IBM **Technical Newsletter**

Re: Form No. Z28-6642-0

This Newsletter No. Z28-2361

Date May 23, 1969

Previous Newsletter Nos. None

BSL Language Specification

This Technical Newsletter, a part of Release 17 of the BSL Compiler, provides replacement pages for BSL Language Specification. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be replaced and/or added are listed below.

Remove	Insert
3-4	3-4
21-22	21-22
23-24	23-24
39-40	39-40, 40.1
43-44	43-44, 44.1
65-66	65-66
67-68	67-68
69-70	69-70

Changes and additions to the text and illustrations are indicated by a vertical line to the left of the change.

Summary of Amendments

The attached amendments include information about three BSL language changes. The first amendment describes and illustrates a new language feature, the GENERATE DATA statement, which enables the user to define data items in assembler language which will be mapped with the compiler-generated constants and user-declared variables. The second amendment discusses a change in the RESTRICT/RELEASE statements which allows the user to specify a register variable name in addition to the actual register number. The third amendment contains information about the acceptance of GEND as a valid abbreviation of the language keyword GENERATED.

Note: Please file this cover letter at the back of the publication to provide a quick reference to changes and a means of checking receipt of all amendments.

CONTENTS

CONTENTS (continued)

Names known in a procedure are also known in procedures internal to that procedure. (The internal procedures should not declare these names.) However, names of item internal to a procedure are not considered known in any containing procedure, and therefore cannot be referenced there.

Examples:

```
P:    PROCEDURE;              /*P is EXTERNAL*/
      DCL A  INTERNAL,
          EV EXTERNAL,
          (INT1,INT2) INTERNAL ENTRY;
      A=0;
      GOTO LAB;
EP:   ENTRY;                  /*EP is EXTERNAL*/
      A=1;
LAB:  CALL INT1;              /*LAB is INTERNAL to P*/
      CALL INT2;
      EV=A;
      RETURN;
INT1:PROCEDURE;               /*INT1 is INTERNAL to P*/
      DCL INTLV;              /*INTLV is INTERNAL to INT1*/
      DO INTLV=1 TO 10;
      A=A+INTLV;              /*valid reference to A*/
      END;
      RETURN;
      END INT1;
INT2:PROCEDURE;               /*INT2 is INTERNAL to P*/
      INTLV=0;                /*invalid; INTLV is internal
                                to INT1, and is not known
                                in INT2*/
      CALL INT1;              /*valid; INT1 is known in P,
                                and therefore also in INT2*/
      RETURN;
      END INT2;
      END P;
```

```
EXTERNAL items:  P, EP, EV
Items INTERNAL to P:  A, LAB, INT1, INT2
Items INTERNAL to INT1:  INTLV
Items INTERNAL to INT2:  none

Items known externally:  P, EP, EV
Items known in P:  P, EP, EV, A, LAB, INT1, INT2
Items known in INT1:  P, EP, EV, A, LAB, INT1, INT2,
                      INTLV
Items known in INT2:  P, EP, EV, A, LAB, INT1, INT2
```

STORAGE CLASS

Data items may be classified according to how they are located. An item may be fixed at a particular location, or it may have a position which varies depending on a locating mechanism.

```
Attributes:     STATIC
                     LOCAL
                     NONLOCAL

                GEND
                GENERATED

                {AUTO      }
                {AUTOMATIC }

                {REG       }
                {REGISTER  }   (register)

                 BASED   [ (locating-expression) ]
```

Fixed Data Areas

The STATIC attribute indicates that main storage is statically assigned for the data item, and never reassigned. STATIC has subclasses LOCAL and NONLOCAL, which indicate the location of the data item relative to the generated code for the declaring procedure.

The LOCAL attribute indicates that the data item is assigned storage, in the same area (CSECT) as for the generated code.

The NONLOCAL attribute specifies that the item is not assigned storage by the declaring procedure. In the case of a NONLOCAL EXTERNAL item, storage assignment is provided by a declaration as LOCAL EXTERNAL in some separately compiled procedure.

Examples:

```
P:PROCEDURE;
  DECLARE L INTERNAL LOCAL STATIC;
       /*storage for L is with the code for P*/
  DECLARE NL STATIC NONLOCAL;
       /*storage for NL is assigned by some other
         procedure having NL as LOCAL EXTERNAL*/
```

User Generated Data

The GENERATED attribute is associated with items defined and insured addressable by the user in a GENERATE statement (explained later on). These items are internal to the procedure, but are not assigned a storage area by the compiler.

The attribute combination NONLOCAL INTERNAL, formerly used to obtain this function, is still recognized by the compiler but should no longer be used.

Examples:

```
  DECLARE DCBISAM GENERATED CHAR(200);
    •
    •
    •
  GENERATE DATA; /*definition of DCBISAM*/
DCBISAM DCB    •••
               •••
$ENDGEN
```

Automatic Storage Allocation

The AUTOMATIC attribute may be used in a reentrant environment to provide an automatic allocation of storage for data on entry to a procedure, and an automatic freeing on exit. (The REENTRANT option is discussed under Procedure Options.) AUTOMATIC should not be used in a nonreentrant environment.

AUTOMATIC data declared in internal procedures will be allocated at the same time as AUTOMATIC data for the outer procedure. This reflects the fact that internal procedures cannot be separately reentrant, and thus only require one data allocation for each allocation of the outer procedure.

Examples:

```
P:PROCEDURE OPTIONS (REENTRANT);
  DECLARE S CHAR(256) AUTOMATIC; /*storage assigned on
    •                              entering P*/
    •
    •
Q:PROCEDURE;
  DECLARE SINQ BIT(32) AUTOMATIC; /*storage also assigned on
    •                              entering P*/
    •
    •
  END Q;

  END P;
```

Data in Registers

The user may associate the REGISTER attribute with a name, to access data located in the registers of the machine. Register specification must be in the range 0 through 15, corresponding to the general registers on System/360. Use of registers requires knowledge of the conventions used by the compiler. These are discussed in the section on Register Usage.

Example:

```
DECLARE R3 REGISTER(3) POINTER(31);
```

Parameters

For some names, the data attributes provided in the declaration are applied to an area located indirectly. The most common example of this is the use of input parameters for a procedure. References to parameters are indirect references through a list of pointers to the corresponding arguments.

Parameters have no attribute keyword to represent 'parameter' storage class. Parameters are indicated as such by their appearance in a parameter list.

Examples:

```
P:PROCEDURE (PARAM1,PARAM2);  /*PARAM1 and PARAM2 are
                                 parameters*/
  DECLARE PARAM1 FIXED(31), PARAM2 CHAR(16);
    •
    •
    •
  END P;
```

Indirect Addressing

The user may obtain indirect data addressing by using the BASED storage class. A name in this class provides a description of an area whose location is based on an associated pointer value. This pointer value is the value of the locating expression in the BASED attribute, or of a pointer associated at a reference to the BASED item. (This topic is explained in the section on Pointer Association.) The expression or pointer yields the address of a storage area. This area is then treated as if it had the attributes associated with the BASED name.

The END Statement

General form:

```
        [ label      ]
END     [ entry-name ] ;
```

The END statement indicates the end of the statements in a group or procedure. With no label or entry name following, it closes out the nearest preceding unclosed group or procedure. With a following label, which must be from a preceding unclosed DO statement, it serves as an end for all unclosed groups up to and including the one started by that DO statement. (This effect is called multiple closure.) An entry name following the END keyword must be the name of the nearest unclosed procedure, thus serving as a check on matching PROCEDURE and END statements.

An END statement which ends a procedure, and which is encountered in the execution path of that procedure, will act as a RETURN for that procedure.

Examples:

```
END;        /*closes nearest group or procedure*/
END DOSET; /*closes all up to DOSET*/
```

The ENTRY Statement

General form:

```
ENTRY [ (parameter [,parameter]... )] ;
```

The ENTRY statement specifies a secondary entry point for a procedure. It is preceded by an entry name by which this entry point is known. A correspondence exists between the arguments of the invocation and the parameters of the entry point, as discussed with the CALL statement. Parameters common to several entry points must have the same position in the parameter list.

Examples:

```
EP:        ENTRY(PAR1,PAR2);
EPNOPAR:   ENTRY;
```

The GENERATE Statement

General form:

```
GEN      }     [{(assembler-text)}]   ;
GENERATE }     [{ DATA            }]
```

The GENERATE statement provides a means of inserting assembly language text into BSL generated code. Use of this facility requires knowledge of compiler code and data generation characteristics.

For the simple GENERATE statement, the form with assembler text in parentheses, the text is mapped starting at column 10 of an output image. The simple GENERATE statement must be wholly contained within a single line (or card). The end of the text is indicated by the following sequence: a right parenthesis, optional blanks, and a semicolon. A label on the statement is placed in the name field of the output card.

Examples:

```
      GENERATE (COPY SECTION    ) ;
L:    GEN (LPSW MYOWNPSW) ;
```

The block GENERATE statement is used to map a series of cards. A label on a block GENERATE statement is put out into the assembler text with an "EQU *". The rest of the input card is ignored. One or more lines (cards) of assembler text following the GENERATE statement are processed until a delimiting control card is encountered. This control card, $ENDGEN, is discussed in the BSL User's Guide.

The input cards for the block GENERATE statement are put out into the assembler text. Mapping of the cards begins with column 1 and continues to the last column before the sequence number. If no compiler options (such as GENMGIN and SEQ) are used to modify the mapping of compiler output, columns 1-72 of the input cards are put into the assembler text (with sequencing in columns 73-80). The user should refer to the BSL User's Guide for the use of these options and their effect on generated code.

Names defined in the assembler text included by a block GENERATE statement are not known by the compiler, since it does not analyze these statements. To make labels and variable names known so that conflicting definitions are not produced, the user should declare such items with the attribute GENERATED in addition to other data attributes. These data items are assumed by the compiler to be addressable, and it is the user's responsibility to insure this.

The first form of the block GENERATE statement has no operand. It should be used to insert in-line, executable assembler text in the user's program.

```
Example:
/* INITIATE I/O OPERATION                                    */
GENERATE;
     SIO   R7
     BC    1,LABELA      HANDLE POSSIBLE ERROR
     BC    2,LABELB      CHANNEL OR SUBCHANNEL IS BUSY
     BC    8,LABELC      I/O CORRECTLY INITIATED
$ENDGEN
```

The second form of the block GENERATE statement has a single operand, the keyword DATA. (It should be noted that 'DATA' is a language keyword only when used in this context.) The GENERATE DATA statement should be used to define data items. This generated data does not appear in-line in the user's program at the point of definition; rather, it is put out at the end of the program following all other data mapping of compiler generated constants and user-declared variables. If a source listing is requested, however, the input cards for the GENERATE DATA statement are listed at the point of definition.

```
Example:
     DCL
     /* DECLARE OF GENERATED VARIABLE                         */
     DCBISAM GENERATED CHAR(200);

     /* GENERATE DATA CONTROL BLOCK                           */
     GENERATE DATA;
     DCBISAM  DCB .....
                  .....
                  .....
     *                 INCLUDE APPROPRIATE COMMENTS
     *                 HERE, IF DESIRED
     $ENDGEN
```

entry point, as discussed with the CALL statement. Procedures can be nested, in which case contained procedures can be invoked only from the immediately containing procedure or from other procedures within the containing procedure. A procedure is syntactically completed by an END statement.

Any internal procedures must be defined immediately before the END statement for the containing procedure. These procedures may themselves have internal procedures, in the same format.

The user may control the entry and exit code produced by the compiler, using the OPTIONS field of the PROCEDURE statement. These options are discussed in a separate section on Procedure Options.

Examples:

```
MAINP:  PROCEDURE(PARAM);
        DCL INTP INTERNAL ENTRY;
        CALL INTP;
        RETURN;
INTP:   PROCEDURE;
        RETURN;
        END INTP;
        END MAINP;
```

The RELEASE Statement

General form:

```
RELEASE ( { register } [ , { register } ] .... ) ;
        { regname  }       { regname  }
```

The RELEASE statement makes the indicated registers available to the compiler in generating code. The user may still explicitly reference the registers, but he should be aware of possible effects both on and by the compiled code.

The registers which may be released are discussed under the topic of Register Usage. Certain registers are preassigned and cannot presently be released or restricted.

At the start of each procedure, including each internal procedure, registers are assumed available for use by the compiler. Declaration of an item as register does not restrict that register.

The register specification in a RELEASE statement can be either the actual register number in decimal form or the name of a data item which has previously been declared with the REGISTER attribute.

Example:

```
DCL
     P1 PTR(31),           /*NON-REG VARIABLE          */
     R7 PTR(31) REG(7);    /*REG VARIABLE              */
RELEASE (6,7,8);           /*LEGAL                     */
RELEASE (P1);              /*ILLEGAL-P1 HAS NOT        */
                           /*   BEEN DECLARED REGISTER*/
RELEASE (R7);              /*LEGAL                     */
RELEASE(6,R7,8);           /*LEGAL                     */
```

The RESTRICT Statement

General form:

$$\text{RESTRICT}\ \left(\ \left\{\begin{matrix}\text{register}\\ \text{regname}\end{matrix}\right\}\ \left[\ ,\ \left\{\begin{matrix}\text{register}\\ \text{regname}\end{matrix}\right\}\ \right]\ldots.\ \right);$$

The RESTRICT statement prohibits the compiler from using the indicated registers. The registers will be used only if explicitly referenced by the programmer.

The registers which may be restricted are discussed under the topic of Register Usage. Certain registers are preassigned and cannot be restricted or released.

At the start of each procedure, including each internal procedure, registers are assumed available for use by the compiler; this includes registers which were restricted in the outer procedure. Declaration of an item as register does not restrict that register.

The register specification in a RESTRICT statement can be either the actual register number in decimal form or the name of a data item which has previously been declared with the REGISTER attribute.

Example:

```
DCL
     P1 PTR(31),           /*NON-REG VARIABLE          */
     R7 PTR(31) REG(7);    /*REG VARIABLE              */
RESTRICT (6,7,8);          /*LEGAL                     */
RESTRICT (P1);             /*ILLEGAL-P1 HAS NOT        */
                           /*   BEEN DECLARED REGISTER*/
RESTRICT (R7);             /*LEGAL                     */
RESTRICT (6,R7,8);         /*LEGAL                     */
```

The RETURN Statement

General form:

```
RETURN  [TO label] ;
```

The RETURN statement terminates execution of the procedure in which it is contained, returning control to the invoking procedure. In the form with no return label, it returns control immediately past the point of invocation. An END statement terminating a procedure will also serve as a return if control ever reaches the statement.

In the form with a return label, the normal return activity is performed, except that control is returned to the point in the calling procedure indicated by the label. The label should be in the invoking procedure, and known in the returning procedure.

Examples:

```
P:   PROCEDURE(L,M);
     DCL L LABEL, XL LABEL EXTERNAL,
          Q POINTER EXT, DL LABEL BASED(Q);
     RETURN;    /*returns to caller past point of call*/
     RETURN TO L; /*returns to label indicated by first
                    parameter*/
     RETURN TO XL; /*returns to label XL, which must be
                     defined in caller as LABEL LOCAL
                     EXTERNAL*/
     RETURN TO DL; /*returns through pointer Q, which
                     must have been set in caller*/
     END P;     /*this would act as a simple return if
                  control ever reached this point*/
```

APPENDIX I: LANGUAGE KEYWORDS

Status	Keyword	Use	References
	ABNL	data attribute	28
	ABNORMAL	data attribute	28
R	ABS	builtin function	32,56
R	ADDR	builtin function	25,32,56
	AUTO	data attribute	22,23
	AUTOMATIC	data attribute	22,23,30,60,61
	BASED	data attribute	16,17,22,24,28,29,57
	BDY	data attribute	26
	BIT	data attribute	15,29,30,55
	BOUNDARY	data attribute	26
R	BY	iteration term	38
	BYTE	boundary choice	26,30
R	CALL	statement header	36,37,39,43,63
	CHAR	data attribute	15
	CHARACTER	data attribute	15,29,30
	CODEREG	procedure option	59,61
	DATA	statement header	40,40.1
	DATAREG	procedure option	59,60,61
R	DCL	statement header	14,38
R	DECLARE	statement header	14,36,38,42
R	DO	statement header	12,35,36,38,39,41,42
	DONTSAVE	procedure option	60,61
	DWORD	boundary choice	26
R	ELSE	false path header	36,41,42
R	END	statement header	12,13,36,38,39,43,44
R	ENTRY	data attribute	17,29,30
R	ENTRY	statement header	11,13,29,36,37,39,42,63
	EXT	data attribute	20
	EXTERNAL	data attribute	20,22,28,29,30
	FIXED	data attribute	15,29,30,33,35
R	GEN	statement header	40,40.1
	GEND	data attribute	22,23
R	GENERATE	statement header	36,40,40.1.59
	GENERATED	data attribute	22,23,40.40.1
R	GO TO	statement header	29,41
R	GOTO	statement header	29,36,41,45,48,49,50
	HWORD	boundary choice	26,30
R	IF	statement header	11,32,36,41,42
	INIT	data attribute	27
	INITIAL	data attribute	27,31
	INT	data attribute	20
	INTERNAL	data attribute	20,23,29,30
	LABEL	data attribute	17,29,30
	LOCAL	data attribute	22,27,30
	NONLOCAL	data attribute	22,23,30
	NORMAL	data attribute	28,30
	NOSAVEAREA	procedure option	61

R - reserved identifier

Status	Keyword	Use	References
	OPTIONS	options header	42,43,59
	OPTIONS	data attribute	63
	POINTER	data attribute	16,29,30,33,35,57
R	PROC	statement header	42
R	PROCEDURE	statement header	11,13,29,36,37,39,42,43,63
	PTR	data attribute	16
	REENTRANT	procedure option	23,59,60,61
	REG	data attribute	22,24
	REGISTER	data attribute	22,24,56,58
R	RELEASE	statement header	36,42,43,44
R	RESTRICT	statement header	36,42,44
R	RETURN	statement header	36,37,39,44.1
R	RETURN TO	statement header	44.1
	SAVE	procedure option	60
	STATIC	data attribute	22,27,30
R	THEN	true path header	41
R	TO	iteration term	38
	VLIST	option choice	63
	WORD	boundary choice	26,30

COMPILE TIME KEYWORDS

Status	Keyword	Use	References
%R	ACT	statement header	52
%R	ACTIVATE	statement header	45,48,52
	CHAR	data attribute	48,49
	CHARACTER	data attribute	46,48,49,52
%R	DCL	statement header	48
%R	DEACT	statement header	52
%R	DEACTIVATE	statement header	45,48,52
%R	DECLARE	statement header	45,46,48,49,51
%R	ELSE	false path header	51
	FIXED	data attribute	46,47,48,49,52
%R	IF	statement header	45,51
%R	INCLUDE	statement header	45,51
%R	THEN	true path header	51

R - reserved identifier
%R - reserved identifier in compile time statements

APPENDIX II: ATTRIBUTE CONFLICTS

The matrix below indicates conflicts between attributes. An X in a position indicates that the attributes conflict. A number in a position indicates that the appropriate note below applies.

	ABNORMAL	AUTOMATIC	BASED	BIT	BOUNDARY	CHARACTER	Component	Dimension	ENTRY	EXTERNAL	FIXED	GENERATED	INITIAL	INTERNAL	LABEL	LOCAL	NONLOCAL	NORMAL	OPTIONS	Parameter	POINTER	REGISTER	STATIC	Structure
ABNORMAL	X																							
AUTOMATIC		X	X							X		X	X			X	X			X			X	
BASED		X	X							X		X	X			X	X			X			X	
BIT				X		X					X													
BOUNDARY					X																			
CHARACTER				X		X					X													
Component	X	X	X				X	1	X	X		X		X	X	X	X		X	X		X	X	
Dimension							1	X	X															
ENTRY	X	X		X	X	X	X	X	X		X		X		X				X		X	X		X
EXTERNAL		X	X				X			X		X		X		X	X			X			X	
FIXED				X		X			X		X										X			
GENERATED		X	X				X			X		X	X			X	X			X			X	
INITIAL		X	X						X			X	X			X	X			X				
INTERNAL							X			X				X			2							
LABEL	X	X		X	X	X	X	X	X		X		X		X				X		X	X		X
LOCAL		X	X				X					X				X	X			X			X	
NONLOCAL		X	X				X					X	X	2		X	X			X			X	
NORMAL	X						X		X									X						
OPTIONS	X	X		X	X	X	X	X			X		X		X				X		X	X		X
Parameter		X	X				X			X		X	X			X	X			X			X	
POINTER				X		X			X		X										X			
REGISTER		X	X	X	X	X	X	X	X	X		X	X		X	X	X		X			X	X	X
STATIC		X	X				X					X											X	
Structure									X															X

1. A component may not have a dimension attribute if a containing structure is dimensioned.

2. The combination NONLOCAL INTERNAL will be recognized as equivalent to GENERATED, but this combination should no longer be used.

INDEX