```
with CODE,
     STACKS,
     ADDRESS_SPACE;

package CONTROL is

     function  READ  (TARGET : STACKS.CONTROL_REFERENCE)
                                          return  STACKS.CONTROL_QUAD;

     procedure WRITE (TARGET : STACKS.CONTROL_REFERENCE;
                      VALUE  : STACKS.CONTROL_QUAD);

     procedure PUSH (WORD    : STACKS.CONTROL_QUAD);

     function  POP  return   STACKS.CONTROL_QUAD;


     procedure INITIALIZE (NEW_MODULE   : STACKS.NAME;
                           DECLARER     : STACKS.NAME;
                           MODULE_TYPE  : STACKS.TYPE_REFERENCE);

     procedure POP    (DOWN_TO : STACKS.CONTROL_DISPLACEMENT);

     procedure RESUME (MODULE  : STACKS.NAME;
                       TOP     : STACKS.CONTROL_DISPLACEMENT;
                       EXTENT  : STACKS.CONTROL_DISPLACEMENT);

     function  TOP           return STACKS.CONTROL_REFERENCE;
     function  EXTENT        return STACKS.CONTROL_DISPLACEMENT;

     function  TOP_PLUS_ONE return STACKS.CONTROL_REFERENCE;
```

```
    PAGE_SIZE      : constant INTEGER :=  16;
    PAGE_COUNT     : constant INTEGER := 100;
    SEGMENT_COUNT  : constant INTEGER :=  20;

    CON_EXTENSION : constant STRING (1 .. 3) := "CON";

    package MEMORY is new ADDRESS_SPACE
                          (WORD                => STACKS.CONTROL_QUAD,
                           SEGMENT_ID          => STACKS.ID,
                           DISPLACEMENT        => STACKS.CONTROL_DISPLACEMENT,
                           PAGE_SIZE           => PAGE_SIZE,
                           NUMBER_OF_PAGES     => PAGE_COUNT,
                           NUMBER_OF_SEGMENTS  => SEGMENT_COUNT);

end CONTROL;
```

```
with PROCESSORS,
     CONVENTION;

package body CONTROL is

     use STACKS;


     PAGE_EXTENT : constant STACKS.CONTROL_DISPLACEMENT
                                        := STACKS.CONTROL_DISPLACEMENT(PAGE_SIZE);

     package REGISTER is

          TOP    : STACKS.CONTROL_REFERENCE;
          EXTENT : STACKS.CONTROL_DISPLACEMENT;

     end REGISTER;
```

```
    function  READ (TARGET : STACKS.CONTROL_REFERENCE)
                                          return STACKS.CONTROL_QUAD is
    begin
        if PROCESSORS.LOCAL (TARGET.STACK.PROCESSOR) then
            return MEMORY.READ (MEMORY.ADDRESS'
                                          (SEGMENT => TARGET.STACK.NUMBER,
                                           OFFSET  => TARGET.OFFSET));
--      else
--          return INTERPROCESSOR.CONTROL.READ (TARGET);
        end if;
    end READ;


    function POP return STACKS.CONTROL_QUAD is
        TEMP : STACKS.CONTROL_QUAD;
    begin
        TEMP := READ (REGISTER.TOP);
        REGISTER.TOP.OFFSET := REGISTER.TOP.OFFSET - 1;
        return TEMP;
    end POP;


    procedure WRITE  (TARGET : STACKS.CONTROL_REFERENCE;
                      VALUE  : STACKS.CONTROL_QUAD) is
    begin
        if PROCESSORS.LOCAL (TARGET.STACK.PROCESSOR) then
            if TARGET.OFFSET >= REGISTER.EXTENT then
                MEMORY.ALLOCATE_NEW_PAGE (TARGET => MEMORY.ADDRESS'
                                                (SEGMENT => TARGET.STACK.NUMBER,
                                                 OFFSET  => REGISTER.EXTENT));
                REGISTER.EXTENT := REGISTER.EXTENT + PAGE_EXTENT;
            end if;

            MEMORY.WRITE (TARGET => MEMORY.ADDRESS'
                                          (SEGMENT => TARGET.STACK.NUMBER,
                                           OFFSET  => TARGET.OFFSET),
                          VALUE  => VALUE);
--      else
--          INTERPROCESSOR.CONTROL.WRITE (TARGET);
        end if;
    end WRITE;


    procedure PUSH (WORD : STACKS.CONTROL_QUAD) is
    begin
        REGISTER.TOP.OFFSET := REGISTER.TOP.OFFSET + 1;
        WRITE (REGISTER.TOP, WORD);
    end PUSH;
```

```
procedure INITIALIZE (NEW_MODULE    : STACKS.NAME;
                      DECLARER      : STACKS.NAME;
                      MODULE_TYPE   : STACKS.TYPE_REFERENCE) is
begin
    MEMORY.CREATE (NEW_MODULE.NUMBER, CON_EXTENSION);
    MEMORY.ALLOCATE_NEW_PAGE (TARGET => MEMORY.ADDRESS'
                                        (SEGMENT => NEW_MODULE.NUMBER,
                                         OFFSET  => 0));
    REGISTER.EXTENT := PAGE_EXTENT;
    REGISTER.TOP    := STACKS.CONTROL_REFERENCE'(STACK  => NEW_MODULE,
                                                 OFFSET => 0);

    WRITE (REGISTER.TOP, CONVENTION.INITIAL_STATE);
    PUSH  (CONVENTION.INITIAL_ALLOCATION);
    PUSH  (CONVENTION.INITIAL_PROGRAM);
    PUSH  (STACKS.CONTROL_QUAD'
                        (OF_KIND           => STACKS.DEPENDENCE_LINK,
                         DECLARER          => DECLARER,
                         MODULE_TYPE       => MODULE_TYPE,
                         DEPEND_FRAME      => CONVENTION.NULL_TYPE_REF,
                         END_VISIBLE_PART  => 0));
    PUSH  (STACKS.CONTROL_QUAD'
                        (OF_KIND          => STACKS.ACCEPT_LINK,
                         IN_RENDEZVOUS    => CONVENTION.NULL_STACK,
                         ACCEPT_FRAME     => 0,
                         ACCEPT_ENTRY     => 0,
                         FAMILY_INDEX     => STACKS.VALUE'
                                             (OF_KIND => STACKS.INTEGER_VAR,
                                              INT_VAL => 0),
                         RENDEZVOUS_REF   => 0,
                         PRIORITY         => 1));
    PUSH  (CONVENTION.EMPTY_STATE);
    PUSH  (CONVENTION.EMPTY_LINK);
end INITIALIZE;
```

```
    procedure POP (DOWN_TO : STACKS.CONTROL_DISPLACEMENT) is
    begin
        REGISTER.TOP.OFFSET := DOWN_TO;
    end POP;


    procedure RESUME (MODULE : STACKS.NAME;
                      TOP    : STACKS.CONTROL_DISPLACEMENT;
                      EXTENT : STACKS.CONTROL_DISPLACEMENT) is
    begin
        REGISTER.TOP    := STACKS.CONTROL_REFERENCE'
                                        (STACK  => MODULE,
                                         OFFSET => TOP);
        REGISTER.EXTENT := EXTENT;
    end RESUME;


    function  TOP return STACKS.CONTROL_REFERENCE is
    begin
        return REGISTER.TOP;
    end TOP;

    function  EXTENT return STACKS.CONTROL_DISPLACEMENT is
    begin
        return REGISTER.EXTENT;
    end EXTENT;

    function TOP_PLUS_ONE return STACKS.CONTROL_REFERENCE is
    begin
        return STACKS.CONTROL_REFERENCE'
                        (STACK  => REGISTER.TOP.STACK,
                         OFFSET => REGISTER.TOP.OFFSET + 1);
    end TOP_PLUS_ONE;

    package body REGISTER is end REGISTER;

end CONTROL;
```