Identification

Implementation of on-conditions in EPL
D.B. Wagner

Discussion

On-conditions in PL/I provide a kind of machine-independent trap-handling mechanism. See the PL/I manual, IBM form C28-6571-3, p. 80, for a detailed discussion. Briefly, the on statement specifies a sequence of statements called an on-unit which is to be executed when a condition arises. Some conditions, such as overflow, may arise as a result of the execution of ordinary code; however the signal statement can be used to simulate the occurrence of any condition. (In the case of the overflow condition, the machine overflow fault is handled by a special fault catcher which uses the signal statement to raise the EPL overflow condition.)

Within a block, on-statements reset each other, but "push-down" on-statements executed in dynamically embracing blocks. The revert statement causes the on-unit which is current in the dynamically embracing block to be reinstated.

Some conditions take an argument, e.g.:

```
     conditions (x_err)
     endfile (user_input)
```

and these require a special treatment described later.

The present Section describes the code generated by EPL for the on, signal, and revert statements and the form of the on-unit.

Condition Prefixes

"Condition prefixes" may be placed on blocks and on single statements to "enable" or "disable" conditions. When a condition is disabled in a block, all signals of that condition, whether implicit or explicit, which occur while control is in that block or any static descendant of it, are suppressed.

In PL/I, eight conditions may be enabled and disabled using condition prefixes:

```
     underflow
     overflow
     zerodivide
     fixedoverflow
     conversion
     size
     subscriptrange
     check (identifier)
```

Because of implementation difficulties, the first four (which depend upon hardware fault handlers) may not appear in condition prefixes in EPL. Because no difficult conversions are supported by EPL, the conversion condition is not necessary and may not appear in condition prefixes. Similarly the check condition is not supported by EPL. This leaves two conditions which may be enabled and disabled in EPL:

```
     size
     subscriptrange
```

When these conditions are enabled, the compiled code checks for them whenever they might occur, and executes the standard code for the signal statement if they do occur. No more need be said here about condition prefixes.

Global Strategy

The on, signal, and revert statements are implemented using the Multics system routines condition, signal, and reversion, described in BD.9.04.

For any of the "standard" conditions which do not take arguments, e.g. overflow, the name given to the system routines is the name of the condition. Thus the statement

```
     signal overflow;
```

is equivalent to the statement

```
     call signal ("overflow");
```

For a programmer-defined condition, e.g. condition (x_007), the name given to the system routines is normally the identifier specified in the statement. Thus the statement

```
     signal condition (x_007)
```

is equivalent to

```
     call signal ("x_007");
```

However for a programmer-defined condition in which the identifier in parentheses happens to be the name of one of the ten standard PL/I conditions which do not take arguments (*conversion*, *fixedoverflow*, *overflow*, *size*, *underflow*, *zerodivide*, *subscriptrange*, *area*, *finish*, and *error*) a special naming convention is used to avoid ambiguity. The name given to the system routines is the name in parentheses with the constant "condition." prefixed to it. Thus the statement

```
     signal condition (overflow);
```

is equivalent to

```
     call signal ("condition.overflow");
```

Implementation

The above is all that needs to be said here from the point of view of global strategies. What follows is merely a discussion of the implementation of the global strategy in greater depth. The Multics condition-handling routines were not designed with PL/I in mind, and a certain amount of fiddling is needed to implement correctly the PL/I rules of how and when things get reverted.

The following discussion applies to each block in the program, and each *on*-condition which may have an *on*-statement executed for it in the block. Versions of the code generated by EPL are shown written in EPL for the sake of clarity. The precise EPLBSA code generated by EPL should eventually be documented in BN.6.07.

A call is kept in the block's automatic storage which is non-zero if and only if an *on*-unit is currently in effect for this *on*-condition in this block. This cell is set to zero by the prologue. Call this cell *x*. Then the code for the *on*-statement is equivalent to:

```
     if x ∧= 0 then call reversion ("name");

     call condition ("name", unit);

     x = 1;

unit:  proc;

     call reversion ("name");
     ... (the on-unit)

     call condition ("name", unit);

     end;
```

where *name* is the name derived from the naming conventions discussed in *Global Strategies*, above.

The code for the *revert* statement is equivalent to:

```
     if x ∧= 0 then do;

          call reversion ("name")

          x = 0;

          end;
```

And in the epilogue (executed at block termination) code equivalent to the following is executed:

```
     if x ∧= 0 then call reversion ("name");
```

The code for the signal statement requires no prologue or epilogue. It is simply

```
     call signal ("name");
```