

# Performance Considerations

This section covers the following topics:

- Formats
  - Arrays
  - Alpha Fields
  - DECIDE ON
  - Numeric Values
  - Variable Positioning
  - Variable Caching
  - NODBG
- 

## Formats

Best performance is achieved when you use the data formats packed numeric (P) and integer (I4) in arithmetic operations.

You should avoid converting data between the formats packed numeric (P), zoned numeric (N), integer (I), and floating point (F), as this causes processing overhead even with optimized code.

As there is no interpretation overhead with optimized code, the differences between the various data formats become much more prominent: with optimized code the performance improvement gained by using format P instead of N, for example, is even higher than with normal code.

### Example:

```
A = A + 1
```

In the above numeric calculation

- with non-optimized code, the P format executes approximately 13 % faster than the N format.
- with optimized code, however, the P format executes approximately 56 % faster than the N format.

The performance gain which would be achieved by applying the Natural Optimizer Compiler to this simple statement is

- with unpacked operands (N): 8 times faster
- with packed operands (P): 15 times faster

## Arrays

Array range operations, such as

```
MOVE A(*) TO B(*)
```

are executed more efficiently than if the same function were programmed using a FOR statement processing loop. This is also true for optimized code.

When indexes are used, integer format I4 should be used to achieve optimum performance.

## Alpha Fields

When moving alpha-literal values to alpha variables, or comparing alpha variables and alpha-literal values, try to make the length of the alpha literal the same as the variable if the difference in length is not large. This will greatly speed up operation, for example:

```
A(A5) := 'XYZAB'  
...  
IF A = 'ABC ' THEN ....
```

## DECIDE ON

When using DECIDE ON with a system variable, array or parameter *operand1*, it is more efficient to move the value to a scalar variable of the same type and length defined in the LOCAL storage section.

## Numeric Values

When using numeric constants in assignments or arithmetic operations, try to force the constants to have the same type as the operation.

### Rules of thumb

- Any numeric constant with or without a decimal but without an exponent is compiled to a packed number having the minimum length and precision to represent the value, unless the constant is an array index or substring starting position or length, in which case it becomes a four-byte integer (I4). This rule applies irrespective of the variable types participating in the operation.
- Operations containing floating point will be executed in floating point. Add **E00** to numeric values to force them to be floating point, for example:

```
ADD 1E00 TO F(F8)
```

- Operations not containing floating point, but containing packed decimal, zoned decimal, date or time variables will be executed in packed decimal. For ADD, SUBTRACT and IF, force numeric constants to have the same number of decimal places as the variable with the highest precision by adding a decimal place and trailing zeros, for example:

```
ADD 1.00 TO P(P7.2)
```

This technique is unnecessary for MULTIPLY and DIVIDE.

## Variable Positioning

To ease the optimization process, try to keep all scalar references at the front of the data section and all array references at the end of the data section.

## Variable Caching

**Applies to NOC Version 3.1.5 only.**

The Natural Optimizer Compiler Version 3.1. and above contains an algorithm to enhance the performance even further. In terms of performance, a statement will differ depending on the types of operands. The statement will execute slower if one or more of the operands is a parameter, array or field of Type N (numeric) or combinations of these types. The NOC analyzes the program flow and determines the variables with one or more of these characteristics that are read two or more times without being written to. It then moves the value of these variables to a temporary cache area where it can be accessed quickly under the following conditions:

- the variable must be accessed often but modified rarely **and**
- the variable is a field of type N (numeric) used often in arithmetic, **or**
- the variable is an array with variable index and accessed without the index being changed, **or**
- the variable is a parameter.

The optimizer counts variable references inside an IF or THEN clause separately to those outside these statements. Thus, a variable **may not** be cached even though there are more than two references in the program because of single references inside an IF statement.



**Warning:**

**If the content of a cached variable is modified with the command MODIFY VARIABLE of the Natural Debugger, only the content of the original variable is modified. The cached value (which may still be used in subsequent statements) remains unchanged. Therefore, variable caching should be used with great care if the Natural Debugger is used. See also the Natural Debugger documentation.**

## NODBG

Once a program has been thoroughly tested and put into production, you should catalog the program with the NODBG option as described in the section Optimizer Options. Without debug code, the optimized statements will execute from 10% to 30% faster.

**Applies to NOC Version 3.1.5 only:**

From NOC Version 3.1 onwards, the code to facilitate debugging is removed when this option is specified, even with INDX or OVFLW options turned on.