Editor's Note:

A revised summary of the material in the A.C.M. library is being printed to encourage further contributions of missing material.

---

As the translation of the Russian paper appearing in this section does not give any easy clues about its subject material or intent, a brief description is attempted here. It is nice to see that English-speaking peoples are not the only experts at obfuscation.

This paper deals with the production of 3-address machine language instructions (for the BESM computer) from algebraic statements of the type found in Fortran, Unicode and other languages. Assembly program characteristics are included. An algorithm is given for creating rough machine language instructions in pseudo-form and then operating upon these to alter them to the most efficient form. For at least the domain of a single formula, a check is made for duplicate strings of any length.

The most pertinent point is that the algorithm itself operates more efficiently than their previous methods. Thus the processor takes an amount of time to produce an efficient object program which is linearly proportional to the number of instructions in the program, NOT to the square of the number as previously. Obviously, when the interaction of instructions over the entire program is considered (as in the Fortran processor) or when the formulae are exceptionally long, this method becomes more valuable as the programs grow larger.

# ON PROGRAMMING OF ARITHMETIC OPERATIONS

A. P. Ershov

Doklady, AN USSR, vol. 118, No. 3, 1958, pp. 427–430

Translated by Morris D. Friedman, Lincoln Laboratory*

The concepts used without explanation are taken from [1].

1°. Programming algorithms of arithmetic operations (AO) consist of three parts.

The first part A1 successively generates the commands of the AO program.

The second part A2 generates a conventional number (CN)* for each command constructed, which denotes the result of the programmed operation, and replaces it in the formula of the programmed expression. Identification of the entries of similar expressions in the AO formula is made during the A2 operation so that similar expressions will not be programmed repeatedly (economy of command).

The third part A3 replaces the CN in the constructed program, which denotes the intermediate results, by a code of operating registers (OR). New principles for constructing the algorithms of A2 and A3 are proposed herein.

2°. Assumptions and Definitions. Programming of arithmetic operations is carried out on a three address computer. The left side of the AO formula is the superposition of binary and unary operations, each of which is realized by a single command. Each command has one bit $\sigma$, which neither enters into the operation code nor into the address part of the command. The A1 algorithm generates the AO commands in the form

| | | | | |
|---|---|---|---|---|
| a | b | c | $\sigma$ | $\theta$ |

where $\theta$ is the operation code, a and b are CN which denote components and c is a CN which denotes the result (if $\theta$ is a unary operation, then $b = 0$; $c \neq 0$ only for a resultant command, i.e., the concluding command of the calculation of the AO formula; the content of the digit $\sigma$ is zero at first). A *Block of Preparatory Operations* (BPO) is a group of n registers with the addresses $L+1, \ldots, L+n$ in which are located the AO commands which are generated by the algorithm A1. A *Block of Resultant*

* Translator's note: Apparently, the author means a number given meaning by certain agreed upon conditions. The translator is grateful to Sheldon Best of MIT for having read the translation and making corrections.

[1] A. P. Ershov: Programming programs for the BESM, Moscow. 1958.

*Commands* (BRC) is a group of registers in which are located all the resultant AO commands in succession. A conventional number of the first kind means a quantity or constant in the formula. A conventional number of the second kind is an intermediate result in the calculation of the formula. It is generated by the algorithm A2 and equals the address of the non-resultant command in the BPO for each such command. A BPO scale is a group of consecutively located registers of the memory with continuous enumeration of the digits, with which the s-th digit of the BPO scale corresponds to the $L+s$ register of the BPO. The scale of the CN of the first kind has a similar apparatus. A *Block of Operating Registers* (BOR) is a group of registers with the addresses $r+1, \ldots, r+m$, where $r+1, \ldots, r+m$ are codes of the operating registers. A *Block of Preparatory Programs* (BPP) is a group of registers in which a preparatory AO program will be placed. The symbol (T) denotes the content of the register T.

3°. In existing command economy methods, the total time of operation of the A2 algorithm is proportional to the square of the number of commands in the AO program.

Shown on figure 1 is a diagram of the A2 algorithm which permits the realization of command economy within a time proportional to the number of commands in the AO program. The basis of the algorithm proposed is the assumption that there exists a certain integer function $F = F(\Theta,a,b)$ $(L+1 \leq F \leq L+n)$ defined for any AO command

| | | | | |
|---|---|---|---|---|
| a | b | c | $\sigma$ | $\Theta$ |

Operation of algorithm A2 is started after construction of the next AO command K by the A1 algorithm (for simplicity, an A2 algorithm is described which does not produce economy of the resultant commands).

Operation 1 investigates whether the command K is the resultant (if not, do operation 2).

Operation 2 calculates $F(\Theta,a,b)$ for the command K and directs the result into the register S. It is evident that $L+1 \leq S \leq L+n$. Let $S = L+p$.

Operation 3 verifies whether $(L+p)$ equals zero (if not, do operation 4).

Operation 4 verifies whether the operation codes, the first two addresses and the digit $\sigma$ agree for the K and $(L+p)$ commands (if yes, output I).

Operation 5 increases p by one if $p < n$ and puts $L+1$ into the register S if $p = n$.

Operations 6–8 perform if the command K is not economized.

Operation 6 investigates the CN from the address part of the command K. If a CN of the first kind is among them, ones are put in the appropriate digits of the scale of the CN of the first kind and in the p-th digit of the BPO scale (zeroes are in all the digits of both scales before the start of the AO programming).

Operation 7 calculates certain quantities needed for the operation of the A3 algorithm for the command K (see 5°).

Operation 8 directs the command K into the $L+p$ register.

Operation 9 performs if K is a resultant command $(c \neq 0)$. If a one is in the digit of the scale of the CN of the first kind corresponding to the CN c, then BPO commands containing a CN of the first kind are scanned by using the BPO scale. Commands containing the CN c are marked by ones in the $\sigma$ digit. Consequently, none of the commands containing the CN c in the address part and having been constructed after K will coincide with any of the commands constructed before K during the execution of operation 4. Then K is transmitted to the next free BRC register.

The A2 diagram has two outputs I and II. A CN of the second kind which denotes the result of a constructed nonresultant command is obtained in the register S at the output I. The output II corresponds to a resultant command.

4°. The duration of the execution of A2 is determined by the number of repetitions of operations 3–5. This number depends on the distribution of the values of $F(\Theta,a,b)$ in the strip $[L+1, L+n]$. [Let us note that the usual command economy algorithms correspond to $F(\Theta,a,b) \equiv L+1$.] Evidently, the most favorable case is the uniform distribution of the $F(\Theta,a,b)$ values in $[L+1, L+n]$ for a random composition of the AO formula. In this case, the mathematical expectation $\Phi$ of the number of repetitions of the operations 3–5 can be calculated as a function of the number of commands k making up the program and of the quantity of registers n in the BPO [$\Phi = \Phi_n(k)$]. A derivation of analytic estimates appears to be difficult and the values of $\Phi_n(k)$ were computed by the Monte Carlo method. Presented on figure 2 are the curves of $\log_{10}\Phi_n(k)$ obtained for $n = 150\ (50)\ 450$. Curves of $\log_{10} k$ and $\log_{10}\frac{k^2}{2}$ are given for comparison. It follows from an analysis of the results obtained that, in practice, not more than one execution of the 3–5 operations will occur in each AO command if the BPO exceeds the number of commands in the AO by not less than one and one-half times for all n.

The simplicity of the calculation and the sufficiently uniform distribution of the values is the unique criterion limiting the choice of $F(\Theta,a,b)$. It is expedient to use the methods of producing uniformly distributed pseudo-random numbers for the actual construction of $F(\Theta,a,b)$. An investigation of the statistical structure of the formulas of the AO to be programmed is of great value in the successful choice of $F(\Theta,a,b)$.

5°. There are definite relations in the order of the performance of the operations entering into the AO formula. These relations are given by a rule that the components are calculated, at the beginning, for the components of the formula and then the operation itself is calculated. Consequently, the formula can be considered as a semi-ordered set of the operations therein. Ordering of the operations, caused by the successive location of the command in the program, occurs in the construction of the program to calculate the formula, consequently, the problem of programming the formula can be formulated as a problem in ordering the operations of the formula by retaining a given semi-order. It is evident that the quantity of OR required to calculate the formula depends on the method of ordering its operations. For example, in order to calculate the formula

$$ab + (cd - ef(gh + ij(kl - mn))) \rightarrow y$$

seven OR are needed to perform the action from left to right while only two OR are needed if the calculation is started with the innermost parenthesis. In this connection, the problem arises of finding such an admissible ordering of the operations of the formula for which the minimum quantity of OR would be required for its calculation.

The problem posed is solved partially by using the algorithm A3 of the ordering of the operations of the formula whose diagram is presented on figure 3. Calculation of the operation 7 of the A2 algorithm for each nonresultant command K of two integer functions whose values are put in the third address of the command before it is transmitted to the BPO is preparatory to the operation of A3. The first function, a function of the order P(K), is given by an inductive definition:

A) If the command K does not contain a CN of the second kind, then $P(K) = 1$.

$B_1$) If a CN of the second kind, denoting the result of the command $K_1$ is an address of the command K, then $P(K) = P(K_1)$.

$B_2$) If CN of the second kind, denoting the result of the commands $K_1$ and $K_2$, are in the first and second addresses of the command K, then

$$P(K) = \begin{cases} \max\{P(K_1)P(K_2)\} & \text{if } P(K_1) \neq P(K_2) \\ P(K_1) + 1 & \text{if } P(K_1) = P(K_2) \end{cases}$$

The second function, the entry counter, is calculated as follows. When a command K is transmitted to the BPO, its entry counter equals 0. If a command K′, containing a CN which denotes the result of the command K, is then transmitted to the BPO, then 1 will be added to the entry counter of the command K.

The algorithm A3 starts to perform after the termination of the operation of A1 and A2.

The operation 1 transmits the next AO resultant command into the register R, starting with the last register of the BRC. Let the command K be in R.

Operation 2 replaces the CN of the second kind in K by a code of OR. If a CN of the second kind L+s enters into K, the content of the L+s register is investigated. The command K′ from L+s is transmitted to the first free register of the BOR. The command K′ in L+s is replaced by the address r+i, which indicates where K′ was transmitted to. If K′ has already been replaced by the r+i address during the processing of one of the preceding AO commands, 1 is subtracted from the entry counter of the command K′ in r+i. The CN L+s in K is replaced by the r+i OR code. If two CN of the second kind $L+s_1$ and $L+s_2$ enter into K, where the commands $K_1$ and $K_2$ are in the $L+s_1$ and $L+s_2$ registers, that one of the commands $K_1$, $K_2$ is transmitted first into the BOR for which the value of the order function is larger.

Operation 3 transmits K to the next register of the BPP, starting with the last register.

Operation 4, scanning from the end of the BOR, finds the first command with entry counter equal to 1. If such a command is not found in the BOR or if no commands are in the BOR, control is transferred to operation 6.

Operation 5 transmits the command found from the r+j register into the R register, puts r+j into the third address of this command and then clears the r+j register.

Operation 6 transfers control to operation 1 if not all the commands are transmitted from the BRC.

The algorithm described solves completely the problem of the most favorable ordering for an AO for which the entry count of each command is 1. This follows from the following two statements which are valid under the above-mentioned limitations:

1. In the interests of the minimum expenditure of operating registers for any binary operation, it is first necessary to calculate those of its components for which the minimum number of OR required for its calculation is larger.

2. The order function for each command equals the minimum quantity of OR required to calculate the expression in which the last operation is realized by the given command.

Moscow University June 27, 1957

## AUTOMATIC PROGRAMMING SYSTEMS

Dear Mr. Bemer,

The attached note enumerates some corrections to my short paper in the May issue of the ACM Communications. The error necessitating these corrections was pointed out to me by John M. Brill in a personal communication, and a copy of my reply to him is also enclosed for your information.

I would appreciate it if you would arrange for these corrections to appear in a forthcoming issue of the Communications.

Sincerely yours,
William H. Kautz
Stanford Research Institute

Correction to "Binary and Truth-Functional Operations on a Decimal Computer with an Extract Command" by William H. Kautz
(ACM Communications *1* (*5*); 12–13 May 1958)

An error in the description given of the extract command in the above paper has been pointed out. This description should read:

$$aEb=\left\{\begin{matrix} a \text{ if } a \text{ is even} \\ a+b-1 \text{ if } a \text{ is odd}\end{matrix}\right\}\text{where } a \text{ and } b \text{ are single decimal digits}$$

In consequence, the inequivalence operation A4 (and the other operations, A6, A7, and A10, which repeat the A4 equation) should be modified as follows:

A4. $X\oplus Y \doteq xE(1^*-y)+yE(1^*-x)$
or $X\oplus Y \doteq (x+y)-2(xEy)$
or $X\oplus Y \doteq (x+y)-(x+y)E0^*$

Also, for no more than 9 variables, $X\oplus Y\oplus Z\oplus \dots \doteq (x+y+z+\dots)-(x+y+z+\dots)E0^*$