##### Assignment statement

The Assignment statement specifies arithmetic calculation or data movement. You can use the assignment statement to assign the value of an arithmetic expression to a data item, assign the value of one item to another, or assign the values of items in one structure to corresponding items (items with the same names) in another structure.

►►result= *numeric-expression*

(R

*operand*

*function-invocation*

; ►◄

##### numeric expression

►► numeric-expressionoperatornumeric-expression ►◄ operand

− (numeric-expression)

+

**Attribute Description**

result The result for an arithmetic assignment statement can be a data item, EZETST, EZEMNO, or EZERCODE. If it is a data item, it can be subscripted, qualified, or both, and it must be numeric, packed, or binary.

For other assignment statements, the result can be a data item, record, map, or certain special function words as listed in the table on [Table 20 on page 451.](#_bookmark338) If the result is a data item, it can be subscripted, qualified, or both.

For Assignment statement examples, see [“Examples for assignment”](#_bookmark300) [on page 383.](#_bookmark300)

function invocation

The operand can be a function invocation of a user defined function or of certain VAGen supplied special function words. A function used in an assignment statement must be defined to have a return value that is compatible with the result of this statement.

For the general syntax of function invocations, see [“Function](#_bookmark309) [invocation statement” on page 400](#_bookmark309) or for details of invoking a specific VAGen supplied function, refer to the description of that word in [“Special function words” on page 451.](#_bookmark338)

operator The following arithmetic operators are supported:

**+** Operands are added.

**−** The second operand is subtracted from the first operand.

**\*** Operands are multiplied.

**/** The first operand is divided by the second operand.

**//** The result is the remainder of a division of the first operand by the second operand. The remainder operator cannot be used with other operators.

All operators, except unary operators, must be surrounded by a space or line boundary, except when preceded or followed by parenthesis.

No operators are allowed on assignment statements involving records, maps, functions, or nonnumeric data.

**Attribute Description**

numeric expression

When the expression is an arithmetic calculation the following true:

v The expression can consist of 1 to 255 operands. Up to 500 characters can be entered using the Assignment Statement Template.

v The number of implicits in an arithmetic statement is limited to 16.

v The arithmetic operators can be any of the five binary operators (+, −, \*, /, //). These operators require two operands.

v The arithmetic operators can be one of the two unary operators (+, −). These operators precede their operand.

v Parentheses can be used to specify the order in which the arithmetic expression is evaluated. When an arithmetic expression contains nested parentheses, the nested expression is evaluated before the expression in which it is contained.

The number of opening and closing parentheses must match, except for the rounding option symbol, (R. A closing parenthesis is not allowed for the rounding option symbol. If you specify the rounding option, you must code it after the last closing parenthesis.

v When you do not use parentheses, the following order of evaluation applies:

* Unary operators are performed first.
* Multiplication and division are performed next.
* Addition and subtraction are performed last.
* Within a statement, the operations are performed according to priority as they are encountered in the statement from left to right. All first priority operations are performed before any second priority operations.

**Note:** Do not use the remainder operator with other arithmetic operators or with the rounding option.

For Assignment statement examples, see [“Examples for assignment”](#_bookmark300) [on page 383.](#_bookmark300)

**Attribute Description**

operand For arithmetic assignment statements, the operand can be a data item, numeric literal, or any numeric special function word. If it is a data item, it can be subscripted, qualified, or both, and must be numeric, packed, or binary.

For other assignment statements, the operand can be a data item, literal, record, map, or certain special function words as listed in the table on [Table 20 on page 451.](#_bookmark338) If the operand is a data item, it can be subscripted, qualified, or both.

**Note:** For all assignment statements where the operand is a numeric literal, you must surround the numeric literal with at least one blank.

Any record data item can be specified as an argument on string function calls. In addition, numeric literals can be specified on calls to some string function words. See “String Function EZE words” to determine whether numeric literals can be used with a particular string function.

(R For arithmetic assignment statements, use (R to round the results after all arithmetic calculations are completed. If you do not specify the rounding option, the result is truncated.

**Note:** The rounding option cannot be used with the remainder operator.

Although the maximum supported length is 18, the maximum length of a data item used as a result of the rounding option is 17. This is because one digit is added internally to the original data item precision when performing the rounding option. Any violations for numeric and packed decimal items are detected at preprocessing time. No detection occurs for binary and numeric map field items if more than 17 digits are used in the rounded arithmetic calculation. Instead, overflow occurs during execution. Numeric implicits are created with a length of 17 instead of 18 when used as result data items of a rounded operation.

If you define an arithmetic statement with more than one operation, all intermediate operations are carried out without the rounding option. Only the result of the last operation is rounded. This is done by adding five to the digit at precision one higher than the precision of the result and then truncating.

For Assignment statement examples, see [“Examples for assignment”](#_bookmark300) [on page 383.](#_bookmark300)

**Attribute Description**

function invocation

The operand can be a function invocation of a user defined function or of certain VAGen supplied special function words. A function used in an assignment statement must be defined to have a return value that is compatible with the result of this statement.

For the general syntax of function invocations, see [“Function](#_bookmark309) [invocation statement” on page 400](#_bookmark309) or for details of invoking a specific VAGen supplied function, refer to the description of that word in [“Special function words” on page 451.](#_bookmark338)

##### Achieving consistent results across environments

Due to truncation of intermediate results, COBOL programs might have different results than GUI or C++ programs for the same arithmetic statements.

To ensure consistent results across environments, use only one binary operator per statement. Multiple addition and subtraction operators can be safely combined if the number of decimal places defined for the result item is greater than or equal to the number of decimal places in any of the operands.

The remainder operator can produce inconsistent results if the result or any of the operands are defined with decimal places greater than zero. To get a consistent remainder with decimal places, use the following algorithm instead of the remainder operator:

quotient = dividend / divisor ;

remainder = dividend - (quotient \* divisor) ;

###### Overflow conditions

You can test and control overflow conditions resulting from arithmetic calculations using EZEOVER and EZEOVERS special function words.

##### Compatibility with CSP/AE arithmetic

Customers moving programs from CSP/AE to VisualAge Generator can specify /MATH=CSPAE when generating the programs for host COBOL environments to ensure that the results of arithmetic expressions are the same in COBOL as they were when running under CSP/AE. If standard COBOL arithmetic is satisfactory, use the default option, /MATH=COBOL, instead for better performance.

Compatibility with CSP/AE is not supported in the test facility, in GUI, or in generated C++ programs.

CSP/AE statements that follow the guidelines for compatibility in the previous section provide consistent results in all environments.

##### Target environments for assignment

Supported in all environments without compatibility considerations.

##### Examples for assignment

The following examples use the Assignment statement as an arithmetic expression or a MOVE statement:

###### An arithmetic expression with parentheses

The following example shows an arithmetic expression that uses parentheses.

PERCENT-CHANGE = (NEW-VALUE - OLD-VALUE) \* 100 / OLD-VALUE;

The processing order in the above example is determined by the parenthesis and the precedence of operators, as follows:

1st intermediate result1 = (NEW-VALUE minus OLD-VALUE)

2nd intermediate result2 = intermediate result1 multiplied by 100 3rd PERCENT-CHANGE = intermediate result2 divided by OLD-VALUE

###### An assignment statement to move or initialize data

When the assignment statement involves data movement and the source expression consists of one operand, the assignment statement works exactly like the MOVE statement.

The following example shows how you use the assignment statement to move data or initialize data:

OLD-VALUE = NEW-VALUE;

###### Example of valid arithmetic statements

The following are valid arithmetic statement examples:

OP1 = OP2 + OP3 \* OP4; /\* Need space around \*, /, +, −, // OP1 = OP2 \* (OP3 + OP4); /\* Parentheses force addition first.

OP1 = OP2 - -OP3; /\* There can be blanks between the unary OP1 = OP2 + -OP3; /\* sign and the operand.

OP1 = OP2 - -(OP + OP); /\* Unary sign before parenthesis is valid. OP1 = OP2 + -(-OP3); /\* OP3 is an operand, (−OP3) is an

/\* operand.

OP1 = -OP2 + OP3; /\* You can start with unary minus. OP1 = OP2 + OP3[R]; /\* [R] is a subscript.

OP1 = (OP2) (R; /\* You can round a single operand. OP1 = OP2 /(OP3 + OP4); /\* No space needed between / and (.

###### Example of arithmetic statements that are not valid

The following arithmetic statement examples are **not** valid:

OP1 = OPERAND1 - - -OPERAND2; /\* Two consecutive signs not allowed. OP1 = OP2+OP3\*OP4; /\* Need spaces around + and \*.

OP1 = OP2 (- OP3); /\* Missing operator.

OP1 = OP2 \*(OP3 +(OP4 / OP5); /\* Unmatched parentheses.

OP1 = OP2 + OP3 \* OP5 [R]; /\* No space allowed before subscript,

###### Example of valid assignment statements

The following are example assignment statements:

MESSAGE\_FIELD = 'Enter option'; /\* Set up message.

MESSAGE-NUMBER = 007; /\* Initialize message number. INIT-MAP = CUST-INFO-RECORD; /\* Initialize map fields.

###### Example assignment statement that is not valid

The following assignment statement is **not** valid:

MESSAGE\_FIELD = ('Enter option'); /\* Do not use ()s with character data

###### Rounded arithmetic statement with multiple operations

The following example uses the /MATH=CSPAE option:

RESULT = OP1 + OP2 + OP3 (R;

|  |  |  |  |
| --- | --- | --- | --- |
| Where:  Field Name | Decimal  Places Length Value | | |
| RESULT | 2 | 4 |  |
| OP1 | 4 | 5 | 1.2345 |
| OP2 | 3 | 4 | 5.678 |
| OP3 | 4 | 4 | .1169 |

This statement is executed as follows:

HOLD = OP1 + OP2 = 1.234 + 5.678 = 6.912 RESULT = HOLD + OP3 = 6.912 + .116 = 7.028 RESULT = 7.028 + 0.005 = 7.033

RESULT = 7.03

Without rounding, the statement is executed as follows:

HOLD = OP1 + OP2 = 1.23 + 5.67 = 6.90 RESULT = HOLD + OP3 = 6.90 + .11 = 7.01

**Note:** Truncation occurs on the operands to match the characteristics of the RESULT field.

###### Arithmetic statement with a negative number

The following example uses the /MATH=CSPAE option.

Without rounding:

A = B - C;

Where: A has 2 decimal places and a length of 3

B = 1.111

C = 3.888

A = -2.78

**Note:** If the number is negative, rounding is applied to the absolute value.

When you use the rounding option, a variable overflow condition can occur, depending on the value and the defined number of characters of the result item.

###### Arithmetic statement with a variable overflow

The following example uses the /MATH=CSPAE option:

Without rounding:

A = B + C;

Where: A has 2 decimal places and a length of 3

B = 8.888

C = 1.111

A = 9.99 with no overflow

With rounding:

A = B + C (R;

Where: A has 2 decimal places and a length of 3

B = 8.888

C = 1.111

A = 0.00 with overflow

The possibility of maximum value overflow increases because the operand value increments.

###### Arithmetic statement with division with a remainder

The following example uses the /MATH=CSPAE option.

Use integers in the operand and result portion of the statement when dividing for remainder. To obtain remainders when dealing with numbers other than integers, use the length and number of decimal places for the result for the remainder.

The following formula is used when calculating the remainder:

REMAINDER = DIVIDEND - (DIVISOR \* QUOTIENT);

**Note:** In the preceding formula, the value in the QUOTIENT has the same length and number of decimal places as the REMAINDER, as illustrated in the following example:

Name Type Length Dec

|  |  |  |  |
| --- | --- | --- | --- |
| REMAINDER1 | NUM | 8 | 3 |
| REMAINDER2 | NUM | 8 | 0 |

1. REMAINDER1 = 12345 // 10000 = 5.000

QUOTIENT1 = 12345 / 10000 = 1.234

Example A is calculated as follows:

REMAINDER1 = 12345 - (10000 \* 1.234) = 5.000

1. REMAINDER2 = 12345 // 10000 = 2345

QUOTIENT2 = 12345 / 10000 = 1

Example B is calculated as follows:

REMAINDER2 = 12345 - (10000 \* 1) = 2345