# The Application of FORTRAN to Automatic Translation

Paul O. Samuelsdorff

## 1. Introduction

A multitude of problem-oriented programming languages are being created in order to facilitate programming in various fields.These languages have two advantages for non-programmers who need a computer for solving their problems: 1. they are not forced to spend much time on learning a complicated machine language; 2. they do not necessarily have to alter their programs when they are obliged to use a different machine. These advantages seem to justify the labour of writing a multitude of compilers that translate each problem-oriented language into the machine language of each machine.

The existence of various problem-oriented languages,however, makes it difficult for the users of computers in various fields to exchange their experience. In addition there is the danger that a certain machine possesses only a limited number of compilers, and that therefore a program may have to be rewritten when for some reason or other a different machine has to be used. The question therefore arises whether it is not possible to have the advantages of problem-oriented languages without multiplying their number.

This paper wants to show that one of the most widespread problem-oriented languages, originally created for mathematicians, may efficiently be used for non-numerical data-processing, if the necessary subroutines are available.

Another object of this paper is to give a concrete picture of how linguistic problems may be handled by the computer to those readers who have only a theoretical knowledge of data-processing. For the benefit of these readers a few definitions will be given, so that the examples will be understood by everybody.

bit = the smalles unit of information that may be stored in the computer, having one of two values symbolized by "1" or "0".

character = a letter of the alphabet, a decimal digit, or one of 11 special signs including the blank. A character is represented in the computer used for our examples (IBM 7094) by a combination of 6 bits.

machine-word = a string of six characters (represented in the computer by a combination of 36 bits).

location = the place in the computer where a machine-word is stored.

array = a combination of locations treated as a unit.

subroutine = a program that may be called up by another program for performing a specific job.

function = a subroutine that in addition to handling a certain job leaves a definite result in a certain location.

IF-statement = an instruction of the program that enables a branching to one of three different statements in the program, depending on whether the value in the parentheses of this statement is negative, zero or positive. Its format is "IF (x) 1,m,n", where 1,m and n are the numbers of the statements to which the program may branch.

The application of a mathematically oriented programming language to linguistic problems will be exemplified by a Hebrew-English translation project written in FORTRAN with the aid of subroutines programmed at the Deutsches Rechenzentrum in Darmstadt. These subroutines handle data smaller than the machine-word (single bits, single characters and their concatenation), the non-numerical machine-word and data larger than the machine-word.

We shall now explain the subroutines used in our project and give examples of their application. In order to avoid a repeated explanation of the variables used in the formats of the subroutines, we introduce the following conventions.

M = a positive integer not greater than 36, referring to one of the 36 bits of a machine-word.

WORD = the location of a machine-word to be handled by the subroutine. If two machine-words are to be handled, we distinguish between WORD1 and WORD2.

BLOCK = the location of the first machine-word of an array containing the characters or the string to be handled by the subroutine. If two arrays are involved, we distinguish between BLOCK1 and BLOCK2.

NA = a positive integer indicating the position of the first character of a string in the array BLOCK to be handled by the subroutine. If two strings are to be handled, we distinguish between NA1 and NA2.

N = an integer, usually positive, indicating the number of characters to be treated by the subroutine. N will be negative if we count the characters from NA to the left.

TABLE = an array containing characters to be used by a subroutine. This table is prepared before the subroutine is called.

The subroutines will be described in the order of the units they handle, starting from the smallest units.

## 2. Bits

### 2.1. PAKR

This routine facilitates the coding of grammatical information. Each bit of a 36-bit machine-word represents a certain kind of grammatical information. Its value is "1" when this information applies to the Hebrew word it is coordinated to and "0" when this information does not apply. The 36 bits of one machine-word, stored in the array CODE, represent the following information.

1 = definite noun
2 = indefinite noun
3 = construct state
4 = pronoun
5 = prepositional phrase
6 = preposition
7 = adverb
8 = numeral
9 = indefinite adjective
10 = definite adjective
11 = indefinite participle
12 = definite participle
13 = ZH
14 = AT
15 = AUTU, AUTH, AUTM, AUTN
16 = KI, AM, KAWR
17 = GM, RQ
18 = LA
19 = IW
20 = verb
21 = infinitive
22 = comma
23 = KL
24 = present
25 = past
26 = future
27 = copula
28 = WL
29 = W
30 = U
31 = masculine singular
32 = feminine singular
33 = masculine plural
34 = feminine plural
35 = no indefinite article
36 = no definite article

(The words in capital letters are Hebrew words that need special treatment.)

It would be very complicated to work out the **figure** to be stored in a certain location for each possible combination of grammatical information. Let us take as an example a masculine singular verb in the past tense. The machine-word coordinated to this verb will have a "1" in bit-positions 20, 25, and 31, and a "0" in all other bit-positions: 000000000000000000010000100000100000. This is the binary equivalent of the decimal number 69 664. Instead of working out this figure, we code a "1" into the 20th, 25th and 31st column of a punched card and leave the other columns blank. The first 36 columns of the card are then read into an array of 36 locations. The 36 machine-words of this array are subsequently packed with the aid of the subroutine PAKR into one location (containing 36 bits), the ones and blanks of the 36 columns on the card resulting in ones and zeros in the respective bits of one machine-word.

## 2.2. KBIT

This subroutine is a function and identifies single bits. If we want to find out whether a word possesses a certain grammatical feature, we have to ask whether the value of the respective bit of the machine-word that contains the desired grammatical information is "1" or "0".

Format: J = KBIT(WORD,M)

The result stored in the location J will be "1" or "0", depending on the value of the respective bit.

The subroutine KBIT enables the control to cause a branching to the specific part of the program needed for the treatment of a member of a certain word-class. A computed GOTO after the dictionary lookup handles this process very efficiently, as seen in the following example.

```
79   DO      90      L = 1,23
     IF (KBIT(CODE(N),L))  999,90,80
80   GO TO (100,200,300,400,500,600,1000,75,920,910,1200,2000,
      13000,4000,5000,1000,75,1000,1000,2200,1000,1000,700),L
90   CONTINUE
```

The value of the variable L must be computed before the GOTO statement is reached. It must be a positive integer not greater than the number of statement-numbers in the parentheses of the GOTO statement, since it determines which statement of the program is to be executed next. L is computed in statement 79. It assumes the values from 1 to 23 as long as the program reaches statement 90. The IF-statement between statement 79 and statement 80 causes the program to branch to statement 90 as long as the Lth bit of the machine-word containing the grammatical information of the Nth word of the sentence equals zero. As soon as the Lth bit equals one, the program will branch to statement 80. (999 is a dummy statement number, since the result of KBIT will never be negative.) Since we are examining the machine-word found in the dictionary together with the Hebrew word we were looking up, statement 80 will cause a branching to that part of the program that handles members of the word-class this Hebrew word belongs to.

An IF-statement using the subroutine KBIT may also enable the program to branch conditionally during the syntactic analysis or the translation process and thus make the solution of ambiguities possible. Specific parts of the program for the solution of ambiguities are reached by statements examining the bits of machine-words located in the array SCODE. For instance for ambiguous verbs the statement will be:"2200 IF (KBIT(SCODE(N),1)) 999,2210,2500". The part of the program solving the ambiguity of the verb starts at statement 2500. The other "1"-bits of SCODE, coded with a verb or a noun, indicate how a preposition is to be translated.

## 2.3. LBIT

This subroutine permits the conversion of "0"-bits into "1"-bits and vice versa. Its format is similar to that of KBIT, namely

J = LBIT(WORD,M)

The result stored at J will be "1" when "0" has been replaced by "1" and "0" when "1" has been replaced by "0".

LBIT is used for storing or deleting grammatical information during the analysis process.

An example is the indication of definite nouns or prepositional phrases during the dictionary lookup. Since in Hebrew the definite article and some of the prepositions consist of one letter that is attached to the following noun, this letter has to be stripped before the word can be found in the dictionary. We use the subroutine LBIT during the dictionary lookup for storing "1" in the first bit-position of CODE(N)and deleting the "1" in the second bit-position if the Nth word of the sentence is a noun preceded by the definite article, or "1" is stored in the fifth bit-position, if the Nth word of the sentence is a noun preceded by a preposition.

## 3. Single characters

### 3.1. IFFP (find first position)

This subroutine makes the identification of single characters possible.

Format: J = IFFP(BLOCK,NA,N,TABLE,IND)

This function finds the first position of a character in the array BLOCK which is equal to a character of TABLE if IND = 1 or unequal to any of the characters of TABLE if IND = 0. The searbh starts from the NAth character of the array. It goes to the right if N is positive and to the left if N is negative.

One of the characters we frequently want to identify is the blank, in order to find word boundaries. This subroutine is used, for instance, when we want to insert an adjective between a preposition and the following noun. We have to look for the blank after the preposition in order to insert the adjective in the right place. An example for a negative N will be given in section 6.3.

### 3.2. SEPO (seek positions)

This subroutine traces the positions of occurrence of a character (or a member of a set of characters) stored in a table.

Format: CALL SEPO (BLOCK,NA,N,TABLE,NSP,NFIND)

The subroutine SEPO examines N characters in the array BLOCK, starting from the NAth character. The number of characters found to be equal to the characters stored in TABLE is stored in the location NFIND. The actual positions of these NFIND characters are stored in the array NSP.

If the only character stored in the table is the blank, dictionary lookup will be very convenient, since the occurrence of word boundaries may be stored for a whole text. NSP(I) will contain the position of the blank after the Ith word of the text. The next word to be processed will therefore easily be found. Its first character will be in the first position after the one indicated by the contents of NSP(I) and its length will be NSP(I+1) - NSP(I) - 1.

### 3.3. ICPC (compare character)

This subroutine compares a character with a set of characters stored in a table.

Format: J = ICPC(BLOCK,NA,TABLE,NTAB)

NTAB is a positive integer stating with how many characters of TABLE we want to compare the NAth character of the array.

The result of this function, stored in J, is zero if the NAth character is unequal to any of the characters of the table, otherwise it is a positive integer not greater than NTAB, indicating that the NAth character of the array BLOCK is equal to the Jth character of TABLE.

In the following example we see how this subroutine is used for choosing between the English indefinite articles "a" (WORD(1)) and "an" (WORD(5)).

```
 1476 K = ICPC (EWD(1,N),1,VOC,5)
 1477 IF (K-1)   1905,1910,1915
 1905 ENG(1,I) = WORD(1)
      I = I+1
      GO TO 1480
 1910 K = ICPC (EWD(1,N),3,VOC,5)
      IF (K)   1905,1915,1905
      ENG(1,I) = WORD(5)
      I = I+1
      GO TO 1480
```

EWD is the array where the English words were brought after the dictionary lookup and ENG where they are brought during the analysis. The first five locations of the table VOC contain the vowels U, O, I, E, A. If the first letter of the noun in question, stored in EWD(1,N), starts with a consonant, K in statement 1476 will be zero, and K-1 in statement 1477 will be negative. If the first letter is U, K-1 =0, and if the first letter is any other vowel, K-1 will be positive. That means, that for any first letter of the noun, except U, statement 1477 decides whether "a" or "an" will be chosen. In the case of U the choice depends on the third letter of the noun, found in statement 1910. "A" will be chosen if the third letter is a vowel, and "an" will be chosen if it is a consonant. This routine will, of course, still have to be refined for the case when the prefix "un" is followed by a vowel.

## 4. Machine-word parts

### 4.1. CSWO (compose word)

This subroutine, composing a machine-word from two parts of two other machine-words, facilitates the insertion of short words like prepositions or the definite article.

Format: S = CSWO (WORD1,WORD2,K)

As a result of this function the first K characters of the machine-word in location WORD1 is composed with the last 6 - K characters of the machine-word in location WORD2. The resulting machine-word is stored in location S. S may be identical with WORD1 or WORD2. We see an example of the application of this subroutine in the part of the program that translates a date into English.

```
 375 EWD(1,IL) = CSWO (WORD(30),EWD(1,IL),2)
     K = IL-1
     CALL SHA2(EWD(1,K),1,32,4,1)
     EWD(1,K) = CSWO (WORD(40),EWD(1,K),3)
     CALL SHA2 (EWD(1,K),1,33,3,1)
     GO TO 330
```

In statement 375 the preposition "of" (WORD(30)), chosen in this case as the translation of the Hebrew prepositional prefix B, is composed with the first part of the name of the month which follows it. The statements following statement 375 translate the Hebrew B before the day of the month by "on the" (WORD(32) and WORD(40)). The shifting subroutine SHA2 will be explained in section 6.2.

## 5. Machine-words

### 5.1. CPWO (compare words)

A special subroutine for comparing non-numerical machine-words is necessary for ignoring the sign-bit. When we make a mathematical comparison between two machine-words (subtracting one word from the other) the first bit is interpreted as a sign (0 = + and 1 = -). This will give a wrong result when we compare a word starting with a letter between A and I with a word starting with a letter between J and Z, since the former have "0" in their first bit-position, while the latter have "1".

In non-numerical data-processing we therefore have to regard the machine-word as an entity of six characters, where the first bit of the first character is part of the binary number representing the whole machine-word.

Format: S = CPWO(WORD1,WORD2)

The machine-words at the two locations WORD1 and WORD2 are compared. S = 1 if the contents of WORD1 is greater than the contents of WORD2, S = 0 if the two machine-words are equal, and S = -1 if the contents of WORD1 is smaller than the contents of WORD2.

## 6. Units larger than the machine-word

### 6.1. CPST (compare strings)

This routine differs from CPWO in enabling us to define the number of characters (N) that we regard as the entity to be compared.

Format: CPST(BLOCK1,NA1,BLOCK2,NA2,N)

The result is similar to that of the comparison of two machine-words: S = 1 if the binary number representing the string in BLOCK1 (starting from NA1) is greater than the binary number representing the string in BLOCK2 (starting from NA2), S = 0 if the two strings are equal, and S = -1 if the binary number representing the string in BLOCK1 is smaller than the binary number representing the string in BLOCK2.

We shall see an example of this subroutine in the dictionary lookup, section 6.5., where the words from the text will be compared to the words in the dictionary.

### 6.2. SHA2 (shift in array)

This subroutine enables us to shift strings of variable length.

Format: CALL SHA2 (BLOCK,NA,N,NSHIFT,IND)

NSHIFT is the number of positions we want to shift a string of N characters, starting from the NAth character in the array BLOCK. NSHIFT may be a positive or a negative integer, depending on whether we want to shift to the right or to the left. IND = 1 if we want to fill the vacant positions by blanks and IND = 0 if we want to fill them by zeros. We saw two examples of this subroutine in section 4.1. Since we reserved six machine-words for one English word, N+NSHIFT = 36 (6x6 characters).

6.3. ICSS (compose strings)

With the aid of this subroutine we may bring a string of variable length (N) from one array (BLOCK1) to another (BLOCK2), starting from any position in the two arrays (NA1 and NA2).

Format: J = ICSS(BLOCK1,NA1,N,BLOCK2,NA2)

The result of this function will be J = NA2+N.

This routine, combined with IFFP (section 3.1.), is very useful in bringing the processed English sentence,word for word, into the output area.

```
 5060 NA2 = 1
      DO   5075   L = 1,I
      N = IFFP (ENG(1,L),36,-36,TAB,0)+1
      IF (N-1)   5075,5075,5070
 5070 NA2 = ICSS (ENG(1,L),1,N,OUT,NA2)
 5075 CONTINUE
```

I is the number of arrays, each consisting of six locations, that contain the English words after the analysis process. IFFP has to look for the first non-blank, since the English equivalent of a Hebrew word may be a word group. For the same reason we have to start the search from the 36th character, going left. We add 1 to the result of the function, so that the English word (or word group) is stored with an additional blank and the result (NA2) of the function ICSS may serve as a parameter of the same function in the next round. The IF-statement before 5070 is a safeguard against the English word being blank.

6.4. A subroutine changing the word order was added to the existing subroutines. Since in Hebrew the adjective always follows the noun it qualifies, we have to change the word order in English.

Format: CALL ADT(N)

N is a positive integer indicating that the adjective is the Nth word in the sentence. It changes position (together with its grammar codes) with the (N-1)th word. This subroutine was entirely written in FORTRAN, without using subroutines.

6.5. The following subroutine was written for the dictionary lookup of the translation project. The words in the dictionary are alphabetically ordered, with the exception of the blank, that stands between R and S, because of the value of its binary equivalent. For this reasen, the blank following the word in the text is added before the word is compared with the dictionary entries (HW).

The following parameters are used by the subroutine:

NA = The position of the first letter of the word in the text that is looked up in the dictionary.

NZ = The number of characters of the word.

IL = The ordinal number of the word in the sentence.

M = The number of entries in the dictionary.

## 5. Machine-words

### 5.1. CPWO (compare words)

A special subroutine for comparing non-numerical machine-words is necessary for ignoring the sign-bit. When we make a mathematical comparison between two machine-words (subtracting one word from the other) the first bit is interpreted as a sign (0 = + and 1 = -). This will give a wrong result when we compare a word starting with a letter between A and I with a word starting with a letter between J and Z, since the former have "0" in their first bit-position, while the latter have "1".

In non-numerical data-processing we therefore have to regard the machine-word as an entity of six characters, where the first bit of the first character is part of the binary number representing the whole machine-word.

Format: S = CPWO(WORD1,WORD2)

The machine-words at the two locations WORD1 and WORD2 are compared. S = 1 if the contents of WORD1 is greater than the contents of WORD2, S = 0 if the two machine-words are equal, and S = -1 if the contents of WORD1 is smaller than the contents of WORD2.

## 6. Units larger than the machine-word

### 6.1. CPST (compare strings)

This routine differs from CPWO in enabling us to define the number of characters (N) that we regard as the entity to be compared.

Format: CPST(BLOCK1,NA1,BLOCK2,NA2,N)

The result is similar to that of the comparison of two machine-words: S = 1 if the binary number representing the string in BLOCK1 (starting from NA1) is greater than the binary number representing the string in BLOCK2 (starting from NA2), S = 0 if the two strings are equal, and S = -1 if the binary number representing the string in BLOCK1 is smaller than the binary number representing the string in BLOCK2.

We shall see an example of this subroutine in the dictionary lookup, section 6.5., where the words from the text will be compared to the words in the dictionary.

### 6.2. SHA2 (shift in array)

This subroutine enables us to shift strings of variable length.

Format: CALL SHA2 (BLOCK,NA,N,NSHIFT,IND)

NSHIFT is the number of positions we want to shift a string of N characters, starting from the NAth character in the array BLOCK. NSHIFT may be a positive or a negative integer, depending on whether we want to shift to the right or to the left. IND = 1 if we want to fill the vacant positions by blanks and IND = 0 if we want to fill them by zeros. We saw two examples of this subroutine in section 4.1. Since we reserved six machine-words for one English word, N+NSHIFT = 36 (6x6 characters).

6.3. ICSS (compose strings)

With the aid of this subroutine we may bring a string of variable length (N) from one array (BLOCK1) to another (BLOCK2), starting from any position in the two arrays (NA1 and NA2).

Format: J = ICSS(BLOCK1,NA1,N,BLOCK2,NA2)

The result of this function will be J = NA2+N.

This routine, combined with IFFP (section 3.1.), is very useful in bringing the processed English sentence,word for word, into the output area.

```
 5060 NA2 = 1
      DO   5075   L = 1,I
      N = IFFP (ENG(1,L),36,-36,TAB,0)+1
      IF (N-1)    5075,5075,5070
 5070 NA2 = ICSS (ENG(1,L),1,N,OUT,NA2)
 5075 CONTINUE
```

I is the number of arrays, each consisting of six locations, that contain the English words after the analysis process. IFFP has to look for the first non-blank, since the English equivalent of a Hebrew word may be a word group. For the same reason we have to start the search from the 36th character, going left. We add 1 to the result of the function, so that the English word (or word group) is stored with an additional blank and the result (NA2) of the function ICSS may serve as a parameter of the same function in the next round. The IF-statement before 5070 is a safeguard against the English word being blank.

6.4. A subroutine changing the word order was added to the existing subroutines. Since in Hebrew the adjective always follows the noun it qualifies, we have to change the word order in English.

Format: CALL ADT(N)

N is a positive integer indicating that the adjective is the Nth word in the sentence. It changes position (together with its grammar codes) with the (N-1)th word. This subroutine was entirely written in FORTRAN, without using subroutines.

6.5. The following subroutine was written for the dictionary lookup of the translation project. The words in the dictionary are alphabetically ordered, with the exception of the blank, that stands between R and S, because of the value of its binary equivalent. For this reasen, the blank following the word in the text is added before the word is compared with the dictionary entries (HW).

The following parameters are used by the subroutine:

NA = The position of the first letter of the word in the text that is looked up in the dictionary.

NZ = The number of characters of the word.

IL = The ordinal number of the word in the sentence.

M = The number of entries in the dictionary.

The following two constants are set up in the subroutine:

N = the first power of 2 greater than M (in our case 1024)

J = ${}_2$Log N (in our case 10)

```
      SUBROUTINE LOOKUP (NA,NZ,IL,M)
      DIMENSION HW(4,1024),HC(1024),HSC(1024),EW(6,1024),EC(1024),
     1TEXT(200),CODE(50),SCODE(50),EWD(6,50),ECODE(50),TESTWD(2)
      COMMON  TEXT,HW,CODE,SCODE,EWD,ECODE,HC,HSC,EW,EC,TESTWD
      TESTWD(1) = 6H000000
      J = 10
      N = 1024
      NI = N/2
 535  N = N-NI
 540  J = J-1
      IF (J)   590,544,544
 544  NI = NI/2
 545  IF (CPST (TEXT,NA,HW(1,N),NZ+1))   535,575,555
 555  N = N+NI
 560  IF (M-N)   565,540,540
 565  N = N-NI
      NI = NI/2
      N = N+NI
      J = J-1
      IF (J)   590,570,570
 570  IF (M-N) 565,544,544
 575  CODE(IL) = HC(N)
      SCODE(IL) = HSC(N)
      DO   580   K = 1,6
 580  EWD(K,IL) = EW(K,N)
      ECODE(IL) = EC(N)
      J = LBIT(TESTWD(1),1)
 590  RETURN
      END
```

The lookup is done by the logarithmic method. We jump into the middle of the array reserved for the dictionary and exclude the lower half of the dictionary from further search if the textword is greater than the word of the dictionary that it was compared with, and exclude the upper half if the word was smaller than the word in the dictionary. This process of halving the searched area is repeated till either the word is found or J, which is decreased by one before each search, is equal to zero. That means, we never have to compare more than ten times to find a word in the dictionary or to know that it cannot be found. Statements 560 and 570 prevent us from searching in the area between M and N. When the word is found, the grammatical information from the dictionary and the English translation of the Hebrew word is delivered to the main program and the first bit-position of TESTWD(1) is set "1".

If the word is not found in the dictionary, the program proceeds as follows. If the first letter is one of the possible Hebrew prefixes (found by ICPC), the word is looked up again, with the first letter stripped and NZ diminished by one. The result of the function ICPC is used by a computed GOTO for a branching to that part of the program that will prepare the insertion of the definite article or the solution of the ambiguity of the preposition.

If the first letter is none of the possible prefixes and the word is not found in the dictionary, the program will use the Hebrew original in the translation and print out a message that the word was not found.

## 7. Conclusion

Apart from the subroutines described above, there are additional subroutines that facilitate information retrieval and other linguistic data-processing. Statistical linguistics may obviously be efficiently treated by a mathematically oriented programming language.

This paper is not supposed to be a plea for FORTRAN, but for a universally applicable problem-oriented language, which will make the progress achieved in programming in one field accessible to the users of computers in all fields. A combination of subroutines written in this language may do the job that is done today by separate problem-oriented languages. A programming language like PL/1, or a language still to be created, may do the job more efficiently than FORTRAN. The idea is that the programs written will have a maximal possibility of application and that programmming examples given in the literature describing computer problems will be understood by all the readers interested.

## 8. Reference

Dipl.-Math. Gisela Schlotter, Verarbeitung nichtnumerischer Daten
Teil I. Unterprogramme (PI-17)

Deutsches Rechenzentrum, Darmstadt