```
FFFFF   RRRR     AAA    M   M  EEEEE  V   V
F       R   R   A   A   MM MM  E      V   V
F       R   R   A   A   M M M  E      V   V
FFF     RRRR    A   A   M   M  EEEE   V   V
F       R R     AAAAA   M   M  E      V   V
F       R  R    A   A   M   M  E       V V
F       R   R   A   A   M   M  EEEEE    V
```

```
 AAA    DDDD     AAA              1      333     333
A   A   D   D   A   A            11     3   3   3   3
A   A   D   D   A   A             1         3       3
A   A   D   D   A   A             1        3       3
AAAAA   D   D   AAAAA             1          3       3
A   A   D   D   A   A     ..      1     3   3   3   3
A   A   DDDD    A   A     ..    111     333     333
```

```
*START* Job FRAMEV Req #367 for DHB     Date  1-Mar-82 14:40:14 Monitor: Rational
File PS:<SIM.CPU>FRAMEV.ADA.133, created: 25-Feb-82 19:30:06
        printed:  1-Mar-82 14:40:14
Job parameters: Request created: 1-Mar-82 14:40:14   Page limit:36   Forms:NORMAL
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:AS
```

```
with CODE,
     STACKS,
     SEGMENTS;

package FRAME is

    function  RESOLVE_REFERENCE (OBJECT : CODE.OBJECT_REFERENCE)
                                         return STACKS.CONTROL_REFERENCE;

    function  READ (OBJECT       : CODE.OBJECT_REFERENCE;
                    LOOK_THROUGH : BOOLEAN)
                                         return STACKS.CONTROL_WORD;

    function  READ_IMPORT (OFFSET : CODE.SCOPE_DELTA)
                                         return STACKS.CONTROL_WORD;


    procedure SAVE_REGISTERS (AT_FRAME : STACKS.CONTROL_REFERENCE);

    procedure PUSH_FOR_CALL (SUBPROG : STACKS.CONTROL_WORD
                                         (OF_KIND => STACKS.SUBPROGRAM_VAR));

    procedure PUSH_FOR_REF  (SUBPROG     : STACKS.CONTROL_WORD
                                         (OF_KIND => STACKS.SUBPROGRAM_VAR);
                             NEW_SCOPE   : STACKS.NAME;
                             IN_UTILITY  : BOOLEAN := FALSE);


    procedure POP  (PARAMETER_COUNT  : CODE.EXIT_POP_COUNT;
                    RESET_DATA_TOP   : BOOLEAN := TRUE;
                    LOAD_INSTRUCTION : BOOLEAN := TRUE);


    procedure INITIALIZE_REGISTERS (FOR_MODULE : STACKS.NAME);

    procedure RESUME_REGISTERS    (FOR_MODULE  : STACKS.NAME;
                                   INNER_FRAME : STACKS.CONTROL_DISPLACEMENT);

    procedure MAKE_PARENT;
    procedure MAKE_BARREN;
```

```
    procedure PUSH_TYPE;
    procedure PUSH_DATA;


    function  REFERENCE_TOP    return CODE.OBJECT_REFERENCE;
    function  REFERENCE_INNER return STACKS.CONTROL_REFERENCE;

    function  OUTER          return STACKS.NAME;
    function  ENCLOSING      return STACKS.CONTROL_REFERENCE;
    function  INNER          return STACKS.CONTROL_DISPLACEMENT;
    function  CONTROL_PRED return STACKS.CONTROL_DISPLACEMENT;
    function  LEX_LEVEL      return CODE.LEXICAL_LEVEL;

    function  FOR_TYPE       return STACKS.TYPE_DISPLACEMENT;
    function  FOR_DATA       return STACKS.DATA_REFERENCE;

    function  BLOCK_START          return SEGMENTS.DISPLACEMENT;
    function  RETURN_ADDRESS       return SEGMENTS.ADDRESS;
    function  RETURN_INSTRUCTION return CODE.INSTRUCTION;

    function  IS_MARKED            return BOOLEAN;
    function  HAS_CHILDREN         return BOOLEAN;
    function  EXECUTING_ACCEPT     return BOOLEAN;
    function  IN_UTILITY           return BOOLEAN;

    procedure IMPORT_SPACE (NAME     : out STACKS.IMPORT_NAME;
                            IS_VALID : out BOOLEAN);

end FRAME;
```

```
FFFFF  RRRR     AAA    M   M  EEEEE  BBBB
F      R   R   A   A   MM MM  E      B   B
F      R   R   A   A   M M M  E      B   B
FFF    RRRR    A   A   M   M  EEEE   BBBB
F      R R     AAAAA   M   M  E      B   B
F      R  R    A   A   M   M  E      B   B
F      R   R   A   A   M   M  EEEEE  BBBB


 AAA   DDDD     AAA          4  4    333   4  4
A   A  D   D   A   A         4  4   3   3  4  4
A   A  D   D   A   A         4  4       3  4  4
A   A  D   D   A   A         44444     3   44444
AAAAA  D   D   AAAAA            4       3      4
A   A  D   D   A   A     ..    4   3   3      4
A   A  DDDD    A   A     ..    4    333       4


*START* Job FRAMEV Req #367 for DHB     Date  1-Mar-82 14:40:14 Monitor: Rational
File PS:<SIM.CPU>FRAMEB.ADA.434,  created:  26-Feb-82 21:49:33
        printed:  1-Mar-82 14:40:30
Job parameters: Request created: 1-Mar-82 14:40:14   Page limit:36   Forms:NORMAL
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:AS
```

```
with TASKS,
     SEGMENTS,
     CONVENTION,
     TYP,
     DATA,
     IMPORT,
     CONTROL,
     COPY,
     RAISES,
     ALLOCATOR,
     INSTRUCTION_UNIT;

package body FRAME is

    use STACKS;

    package REGISTER is

        IS_MARKED               : BOOLEAN;

        OUTER_FRAME             : STACKS.NAME;
        ENCLOSING_FRAME         : STACKS.CONTROL_REFERENCE;
        INNER_FRAME             : STACKS.CONTROL_DISPLACEMENT;
        CONTROL_PRED            : STACKS.CONTROL_DISPLACEMENT;
        CURRENT_LEX             : CODE.LEXICAL_LEVEL;

        TYPE_FRAME              : STACKS.TYPE_DISPLACEMENT;
        DATA_FRAME              : STACKS.DATA_REFERENCE;

        BLOCK_START             : SEGMENTS.DISPLACEMENT;
        RETURN_ADDRESS          : SEGMENTS.ADDRESS;
        RETURN_INSTRUCTION      : CODE.INSTRUCTION;

        EXECUTING_ACCEPT        : BOOLEAN;
        HAS_CHILDREN            : BOOLEAN;
        IN_UTILITY              : BOOLEAN;

        IMPORT_NAME             : STACKS.IMPORT_NAME;
        IMPORT_NAME_VALID       : BOOLEAN;

    end REGISTER;


    procedure PUSH_TYPE is
    begin
        REGISTER.TYPE_FRAME := TYP.TOP().OFFSET;
    end PUSH_TYPE;


    procedure PUSH_DATA is
    begin
        REGISTER.DATA_FRAME := DATA.IN_STACK.TOP ();
    end PUSH_DATA;
```

```
function REFERENCE_INNER return STACKS.CONTROL_REFERENCE is
begin
    return STACKS.CONTROL_REFERENCE'(STACK  => CONTROL.CURRENT_STACK (),
                                     OFFSET => REGISTER.INNER_FRAME);
end REFERENCE_INNER;


function REFERENCE_TOP return CODE.OBJECT_REFERENCE is
    TOP_REF : CODE.OBJECT_REFERENCE (LEVEL => REGISTER.CURRENT_LEX);
    OFFSET  : STACKS.CONTROL_DISPLACEMENT;
begin
    OFFSET  := CONTROL.CURRENT_OFFSET () - REGISTER.INNER_FRAME;
    case REGISTER.CURRENT_LEX is
        when  0 | 1 =>
            TOP_REF.SCOPE_OFFSET := CODE.SCOPE_DELTA (OFFSET);

        when others =>
            TOP_REF.FRAME_OFFSET := CODE.FRAME_DELTA (OFFSET);
    end case;
    return TOP_REF;
end REFERENCE_TOP;


procedure VALIDATE_IMPORT_NAME is
    STATIC_INFO : STACKS.CONTROL_WORD
                              (OF_KIND => STACKS.STATIC_CONNECTION);
begin
    if not REGISTER.IMPORT_NAME_VALID then
        STATIC_INFO := CONTROL.READ
                              (STACKS.CONTROL_REFERENCE'
                                  (STACK  => REGISTER.OUTER_FRAME,
                                   OFFSET => STACKS.STATIC_OFFSET));
        REGISTER.IMPORT_NAME := STATIC_INFO.MODULE_IMPORTS;
        REGISTER.IMPORT_NAME_VALID := TRUE;
    end if;
end VALIDATE_IMPORT_NAME;


procedure INVALIDATE_IMPORT_NAME is
begin
    REGISTER.IMPORT_NAME := CONVENTION.NULL_IMPORT_SPACE;
    REGISTER.IMPORT_NAME_VALID := FALSE;
end INVALIDATE_IMPORT_NAME;
```

```
function RESOLVE_REFERENCE (OBJECT : CODE.OBJECT_REFERENCE)
                                        return STACKS.CONTROL_REFERENCE is
    TARGET       : STACKS.CONTROL_REFERENCE;
    STATIC_STATE : STACKS.CONTROL_WORD
                          (OF_KIND => STACKS.ACTIVATION_STATE);
begin
    if OBJECT.LEVEL = 0  or else
            (OBJECT.LEVEL = 1 and then OBJECT.SCOPE_OFFSET < 0) then
        RAISES.SIM_ERROR ("FRAME.RESOLVE_REFERENCE");

    elsif OBJECT.LEVEL > REGISTER.CURRENT_LEX then
        RAISES.INSTRUCTION_ERROR;

    elsif OBJECT.LEVEL = 1 then
        TARGET.STACK  := REGISTER.OUTER_FRAME;
        TARGET.OFFSET := STACKS.OUTER_STATE +
                                        STACKS.CONTROL_DISPLACEMENT
                                            (INTEGER (OBJECT.SCOPE_OFFSET));

    elsif OBJECT.LEVEL = REGISTER.CURRENT_LEX then
        TARGET.STACK  := CONTROL.CURRENT_STACK ();
        TARGET.OFFSET := REGISTER.INNER_FRAME +
                                        STACKS.CONTROL_DISPLACEMENT
                                           (INTEGER (OBJECT.FRAME_OFFSET));
    else
        TARGET := REGISTER.ENCLOSING_FRAME;
        for I in reverse OBJECT.LEVEL ..
                          CODE.LEXICAL_LEVEL
                              (INTEGER (REGISTER.CURRENT_LEX) - 2) loop
            STATIC_STATE  := CONTROL.READ (TARGET);

            if STATIC_STATE.OF_KIND /= STACKS.ACTIVATION_STATE then
                RAISES.SIM_ERROR ("FRAME.RESOLVE_REFERENCE");
            else
                TARGET := STATIC_STATE.ENCLOSING_FRAME;
            end if;
        end loop;
        TARGET.OFFSET := TARGET.OFFSET +
                                        STACKS.CONTROL_DISPLACEMENT
                                            (INTEGER (OBJECT.FRAME_OFFSET));
    end if;

    return TARGET;
end RESOLVE_REFERENCE;


function READ_IMPORT (OFFSET : CODE.SCOPE_DELTA)
                                        return STACKS.CONTROL_WORD is
begin
    VALIDATE_IMPORT_NAME;
    return IMPORT.READ (IN_SPACE  => REGISTER.IMPORT_NAME,
                        AT_OFFSET => OFFSET);
end READ_IMPORT;
```

```
function READ (OBJECT       : CODE.OBJECT_REFERENCE;
               LOOK_THROUGH : BOOLEAN)
                                        return STACKS.CONTROL_WORD is
    OBJECT_VALUE : STACKS.CONTROL_WORD;
begin
    if OBJECT.LEVEL /= 0 then
        OBJECT_VALUE := CONTROL.READ (RESOLVE_REFERENCE (OBJECT));
    else
        OBJECT_VALUE := READ_IMPORT  (OBJECT.SCOPE_OFFSET);
    end if;

    if LOOK_THROUGH and then
          OBJECT_VALUE.OF_KIND = STACKS.REFERENCE_VAR and then
                OBJECT_VALUE.REF_SUBPROG = CODE.NOT_SUBPROG then
        OBJECT_VALUE := CONTROL.READ (OBJECT_VALUE.REF_TO_VALUE);
    end if;

    return OBJECT_VALUE;
end READ;
```

```
procedure WRITE_STATE (AT_FRAME : STACKS.CONTROL_REFERENCE) is
begin
    CONTROL.WRITE
            (TARGET => AT_FRAME,
             VALUE  => STACKS.CONTROL_WORD'
                    (OF_KIND          => STACKS.ACTIVATION_STATE,
                     OUTER_FRAME      => REGISTER.OUTER_FRAME,
                     ENCLOSING_FRAME  => REGISTER.ENCLOSING_FRAME,
                     CONTROL_PRED     => REGISTER.CONTROL_PRED,
                     CURRENT_LEX      => REGISTER.CURRENT_LEX,
                     RETURN_INSERT    => REGISTER.RETURN_INSTRUCTION,
                     IN_ACCEPT_BODY   => REGISTER.EXECUTING_ACCEPT,
                     IN_UTILITY       => REGISTER.IN_UTILITY,
                     HAS_CHILDREN     => REGISTER.HAS_CHILDREN));
end WRITE_STATE;


procedure WRITE_LINK  (AT_FRAME : STACKS.CONTROL_REFERENCE) is
begin
    CONTROL.WRITE
            (TARGET => STACKS.CONTROL_REFERENCE'
                           (STACK  => AT_FRAME.STACK,
                            OFFSET => AT_FRAME.OFFSET + 1),
             VALUE  => STACKS.CONTROL_WORD'
                    (OF_KIND          => STACKS.ACTIVATION_LINK,
                     TYPE_FRAME       => REGISTER.TYPE_FRAME,
                     DATA_FRAME       => REGISTER.DATA_FRAME,
                     BLOCK_START      => REGISTER.BLOCK_START,
                     RETURN_ADDRESS   => REGISTER.RETURN_ADDRESS,
                     FRAME_MARKED     => REGISTER.IS_MARKED));
end WRITE_LINK;


procedure SAVE_REGISTERS (AT_FRAME : STACKS.CONTROL_REFERENCE) is
begin
    WRITE_STATE (AT_FRAME);
    WRITE_LINK  (AT_FRAME);
end SAVE_REGISTERS;
```

```
procedure MARK_STACK (FOR_SUBPROG : STACKS.CONTROL_WORD
                                    (OF_KIND => STACKS.SUBPROGRAM_VAR)) is
begin
    if FOR_SUBPROG.SUBPROG_SORT = CODE.FOR_OPERATION or else
                FOR_SUBPROG.SUBPROG_SORT = CODE.NOT_SUBPROG then
        RAISES.SIM_ERROR ("FRAME.MARK_STACK");
    end if;

    if not REGISTER.IS_MARKED then
        REGISTER.IS_MARKED := TRUE;
        SAVE_REGISTERS (AT_FRAME => REFERENCE_INNER ());
    end if;

    INSTRUCTION_UNIT.FETCH (NEXT      => REGISTER.RETURN_INSTRUCTION,
                            WITH_READ => FALSE);
    REGISTER.RETURN_ADDRESS := INSTRUCTION_UNIT.CURRENT_PROGRAM_COUNTER ();

    REGISTER.CONTROL_PRED     := REGISTER.INNER_FRAME;
    REGISTER.INNER_FRAME      := CONTROL.TOP ().OFFSET + 1;
    REGISTER.ENCLOSING_FRAME  := FOR_SUBPROG.DECLARE_FRAME;
    REGISTER.CURRENT_LEX      := FOR_SUBPROG.LEX_LEVEL;
    REGISTER.DATA_FRAME       := DATA.IN_STACK.TOP ();
    REGISTER.BLOCK_START      := FOR_SUBPROG.BLOCK_BEGIN;
    REGISTER.EXECUTING_ACCEPT := (FOR_SUBPROG.SUBPROG_SORT
                                      = CODE.FOR_ACCEPT);
    CONTROL.PUSH_FOR_FRAME;

    REGISTER.IS_MARKED := FALSE;

    INSTRUCTION_UNIT.SET_PROGRAM_COUNTER
                        (SEGMENTS.ADDRESS'
                            (SEGMENT => FOR_SUBPROG.CODE_SEGMENT,
                             OFFSET  => FOR_SUBPROG.BLOCK_BEGIN,
                             INDEX   => 5));
    INSTRUCTION_UNIT.INSERT (FOR_SUBPROG.FIRST_INSERT);
end MARK_STACK;
```

```
procedure PUSH_FOR_CALL (SUBPROG : STACKS.CONTROL_WORD
                                   (OF_KIND => STACKS.SUBPROGRAM_VAR)) is
begin
    MARK_STACK (SUBPROG);
    REGISTER.TYPE_FRAME    := TYP.ALLOCATE(1).OFFSET;
    REGISTER.HAS_CHILDREN  := FALSE;
    REGISTER.IN_UTILITY    := FALSE;
end PUSH_FOR_CALL;


procedure PUSH_FOR_REF (SUBPROG     : STACKS.CONTROL_WORD
                                      (OF_KIND => STACKS.SUBPROGRAM_VAR);
                        NEW_SCOPE   : STACKS.NAME;
                        IN_UTILITY  : BOOLEAN := FALSE) is
begin
    if SUBPROG.SUBPROG_SORT /= CODE.FOR_OPERATION then
        MARK_STACK (SUBPROG);
        REGISTER.OUTER_FRAME := NEW_SCOPE;
        INVALIDATE_IMPORT_NAME;

        if not IN_UTILITY then
            REGISTER.TYPE_FRAME    := TYP.ALLOCATE(1).OFFSET;
            REGISTER.HAS_CHILDREN  := FALSE;
        end if;
        REGISTER.IN_UTILITY := IN_UTILITY;
    else
        INSTRUCTION_UNIT.INSERT (SUBPROG.FIRST_INSERT);
    end if;
end PUSH_FOR_REF;
```

```
procedure INITIALIZE_REGISTERS (FOR_MODULE  : STACKS.NAME) is
begin
    REGISTER.IS_MARKED           := TRUE;
    REGISTER.OUTER_FRAME         := FOR_MODULE;
    REGISTER.ENCLOSING_FRAME := STACKS.CONTROL_REFERENCE'
                                        (STACK  => FOR_MODULE,
                                         OFFSET => 0);
    REGISTER.INNER_FRAME         := STACKS.OUTER_STATE;
    REGISTER.CONTROL_PRED        := 0;
    REGISTER.CURRENT_LEX         := 1;
    REGISTER.TYPE_FRAME          := 0;
    REGISTER.DATA_FRAME          := STACKS.DATA_REFERENCE'
                                        (STACK  => FOR_MODULE,
                                         OFFSET => 0);
    REGISTER.BLOCK_START         := 0;
    REGISTER.RETURN_ADDRESS      := CONVENTION.NULL_ADDRESS;
    REGISTER.RETURN_INSTRUCTION := CODE.NO_OP;
    REGISTER.HAS_CHILDREN        := TRUE;
    REGISTER.IN_UTILITY          := FALSE;
    REGISTER.EXECUTING_ACCEPT    := FALSE;
    INVALIDATE_IMPORT_NAME;
end INITIALIZE_REGISTERS;


procedure RESTORE_REGISTERS (CONTROL_STATE : STACKS.CONTROL_WORD
                                    (OF_KIND => STACKS.ACTIVATION_STATE);
                             CONTROL_LINK  : STACKS.CONTROL_WORD
                                    (OF_KIND => STACKS.ACTIVATION_LINK)) is
begin
    REGISTER.OUTER_FRAME          := CONTROL_STATE.OUTER_FRAME;
    REGISTER.ENCLOSING_FRAME      := CONTROL_STATE.ENCLOSING_FRAME;
    REGISTER.CONTROL_PRED         := CONTROL_STATE.CONTROL_PRED;
    REGISTER.RETURN_INSTRUCTION   := CONTROL_STATE.RETURN_INSERT;
    REGISTER.CURRENT_LEX          := CONTROL_STATE.CURRENT_LEX;
    REGISTER.HAS_CHILDREN         := CONTROL_STATE.HAS_CHILDREN;
    REGISTER.IN_UTILITY           := CONTROL_STATE.IN_UTILITY;
    REGISTER.EXECUTING_ACCEPT     := CONTROL_STATE.IN_ACCEPT_BODY;

    REGISTER.TYPE_FRAME           := CONTROL_LINK.TYPE_FRAME;
    REGISTER.DATA_FRAME           := CONTROL_LINK.DATA_FRAME;
    REGISTER.BLOCK_START          := CONTROL_LINK.BLOCK_START;
    REGISTER.RETURN_ADDRESS       := CONTROL_LINK.RETURN_ADDRESS;
    REGISTER.IS_MARKED            := CONTROL_STATE.FRAME_MARKED;
end RESTORE_REGISTERS;
```

```
procedure POP (PARAMETER_COUNT   : CODE.EXIT_POP_COUNT;
               RESET_DATA_TOP    : BOOLEAN := TRUE;
               LOAD_INSTRUCTION  : BOOLEAN := TRUE) is
    BOUNDARY    : STACKS.CONTROL_DISPLACEMENT;
    STATE_WORD  : STACKS.CONTROL_WORD
                              (OF_KIND => STACKS.ACTIVATION_STATE);
    LINK_WORD   : STACKS.CONTROL_WORD
                              (OF_KIND => STACKS.ACTIVATION_LINK);
begin
    BOUNDARY := REGISTER.INNER_FRAME -
                              STACKS.CONTROL_DISPLACEMENT
                                   (INTEGER (PARAMETER_COUNT) + 1);
    if BOUNDARY > REGISTER.CONTROL_PRED then
        CONTROL.POP (DOWN_TO => BOUNDARY);
    else
        RAISES.INSTRUCTION_ERROR;
    end if;

    if not REGISTER.IN_UTILITY then
        TYP.POP (DOWN_TO => REGISTER.TYPE_FRAME);
    end if;

    if RESET_DATA_TOP then
        DATA.IN_STACK.POP (DOWN_TO => REGISTER.DATA_FRAME);
    end if;

    if LOAD_INSTRUCTION then
        INSTRUCTION_UNIT.INSERT   (NEXT => REGISTER.RETURN_INSTRUCTION);
        INSTRUCTION_UNIT.SET_PROGRAM_COUNTER (REGISTER.RETURN_ADDRESS);
    end if;

    if REGISTER.CONTROL_PRED = 0 then
        RAISES.SIM_ERROR ("FRAME.POP");

    elsif REGISTER.CONTROL_PRED = STACKS.OUTER_STATE then
        INITIALIZE_REGISTERS (FOR_MODULE  => CONTROL.CURRENT_STACK ());

    else
        REGISTER.INNER_FRAME := REGISTER.CONTROL_PRED;
        STATE_WORD := CONTROL.READ (STACKS.CONTROL_REFERENCE'
                                    (STACK  => CONTROL.CURRENT_STACK (),
                                     OFFSET => REGISTER.CONTROL_PRED));
        LINK_WORD  := CONTROL.READ (STACKS.CONTROL_REFERENCE'
                                    (STACK  => CONTROL.CURRENT_STACK (),
                                     OFFSET => REGISTER.CONTROL_PRED + 1));
        RESTORE_REGISTERS (CONTROL_STATE => STATE_WORD,
                           CONTROL_LINK  => LINK_WORD);
    end if;
end POP;
```

```
procedure RESUME_REGISTERS (FOR_MODULE   : STACKS.NAME;
                            INNER_FRAME : STACKS.CONTROL_DISPLACEMENT) is
    STATE_WORD : STACKS.CONTROL_WORD
                            (OF_KIND => STACKS.ACTIVATION_STATE);
    LINK_WORD  : STACKS.CONTROL_WORD
                            (OF_KIND => STACKS.ACTIVATION_LINK);
begin
    REGISTER.INNER_FRAME := INNER_FRAME;

    STATE_WORD := CONTROL.READ (STACKS.CONTROL_REFERENCE'
                                    (STACK  => FOR_MODULE,
                                     OFFSET => STACKS.OUTER_STATE));
    LINK_WORD  := CONTROL.READ (STACKS.CONTROL_REFERENCE'
                                    (STACK  => FOR_MODULE,
                                     OFFSET => STACKS.OUTER_LINK));
    RESTORE_REGISTERS (CONTROL_STATE => STATE_WORD,
                       CONTROL_LINK  => LINK_WORD);
    INVALIDATE_IMPORT_NAME;
end RESUME_REGISTERS;


procedure MAKE_PARENT is
begin
    if not REGISTER.HAS_CHILDREN and then REGISTER.IS_MARKED then
        REGISTER.HAS_CHILDREN := TRUE;
        WRITE_LINK (AT_FRAME => REFERENCE_INNER ());
    else
        REGISTER.HAS_CHILDREN := TRUE;
    end if;
end MAKE_PARENT;


procedure MAKE_BARREN is
begin
    if REGISTER.HAS_CHILDREN and then REGISTER.IS_MARKED then
        REGISTER.HAS_CHILDREN := FALSE;
        WRITE_LINK (AT_FRAME => REFERENCE_INNER ());
    else
        REGISTER.HAS_CHILDREN := FALSE;
    end if;
end MAKE_BARREN;
```

```
function IS_MARKED return BOOLEAN is
begin
    return REGISTER.IS_MARKED;
end IS_MARKED;


function OUTER return STACKS.NAME is
begin
    return REGISTER.OUTER_FRAME;
end OUTER;


function ENCLOSING return STACKS.CONTROL_REFERENCE is
begin
    return REGISTER.ENCLOSING_FRAME;
end ENCLOSING;


function INNER return STACKS.CONTROL_DISPLACEMENT is
begin
    return REGISTER.INNER_FRAME;
end INNER;


function CONTROL_PRED return STACKS.CONTROL_DISPLACEMENT is
begin
    return REGISTER.CONTROL_PRED;
end CONTROL_PRED;
```

```
function LEX_LEVEL return CODE.LEXICAL_LEVEL is
begin
    return REGISTER.CURRENT_LEX;
end LEX_LEVEL;


function FOR_TYPE return STACKS.TYPE_DISPLACEMENT is
begin
    return REGISTER.TYPE_FRAME;
end FOR_TYPE;


function FOR_DATA return STACKS.DATA_REFERENCE is
begin
    return REGISTER.DATA_FRAME;
end FOR_DATA;


function BLOCK_START return SEGMENTS.DISPLACEMENT is
begin
    return REGISTER.BLOCK_START;
end BLOCK_START;


function RETURN_ADDRESS return SEGMENTS.ADDRESS is
begin
    return REGISTER.RETURN_ADDRESS;
end RETURN_ADDRESS;


function RETURN_INSTRUCTION return CODE.INSTRUCTION is
begin
    return REGISTER.RETURN_INSTRUCTION;
end RETURN_INSTRUCTION;


function EXECUTING_ACCEPT return BOOLEAN is
begin
    return REGISTER.EXECUTING_ACCEPT;
end EXECUTING_ACCEPT;


function HAS_CHILDREN return BOOLEAN is
begin
    return REGISTER.HAS_CHILDREN;
end HAS_CHILDREN;


function IN_UTILITY return BOOLEAN is
begin
    return REGISTER.IN_UTILITY;
end IN_UTILITY;
```

```
    procedure IMPORT_SPACE (NAME     : out STACKS.IMPORT_NAME;
                            IS_VALID : out BOOLEAN) is
    begin
        IS_VALID := REGISTER.IMPORT_NAME_VALID;
        if IS_VALID then
            NAME := REGISTER.IMPORT_NAME;
        else
            NAME := CONVENTION.NULL_IMPORT_SPACE;
        end if;
    end IMPORT_SPACE;


    package body REGISTER is end REGISTER;

end FRAME;.
```