COBOL PROGRAMMING (4)

R. Ramaswamy and T. V. Krishnamurthy

ADD Statement

The Add statement enables us to find the sum of two or more numeric fields and store the result. The Add format can be given in the following two forms:

```
Add { literal-1    }     [literal-2    ]
    { identifier-1 } ,   [identifier-2 ]
...TO (identifier-m) . [identifier-n]

Add { literal-1    }   { literal-2    }
    { identifier-1 }   { identifier-2 }
...GIVING (identifier-m), [identifier-n]
```

In the first format all the operands preceding the word TO are added together and the result is added to each of the item following the word TO and stored in the respective fields. For example, if we write.

ADD OLD-STOCK TO MEAN-STOCK

the old stock will be added to the value already in the mean stock and the result stored in the mean stock field. The value of the field preceding the word TO is left unchanged. If we write

ADD ALLOWANCE, SPECIAL-PAY TO SALARY

the value of allowance and special pay will be added together and the same will be added to the salary and stored in the field of the salary.If the values of the three fields before computation are 23.00, 47.00 and 200.00 respectively, their values after computation will become 23.00, 47.00 and 270.00 respectively. Suppose, we write

ADD ALLOWANCE TO SPECIAL-PAY, SALARY

the value of the three fields after computation will become 23.00, 70.00 and 223.00 respectively.

In the second format also the operands preceding the word GIVING are all added together and the result is stored in each of the fields succeeding the word GIVING. The difference is that there is no further addition with the values already present in the fields succeeding the word GIVING. The old values, if any, are simply replaced by the total obtained from the fields on the left of the word GIVING. Suppose, we write

ADD A, B GIVING C

and if the original values of the three fields are 23, 47 and 200 respectively, the values of the three fields after computation will become 23, 47 and 70 respectively. Another advantage in this format is that the giving field or the storing field can be given editing specifications so that the result can be stored in the edited form.

Note: Braces ({ }) enclosing a number of items indicate that one of the items must be used. Square brackets indicate that the enclosed items are optional as required by the program. When the brackets contain more than one item, one or more may be used as required.

Rounded option

When an arithmetic operation is performed the result may contain more decimal places than provided in that field. For example, if the result of a calculation comes out as 450.376 and if the field has provision for only two decimals, the number will be stored as 450.37 after truncating the last digit. It is more accurate to round off the number to the second decimal place, and this can be specified by the ROUNDED option. Rounded specifications prevent truncation after decimal point alignment, if the number of digits to the right of the decimal place exceeds the number of positions allotted in the result item. When ROUNDED is specified the least significant digit of the retained result is incremented by 1 if the most significant of the excess digit is 5 or greater. The following examples will show the use of ROUNDED option.

Result of calculation	*Result without option*	*Result with Rounded option*
523.467	523.46	523.47
276.985	276.98	276.99
256.301	256.30	256.30

In a typical ADD statment, the ROUNDED option is given as follows:

ADD A, B, C GIVING S ROUNDED
ADD A, B, C TO D ROUNDED

Size error option

A size error condition is said to exist if after decimal point alignment, the value of the result cannot be contained in the result item. This applies only to the integer part of the answer and not to the decimal position, since excess decimal places will be truncated. If a result 2579 is to be placed in a field with only three digits capacity, a size error condition arises. Checking for size error condition is carried out

---

This is the fourth part of the serial on Cobol Programming being published regularly since January 1978. *Mr R. Ramaswamy* is lecturer in physics at Thiagarajar College of Engineering, Madurai and Mr *T.V. Krishnamurthy* is system programmer with the K. C.P. Limited, Madras.

Earlier they co-authored a serial on 'Computer Language: Fortran IV' which was published in the January 1976 to May 1976 issues of EFY. Their first co-authored serial on 'Computer Languages' in EFY appeared in the May 1974 to April 1975 issues. They have also jointly authored a book entitled 'Teach Yourself Computers.'

only on the final result of the calculation and is done after rounding if the rounded option has been specified.

Rounding option precedes the checking for the size error condition. If a size error occurs during the execution of a statement not specifying the size error option, the result is unpredicatable. A size error option does not alter the result. By giving this option the computer is instructed as to what is to be done in case the size error condition arises during computation. One specification may be to ask the computer to stop. Another specification may be to ask the computer to stop. Another specification may be to ask the computer to go to some other statement somewhat as follows:

ADD TO B ON SIZE-ERROR GO TO OVER-FLOW
ADD A TO B ON SIZE-ERROR STOP

Subtract statement

This statment is used to subtract one or more values from another value. The Subtract statement can be written in the following two formats:

SUBTRACT { identifier-1 / literal-1 } [identifier-2 / literal-2] ...FROM{identifier-m} [identifier-n]

SUBTRACT { identifier-1 / literal-1 } [identifier-2 / literal-2] ...FROM(identifier -m) GIVING(identifier-n.)

In the first format, all the operands preceding the word FROM are added together. This total is then subtracted from the item following the word FROM and the result is stored in each of the operands. For example, if we write

SUBTRACT A, B, C FROM D, E, F

the values of the first three fields will be added together and this sum will be subtracted from each of remaining fields in turn. Results stored in D, E and F are (D-(A + B + C)), (E −(A+ B +C)) and (F − (A + B + C)) respectively. The values originally stored in D, E and F are lost. In the second format all the operands preceding the word FROM are added together. This total is then subtracted from the operand immediately following the word FROM and the result is stored in the data item specified following the word GIVING. It must be noted that the value of the identifier-m is not changed.

In the previous example, if we want to retain the value of D, we can write

SUBTRACT A, B, C FROM D GIVING K

The above statement causes the computation (D− (A + B +C)) and storage of the result in the location K. Only the initial value of K is lost, but not the values of A, B, C and D. Only one identifier may appear after the word FROM and one after the word GIVING. In some computers more than one identifier can be present after the word GIVING. In that case each identifier will store only the value of K as computed above. Rounded specification and size error option can be given to the result of computation. For example, one can write

SUBTRACT A, B, C FROM D GIVING K ROUNDED, ON SIZE ERROR GO TO PARA-2.

Multiply statement

This statement is used to obtain the product of two data item values. The Multiply statement can be written in the following two forms;

MULTIPLY { identifier-1 / literal-1 }
BY (identifier-2), [identifier-3]

MULTIPLY { identifier-1 / literal-1 }
BY { identifier-2 / literal-2 }
GIVING identifier-3

In the first format, the value in the identifier-1 or the value of the literal-1 is multiplied by the value of the identifier-2 and the result is stored in the identifier-2. Similarly, the identifier-1 is multiplied by the identifier-3 and the product is stored in the identifier-3. That is, each product replaces the corresponding multiplier. For example, if we write

MULTIPLY A BY B, C, D

The products AB, AC and AD are stored in the locations B, C and D respectively.

In the second format, the single product is stored in each of the identifier following the word GIVING. The value of the first two fields are not changed by the execution of the statements. Suppose we write

MULTIPLY A BY B GIVING C, D, E.

the product AB is stored in each of the locations C, D and E, removing the original contents of C, D and E. Rounded specifications and size error option can be given for the final result.

Divide statement

This statement is used to divide one data item by another data item and calculate the quotient and the remainder if required. The DIVIDE statement can be written in the following five forms:

DIVIDE { identifier-1 / literal-1 } INTO (indentifer-2), (identifier-3)

DIVIDE { indentifier-1 / literal-1 } INTO (identifier-2), GIVING identifier-3

DIVIDE { identifier-1 / literal-1 } INTO (identifier-2), GIVING identifier-3 REMAINDER (identifier-4)

DIVIDE { identifier-1 / literal-1) } BY (identifier-2) GIVING (identifier-3)

DIVIDE { identifier-1 / literal-1 } BY (identifier-2) GIVING identifier-3 REMAINDER identifier-4

In the first format, the value in the identifier-1 is divided into the identifier-2 and the quotient replaces the value of the identifier-2 discarding the remainder. The calculation can be summarised as follows:

$$\text{identifier-2} = \frac{\text{identifier-2}}{\text{identifier-1}}$$

Similarly, the identifier-3 is divided by the identifier-1 and the quotient is stored in the identifier-3, discarding the remainder. In the second format the single quotient obtained by dividing the identifier-2 by the identifier-1 is stored in each data item specified following the word GIVING. In the third format both the quotient and the remainder are produced. The fourth and the fifth formats use the word BY instead of the word INTO. This causes the identifier-1 to be divided by the identifier-2. In the fourth format the quotient alone is stored in each of the identifiers following the word GIVING. In the fifth format both the quotient and the remainder are produced. When we use the GIVING form both the dividend and the divisor are saved, whereas in other cases, the dividend is replaced by the quotient. Rounded and size error options can be given for the result in all cases. The following are examples of valid DIVIDE statements:

1. DIVIDE A INTO B
2. DIVIDE A INTO B GIVING C, D, E
3. DIVIDE B BY A GIVING C, D, E
4. DIVIDE B BY A ON SIZE ERROR GO TO OVER-FLOW
5. DIVIDE B BY A GIVING C, D, E ROUNDED
6. DIVIDE B BY A GIVING C REMAINDER D

Accept statement

The Accept statement is used to read low-volume data from a designated low-speed hardware device. The general format is as follows:

ACCEPT (identifier) FROM (mnemonic-name-1)

The mnemonic name is the programmer assigned name for the hardware device which is to accept the data. This name must have appeared in the Special Names paragraph of the environment division before it is used in the procedure division. The maximum length of data which the hardware device can transfer through this statement depends on the implementor.

This statement is usually used to give the computer messages from console. Occasionally this statement can also be used to give the computer messages from the card reader. With an accept statement only one card can be read at a time and after executing it the computer will output a message 'Awaiting Reply', and wait until the operator gives the next instruction manually. Thus we see that there is a wastage of computer time in using the Accept Statement and so this statement is used very sparingly except when the operator wants to give some special message to the computer.

Display statement

The Display statement is used to write a low-volume data on a designated low-speed hardware device. The general format is as follows:

DISPLAY { literal-1 / identifier-1 }
UPON mnemonic-name-1

This mnemonic name is the programmer assigned name for the line printer which is to display the identifier-1 or the literal-1. If UPON and the succeeding words are not given, the display is made on the console typewriter. If the mnemonic name is given in the statement, it must have occurred in the special names paragraph of the environment divison. The Display statement is usually used in conjunction with the Accept statement. The Display and the Accept statements must be restrictively used, firstly because they take lot of computer time for execution, and secondly because they require the attention of the operator every time a line is accepted or displayed. The following are some of the valid Accept and Display statements:

ACCEPT KARD-IN FROM CARD-READER-1
DISPLAY KARD-IN UPON PRINTER-1
DISPLAY KARD-IN
ACCEPT KARD-IN

In the last two statements the transfer is through the console typewriter.

Move statement

The MOVE statement is used to transfer data or copy data from one computer storage location (identified by data name) to another storage location (identified by another data name). The general form of the statement is as follows:

MOVE { data-name-1 / literal }
TO (data-name-2) [data-name-3]...

The operand to the left of TO is referred to as the sending area, while the operands to the right of TO are referred to as the receiving areas. It must be noted that the contents of the sending area remain unchanged by the MOVE operation. The following are some examples of valid MOVE statements:

MOVE PAY-IN TO PAY-OUT
MOVE AREA-1, TO AREA-2, AREA-3, AREA-4
MOVE 'GAIN' TO FIELD-1

When one moves one data item to another data item, the size of the receiving field must be as large as the field being sent, otherwise the data will be truncated. Truncation is determined by the receiving field. When the receiving field is alphanumeric and smaller than the sending field, the excess right-hand characters of the sending field will be truncated. When the receiving field is numeric and smaller, excess left-hand characters of the sending field will be truncated. When the receiving field is alphanumeric and

larger, the dats is placed in the leftmost positions of the receiving field. The remaining receiver positions are automatically filled with spaces. When the receiving field is computational and larger, the data is placed in the rightmost positions of the receiving field (rightmost sending digit goes to rightmost receiving digit etc). The remaining receiver positions are automatically filled with zeros. Numeric data moved into a field with edit symbols is handled according to the rules for numeric fields which we will be seeing later.

The following table summarises the valid and the invalid data movements:

Sending field	*Receiving field*	*Legality*
Alphabetic	Numeric	Illegal
Alphabetic	Alphanumeric	Legal
Numeric	Alphanumeric	Legal for integers only
Numeric	Alphabetic	Illegal
Alphanumeric	Numeric	Legal for integers only
Alphanumeric	Alphabetic	Legal for alphabetic characters only

Stop Statement

The STOP statement is used to make either a temporary stop or a permanent stop of the computer processing of a particular program. The general form is

STOP { literal / RUN }

When a literal is used, the literal will be displayed on the console typewriter, so that the operator may know where the program has halted and take appropriate action for its further continuance. When the statement is given as STOP RUN the processing of the program is terminated permanently.

To be continued next month