```
 CCCC   OOO   DDDD   EEEEE
C      O   O  D   D  E
C      O   O  D   D  E
C      O   O  D   D  EEEE
C      O   O  D   D  E
C      O   O  D   D  E
 CCCC   OOO   DDDD   EEEEE
```

```
 AAA   DDDD    AAA            1     888    999
A   A  D   D  A   A          11    8   8  9   9
A   A  D   D  A   A           1    8   8  9   9
A   A  D   D  A   A           1     888    9999
AAAAA  D   D  AAAAA           1    8   8       9
A   A  D   D  A   A    ..     1    8   8       9
A   A  DDDD   A   A    ..   111    888    999
```

```
*START* Job CODE Req #187 for DHB    Date 27-Feb-82 15:37:18 Monitor: Rational M
File PS:<SIM.DEF>CODE.ADA.189, created: 25-Feb-82 14:15:42
        printed: 27-Feb-82 15:37:19
Job parameters: Request created:27-Feb-82 15:35:51   Page limit:18   Forms:NORMAL
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:A
```


```
with BASE,
     SEGMENTS;

ckage CODE is

    type OPERAND_CLASS is
                (DISCRETE_CLASS,     FLOAT_CLASS,              ARRAY_CLASS,
                 RECORD_CLASS,       VARIANT_RECORD_CLASS,     ACCESS_CLASS,
                 MODULE_CLASS,       SELECT_CLASS,             ENTRY_CLASS,
                 FAMILY_CLASS,       EXCEPTION_CLASS,          SEGMENT_CLASS,
                 ANY_CLASS);


    type DECL_PRIVACY is (IS_PUBLIC,    IS_PRIVATE,    IS_LOCAL);

    type DECL_STATE   is (COMPLETE,          INCOMPLETE,
                          CONSTRAINED,       DERIVED);

    type DECL_OPTION  is (DERIVE_PRIVACY,  MAKE_UNSIGNED,     GARBAGE_COLLECT,
                          CREATE_QUEUE,    MAKE_CONSTRAINED,  ALIGN,
                          NO_OPTION);

    type OPTION_SET is
        record
            OPTION_1 : DECL_OPTION;
            OPTION_2 : DECL_OPTION;
            OPTION_3 : DECL_OPTION;
        end record;


    type VAR_VISIBILITY  is (IS_VISIBLE, IS_HIDDEN);

    type VAR_OPTION      is (UNCHECKED,   WITH_PARAM,    DUPLICATE,
                             NONE,        DATA_MODULE,   HEAP_MODULE);

    subtype SIMPLE_OPTION          is VAR_OPTION range  DUPLICATE .. NONE;
    subtype ARRAY_OPTION           is VAR_OPTION range  UNCHECKED .. NONE;
    subtype VARIANT_RECORD_OPTION  is VAR_OPTION range WITH_PARAM .. NONE;
    subtype MODULE_OPTION          is VAR_OPTION range NONE .. HEAP_MODULE;


    type SUBPROG_SORT is (FOR_CALL,          FOR_ACCEPT,
                          FOR_OPERATION,     FOR_ELABORATION,
                          FOR_OUTER_CALL,    FOR_OUTER_ACCEPT,
                          NOT_SUBPROG);
```

```
type LEXICAL_LEVEL is new INTEGER range     0 ..   15;
type SCOPE_DELTA   is new INTEGER range     0 ..  511;
type FRAME_DELTA   is new INTEGER range - 256 .. 255;

type OBJECT_REFERENCE (LEVEL : LEXICAL_LEVEL := 0) is
    record
        case LEVEL is
            when 0 .. 1 => SCOPE_OFFSET : SCOPE_DELTA;

            when others => FRAME_OFFSET : FRAME_DELTA;
        end case;
    end record;


type JUMP_OFFSET    is new INTEGER range - 2 ** 10 .. 2 ** 10 - 1;

type CASE_MAXIMUM   is new INTEGER range   0 .. 2 **  9 - 1;


type EXIT_POP_COUNT is new INTEGER range   0 .. 2 **  8 - 1;


FIELD_INDEX_SIZE : constant INTEGER := 8;

type FIELD_INDEX is new INTEGER range 0 .. 2 ** FIELD_INDEX_SIZE - 1;

type FIELD_SORT  is (DISCRIM, FIXED, VARIANT);


type LIT (OF_KIND : OPERAND_CLASS := DISCRETE_CLASS) is
    record
        case OF_KIND is
            when DISCRETE_CLASS =>
                DISCRETE_LITERAL : BASE.DISCRETE_LITERAL;

            when ARRAY_CLASS    =>
                ARRAY_LITERAL    : BASE.ARRAY_LITERAL;

            when others =>
                null;
        end case;
    end record;


type LONG_LIT (OF_KIND : OPERAND_CLASS := DISCRETE_CLASS) is
    record
        case OF_KIND is
            when DISCRETE_CLASS  =>
                DISCRETE_LITERAL : BASE.DISCRETE;

            when FLOAT_CLASS =>
                FLOAT_LITERAL    : BASE.REAL;

            when others =>
                null;
        end case;
    end record;
```

```
type OPERATOR is
            (EQUAL_OP,                  NOT_EQUAL_OP,          GREATER_OP,
             LESS_OP,                   GREATER_EQUAL_OP,      LESS_EQUAL_OP,
             FIRST_OP,                  LAST_OP,               VAL_OP,
             POS_OP,                    IS_CONSTRAINED_OP,     IS_TERMINATED_OP,
             SIZE_OP,                   ADDRESS_OP,            TYPE_SIZE_OP,
             TYPE_ADDRESS_OP,           LENGTH_OP,             COUNT_OP,
             BELOW_RANGE_OP,            ABOVE_RANGE_OP,        IN_RANGE_OP,
             NOT_IN_RANGE_OP,           IN_TYPE_OP,            NOT_IN_TYPE_OP,
             NOT_OP,                    AND_OP,                OR_OP,
             XOR_OP,                    UNARY_MINUS_OP,        ABS_OP,
             SUCC_OP,                   PRED_OP,               INCREMENT_OP,
             DECREMENT_OP,              PLUS_OP,               MINUS_OP,
             TIMES_OP,                  DIVIDE_OP,             MOD_OP,
             REM_OP,                    CONVERT_OP,            CONVERT_ACTUAL_OP,
             RUN_UTILITY_OP,            MAKE_CONSTANT_OP,      COMPLETE_OP,
             CONSTRAIN_COMPLETE_OP,     DERIVE_COMPLETE_OP,    BOUNDS_OP,
             REVERSE_BOUNDS_OP,         BOUNDS_CHECK_OP,       SLICE_OP,
             SLICE_ASSIGN_OP,           CONCATENATE_OP,        ELEMENT_TYPE_OP,
             SET_BOUNDS_SITE_OP,        SET_MIN_OP,            SET_MAX_OP,
             SET_VARIANT_OP,            FIELD_READ_OP,         FIELD_WRITE_OP,
             FIELD_REF_OP,              FIELD_TYPE_OP,         FIELD_EXE_OP,
             GUARD_WRITE_OP,            ENTRY_CALL_OP,         COND_CALL_OP,
             TIMED_CALL_OP,             FAMILY_CALL_OP,        FAMILY_COND_OP,
             FAMILY_TIMED_OP,           RENDEZVOUS_OP,         ACTIVATE_OP,
             DEQUEUE_OP,                ABORT_OP,              CONTINUE_OP,
             NULL_OP,                   NEW_OP,                ALL_READ_OP,
             ALL_WRITE_OP,              ALL_REF_OP,            RAISE_OP,
             RAISED_EXCEPTION_OP,       RAISED_ADDRESS_OP,     RAISED_SCOPE_OP,
             LOAD_OP,                   RELOAD_OP,             SEGMENT_NUMBER_OP);

subtype EQUALITY_OP    is OPERATOR range        EQUAL_OP .. NOT_EQUAL_OP;
subtype RELATIONAL_OP  is OPERATOR range      GREATER_OP .. LESS_EQUAL_OP;
subtype ATTRIBUTE_OP   is OPERATOR range        FIRST_OP .. COUNT_OP;
subtype RANGE_OP       is OPERATOR range BELOW_RANGE_OP .. NOT_IN_TYPE_OP;
subtype LOGICAL_OP     is OPERATOR range          NOT_OP .. XOR_OP;
subtype UNARY_OP       is OPERATOR range UNARY_MINUS_OP .. DECREMENT_OP;
subtype ARITHMETIC_OP  is OPERATOR range         PLUS_OP .. REM_OP;
subtype CONVERSION_OP  is OPERATOR range     CONVERT_OP .. CONVERT_ACTUAL_OP;
subtype VARIABLE_OP    is OPERATOR range RUN_UTILITY_OP .. MAKE_CONSTANT_OP;
subtype COMPLETION_OP  is OPERATOR range  COMPLETE_OP .. DERIVE_COMPLETE_OP;
subtype TYPE_OP        is OPERATOR range       BOUNDS_OP .. BOUNDS_CHECK_OP;
subtype ARRAY_OP       is OPERATOR range        SLICE_OP .. SET_MAX_OP;
subtype FIELD_OP       is OPERATOR range SET_VARIANT_OP .. FAMILY_TIMED_OP;
subtype COMPONENT_OP   is OPERATOR range  FIELD_READ_OP .. FIELD_REF_OP;
subtype TASKING_OP     is OPERATOR range  RENDEZVOUS_OP .. ABORT_OP;
subtype ELABORATE_OP   is OPERATOR range    CONTINUE_OP .. CONTINUE_OP;
subtype ACCESS_OP      is OPERATOR range         NULL_OP .. ALL_REF_OP;
subtype EXCEPTION_OP   is OPERATOR range        RAISE_OP .. RAISED_SCOPE_OP;
subtype SEGMENT_OP     is OPERATOR range         LOAD_OP .. SEGMENT_NUMBER_OP;
```

```
type SPECIAL_OPERATOR is
             (NIL,

              CALL_REFERENCE,       ACTIVATE_SUBPROG,
              DELETE_ITEM,          SET_VISIBILITY,

              ACCEPT_ACTIVATION,    ACTIVATE_ALL,
              SIGNAL_ACTIVATED,     ACCEPT_TERMINATION,   SELECT_TERMINATION,
              NAME_MODULE,          DELAY_MODULE,

              PUSH_CONTROL,         POP_CONTROL,          SWAP_CONTROL,
              MARK_AUXILIARY,       MARK_DATA,            MARK_TYPE,
              POP_AUXILIARY,        POP_DATA,             POP_TYPE,
              PUSH_DATA,            PUSH_TYPE,

              SET_DEBUG_MASK,       SET_DEBUG_SUBPROG,    SET_DEBUG_SCOPE,
              ESTABLISH_FRAME,      QUERY_FRAME,

              INTRODUCE_IMPORT,     REMOVE_IMPORT,        OVERWRITE_IMPORT);


subtype REFERENCE_OP   is SPECIAL_OPERATOR
                          range     CALL_REFERENCE ..  SET_VISIBILITY;

subtype ACTIVATION_OP  is SPECIAL_OPERATOR
                          range ACCEPT_ACTIVATION ..  SELECT_TERMINATION;

subtype STACK_OP       is SPECIAL_OPERATOR
                          range       PUSH_CONTROL ..  PUSH_TYPE;

subtype DEBUG_OP       is SPECIAL_OPERATOR
                          range     SET_DEBUG_MASK ..  QUERY_FRAME;

subtype IMPORT_OP      is SPECIAL_OPERATOR
                          range  INTRODUCE_IMPORT ..  OVERWRITE_IMPORT;
```

```
type OP_CODE is (DECLARE_TYPE,     DECLARE_VARIABLE,   DECLARE_SUBPROGRAM,
                 LOAD,             REFERENCE,          STORE,
                 CALL,
                 EXIT_PROCEDURE,   EXIT_FUNCTION,      EXIT_ACCEPT,
                 POP_PROCEDURE,    POP_FUNCTION,
                 OPERATE,          OPERATE_SPECIAL,
                 JUMP,             JUMP_FALSE,         JUMP_TRUE,
                 JUMP_CASE,        CASE_LABEL,
                 SHORT_LITERAL,    LITERAL,            LONG_LITERAL,
                 LITERAL_VALUE,    LONG_VALUE,         SEGMENT_VALUE,
                 HEADER,           HANDLER,
                 LOCALS,           PARAMETERS);


type INSTRUCTION (FOR_OP : OP_CODE := OPERATE_SPECIAL) is
    record
        case FOR_OP is
            when DECLARE_TYPE =>
                TYPE_CLASS    : OPERAND_CLASS;
                TYPE_PRIVACY  : DECL_PRIVACY;
                TYPE_STATE    : DECL_STATE;
                TYPE_OPTIONS  : OPTION_SET;

            when DECLARE_VARIABLE =>
                VARIABLE_CLASS      : OPERAND_CLASS;
                VARIABLE_VISIBILITY : VAR_VISIBILITY;
                VARIABLE_OPTION     : VAR_OPTION;

            when DECLARE_SUBPROGRAM =>
                SUBPROGRAM_SORT       : SUBPROG_SORT;
                SUBPROGRAM_VISIBILITY : VAR_VISIBILITY;

            when LOAD | REFERENCE | STORE | CALL =>
                OBJECT      : OBJECT_REFERENCE;

            when EXIT_PROCEDURE | EXIT_FUNCTION | EXIT_ACCEPT =>
                POP_COUNT   : EXIT_POP_COUNT;

            when POP_PROCEDURE  | POP_FUNCTION =>
                TO_LEVEL    : LEXICAL_LEVEL;

            when OPERATE =>
                OPERAND     : OPERAND_CLASS;
                OPERATION   : OPERATOR;
                FIELD       : FIELD_INDEX;
                FIELD_KIND  : FIELD_SORT;

            when OPERATE_SPECIAL =>
                SPECIAL_OP  : SPECIAL_OPERATOR;

            when JUMP | JUMP_FALSE | JUMP_TRUE =>
                RELATIVE    : JUMP_OFFSET;

            when JUMP_CASE  =>
                CASE_MAX    : CASE_MAXIMUM;
```

```
                when CASE_LABEL =>
                    CHOICE       : SEGMENTS.REFERENCE;

                when SHORT_LITERAL  =>
                    SHORT_VALUE : BASE.SHORT_LITERAL;

                when LITERAL | LONG_LITERAL =>
                    VALUE_CLASS      : OPERAND_CLASS;
                    VALUE_POINTER    : SEGMENTS.REFERENCE;

                when LITERAL_VALUE  =>
                    LIT_VALUE        : LIT;

                when LONG_VALUE     =>
                    LONG_LIT_VALUE   : LONG_LIT;

                when SEGMENT_VALUE =>
                    SEGMENT_NUMBER   : SEGMENTS.NAME;

                when HEADER       =>
                    BODY_START    : SEGMENTS.REFERENCE;

                when HANDLER      =>
                    EXCEPT_START : SEGMENTS.REFERENCE;

                when LOCALS       =>
                    END_LOCALS    : SCOPE_DELTA;

                when PARAMETERS   =>
                    PARAM_END     : FRAME_DELTA;
            end case;
        end record;


    NO_OP : constant INSTRUCTION := INSTRUCTION'(OPERATE_SPECIAL, NIL);


    type WORD    is array (SEGMENTS.INSTRUCTION_INDEX) of INSTRUCTION;

    type SEGMENT is array (SEGMENTS.DISPLACEMENT range <>) of WORD;


private

    for OPTION_SET use
        record
            OPTION_1 at 0 range  0 ..  7;
            OPTION_2 at 0 range  8 .. 15;
            OPTION_3 at 0 range 16 .. 23;
        end record;

end CODE;


package body CODE is end CODE;
```