds.

Case 2: $(A \neq 0$ and $B = 0)$ . 38 microseconds.

Case 3: $(A = 0$ and $B \neq 0)$ . $36n + 60$ microseconds.

Case 4: $(A \cdot B < 0)$ . $14h + 41k + 126$ microseconds.

Case 5: $(A \cdot B > 0$ and $A - B \neq 0)$ . $14h + 77k + 140$ microseconds.

Case 6: $(A \cdot B > 0$ and $A - B = 0)$ . $54h + 140$ microseconds.

## 3.3. Multiplication

D = IMADD(A,B,C)

A is a non-zero L-integer. B is a non-zero $\beta$-digit. C is a $\beta$-digit such that $A \cdot B \cdot C \geq 0$ . D is the L-integer $A \cdot B + C$ . The list representing A

is altered to represent D , another cell being suffixed to the list if necessary.

Algorithm

(1) T ← A ; G ← C ; H ← 0 .

(2) F ← FIRST(T) ; E ← B ; MPY(E,F); ADD3(G,F,H); ALTER(F,T); H ← E ; E ← T ; T ← TAIL(T) ; if T ≠ 0 , go to (2) .

(3) D ← A ; G ← G + H ; if G ≠ 0 , (F ← PFA(G,0) ; SSUCC(F,E)) ; return.

Computing Time: ∿L(A) .

On the UNIVAC 1108, the computing time of IMADD(A,B,C) is approximately 32 L(A) + 20 L(D) + 60 microseconds.

C = IPROD(A,B)

A and B are L-integers. C = A · B , an L-integer.

Algorithm

(1) C ← 0 ; if A = 0 or B = 0 , return.

(2) S ← PFA(0,0) ; T ← PFA(0,S) ; M ← LENGTH(A) + LENGTH(B) - 2 ; C ← T ; if M = 0 , go to (3); for I = 1,...,M do (C ← PFA(0,C)) .

(3) V ← B ; W ← C .

(4) X ← A ; ADV(BJ,V); Z ← W ; G ← 0 .

(5) ADV(AI,X); F ← BJ ; MPY(AI,F); E ← FIRST(Z) ;
ADD3(G,E,F); ALTER(E,Z); G ← AI + G : Z ← TAIL(Z) ;
if X ≠ 0 , go to (5).

(6) ALTER(G,Z); W ← TAIL(W) ; if V ≠ 0 , go to (4).

(7) If G = 0 , (SSUCC(0,T); DECAP(G,S)); return.

Computing Time: ∿1 for A = 0 or B = 0 , otherwise, ∿L(A) · L(B) .

On the UNIVAC 1108, the computing time of IPROD(A,B) is approximately 30 microseconds for A = 0 or B = 0 ; otherwise, approximately 62mn + 23m + 57n + 183 microseconds, where m = L(A) and n = L(B) .

B = IPOWER(A,N)

A is an L-integer, N is a non-negative FORTRAN integer. $B = A^N$ , an L-integer ($0^0$ is taken to be one).

Algorithm

(1) If N = 0 , (B ← PFA(1,0) ; return); B ← BORROW(A) ;
if A = 0 , return; J ← 1 .

(2) If J = N , return; C ← IPROD(B,A) ; ERLA(B);
B ← C ; J ← J + 1 ; go to (2).

Computing Time: ∿1 for A = 0 or N ≤ 1 , ∿N for |A| = 1 and N ≥ 2 , and $\sim m^2N^2$ otherwise, where m = L(A) .

For $m \geq 3$ and $N \geq 3$, the observed computing time of IPOWER(A,N) on the UNIVAC 1108 is approximately 0.3 $m^2N^2$ milliseconds, where $m = L(A)$.

## 3.4. Division

L = IQRS(A,B)

A is an L-integer. B is a non-zero β-digit. L is the list (Q,R), where Q = quot (A,B), an L-integer, and R = rem (A,B), a β-digit.

Algorithm

(1) Q ← 0 ; E ← 0 ; if A = 0, go to (3); C ← CINV(A).

(2) DECAP(F,C); QR(E,F,B); if Q ≠ 0 or E ≠ 0,
Q ← PFA(E,Q) ; E ← F ; if C ≠ 0, go to (2).

(3) L ← PFL(Q,PFA(E,0)) ; return.

Computing Time: ∿L(A).

On the UNIVAC 1108, the computing time of IQRS(A,B) is approximately 65 microseconds for A = 0, approximately 92m + 97 microseconds for A ≠ 0, where m = L(A).

L = IQR(A,B)

A and B are L-integers. L is the list (Q,R) where Q = quot (A,B) and R = rem (A,B), both L-integers.

Algorithm

(1) [A = 0 or B = 0 .] Q ← 0 ; if A ≠ 0 and B ≠ 0 , go to (2); R ← BORROW(A) ; go to (19).

(2) [B single-precision.] If TAIL(B) ≠ 0 , go to (3); B1 ← FIRST(B) ; L ← IQRS(A,B1) ; DECAP(Q,L); DECAP(R,L); if R ≠ 0 , R ← PFA(R,0) ; go to (19).

(3) [Compute lengths.] M ← LENGTH(A) ; N ← LENGTH(B) .

(4) [M < N .] K ← M - N ; if $K \geq 0$ , go to (5); R ← BORROW(A) ; go to (19).

(5) [Compute signs and normalizer.] S ← ISIGNL(A) ; U ← B ; for I = 1,..., N - 1 , do U ← TAIL(U) ; C ← FIRST(U) ; T ← 1 ; if C < 0 , (T ← -1 ; C ← -C) ; D ← [β/(C + 1)] ; W ← S · T .

(6) [Normalize.] E ← PFA(S·D,0) ; $\overline{A}$ ← IPROD(A,E) ; ALTER(T·D,E); $\overline{B}$ ← IPROD(B,E) ; ERLA(E) .

(7) [Form lists of cell locations.] U ← $\overline{A}$ ; L1 ← 0 ; for I = 1,2,..., K + 1 , do (L1 ← PFA(U,L1) ; U ← TAIL(U)) ; L2 ← L1 ; if N = 2 , go to (8); for I = 1,2,...,N - 2 , do (L2 ← PFA(U,L2) ; U ← TAIL(U)) .

(8) [Extend $\overline{A}$ , if necessary.] If TAIL(U) ≠ 0 , go to (9); V ← PFA(0,0) ; SSUCC(V,U).

(9) [Extract leading digits of $\overline{B}$ .] $\overline{U}$ ← B ; if N ≠ 2 , (for I = 1,2,...,N - 2 , do U ← TAIL(U)) ; ADV(B2,U); ADV(B1,U).

(10) [Extract leading digits of $\overline{A}$ .] U ← FIRST(L2) ; ADV(A3,U); ADV(A2,U); ADV(A1,U).

(11) [Compute $\overline{q}$ .] If A1 $\geq$ B1 , ($\overline{Q}$ ← $\beta$ - 1 ; go to (12)); E ← A1 ; F ← A2 ; QR(E,F,B1); $\overline{Q}$ ← E .

(12) [If r < 0 , decrease $\overline{q}$ and repeat.] C1 ← $\overline{Q}$ ; C2 ← B1 ; MPY(C1,C2); R1 ← A1 - C1 ; R2 ← A2 - C2 ; if R2 < 0 , (R2 ← R2 + $\beta$ ; R1 ← R1 - 1) ; if R1 > 0 , go to (13); C1 ← $\overline{Q}$ ; C2 ← B2 ; MPY(C1,C2); R1 ← R2 - C1 ; if R1 > 0 , go to (13); R2 ← A3 - C2 ; if R1 < 0 or R2 < 0 , ($\overline{Q}$ ← $\overline{Q}$ - 1 ; go to (12)).

(13) [Initialize for subtraction.] U ← FIRST(L1) ; V ← $\overline{B}$ ; C1 ← 0 .

(14) [Subtract $\overline{q}\beta^j\overline{B}$ from $\overline{A}$ .] E ← -$\overline{Q}$ ; ADV(F,V); MPY(E,F); A1 ← FIRST(U) ; ADD3(C1,A1,F); if A1 < 0 , (A1 ← A1 + $\beta$ ; C1 ← C1 - 1) ; C1 ← C1 + E ; ALTER(A1,U); X ← U ; U ← TAIL(U) ; if V ≠ 0 , go to (14).

(15) [Finish subtraction; result negative?] DECAP(A1,U); SSUCC(0,X); A1 ← A1 + C1 ; if A1 = 0 , go to (17); U ← FIRST(L1) ; V ← $\overline{B}$ ; C1 ← 0 ; $\overline{Q}$ ← $\overline{Q}$ - 1 .

(16) [Add $\beta^j\overline{B}$ to $\overline{A}$ .] A1← FIRST(U) ; ADV(B1,V); ADD3(C1,A1,B1); ALTER(A1,U); U ← TAIL(U) ; if V ≠ 0 , go to (16).

(17) [Return for next digit, if any.] If $\overline{Q} \neq 0$ or $Q \neq 0$ , $Q \leftarrow PFA(W \cdot \overline{Q},Q))$ ; $L1 \leftarrow TAIL(L1)$ ; DECAP(U,L2); if $L1 \neq 0$ , go to (10).

(18) [Divide $\overline{A}$ by normalizer and erase lists.] $L \leftarrow IQRS(\overline{A},S \cdot D)$ ; DECAP(R,L); DECAP(U,L); ERLA($\overline{B}$); ERLA(L2); ERLA($\overline{A}$).

(19) [Set $L \leftarrow (Q,R)$ and return.] $L \leftarrow PFL(Q,PFL(R,0))$ ; return.

Computing Time: $\sim 1$ for $A = 0$ or $B = 0$ ; otherwise $\sim n$ for $m < n$ and $\sim n(m-n+1)$ for $m \geq n$ , where $m = L(A)$ and $n = L(B)$ .

The average computing time of IQR can be approximated by a function of $n = L(B)$ and $q = L(Q)$ . On the UNIVAC 1108, for $Q \neq 0$ , the average computing time is $90\ q + 190$ microseconds for $n = 1$ , and $75\ nq + 735\ n + 230\ q + 570$ microseconds for $n > 1$ .

## Q = IQ(A,B)

A and B are L-integers. Q = quot (A,B) , an L-integer.

Algorithm

(1) $L \leftarrow IQR(A,B)$ ; DECAP(Q,L); DECAP(R,L); ERLA(R); return.

R = IREM(A,B)

A and B are L-integers. R = rem (A,B) , an L-integer.

Algorithm

(1) L ← IQR(A,B); DECAP(Q,L); DECAP(R,L); ERLA(Q); return.

The theoretical computing times for IQ and IREM are the same as for IQR; the empirical computing times are nearly the same.

## 3.5. Greatest Common Divisors

We denote by gcd (a,b) the greatest common divisor of two integers. The greatest common divisor function is so defined that gcd (0,a) = gcd (a,0) = |a| (in particular, gcd (0,0) = 0) and gcd (a,b) > 0 if $a \neq 0$ or $b \neq 0$ . The following algorithm is an implementation of Lehmer's version of the Euclidean algorithm for the g.c.d.

Z = IGCD(I,J)

I and J are arbitrary L-integers. Z = gcd(I,J) , an L-integer.

Algorithm

(1) [I or J zero?] If I = 0 , (Z ← IABSL(J) ; return); if J = 0 , (Z ← IABSL(I) ; return).

(2) [Construct A = |I| .] A ← 0 ; T ← I .

(3) ADV(D,T); A ← PFA(|D|,A) ; if T ≠ 0 , go to (3).

(4) A ← INV(A) .

(5) [Construct B = |J| .] B ← 0 ; T ← J .

(6) ADV(D,T); B ← PFA(|D|,B) ; if T ≠ 0 , go to (6).

(7) B ← INV(B) .

(8) [Compare A and B .] S ← ICOMP(A,B) ; if S = 0 , ERLA(A); Z ← B ; return) ; if S < 0 , (T ← A ; A ← B ; B ← T) .

(9) [B = 0 ?] If B = 0 , (Z ← A ; return) .

(10) [B single-precision?] If TAIL(B) ≠ 0 , go to (12); C ← IREM(A,B) ; ERLA(A); if C = 0 , (Z ← B ; return) ; DECAP(A,B); DECAP(B,C).

(11) C ← rem(A,B) ; A ← B ; B ← C ; if B ≠ 0 , go to (11); Z ← PFA(A,0) ; return.

(12) [Compute normalizer, d .] T ← A .

(13) ADV(F,T); if T ≠ 0 , go to (13); D ← 1 ; E ← 0 ; F ← F + 1 ; if F ≠ β , QR(D,E,F).

(14) [Compute A' , A" , and N = length (A) .] T ← A ; N ← 0 ; C ← 0 ; G ← 0 .

(15) ADV(E,T); F ← D ; MPY(E,F); ADD3(C,F,G); G ← E ;
N ← N + 1 ; if T ≠ 0 , go to (15).

(16) A' ← F ; A" ← A' + 1 .

(17) [Compute B' , B" and M = length (B) .] T ← B ;
M ← 0 ; C ← 0 ; G ← 0 .

(18) ADV(E,T); F ← D ; MPY(E,F); ADD3(C,F,G); G ← E ;
M ← M + 1 ; if T ≠ 0 , go to (18).

(19) B" ← F ; if M < N , B" ← C + G ; if M < N - 1 ,
B" ← 0 ; B' ← B" + 1 .

(20) [Exceptional condition?] If B" = 0 or A' < B'
or A" = β , go to (25).

(21) [Initialize X,Y,U,V.] X ← 1 ; Y ← 0 ; U ← 0 ;
V ← 1 .

(22) [B' = 0 or B" = 0 ?] If B' = 0 or B" = 0 ,
go to (26).

(23) [Q' = Q" ?] Q' ← [A'/B'] ; Q" ← [A"/B"] ; if Q' = Q" ,
go to (24); if Y = 0 , go to (25); go to (26).

(24) [Modify A' , A" , B' , B" , X , Y , U , V .]
T ← A' - Q' B' ; A' ← B' ; B' ← T ; T ← A" - Q' B" ;
A" ← B" ; B" ← T ; T ← X - Q' U ; X ← U ; U ← T ;
T ← Y - Q' V ; Y ← V ; V ← T ; go to (22).

(25) [Divide A by B .] T ← IREM(A,B) ; ERLA(A); A ← B ;
B ← T ; go to (9).

(26) [Replace A by AX + BY , B by AU + BV .] T1 ← A ;
T2 ← B ; C1 ← 0 ; C2 ← 0 ; L1 ← 0 ; L2 ← 0 .

(27) A1 ← FIRST(T1) ; A2 ← FIRST(T2) ; E1 ← A1 ; F1 ← X ;
MPY(E1,F1); E2 ← A2 ; F2 ← Y ; MPY(E2,F2); ADD3(C1,F1,F2);
C1 ← C1 + E1 + E2 ; if F1 < 0 , (F1 ← F1 + β ;
C1 ← C1 - 1) ; ALTER(F1,T1); if F1 ≠ 0 , L1 ← T1 ;
E1 ← A1 ; F1 ← U ; MPY(E1,F1); E2 ← A2 ; F2 ← V ;
MPY(E2,F2); ADD3(C2,F1,F2); C2 ← C2 + E1 + E2 ; if
F1 < 0 , (F1 ← F1 + β ; C2 ← C2 - 1) ; ALTER(F1,T2);
if F1 ≠ 0 , L2 ← T2 ; U1 ← TAIL(T1) ; U2 ← TAIL(T2) ;
if U1 ≠ 0 and U2 = 0 , (U2 ← PFA(0,0 ; SSUCC(U2,T2)) ;
T1 ← U1 ; T2 ← U2 ; if T1 ≠ 0 , go to (27).

(28) U1 ← TAIL(L1) ; SSUCC(0,L1); ERLA(U1); if L2 ≠ 0 ,
(U2 ← TAIL(L2) ; SSUCC(0,L2); ERLA(U2)); go to (9).

Computing Time: Let $t = L(I)$ , $u = L(J)$ , $Z = \gcd(I,J)$ , $k = L(Z)$ , $m = \max(t,u)$ and $n = \min(t,u)$ . The maximum and average computing times are $\sim m$ if $I = 0$ or $J = 0$ , $\sim n(m - k + 1)$ otherwise. The minimum computing time is $\sim 1$ if $I = 0$ or $J = 0$ , $\sim n(m - n + 1) + k(n - k + 1)$ otherwise.

The following table contains some representative empirically observed computing times of IGCD for the UNIVAC 1108. These times are given in milliseconds for the cases $1 \leq k < m = n \leq 10$ . Inputs A and B for these cases were obtained by generating random relatively prime integers $\overline{A}$ and $\overline{B}$ of length $m - k$ and a random integer C of

length $k$ , and setting $A = \overline{A}C$ , $B = \overline{B}C$ , so that $C = \pm \gcd(A,B)$ . Computing times for cases in which $m > n$ can be approximated by adding to these times the time for the division of $A$ , an m-digit number, by $B$ , an n-digit number, using the computing time formula for IQR.

IGCD Computing Times

| n \ k | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 4.8 | | | | | | | | |
| 3 | 7.8 | 8.0 | | | | | | | |
| 4 | 11.8 | 14.3 | 7.6 | | | | | | |
| 5 | 17.0 | 21.0 | 13.2 | 9.7 | | | | | |
| 6 | 23.9 | 23.1 | 19.8 | 15.6 | 11.5 | | | | |
| 7 | 28.0 | 29.6 | 25.1 | 21.2 | 19.8 | 11.0 | | | |
| 8 | 34.9 | 36.4 | 31.2 | 28.6 | 26.0 | 19.5 | 14.6 | | |
| 9 | 41.0 | 42.0 | 38.1 | 37.9 | 31.7 | 25.7 | 22.7 | 15.5 | |
| 10 | 50.6 | 57.6 | 46.6 | 44.0 | 38.6 | 34.2 | 33.8 | 26.1 | 20.1 |

The least common multiple of integers a and b , lcm(a,b) is so defined that lcm(a,0) = lcm(0,a) = 0 and lcm(a,b) = |a · b/gcd(a,b)| if $a \neq 0$ and $b \neq 0$ . The following algorithm uses these formulas to compute lcm(a,b) for any two L-integers a and b .

C = ILCM(A,B)

A and B are arbitrary L-integers. C is the least common multiple of A and B , an L-integer.

Algorithm

(1) C ← 0 ; if A = 0 or B = 0 , return.

(2) D ← IGCD(A,B) ; E ← IQ(B,D) ; ERLA(D); F ← IPROD(A,E) ; ERLA(E); C ← IABSL(F) ; ERLA(F); return.

Computing Time: Let t = L(A) , u = L(B) , C = lcm(A,B) , k = L(C) , m = max(t,u) and n = min(t,u) . The computing time of ILCM(A,B) is ∿1 for A = 0 or B = 0 , ∿n(m - k + 1) otherwise.

## 3.6. Input-Output and Conversion

If a is a non-zero integer and $|a| = \sum_{i=0}^{n} a_i \cdot 10^i$ , $a_n \neq 0$ , in base 10 radix representation, the character list representation of a is the list $(S,a_n,\ldots,a_1,a_0)$ , where S is 36 (the SAC-1 character code for "+") if $a > 0$ and S is 37 (the SAC-1 character code for "-") if $a < 0$ .

The character list representation of the integer zero is the list (36,0). The sign S = 36 may be omitted from the list when $a \geq 0$ .

B = IDTOH(A)

A is the character list representation of an integer a . B is the list $(b_m,\ldots,b_1,b_0)$ , where $a = \sum_{i=0}^{m} b_i \theta^i$ is the base $\theta$ representation of a . B is the null list if a = 0 .

Algorithm

(1) U ← A ; B ← 0 ; S ← 1 ; if FIRST(U) $\geq$ 10 , (ADV(V,U) ; if V = 37 , S ← -1) ; if FIRST(U) = 0 , return; V ← INV(U) ; U ← V .

(2) BI ← 0 ; M ← 1 .

(3) ADV(AI,U) ; BI ← AI · M + BI ; M ← 10 · M ; if $M \neq \theta$ and $U \neq 0$ , go to (3).

(4) B ← PFA(S · BI, B) ; if $U \neq 0$ , go to (2); U ← INV(V) ; return.

Computing Time: $\sim L(a)$ .

On the UNIVAC 1108, the computing time of IDTOH is arrpoximately $18\lceil n/9 \rceil + 38n + 60$ microseconds, where $n = L_{10}(a)$ .

B = IHTOB(A)

A is a list $(a_m,\ldots,a_1,a_0)$ , where $a = \sum_{i=0}^{m} a_i \theta^i$

is the base $\theta$ representation of an integer $a$. B is the list $(b_0,b_1,\ldots,b_n)$, where $a = \sum_{i=0}^{n} b_i \beta^i$ is the base $\beta$ representation of $a$. If $a = 0$, then A and B are the null list.

Algorithm

(1) U ← A ; B ← 0 ; if U = 0 , return; ADV(AI,U) ;
B ← PFA(AI,0) .

(2) If U = 0 , return; ADV(AI,U) ; B ← IMADD(B,θ,AI) ;
go to (2).

Computing Time: $\sim L(a)^2$ .

On the UNIVAC 1108, the computing time of IHTOB is approximately $26\lceil n/9\rceil^2 + 100\lceil n/9\rceil + 44$ microseconds, where $n = L_{10}(a)$ .

B = IDTOB(A)

A is the character list representation of an integer $a$. B is the base $\beta$ radix representation of $a$, the internal canonical form.

Algorithm

(1) C ← IDTOH(A) ; B ← IHTOB(C) ; ERLA(C) ; return.

Computing Time: $\sim L(a)^2$ .

On the UNIVAC 1108, the computing time of IDTOB is approximately $26\lceil n/9 \rceil^2 + 118\lceil n/9 \rceil + 58n + 126$ microseconds, where $n = L_{10}(a)$ .

A = IREAD(U)

U is a FORTRAN integer designating an input unit. An L-integer is read from unit U and converted to internal canonical form. A is the location of the internal list representation. However, if unit U was initially positioned at an endfile record, then A = -1 . The external canonical form of the integer A must begin in column 1 of the first record on unit U , and may extend over any number of 72-character records. The last character must be followed by a blank. Thus, if there are n characters in the external form, the number of records read will be $\lceil (n+1)/72 \rceil$ . If A is positive, the initial plus sign may be omitted. If A is zero, the external form may be either '+0' or '0' .

Algorithm

(1) B ← 0 ; READ(U,RECORD) ; if RECORD(1) = -1 , (A ← -1 ; return).

(2) For N = 1,...,72 do (if RECORD(N) = 44 , go to (4) ; B ← PFA(RECORD(N),B)).

(3) READ(U,RECORD) ; if RECORD(1) $\neq$ -1 , go to (2).

(4) B $\leftarrow$ INV(B) ; A $\leftarrow$ IDTOB(B) ; ERLA(B) ; return.

Computing Time: $\sim L(A)^2$ .

On the UNIVAC 1108, the computing time of IREAD is approximately $6925k + 26\lceil n/9 \rceil^2 + 118\lceil n/9 \rceil + 96n + 157$ microseconds, where $k$ is the number of records read and $n = L_{10}(a)$ . Computing time here refers to the time of the central processing unit. Elapsed real time may of course be considerably greater, depending on the characteristics and status of the input unit U .

B = IBTOH(A)

A is the base $\beta$ internal canonical form of an integer $a$ . B is the list $(b_n,\ldots,b_1,b_0)$ , where $a = \sum_{i=0}^{n} b_i \theta^i$ in base $\theta$ representation. If $a = 0$ then A and B are the null list.

Algorithm

(1) B $\leftarrow$ 0 ; Q $\leftarrow$ BORROW(A) .

(2) If Q = 0 , return; L $\leftarrow$ IQRS(Q,$\theta$) ; ERLA(Q) ;
DECAP(Q,L) ; DECAP(BI,L) ; B $\leftarrow$ PFA(BI,B) ; go to (2).

Computing Time: $\sim L(a)^2$ .

On the UNIVAC 1108, the computing time of IBTOH is approximately $109\lceil n/9 \rceil^2 + 110\lceil n/9 \rceil + 30$ microseconds, where $n = L_{10}(a)$ .

B = IHTOD(A)

A is the list $(a_m,\ldots,a_1,a_0)$ , where $\sum_{i=0}^{m} a_i\,\theta^i$ is the base $\theta$ representation of an integer $a$ (A is the null list if $a = 0$ ). B is the character list representation of $a$ .

Algorithm

(1) If $A = 0$ , (B ← PFA(36,PFA(0,0)) ; return);
B ← 0 ; U ← A ; S ← FIRST(A) .

(2) If $U = 0$ , go to (4); ADV(AI,U) ; M ← [θ/10] .

(3) BI ← [AI/M] ; AI ← AI − BI · M ; if $B \neq 0$ or $BI \neq 0$ , B ← PFA(|BI|,B) ; M ← [M/10] ; if $M \neq 0$ , go to (3); go to (2).

(4) If $S > 0$ , S ← 36 ; if $S < 0$ , S ← 37 ;
B ← PFA(S,INV(B)) ; return.

Computing Time: $\sim L(a)$ .

On the UNIVAC 1108, the computing time of IHTOD is approximately $26\lceil n/9\rceil + 56n + 46$ microseconds, where $n = L_{10}(a)$ .

B = IBTOD(A)

A is the base β internal canonical form of an integer $a$ . B is the character list representation of $a$ .

Algorithm

(1) C ← IBTOH(A) ; B ← IHTOD(C) ; ERLA(C) ; return.

Computing Time: $\sim L(a)^2$ .

On the UNIVAC 1108, the computing time of IBTOD is approximately $109\lceil n/9\rceil^2 + 146\lceil n/9\rceil + 56n + 86$ microseconds, where $n = L_{10}(a)$ .

IWRITE(U,A)

U is an output unit number. A is an integer in base β radix representation (internal canonical form). A is converted to decimal (external canonical form) and written on unit U as a sequence of $\lceil (n + 1)/72\rceil$ records, where n is the number of characters in the external canonical form. The first character (the sign) is placed in column 1 and the last record is filled out with blanks. Each record consists of 72 characters, except that if U designates printed output each record contains 73 characters, the first being a blank designating single-space printing.

Algorithm

(1) B ← IBTOD(A).

(2) For N = 1 , ... , 72 do (if B = 0 , go to (4);
DECAP(C,B) ; RECORD(N) ← C).

(3) WRITE(U,RECORD) ; go to (2).

(4) For I = N , ..., 72 do RECORD(I) ← 44 .

(5) WRITE(U,RECORD) ; return.

Computing Time: $\sim L(A)^2$ .

On the UNIVAC 1108, the computing time is approximately $7620k + 109\lceil n/9 \rceil^2 + 146\lceil n/9 \rceil + 76n + 102$ microseconds, where $k$ is the number of records written and $n = L_{10}(a)$ . Computing time here refers only to the time of the central processing unit, which may be considerably less than the elapsed real time.

## 4. Fortran Program Listings

```
      SUBROUTINE ADD3(A,B,C)
      COMMON /TR3/ BETA,THETA
1     B=A+B+C
2     A=S/BETA
3     B=B-A*BETA
      RETURN
      END



      SUBROUTINE COMPAT(S,A,B)
      COMMON /TR3/ BETA,THETA
      INTEGER S,A,B,BETA
1     B=A+3
      IF (S.LT.0) GO TO 3
2     IF (B.GE.0) GO TO 4
      B=B+BETA
      A=-1
      RETURN
3     IF (B.LE.0) GO TO 4
      B=B-BETA
      A=1
      RETURN
4     A=0
      RETURN
      END



      INTEGER FUNCTION IABSL(A)
      INTEGER A,B,S,BORROW,INEG,ISIGNL
10    S=ISIGNL(A)
20    IF (S.LT.0) GO TO 30
      B=BORROW(A)
      GO TO 31
30    B=INEG(A)
31    IABSL=B
      RETURN
      END



      INTEGER FUNCTION IBTOD(A)
      INTEGER A,C
1     C=IBTOH(A)
      IBTOD=IHTOD(C)
      CALL ERLA(C)
      RETURN
      END



      INTEGER FUNCTION IBTOH(A)
      COMMON /TR3/BETA,THETA
```

```
      INTEGER A,B,EETA,BI,Q,THETA,BORROW,PFA
10    B=0
      Q=BORROW(A)
20    IF (Q.EQ.0) GO TO 21
      L=IQRS(Q,THETA)
      CALL ERLA(Q)
      CALL DECAP(Q,L)
      CALL DECAP(BI,L)
      B=PFA(BI,B)
      GO TO 20
21    IBTOH=B
      RETURN
      END



      INTEGER FUNCTION ICOMP(A,B)
      INTEGER A,B,S,T,U,V,X,Y
      INTEGER FIRST,TAIL,ISIGNL
1     U=ISIGNL(A)
      V=ISIGNL(B)
      S=U-V
      IF(U.EQ.0.OR.V.EQ.0)GO TO 41
      IF(S.NE.0)GO TO 4
      X=A
      Y=B
2     T=FIRST(X)-FIRST(Y)
      IF (T.NE.0) S=T
      X=TAIL(X)
      Y=TAIL(Y)
      IF (X.EQ.0) GO TO 3
      IF (Y.NE.0) GO TO 2
      S=U
      GO TO 41
3      IF(Y.EQ.0) GO TO 4
      S=-U
      GO TO 41
4     IF (S.GT.0) S=1
      IF (S.LT.0) S=-1
41    ICOMP=S
      RETURN
      END



      INTEGER FUNCTION IDIF(A,B)
      INTEGER A,B,C,F,G,AI,BI,DI,EI,S,T,U,V
      INTEGER BORROW,FIRST,INV,PFA,TAIL,ISIGNL,INEG
10    IF (A.NE.0) GO TO 11
      C=INEG(B)
      GO TO 161
11    IF (B.NE.0) GO TO 12
      C=BORROW(A)
      GO TO 161
```

```
12    C=0
      U=A
      V=B
20    S=ISIGNL(A)
      T=ISIGNL(B)
      IF (S+T.NE.0) GO TO 80
      DI=0
30    CALL ADV(AI,U)
      CALL ADV(BI,V)
      BI=-BI
      CALL ADD3(DI,AI,BI)
      C=PFA(AI,C)
40    IF (U.EQ.0) GO TO 60
      IF (V.NE.0) GO TO 30
50    CALL ADV(AI,U)
      CALL ADD3(DI,AI,0)
      C=PFA(AI,C)
      IF (U.NE.0) GO TO 50
      GO TO 70
60    IF (V.EQ.0) GO TO 70
      CALL ADV(BI,V)
      BI=-BI
      CALL ADD3(DI,BI,0)
      C=PFA(BI,C)
      GO TO 60
70    IF (DI.NE.0) C=PFA(DI,C)
      C=INV(C)
      GO TO 151
80    F=0
90    CALL ADV(AI,U)
      CALL ADV(BI,V)
      DI=AI-BI
      IF (DI.NE.0) F=DI
      C=PFA(DI,C)
100   IF (U.EQ.0) GO TO 120
      IF (V.NE.0) GO TO 90
110   CALL ADV(DI,U)
      IF (DI.NE.0) F=DI
      C=PFA(DI,C)
      IF (U.NE.0) GO TO 110
      GO TO 130
120   IF (V.EQ.0) GO TO 130
      CALL ADV(DI,V)
      DI=-DI
      IF (DI.NE.0) F=DI
      C=PFA(DI,C)
      GO TO 120
130   IF (F.NE.0) GO TO 140
      CALL ERLA(C)
      C=0
      GO TO 161
140   C=INV(C)
      S=1
      IF (F.LT.0) S=-1
```

```
      F=0
      EI=0
      U=C
150   DI=FIRST(U)
      CALL COMPAT(S,EI,DI)
      IF (DI.NE.0) F=U
      CALL ALTER(DI,U)
      U=TAIL(U)
      IF (U.NE.0) GO TO 150
160   G=TAIL(F)
      CALL SSUCC(C,F)
      CALL ERLA(G)
161   IDIF=C
      RETURN
      END



      INTEGER FUNCTION IDTOB(A)
      INTEGER A,C
1     C=IDTOH(A)
      IDTOB=IHTOB(C)
      CALL ERLA(C)
      RETURN
      END



      INTEGER FUNCTION IDTOH(A)
      COMMON/TRB/BETA,THETA
      INTEGER BETA,THETA,A
      INTEGER FIRST,PFA,INV
      INTEGER U,S,BI,M,AI,V
10    U = A
      IDTOH = 0
      S=1
      IF(FIRST(U).LT.10)GO TO 15
      CALL ADV(V,U)
      IF(V.EQ.37)S=-1
  15  IF(FIRST(U).EQ.0)RETURN
      V = INV(U)
      U = V
20    BI=0
      M = 1
30    CALL ADV(AI,U)
      BI=AI*M+BI
      M = 10*M
      IF(M.NE.THETA.AND.U.NE.0)GO TO 30
      IDTOH = PFA(S*BI,IDTOH)
      IF(U.NE.0)GO TO 20
      U = INV(V)
      RETURN
      END
```

```
      INTEGER FUNCTION IGCD(I,J)
      COMMON /TR3/BETA,THETA
      INTEGER BETA,THETA
      INTEGER FIRST,PFA,TAIL
      INTEGER A,A1,A2,ADPRIM,APRIME,B,BDPRIM,BPRIME
      INTEGER C,C1,C2,D,E,E1,E2,F,F1,F2,G,GDPRIM,GPRIME
      INTEGER S,T,T1,T2,U,U1,U2,V,X,Y
10    IF (I.NE.0) GO TO 11
      IGCD=IABSL(J)
      RETURN
11    IF (J.NE.0) GO TO 20
      IGCD=IABSL(I)
      RETURN
20    A=0
      T=I
30    CALL ADV(D,T)
      A=PFA(IABS(D),A)
      IF (T.NE.0) GO TO 30
40    A=INV(A)
50    B=0
      T=J
60    CALL ADV(D,T)
      B=PFA(IABS(D),B)
      IF (T.NE.0) GO TO 60
70    B=INV(B)
80    S=ICOMP(A,B)
      IF (S.NE.0) GO TO 81
      CALL ERLA(A)
      IGCD=B
      RETURN
81    IF (S.GE.0) GO TO 90
      T=A
      A=B
      B=T
90    IF (B.NE.0) GO TO 100
      IGCD=A
      RETURN
100   IF (TAIL(B).NE.0) GO TO 120
      C=IREM(A,B)
      CALL ERLA(A)
      IF (C.NE.0) GO TO 101
      IGCD=B
      RETURN
101   CALL DECAP(A,B)
      CALL DECAP(B,C)
110   C=A-A/B*B
      A=B
      B=C
      IF (B.NE.0) GO TO 110
      IGCD=PFA(A,0)
      RETURN
120   T=A
```

```
130   CALL ADV(F,T)
      IF (T.NE.0) GO TO 130
      D=1
      E=0
      F=F+1
      IF (F.NE.BETA) CALL QR(D,E,F)
140   T=A
      N=0
      C=0
      G=0
150   CALL ADV(E,T)
      F=0
      CALL MPY(E,F)
      CALL ADD3(C,F,G)
      G=E
      N=N+1
      IF (T.NE.C) GO TO 150
160   APRIME=F
      ADPRIM=APRIME+1
170   T=B
      M=0
      C=0
      G=C
180   CALL ADV(E,T)
      F=D
      CALL MPY(E,F)
      CALL ADD3(C,F,C)
      G=E
      M=M+1
      IF (T.NE.0) GO TO 180
190   BDPRIM=F
      IF(M.LT.N) BDPRIM=C+G
      IF (M.LT.N-1) BDPRIM=0
      BPRIME=BDPRIM+1
200   IF (BDPRIM.EQ.0.OR.APRIME.LT.BPRIME.OR.ADPRIM.EQ.BETA)
     C GO TO 250
210   X=1
      Y=0
      U=0
      V=1
220   IF (BPRIME.EQ.0.OR.BDPRIM.EQ.0) GO TO 260
      IF (BPRIME.EQ.0.OR.BDPRIM.EQ.0) GO TO 260
230   QPRIME=APRIME/BPRIME
      QDPRIM=ADPRIM/BDPRIM
      IF (QPRIME.EQ.QDPRIM) GO TO 240
      IF (Y.EQ.0) GO TO 250
      GO TO 260
240   T=APRIME-QPRIME*BPRIME
      APRIME=BPRIME
      BPRIME=T
      T=ADPRIM-QPRIME*BDPRIM
      ADPRIM=BDPRIM
      BDPRIM=T
      T=X-QPRIME*U
```

```
      X=U
      U=T
      T=Y-QPRIME*V
      Y=V
      V=T
      GO TO 220
  250 T=IREM(A,B)
      CALL ERLA(A)
      A=B
      B=T
      GO TO 90
  260 T1=A
      T2=B
      C1=0
      C2=0
      L1=0
      L2=0
  270 A1=FIRST(T1)
      A2=FIRST(T2)
      E1=A1
      F1=X
      CALL MPY(E1,F1)
      E2=A2
      F2=Y
      CALL MPY(E2,F2)
      CALL ADD3(C1,F1,F2)
      C1=C1+E1+E2
      IF (F1.GE.0) GO TO 271
      F1=F1+BETA
      C1=C1-1
  271 CALL ALTER(F1,T1)
      IF (F1.NE.0) L1=T1
      E1=A1
      F1=U
      CALL MPY(E1,F1)
      E2=A2
      F2=V
      CALL MPY(E2,F2)
      CALL ADD3(C2,F1,F2)
      C2=C2+E1+E2
      IF (F1.GE.0) GO TO 272
      F1=F1+BETA
      C2=C2-1
  272 CALL ALTER(F1,T2)
      IF (F1.NE.0) L2=T2
      U1=TAIL(T1)
      U2=TAIL(T2)
      IF (U1.EQ.0.OR.U2.NE.0) GO TO 273
      U2=PFA(0,0)
      CALL SSUCC(U2,T2)
  273 T1=U1
      T2=U2
      IF (T1.NE.0) GO TO 270
  280 U1=TAIL(L1)
```

```
      CALL SSUCC(C,L1)
      CALL ERLA(U1)
      IF (L2.EQ.0) GO TO 90
      U2=TAIL(L2)
      CALL SSUCC(C,L2)
      CALL ERLA(U2)
      GO TO 90
      END



      INTEGER FUNCTION IHTOB(A)
      COMMON /TR3/BETA,THETA
      INTEGER A,AI,B,BETA,THETA,U,PFA
10    U=A
      B=0
      IF (U.EQ.0) GO TO 21
      CALL ADV(AI,U)
      B=PFA(AI,0)
20    IF (U.EQ.0) GO TO 21
      CALL ADV(AI,U)
      B=IMADD(B,THETA,AI)
      GO TO 20
21    IHTOB=B
      RETURN
      END



      INTEGER FUNCTION IHTOD(A)
      COMMON/TR3/BETA,THETA
      INTEGER BETA,THETA,A
      INTEGER PFA,FIRST,INV
      INTEGER U,S,AI,M,BI
10    IF(A.NE.0)GO TO 15
      IHTOD = PFA(36,PFA(0,0))
      RETURN
15    IHTOD = 0
      U = A
      S = FIRST(A)
20    IF(U.EQ.0)GO TO 40
      CALL ADV(AI,U)
      M = THETA/10
30    BI = AI/M
      AI=AI-BI*M
      IF(IHTOD.NE.0.OR.BI.NE.0)IHTOD = PFA(IABS(BI),IHTOD)
      M = M/10
      IF(M)30,20,30
40    IF(S.GT.0)S=36
      IF(S.LT.0)S=37
      IHTOD = PFA(S,INV(IHTOD))
      RETURN
      END
```

```
      INTEGER FUNCTION ILCM(A,B)
      INTEGER A,B,D,E,F
      ILCM = 0
      IF(A.EQ.0)RETURN
      IF(B.EQ.0)RETURN
      D = IGCD(A,B)
      E=IQ(B,D)
      CALL ERLA(D)
      F=IPROD(A,E)
      CALL ERLA(E)
      ILCM=IABSL(F)
      CALL ERLA(F)
      RETURN
      END



      INTEGER FUNCTION IMADD(A,B,C)
      INTEGER A,B,C,D,E,F,G,H,T
      INTEGER FIRST,TAIL,PFA
10    T=A
      G=C
      H=0
20    F=FIRST(T)
      E=B
      CALL MPY(E,F)
      CALL ADD3(G,F,H)
      CALL ALTER(F,T)
      H=E
      E=T
      T=TAIL(T)
      IF (T.NE.0) GO TO 20
30    D=A
      G=G+H
      IF (G.EQ.0) GO TO 31
      F=PFA(G,0)
      CALL SSUCC(F,E)
31    IMADD=D
      RETURN
      END



      INTEGER FUNCTION INEG(A)
      INTEGER A,AI,B,U,INV,PFA
10    B=0
      IF (A.EQ.0) GO TO 31
      U=A
20    CALL ADV(AI,U)
```

```
      B=PFA(-AI,B)
      IF (U.NE.0) GO TO 20
30    B=INV(B)
31    INEG=B
      RETURN
      END


      INTEGER FUNCTION IPOWER(A,N)
      INTEGER A,B,C,BORROW,PFA,IPROD
10    IF (N.NE.0) GO TO 11
      B=PFA(1,0)
      GO TO 21
11    B=BORROW(A)
      IF (A.EQ.0) GO TO 21
      J=1
20    IF (J.EQ.N) GO TO 21
      C=IPROD(B,A)
      CALL ERLA(B)
      B=C
      J=J+1
      GO TO 20
21    IPOWER=B
      RETURN
      END


      INTEGER FUNCTION IPROD(A,B)
      INTEGER A,B,C,E,F,G,S,T,U,V,W,X,Z,AI,BJ
      INTEGER FIRST,TAIL,PFA,LENGTH
10    C=0
      IF (A.EQ.0.OR.B.EQ.0) GO TO 71
20    S=PFA(0,0)
      T=PFA(0,S)
      M=LENGTH(A)+LENGTH(B)-2
      C=T
      IF (M.EQ.0) GO TO 30
      DO 21 I=1,M
21    C=PFA(0,C)
30    V=B
      W=C
40    X=A
      CALL ADV(BJ,V)
      Z=W
      G=0
50    CALL ADV(AI,X)
      F=BJ
      CALL MPY(AI,F)
      E=FIRST(Z)
      CALL ADD3(G,E,F)
      CALL ALTER(E,Z)
```

```
      G=AI+G
      Z=TAIL(Z)
      IF (X.NE.C) GO TO 50
60    CALL ALTER(G,Z)
      W=TAIL(W)
      IF(V.NE.C) GO TO 40
70    IF (G.NE.C) GO TO 71
      CALL SSUCC(C,T)
      CALL DECAP(G,S)
71    IPROD=C
      RETURN
      END



      INTEGER FUNCTION IQ(A,B)
      INTEGER A,B,L,Q,R,IQR
1     L=IQR(A,B)
      CALL DECAP(Q,L)
      CALL DECAP(R,L)
      CALL ERLA(R)
      IQ=Q
      RETURN
      END



      INTEGER FUNCTION IQR(A,B)
      COMMON /TR3/ BETA,THETA
      INTEGER BETA,THETA
      INTEGER BORROW,FIRST,IPROD,IQRS,ISIGNL,LENGTH,PFA,PFL,TAIL
      INTEGER A,A1,A2,A3,ABAR,B,B1,B2,BBAR,C,C1,C2,D,E,F,I,J,K
      INTEGER L,L1,L2,M,N,Q,QBAR,R,R1,R2,S,T,U,V,W,X
10    Q=C
      IF (A.NE.C.AND.B.NE.C) GO TO 20
      R=BORROW(A)
      GO TO 190
20    IF (TAIL(B).NE.C) GO TO 30
      B1=FIRST(B)
      L=IQRS(A,B1)
      CALL DECAP(G,L)
      CALL DECAP(R,L)
      IF (R.NE.C) R=PFA(R,C)
      GO TO 190
30    M=LENGTH(A)
      N=LENGTH(B)
40    K=M-N
      IF (K.GE.C) GO TO 50
      R=BORROW(A)
      GO TO 190
50    S=ISIGNL(A)
      U=B
      J=N-1
      DO 51 I=1,J
```

```
 51   U=TAIL(U)
      C=FIRST(U)
      T=1
      IF (C.GE.0) GO TO 52
      T=-1
      C=-C
 52   D=BETA/(C+1)
      W=S*T
 60   E=PFA(S*D,0)
      ABAR=IPROD(A,E)
      CALL ALTER(T*D,E)
      BBAR=IPROD(B,E)
      CALL ERLA(E)
 70   U=ABAR
      L1=0
      J=K+1
      DO 71 I=1,J
      L1=PFA(U,L1)
 71   U=TAIL(U)
      L2=L1
      J=N-2
      IF (J.EQ.0) GO TO 80
      DO 72 I=1,J
      L2=PFA(U,L2)
 72   U=TAIL(U)
 80   IF (TAIL(U).NE.0) GO TO 90
      V=PFA(C,0)
      CALL SSUCC(V,U)
 90   U=BBAR
      J=N-2
      IF (J.EQ.0) GO TO 92
      DO 91 I=1,J
 91   U=TAIL(U)
 92   CALL ADV(B2,U)
      CALL ADV(B1,U)
100   U=FIRST(L2)
      CALL ADV(A3,U)
      CALL ADV(A2,U)
      CALL ADV(A1,U)
110   IF (A1.LT.B1) GO TO 111
      QBAR=BETA-1
      GO TO 120
111   E=A1
      F=A2
      CALL QR(E,F,B1)
      QBAR=E
120   C1=QBAR
      C2=B1
      CALL MPY(C1,C2)
      R1=A1-C1
      R2=A2-C2
      IF (R2.GE.0) GO TO 121
      R2=R2+BETA
      R1=R1-1
```

```
121   IF (R1.GT.0) GO TO 130
      C1=QBAR
      C2=B2
      CALL MPY(C1,C2)
      R1=R2-C1
      IF (R1.GT.0) GO TO 130
      R2=A3-C2
      IF (R1.GE.0.AND.R2.GE.0) GO TO 130
      QBAR=QBAR-1
      GO TO 120
130   U=FIRST(L1)
      V=B3AR
      C1=0
140   E=-QBAR
      CALL ADV(F,V)
      CALL MPY(E,F)
      A1=FIRST(U)
      CALL ADD3(C1,A1,F)
      IF (A1.GE.0) GO TO 141
      A1=A1+BETA
      C1=C1-1
141   C1=C1+E
      CALL ALTER(A1,U)
      X=U
      U=TAIL(U)
      IF (V.NE.0) GO TO 140
150   CALL DECAP(A1,U)
      CALL SSUCC(0,X)
      A1=A1+C1
      IF (A1.EQ.0) GO TO 170
      U=FIRST(L1)
      V=B3AR
      C1=0
      QBAR=QBAR-1
160   A1=FIRST(U)
      CALL ADV(B1,V)
      CALL ADD3(C1,A1,B1)
      CALL ALTER(A1,U)
      U=TAIL(U)
      IF (V.NE.0) GO TO 160
170   IF (QBAR.NE.0.OR.Q.NE.0) Q=PFA(W*QBAR,Q)
      L1=TAIL(L1)
      CALL DECAP(U,L2)
      IF (L1.NE.0) GO TO 100
180   L=IQRS(ABAR,S*D)
      CALL DECAP(R,L)
      CALL DECAP(U,L)
      CALL ERLA(BBAR)
      CALL ERLA(L2)
      CALL ERLA(ABAR)
190   IQR=PFL(Q,PFL(R,0))
      RETURN
      END
```

```
      INTEGER FUNCTION IQRS(A,B)
      INTEGER A,B,C,E,F,Q
      INTEGER CINV,PFA,PFL
1     Q=0
      E=0
      IF(A.EQ.0) GO TO 3
      C=CINV(A)
2     CALL DECAP(F,C)
      CALL QR(E,F,B)
      IF(Q.NE.0.OR.E.NE.0) Q=PFA(E,Q)
      E=F
      IF(C.NE.0) GO TO 2
3     IQRS=PFL(Q,PFA(E,0))
      RETURN
      END



      INTEGER FUNCTION IREAD(U)
      COMMON/TR1/AVAIL,STAK,RECORD(72)
      INTEGER AVAIL,STAK,RECORD,U
      INTEGER PFA,IDTOB,INV
      INTEGER B,N
 10   B = 0
      CALL READ(U,RECORD)
      IF(RECORD(1).NE.-1)GO TO 20
      IREAD = -1
      RETURN
 20   DO 28 N = 1,72
      IF(RECORD(N).EQ.44)GO TO 40
 28   B = PFA(RECORD(N),B)
 30   CALL READ(U,RECORD)
      IF(RECORD(1).GE.0)GO TO 20
 40   B = INV(B)
      IREAD = IDTOB(B)
      CALL ERLA(B)
      RETURN
      END



      INTEGER FUNCTION IREM(A,B)
      INTEGER A,B,L,Q,R
1     L=IQR(A,B)
      CALL DECAP(Q,L)
      CALL DECAP(R,L)
      CALL ERLA(Q)
      IREM=R
      RETURN
      END
```

```
      INTEGER FUNCTION ISIGNL(A)
      INTEGER A,B,S,T
10    S=0
      IF (A.EQ.0) GO TO 31
      T=A
20    CALL ADV(B,T)
      IF (B.EQ.0) GO TO 20
30    S=1
      IF (B.LT.0) S=-1
31    ISIGNL=S
      RETURN
      END



      INTEGER FUNCTION ISUM(A,B)
      INTEGER A,B,C,F,G,AI,BI,DI,EI,S,T,U,V
      INTEGER BORROW,FIRST,INV,PFA,TAIL,ISIGNL
10    IF (A.NE.0) GO TO 11
      C=BORROW(B)
      GO TO 161
11    IF (B.NE.0) GO TO 12
      C=BORROW(A)
      GO TO 161
12    C=0
      U=A
      V=B
20    S=ISIGNL(A)
      T=ISIGNL(B)
      IF (S+T.EQ.0) GO TO 80
      DI=0
30    CALL ADV(AI,U)
      CALL ADV(BI,V)
      CALL ADD3(DI,AI,BI)
      C=PFA(AI,C)
40    IF (U.NE.0) GO TO 41
      U=V
      GO TO 60
41    IF (V.NE.0) GO TO 30
50    CALL ADV(AI,U)
      CALL ADD3(DI,AI,0)
      C=PFA(AI,C)
60    IF (U.NE.0) GO TO 50
70    IF (DI.NE.0) C=PFA(DI,C)
      C=INV(C)
      GO TO 161
80    F=0
90    CALL ADV(AI,U)
      CALL ADV(BI,V)
      DI=AI+BI
      IF (DI.NE.0) F=DI
      C=PFA(DI,C)
100   IF (U.NE.0) GO TO 101
```

```
      U=V
      GO TO 120
 101  IF (V.NE.0) GO TO 90
110   CALL ADV(F,U)
      C=PFA(F,C)
 120  IF (U.NE.0) GO TO 110
 130  IF (F.NE.0) GO TO 140
      CALL ERLA(C)
      C=0
      GO TO 161
 140  C=INV(C)
      S=1
      IF (F.LT.0) S=-1
      F=0
      EI=0
      U=C
 150  DI=FIRST(U)
      CALL COMPAT(S,EI,DI)
      IF (DI.NE.0) F=U
      CALL ALTER(DI,U)
      U=TAIL(U)
      IF (U.NE.0) GO TO 150
 160  G=TAIL(F)
      IF(G.EQ.0) GO TO 161
      CALL SSUCC(C,F)
      CALL ERLA(G)
 161  ISUM=C
      RETURN
      END


      SUBROUTINE IWRITE(U,A)
      COMMON/TR1/AVAIL,STAK,RECORD(72)
      INTEGER AVAIL,STAK,RECORD,U,A
      INTEGER IBTOD,N,B,C
 10   B = IBTOD(A)
 20   DO 28 N = 1,72
      IF(B.EQ.0)GO TO 40
      CALL DECAP(C,B)
 28   RECORD(N)=C
 30   CALL WRITE(U,RECORD)
      GO TO 20
 40   DO 48 I=N,72
 48   RECORD(I)=44
 50   CALL WRITE(U,RECORD)
      RETURN
      END


      SUBROUTINE MPY(A,B)
      COMMON /TR3/ BETA,THETA
      INTEGER A,B,BETA
```

```
      B=A*B
      A=B/BETA
      B=B-A*BETA
      RETURN
      END


      SUBROUTINE GR(A,B,C)
      COMMON /TR3/ BETA,THETA
      INTEGER A,B,C,E,BETA,THETA
1     E=A*BETA+B
2     A=E/C
3     B=E-C*A
      RETURN
      END
```

## 5. A Test Program

The following Fortran main program may be used as a partial test of whether this system has been correctly implemented. The test provided is very limited, but it is at least complete in the sense that each subprogram of the Integer Arithmetic System is executed at least once. The test program also serves to illustrate some features common to SAC-1 user programs.

This program reads three L-integers, A , B and C , and performs the following arithmetic:

$$D = A + B$$

$$E = \mathrm{lcm}(D,C)$$

$$F = \mathrm{gcd}(D,C)$$

$$G = E \cdot F$$

$$H = G^2$$

$$I = D \cdot C$$

$$J = I^2$$

$$K = H - J$$

Each result is printed using IWRITE and all integers are erased using ERLA. The length L of the available space list is computed and printed at the beginning and again at the end; the length should be the same in each case. Since $\mathrm{gcd}(u,v) \cdot \mathrm{lcm}(u,v) = |u \cdot v|$ for all integers u and v , the program should produce results $G = |I|$ ,

H = J and K = 0 .

Note that the values assigned to BETA, THETA, IN and OUT at the beginning of the program depend on the SAC-1 implementation and will generally need to be changed. Also the second argument of the subroutine BEGIN, the dimension of the array SPACE, must be a multiple of the number of words per cell.

## Test Program

```
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR3/ BETA,THETA
      INTEGER AVAIL,STAK,RECORD,BETA,THETA
      INTEGER IN,OUT,SPACE
      INTEGER A,B,C,D,E,F,G,H,I,J,K,L,
      DIMENSION SPACE(1000)
      BETA = 2**33
      THETA = 10**9
      IN = 5
      OUT = 6
      CALL BEGIN(SPACE,1000)
      L = LENGTH(AVAIL)
      PRINT 1,L
    1 FORMAT(/3H L=,I6)
```

```
      A = IREAD(IN)
      B = IREAD(IN)
      C = IREAD(IN)
      D = ISUM(A,B)
      CALL ERLA(A)
      CALL ERLA(B)
      PRINT 2
    2 FORMAT(/3H D=)
      CALL IWRITE(OUT,D)
      E = ILCM(D,C)
      PRINT 3
    3 FORMAT(/3H E=)
      CALL IWRITE(OUT,E)
      F = IGCD(D,C)
      PRINT 4
    4 FORMAT(/3H F=)
      CALL IWRITE(OUT,F)
      G = IPROD(E,F)
      PRINT 5
    5 FORMAT(/3H G=)
      CALL IWRITE(OUT,G)
      CALL ERLA(E)
      CALL ERLA(F)
```

```
      H = IPOWER(G,2)
      CALL ERLA(G)
      PRINT 6
    6 FORMAT(/3H H=)
      CALL IWRITE(OUT,H)
      I = IPROD(D,C)
      CALL ERLA(C)
      CALL ERLA(D)
      PRINT 7
    7 FORMAT(/3H I=)
      CALL IWRITE(OUT,I)
      J = IPOWER(I,2)
      CALL ERLA(I)
      PRINT 8
    8 FORMAT(/3H J=)
      CALL IWRITE(OUT,J)
      K = IDIF(H,J)
      CALL ERLA(H)
      CALL ERLA(J)
      PRINT 9
    9 FORMAT(/3H K=)
      CALL IWRITE(OUT,K)
      CALL ERLA(K)
      L = LENGTH(AVAIL)
      PRINT 1,L
      STOP
      END
```

## Input Data

+561237854126098503214568621456878

-856412302147521463214569879874569

+957648214632857641336412458769 58

## Output

L = 500

D =

-29517444802142296000000125841769 1

E =

+9422442771765164544673661962504429829756270127704019614918821326

F =

+3

G =

+2826732831529549363402098588751328948926881038311205884475646397 8

H =

+799041850084706370360814639852386554185928739454249807606290227998154401871648676345518560272396656915991920442376880614011584484

I =

-2826732831529549363402098588751328948926881038311205884475646397 8

J =

+799041850084706370360814639852386554185928739454249807606290227998154401871648676345518560272396656915991920442376880614011584484

K =

+0

L = 500

## REFERENCES

[1] Fortran vs. Basic Fortran-A Programming Language for Information Processing on Automatic Data Processing Systems, Comm. A.C.M., Vol. 7, No. 10 (Oct. 1964), pp. 591-625.

[2] Collins, G. E., The SAC-1 List Processing System, Univ. of Wisconsin Computer Sciences Dept. Tech. Report No. 129 (34 pages), July 1971.

[3] Collins, G. E., and J. R. Pinkert, The Revised SAC-1 Integer Arithmetic System, Univ. of Wisconsin Computing Center Tech. Report No. 9 (50 pages), Nov. 1968.

[4] Collins, G. E., The SAC-1 Polynomial System, Univ. of Wisconsin Computer Sciences Dept. Tech. Report No. 115 (66 pages), March 1971.

[5] Collins, G. E., The SAC-1 Rational Function System, Univ. of Wisconsin Computer Sciences Dept. Tech. Report No. 135 (32 pages), Sept. 1971.

[6] Collins, G. E., L. E. Heindel, E. Horowitz, M. T. McClellan, and D. R. Musser, The SAC-1 Modular Arithmetic System, Univ. of Wisconsin Computing Center Tech. Report No. 10 (50 pages), June 1969.

[7] Collins, G. E., and E. Horowitz, The SAC-1 Partial Fraction Decomposition and Rational Function Integration System, Univ. of Wisconsin Computer Sciences Dept. Tech. Report No. 80 (47 pages), Feb. 1970.

[8] Collins, G. E., and L. E. Heindel, The SAC-1 Polynomial Real Zero System, Univ. of Wisconsin Computer Sciences Dept. Tech. Report No. 93 (72 pages), Aug. 1970.

[9] Collins, G. E., The SAC-1 Polynomial GCD and Resultant System, Univ. of Wisconsin Computer Sciences Dept. Tech. Report No. 145 (94 pages), Feb. 1972.

[10] Collins, G. E., and M. T. McClellan, The SAC-1 Polynomial Linear Algebra System, Univ. of Wisconsin Computer Sciences Dept. Tech. Report No. 154 (107 pages), April 1972.

[11] Collins, G. E., and D. R. Musser, The SAC-1 Polynomial Factorization System, Univ. of Wisconsin Computer Sciences Dept. Tech. Report No. 157 (65 pages), March, 1972.

[12] Collins, G. E., The Calculation of Multivariate Polynomial Resultants, Jour. A.C.M., Vol. 18, No. 4 (Oct. 1971), pp. 515-532.

[13] Knuth, D. E., The Art of Computer Programming, Vol. 2 (Seminumerical Algorithms), Addison Wesley Publ. Co., 1969.

[14] Musser, D. R., Algorithms for Polynomial Factorization, Univ. of Wisconsin Computer Sciences Dept. Tech. Report No. 134 (174 pages), Sept. 1971.

# ALGORITHM INDEX


| | Page | | Page |
|---|---|---|---|
| ADD3 | 7 | INEG | 13 |
| COMPAT | 11 | IPOWER | 20 |
| IABSL | 13 | IPROD | 19 |
| IBTOD | 35 | IQ | 24 |
| IBTOH | 34 | IQR | 21 |
| ICOMP | 14 | IQRS | 21 |
| IDIF | 17 | IREAD | 33 |
| IDTOB | 32 | IREM | 25 |
| IDTOH | 31 | ISIGNL | 12 |
| IGCD | 25 | ISUM | 15 |
| IHTOB | 31 | IWRITE | 36 |
| IHTOD | 35 | MPY | 9 |
| ILCM | 30 | QR | 10 |
| IMADD | 18 | | |