```
M    M    OOO    V    V  EEEEEE  BBBB
MM  MM   O   O   V    V  E       B   B
M M M    O   O   V    V  E       B   B
M  M     O   O   V    V  EEEE    BBBB
M   M    O   O   V    V  E       B   B
M    M   O   O    V  V   E       B   B
M    M    OOO      V    EEEEEE  BBBB


 AAA    DDDD     AAA              1      OOO     999
A   A   D   D   A   A            11     O   O  9   9
A   A   D   D   A   A             1     O  OO  9   9
A   A   D   D   A   A             1     O O O   9999
AAAAA   D   D   AAAAA             1     OO  O      9
A   A   D   D   A   A     ..      1     O   O      9
A   A   DDDD    A   A     ..     111     OOO     999


*START* Job MOVEB Req #365 for DHB     Date  1-Mar-82 14:35:52 Monitor: Rational
File PS:<SIM.CPU>MOVEB.ADA.109, created: 24-Feb-82  1:11:29
        printed:  1-Mar-82 14:35:52
Job parameters: Request created: 1-Mar-82 14:35:51   Page limit:18   Forms:NORMAL
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:AS
```

```
with STACKS,
     DATA,
     CONTROL,
     OPS,
     COPY,
     FRAME,
     RAISES;

separate (EXECUTION_UNIT)
package body MOVE is

    use CODE;


    function DEREFERENCE_RENAMING (VALUE_RENAME : STACKS.CONTROL_WORD)
                                          -- (OF_KIND => STACKS.VALUE_REF)
                                                return STACKS.CONTROL_WORD is
        VALUE_TYPE : STACKS.TYPE_LINK;   -- (OF_KIND => STACKS.VALUE_VAR)
        VALUE_SIZE : STACKS.VALUE_SIZE;
        VALUE_ITEM : STACKS.VALUE;
    begin
        VALUE_TYPE := OPS.VALUE_TYPE_FOR_REF (VALUE_RENAME.REF_TYPE);

        case VALUE_TYPE.OF_KIND is
            when STACKS.DISCRETE_VAR =>
                VALUE_SIZE := OPS.DISCRETE_VAR_SIZE (VALUE_TYPE);
                VALUE_ITEM := DATA.READ_DISCRETE
                                          (SIZE      => VALUE_SIZE,
                                           IS_SIGNED => VALUE_TYPE.IS_SIGNED,
                                           LOCATION  => VALUE_RENAME.REF_ITEM);

            when STACKS.FIXED_SIZE_VAR =>
                VALUE_ITEM := DATA.READ_VALUE
                                          (OF_KIND   => VALUE_TYPE.OF_KIND,
                                           LOCATION  => VALUE_RENAME.REF_ITEM);

            when others =>
                RAISES.SIM_ERROR ("MOVE.DEFERENCE_RENAMING");
        end case;

        return OPS.MAKE_VALUE_VAR (VALUE_ITEM, VALUE_TYPE);
    end DEREFERENCE_RENAMING;
```

```
procedure VALUE (OBJECT : CODE.OBJECT_REFERENCE) is
    VAR_COPY    : STACKS.CONTROL_WORD;
    REF_VALUE   : STACKS.VALUE;
begin
    VAR_COPY  := FRAME.READ (OBJECT, LOOK_THROUGH => TRUE);

    case VAR_COPY.OF_KIND is
        when STACKS.TYPED_VAR =>
            OPS.MAKE_LOCAL (VAR_COPY);

        when STACKS.REFERENCE_VAR  =>
            if VAR_COPY.REF_SUBPROG /= CODE.NOT_SUBPROG then
                RAISES.INSTRUCTION_ERROR;
            else
                RAISES.SIM_ERROR ("MOVE.VALUE");
            end if;

        when STACKS.SUBPROGRAM_VAR =>
            if OBJECT.LEVEL /= 0 then
                RAISES.INSTRUCTION_ERROR;
            else
                RAISES.SIM_ERROR ("MOVE.VALUE");
            end if;

        when STACKS.SELECT_VAR    | STACKS.EXCEPTION_VAR |
             STACKS.DELETION_KEY  | STACKS.BREAKPOINT_KEY =>
            null;

        when others =>
            RAISES.INSTRUCTION_ERROR;
    end case;

    if VAR_COPY.OF_KIND in STACKS.VALUE_REF'FIRST
                                .. STACKS.VALUE_REF'LAST then
        VAR_COPY := DEREFERENCE_RENAMING (VAR_COPY);
    end if;

    CONTROL.PUSH (VAR_COPY);
end VALUE;
```

```
function DEREFERENCE (REF_TYPE : STACKS.TYPE_LINK)
                                                    return STACKS.TYPE_LINK is
begin
    case REF_TYPE.OF_KIND is
        when STACKS.DISCRETE_REF =>
            return STACKS.TYPE_LINK'
                    (OF_KIND    => STACKS.DISCRETE_VAR,
                     PATH       => REF_TYPE.PATH,
                     FOR_TYPE   => REF_TYPE.FOR_TYPE,
                     FOR_VAR    => REF_TYPE.FOR_VAR,
                     IS_SIGNED  => REF_TYPE.IS_SIGNED);

        when STACKS.FLOAT_REF =>
            return STACKS.TYPE_LINK'
                    (OF_KIND    => STACKS.FLOAT_VAR,
                     PATH       => REF_TYPE.PATH,
                     FOR_TYPE   => REF_TYPE.FOR_TYPE,
                     FOR_VAR    => REF_TYPE.FOR_VAR);

        when STACKS.ACCESS_REF =>
            return STACKS.TYPE_LINK'
                    (OF_KIND    => STACKS.ACCESS_VAR,
                     PATH       => REF_TYPE.PATH,
                     FOR_TYPE   => REF_TYPE.FOR_TYPE,
                     FOR_VAR    => REF_TYPE.FOR_VAR);

        when STACKS.MODULE_REF =>
            return STACKS.TYPE_LINK'
                    (OF_KIND    => STACKS.MODULE_VAR,
                     PATH       => REF_TYPE.PATH,
                     FOR_TYPE   => REF_TYPE.FOR_TYPE,
                     FOR_VAR    => REF_TYPE.FOR_VAR);

        when others =>
            RAISES.SIM_ERROR ("MOVE.DEREFERENCE");
    end case;
end DEREFERENCE;
```

```
procedure RESULT (OBJECT : CODE.OBJECT_REFERENCE) is
    OLD_VALUE  : STACKS.CONTROL_WORD;
    IMPORT_REF : STACKS.CONTROL_WORD;
    NEW_VALUE  : STACKS.CONTROL_WORD;
    TARGET     : STACKS.CONTROL_REFERENCE;
begin
    OLD_VALUE  := FRAME.READ (OBJECT, LOOK_THROUGH => TRUE);
    NEW_VALUE  := CONTROL.READ (CONTROL.TOP ());

    case OLD_VALUE.OF_KIND is
        when STACKS.VALUE_VAR =>
            if OBJECT.LEVEL = 0 then
                IMPORT_REF := FRAME.READ_IMPORT (OBJECT.SCOPE_OFFSET);

                if IMPORT_REF.OF_KIND = STACKS.REFERENCE_VAR then
                    TARGET := IMPORT_REF.REF_TO_VALUE;
                else
                    RAISES.INSTRUCTION_ERROR;
                end if;
            else
                TARGET := FRAME.RESOLVE_REFERENCE (OBJECT);
            end if;
            COPY.VALUE (NEW_VALUE => NEW_VALUE,
                        OLD_VALUE => OLD_VALUE,
                        TARGET    => TARGET);

        when STACKS.VALUE_REF =>
            COPY.VALUE (NEW_VALUE => NEW_VALUE,
                        OLD_TYPE  => DEREFERENCE (OLD_VALUE.REF_TYPE),
                        OLD_REF   => OLD_VALUE.REF_ITEM);

        when STACKS.STRUCTURE_VAR =>
            COPY.VALUE (NEW_VALUE => NEW_VALUE,
                        OLD_TYPE  => OLD_VALUE.REF_TYPE,
                        OLD_REF   => OLD_VALUE.REF_ITEM);

        when STACKS.DELETION_KEY =>
            if NEW_VALUE.OF_KIND in STACKS.VARIABLE'FIRST
                                    .. STACKS.VARIABLE'LAST then

                if OLD_VALUE.SETS_VISIBILITY and then
                        NEW_VALUE.OF_KIND in STACKS.TYPED_VAR'FIRST
                                          .. STACKS.TYPED_VAR'LAST then
                    OPS.MAKE_VISIBLE (NEW_VALUE);
                end if;
                CONTROL.WRITE (TARGET => FRAME.RESOLVE_REFERENCE (OBJECT),
                               VALUE  => NEW_VALUE);
            else
                RAISES.INSTRUCTION_ERROR;
            end if;

        when others =>
            RAISES.INSTRUCTION_ERROR;
    end case;

    CONTROL.POP (DOWN_TO => CONTROL.TOP_MINUS_ONE().OFFSET);
end RESULT;
```

```
    procedure REFERENCE (OBJECT : CODE.OBJECT_REFERENCE) is
        REF_WORD : STACKS.CONTROL_WORD;
        REF_SITE : STACKS.CONTROL_REFERENCE;
        NEW_REF  : STACKS.CONTROL_WORD (OF_KIND => STACKS.REFERENCE_VAR);
    begin
        REF_WORD := FRAME.READ (OBJECT, LOOK_THROUGH => FALSE);

        case REF_WORD.OF_KIND is
            when STACKS.VALUE_VAR | STACKS.SUBPROGRAM_VAR |
                 STACKS.ENTRY_VAR =>
                if OBJECT.LEVEL /= 0 then
                    NEW_REF.REF_TO_VALUE := FRAME.RESOLVE_REFERENCE (OBJECT);
                    NEW_REF.REF_IN_SCOPE := FRAME.OUTER ();
                    if REF_WORD.OF_KIND = STACKS.SUBPROGRAM_VAR then
                        NEW_REF.REF_SUBPROG := REF_WORD.SUBPROG_SORT;
                    else
                        NEW_REF.REF_SUBPROG := CODE.NOT_SUBPROG;
                    end if;
                    CONTROL.PUSH (NEW_REF);
                else
                    RAISES.INSTRUCTION_ERROR;
                end if;

            when STACKS.INDIRECT_VAR | STACKS.REFERENCE_VAR =>
                CONTROL.PUSH (REF_WORD);

            when others =>
                RAISES.INSTRUCTION_ERROR;
        end case;
    end REFERENCE;

end MOVE;
```