```
with CODE;

package INSTRUCTION_UNIT is

    procedure FETCH  (NEXT : out CODE.INSTRUCTION;
                      WITH_READ : BOOLEAN := TRUE);

    procedure SET_PROGRAM_COUNTER (LOCATION : CODE.ADDRESS);

    procedure INSERT (NEXT : CODE.INSTRUCTION);

    procedure CLEAR_INSERTION;

    procedure SET_BREAKPOINT (AT_PC : CODE.ADDRESS);

    function  CURRENT_INSTRUCTION_BUFFER return CODE.QUAD_WORD;

    function  CURRENT_PROGRAM_COUNTER    return CODE.ADDRESS;

    function  CURRENT_INSERT return CODE.INSTRUCTION;

    function  NEXT_INSTRUCTION return CODE.INSTRUCTION;

    BREAK_OCCURRED : exception;

end INSTRUCTION_UNIT;
```

```
with PROGRAM;

package body INSTRUCTION_UNIT is

    NO_OP_BUFFER          : constant CODE.QUAD_WORD := (others => CODE.NO_OP);
    NO_ADDRESS            : constant CODE.ADDRESS   := CODE.ADDRESS'(0, 0, 0);

    CURRENT_PC            : CODE.ADDRESS   := NO_ADDRESS;
    INSTRUCTION_BUFFER    : CODE.QUAD_WORD := NO_OP_BUFFER;

    INSERTED_INSTRUCTION  : CODE.INSTRUCTION := CODE.NO_OP;
    INSERT_AVAILABLE      : BOOLEAN  := FALSE;

    BREAKPOINT            : CODE.ADDRESS := NO_ADDRESS;
    BREAKPOINT_SET        : BOOLEAN := FALSE;

    procedure FILL_BUFFER is
    begin
        INSTRUCTION_BUFFER := PROGRAM.MEMORY.READ
                                  (PROGRAM.MEMORY.ADDRESS'
                                       (SEGMENT => CURRENT_PC.SEGMENT,
                                        OFFSET  => CURRENT_PC.QUAD_WORD));
    exception
        when PROGRAM.MEMORY.ADDRESS_NOT_WRITTEN =>
            INSTRUCTION_BUFFER := NO_OP_BUFFER;
    end FILL_BUFFER;


    procedure FETCH (NEXT       : out CODE.INSTRUCTION;
                     WITH_READ : BOOLEAN := TRUE) is
    begin
        if INSERT_AVAILABLE then
            NEXT := INSERTED_INSTRUCTION;
            INSERTED_INSTRUCTION := CODE.NO_OP;
            INSERT_AVAILABLE     := FALSE;
        else
            if BREAKPOINT_SET and then BREAKPOINT = CURRENT_PC then
                BREAKPOINT_SET := FALSE;
                raise BREAK_OCCURRED;
            end if;
            NEXT := INSTRUCTION_BUFFER(CURRENT_PC.HALF_WORD);
            if  CURRENT_PC.HALF_WORD <  CODE.HALF_WORD_INDEX'LAST then
                CURRENT_PC.HALF_WORD :=
                        CODE.HALF_WORD_INDEX'SUCC (CURRENT_PC.HALF_WORD);
            else
                CURRENT_PC.QUAD_WORD :=
                        CODE.SEGMENT_INDEX'SUCC (CURRENT_PC.QUAD_WORD);
                CURRENT_PC.HALF_WORD := CODE.HALF_WORD_INDEX'FIRST;

                if WITH_READ then
                   FILL_BUFFER;
                end if;
            end if;
        end if;
    end FETCH;
```

```
procedure SET_PROGRAM_COUNTER (LOCATION : CODE.ADDRESS) is
begin
    CURRENT_PC := LOCATION;
    FILL_BUFFER;
end SET_PROGRAM_COUNTER;


procedure INSERT (NEXT : CODE.INSTRUCTION) is
begin
    INSERTED_INSTRUCTION := NEXT;
    INSERT_AVAILABLE     := TRUE;
end INSERT;


procedure CLEAR_INSERTION is
begin
    INSERT_AVAILABLE     := FALSE;
    INSERTED_INSTRUCTION := CODE.NO_OP;
end CLEAR_INSERTION;


procedure SET_BREAKPOINT (AT_PC : CODE.ADDRESS) is
begin
    BREAKPOINT := AT_PC;
    BREAKPOINT_SET := TRUE;
end SET_BREAKPOINT;
```

```
    function CURRENT_PROGRAM_COUNTER return CODE.ADDRESS is
    begin
        return CURRENT_PC;
    end;


    function CURRENT_INSTRUCTION_BUFFER return CODE.QUAD_WORD is
    begin
        return INSTRUCTION_BUFFER;
    end;


    function CURRENT_INSERT return CODE.INSTRUCTION is
    begin
        if INSERT_AVAILABLE then
            return INSERTED_INSTRUCTION;
        else
            return CODE.NO_OP;
        end if;
    end CURRENT_INSERT;


    function NEXT_INSTRUCTION return CODE.INSTRUCTION is
    begin
        if INSERT_AVAILABLE then
            return INSERTED_INSTRUCTION;
        else
            return INSTRUCTION_BUFFER(CURRENT_PC.HALF_WORD);
        end if;
    end NEXT_INSTRUCTION;

end INSTRUCTION_UNIT;
```