At present, the multiplication time for the computer is 3 milliseconds, and the addition time is 80 microseconds. These present times are partly due to safety factor delays which have been built into the arithmetic control circuitry by means of mono-stable flip-flops. With the present arithmetic control, a multiplication time of 1 millisecond and an addition time of 50 microseconds seems feasible.

MACHINE AIDS TO CODING

By

Earl J. Isaac
Naval Research Laboratory, Washington

Coding for digital computers is a process of translating from one language to another. A problem to be computed might be expressed in algebraic notation of several types or in one of several forms of logical symbolism. This notation or symbolism must be translated into the machine language. In many problems, particularly those involving iterative processing of many variables, the translation is reasonably direct and can be written out and checked with little difficulty. Some problems, however, require a great many instructions dealing with few variables and arranged in complex alternative paths. In these cases the translation process becomes formidable and the most extensive checking often fails to eliminate clerical errors.

The translation can be divided roughly into two steps, translation of grammar and translation of words. Usually the translation of grammar, passing from one structure convention to another, is the more difficult. The translation of words, substituting one set of arbitrary symbols for another, is a relatively simple process. As an example, consider the algebraic expression $y=ax^2+bx+c$. Let the symbol for addition be A, for multiplication, M. Assume the machine design requires a three address code of the form operator-order-operand-result location.

The translation of the grammar would then be:

```
x M a  y
y A b  y
y M x  y
y A c  y
```

i.e., a sequence of four separate instructions that expresses the fact that $y=ax^2+bx+c$ in the grammar of the machine. The translation is complex and in general not unique.

The next step is the substitution of machine symbols for the algebraic symbols. Let us make the correspondence

a ~	001
b ~	002
c ~	003
x ~	004
y ~	005

Assume that the machine symbol for addition is 1 and for multiplication 2. The translation is then

```
004  2  001  005
005  1  002  005
005  2  004  005
005  1  003  005
```

Since this step is merely a replacement of equivalent symbols it is easily done. Nevertheless, in long and complicated codes, clerical errors are almost inevitable, and long and arduous checking is necessary to avoid them. To simplify this process, an obvious recourse is to use auxiliary machines. If the coder wrote in machine grammar but used arbitrary symbols for the words, the auxiliary equipment could perform the substitution.

The use of an arbitrary symbolism in coding instead of reference to specific machine locations has two principle advantages.

The first advantage is that the arbitrary symbolism permits the use of a redundant dictionary; the second is that major rearrangements of specific locations without altering the relative coding are possible. A redundant dictionary, i.e., the use of more symbols than necessary, permits the coder to use mnemonic symbols for variables, so that the symbols serve as cues to the physical quantity they represent. Redundancy in machine symbolism is costly from an engineering standpoint, but an arbitrary language with no redundancy is difficult to understand, particularly when the symbols have no relationship to previous experience. For example, we can specify a dictionary code as follows:

dogs $\sim$	1
fleas $\sim$	2
midges $\sim$	3
germs $\sim$	4
have $\sim$	5

then write

1 5 2 3 5 4 2 5 3

which translates

"Dogs have fleas, midges have germs, fleas have midges."

However, this symbolism requires almost constant reference to the dictionary to translate. If, instead, we specify the dictionary:

dogs $\sim$	DO
fleas $\sim$	FL
midges $\sim$	MD
germs $\sim$	GR
have $\sim$	HV

then write

DO HV FL MD HV GR FL HV MD

the phrase can easily be translated without recourse to the dictionary. By using symbols that suggest the quantity to which they correspond the coder can readily interpret his code both while constructing it and while checking it.

The second main advantage is freedom of rearrangement. A block arrangement has many advantages for checking purposes. For example, variables of the same type should be grouped together and separated from instructions. If changes or corrections are made in the code, a reassignment of locations is often necessary in order to maintain the block arrangement. This disrupts a machine coding but has no effect on the relative coding.

The translation can be done with the computer itself or by elementary auxiliary equipment. In a practical application an IBM sorter and reproducer are used to translate codes for the National Bureau of Standards SEAC. The instructions written with a redundant dictionary are punched into one deck and the dictionary words themselves, together with the corresponding machine addresses, are punched into another. The two decks are then sorted together. The substitution is made by an interspersed gang punching run on an IBM reproducer.

As indicated above the translation of the structure appears to be considerably more difficult. Some may find that writing a code directly in terms of the allowed operations of the particular machine is as convenient as any other form. The use of subroutines permits the coder to think in terms of functions that are complex combinations of the elementary arithmetic and logical operations of the machine. This is in effect a different structure than that permitted by the basic machine. The translation problem in this case is the replacement of the coder's expression for the complex operation by the equivalent structure in the machine terminology. A system is now under development at the Naval Research Laboratory to perform this translation for a single-address computer, the NAREC. For purely arithmetic operations this system will permit the coder to specify in each instruction, written using a redundant dictionary, up to six arguments for a subroutine, designation of the subroutine, where to store the results of the subroutine and an arbitrary transfer of control if necessary. The translation process uses only an IBM sorter and reproducer and takes place in two steps: (1) translation of the structure, and (2) translation of the words. In the first step the deck containing the instructions is sorted by subroutines and the number of occurrences of each subroutine is counted. Standard sets of cards are prepared for each subroutine. These contain the complete coded pattern of the subroutines with constants in redundant form and with blanks left for the positions occupied by the arguments. In addition, the subroutine decks contain punches that control the read feed of the reproducer and the timing of selectors that punch the arguments into the appropriate positions in the subroutine deck. The instruction deck is placed in the read feed, the subroutine decks in the punch feed. As each instruction card passes under the reading brushes the information is punched into the first card of the subroutine deck. This information is gang-punched back into successive cards of the subroutine deck while the read feed is held. Selectors then transfer the symbols from the gang-punched columns into appropriate positions on the subroutine cards. When a subroutine is completed the next instruction card is read. This process results in a deck containing a set of instructions in terms of the elementary operations of the computer but with symbols still from a redundant dictionary. The second step of word translation is performed as described previously.

The computer itself can obviously perform the processes described above. Only a careful analysis can determine whether or not any advantage would be gained by using the computer for this purpose instead of the elementary machinery now in use.

COMPUTER AIDS TO CODE CHECKING

By

Ira C. Diehm
National Bureau of Standards, Washington

When a complex routine is tried on a computer for the first time, it is seldom found to be free from error. The trend toward automatic performance of the clerical parts of the coding process should reduce the number of coding errors. This mechanization of coding is the subject of several papers at this conference. Nevertheless, a significant amount of valuable computer time will continue to be devoted to the search for coding errors.

Careful proofreading and clerical checking are obvious but important