

with CODE;

package EXECUTION_UNIT is

procedure DISPATCH (NEXT : CODE.INSTRUCTION);

end EXECUTION_UNIT;

end EXECUTION until

DISPATCH (NEXT : CODE INSTRUCTION)

back EXECUTION until

STOP CODE

with STACKS,
CONTROL;

package body EXECUTION_UNIT is

package MOVE is

procedure VALUE (OBJECT : CODE.OBJECT_REFERENCE);
procedure RESULT (OBJECT : CODE.OBJECT_REFERENCE);
procedure REFERENCE (OBJECT : CODE.OBJECT_REFERENCE);

procedure POP;
procedure SWAP;

end MOVE;

package PUSH_LITERAL is

procedure SHORT (VALUE : CODE.SHORT_LITERAL);
procedure BY_REFERENCE (REFERENCE : CODE.REFERENCE;
CLASS : CODE.OPERAND_CLASS);

end PUSH_LITERAL;

package JUMP is

procedure UNCONDITIONAL (LOCATION : CODE.REFERENCE);
procedure CONDITIONAL (LOCATION : CODE.REFERENCE;
CONDITION : BOOLEAN);
procedure CASE_IS (MAXIMUM : CODE.CASE_MAXIMUM);

end JUMP;

package OBJECT_OP is

procedure TYPE_DECL (CLASS : CODE.OPERAND_CLASS;
VISIBILITY : CODE.VISIBILITY;
LIMITATION : CODE.LIMITATION;
COMPLETE : CODE.DECL_STATE);
procedure VAR_DECL (CLASS : CODE.OPERAND_CLASS;
VISIBILITY : CODE.VISIBILITY;
PARAM : CODE.PARAMETERIZATION;
ALLOC : CODE.DATA_ALLOCATION);
procedure DISPATCH (OPERAND : CODE.OPERAND_CLASS;
OPERATOR : CODE.OPERATOR;
FIELD : CODE.FIELD_INDEX;
FIELD_KIND : CODE.FIELD_SORT);

end OBJECT_OP;

with STACKS,
CONTROL

package body EXECUTION_UNIT is

package MOVE is

procedure VALUE (OBJECT : CODE.OBJECT_REFERENCE);
procedure RESULT (OBJECT : CODE.OBJECT_REFERENCE);
procedure REFERENCE (OBJECT : CODE.OBJECT_REFERENCE);

procedure POP;
procedure SWAP;

end MOVE;

package PUSH_LITERAL is

procedure SHORT (VALUE : CODE.SHORT_LITERAL);

procedure BY_REFERENCE (REFERENCE : CODE.REFERENCE;
CLASS : CODE.OPERAND_CLASS);

end PUSH_LITERAL;

package JUMP is

procedure UNCONDITIONAL (LOCATION : CODE.REFERENCE);
procedure CONDITIONAL (LOCATION : CODE.REFERENCE;
CONDITION : BOOLEAN);

procedure CASE_IS (MAXIMUM : CODE.CASE_MAXIMUM);

end JUMP;

package OBJECT_OP is

procedure TYPE_DECL (CLASS : CODE.OPERAND_CLASS;
VISIBILITY : CODE.VISIBILITY;
LIMITATION : CODE.LIMITATION;
COMPLETE : CODE.DECL_STATE);

procedure VAR_DECL (CLASS : CODE.OPERAND_CLASS;
VISIBILITY : CODE.VISIBILITY;
PARAM : CODE.PARAMETERIZATION;
ALLOC : CODE.DATA_ALLOCATION);

procedure DISPATCH (OPERAND : CODE.OPERAND_CLASS;
OPERATOR : CODE.OPERATOR;
FIELD : CODE.FIELD_INDEX;
FIELD_KIND : CODE.FIELD_SORT);

end OBJECT_OP;

package SUBPROGRAM_OP is

```
procedure DECL (START      : CODE.SEGMENT_INDEX;
                VISIBILITY : CODE.VISIBILITY);

procedure CALL (SUBPROGRAM_REF : CODE.OBJECT_REFERENCE);

procedure CALL (SUBPROGRAM_VALUE : STACKS.CONTROL_QUAD
                (OF_KIND => STACKS.PROCEDURE_VAR));

procedure EXIT_PROCEDURE (PARAMS : CODE.IN_PARAMETER_COUNT);

procedure EXIT_FUNCTION  (PARAMS : CODE.IN_PARAMETER_COUNT);

procedure EXIT_ACCEPT    (PARAMS : CODE.IN_PARAMETER_COUNT);

procedure POP_PROC (LEVELS : CODE.LEXICAL_LEVEL);

procedure POP_FUNC (LEVELS : CODE.LEXICAL_LEVEL);
```

end SUBPROGRAM_OP;

package MODULE_OP is

```
procedure TYPE_DECL (MODULE_KIND : STACKS.MODULE_VAR;
                    VISIBILITY   : CODE.VISIBILITY;
                    COMPLETE     : CODE.DECL_STATE);

procedure VAR_DECL  (CLASS      : CODE.OPERAND_CLASS;
                    VISIBILITY  : CODE.VISIBILITY;
                    CHILD_ALLOC : CODE.DATA_ALLOCATION);

procedure DISPATCH  (OPERAND      : CODE.OPERAND_CLASS;
                    OPERATION    : CODE.OPERATOR;
                    FIELD        : CODE.FIELD_INDEX);
```

end MODULE_OP;

package TASKING_OP is

```
procedure DECL_CONSTRUCT (OPERAND : CODE.OPERAND_CLASS);

procedure DISPATCH  (OPERAND : CODE.OPERAND_CLASS;
                    OPERATOR : CODE.OPERATOR;
                    FIELD    : CODE.FIELD_INDEX);
```

end TASKING_OP;

end TASKING-OB:

FIELD : CODE FIELD-INDEX)

OBSERVATION : CODE OBSERVATION)

blockable DISPATCH (OBSERVATION : CODE OBSERVATION-CLASS)

blockable DESC-CONSTRAINT (OBSERVATION : CODE OBSERVATION-CLASS)

blockable TASKING-OB IS

end MODULE-OB:

FIELD : CODE FIELD-INDEX)

OBSERVATION : CODE OBSERVATION)

blockable DISPATCH (OBSERVATION : CODE OBSERVATION-CLASS)

CHILD-ASSOC : CODE DATA-ASSOCIATION)

VISIBILITY : CODE VISIBILITY)

blockable LAB-DESC (CLASS : CODE OBSERVATION-CLASS)

COMPLETE : CODE DESC-STATE)

VISIBILITY : CODE VISIBILITY)

blockable LAB-DESC (MODULE-KIND : STACKS-MODULE-LAB)

blockable MODULE-OB IS

end SUBPROCESSOR-OB:

blockable BOB-ELMS (TELEGS : CODE TEXICOM-TELEG)

blockable BOB-PROC (TELEGS : CODE TEXICOM-TELEG)

blockable EXIT-ACCEPT (PARAMS : CODE IN-PARAMETER-COUNT)

blockable EXIT-ELIMINATION (PARAMS : CODE IN-PARAMETER-COUNT)

blockable EXIT-PROCEDURE (PARAMS : CODE IN-PARAMETER-COUNT)

(OF-KIND => STACKS-PROCEDURE-LAB))

blockable CALL (SUBPROCESSOR-LABOR : STACKS-COMPILE-LOAD)

blockable CALL (SUBPROCESSOR-REF : CODE OBJECT-REFERENCE)

VISIBILITY : CODE VISIBILITY)

blockable DESC (STATE : CODE SEGMENT-INDEX)

blockable SUBPROCESSOR-OB IS

```
package SYSTEM_OP is
  procedure DISPATCH (OPERATOR : CODE.SYSTEM_OP);
end SYSTEM_OP;

package ERROR is
  procedure ILLEGAL_INSTRUCTION;
  procedure DISPATCH (OPERATOR : CODE.OPERATOR);
end ERROR;
```

end ERROR

procedure DISPATCH (OPERATOR : CODE OPERATOR)

procedure INTEGRATION

backspace ERROR 12

end SYSTEM_OB

procedure DISPATCH (OPERATOR : CODE SYSTEM_OB)

backspace SYSTEM_OB 12

```

procedure DISPATCH (NEXT : CODE.INSTRUCTION) is
begin
  case NEXT.FOR_OP is
    when CODE.VAL =>
      MOVE.VALUE (NEXT.OBJECT);

    when CODE.STO =>
      MOVE.RESULT (NEXT.OBJECT);

    when CODE.REF =>
      MOVE.REFERENCE (NEXT.OBJECT);

    when CODE.DECL =>
      OBJECT_OP.TYPE_DECL (NEXT.DECL_CLASS,
                           NEXT.DECL_VISIBLE,
                           NEXT.DECL_LIMITED,
                           NEXT.DECL_COMPLETE);

    when CODE.VAR =>
      OBJECT_OP.VAR_DECL (NEXT.VAR_CLASS,
                          NEXT.VAR_VISIBLE,
                          NEXT.VAR_PARAM,
                          NEXT.VAR_ALLOC);

    when CODE.PROC =>
      SUBPROGRAM_OP.DECL (NEXT.START, NEXT.PROC_VISIBLE);

    when CODE.SHORT_LIT =>
      PUSH_LITERAL.SHORT (NEXT.SHORT_LIT_VALUE);

    when CODE.LIT | CODE.LONG_LIT =>
      PUSH_LITERAL.BY_REFERENCE (NEXT.VAL_POINTER, NEXT.VAL_CLASS);

    when CODE.JUMP =>
      JUMP.UNCONDITIONAL (NEXT.LOCATION);

    when CODE.JUMPT =>
      JUMP.CONDITIONAL (NEXT.LOCATION, TRUE);

    when CODE.JUMPF =>
      JUMP.CONDITIONAL (NEXT.LOCATION, FALSE);

    when CODE.CASE_JUMP =>
      JUMP.CASE_IS (NEXT.CASE_MAX);
  end case;
end DISPATCH;

```

ADWB: CASE IS (NEXT: CASE MAX):
when CODE: CASE ADWB =>

ADWB: CONDITIONAL (NEXT: LOCATION: EAGE):
when CODE: ADWB =>

ADWB: CONDITIONAL (NEXT: LOCATION: TBDE):
when CODE: ADWB =>

ADWB: DISCONDITIONAL (NEXT: LOCATION):
when CODE: ADWB =>

ADWB: TIERAL: BY REFERENCE (NEXT: AVG: BOINTER: NEXT: AVG: CGASS):
when CODE: TIL: CODE: COND: TIL =>

ADWB: TIERAL: SHOWT (NEXT: SHOWT: TIL: AVGDE):
when CODE: SHOWT: TIL =>

ADWB: BOBBOBAM: OF: DESC: (NEXT: START: NEXT: BOB: AISIBGE):
when CODE: BOB =>

NEXT: AVG: AGGOS:
NEXT: AVG: BAKAM:
NEXT: AVG: AISIBGE:
OBJECT: OF: AVG: DESC: (NEXT: AVG: CGASS):
when CODE: AVG =>

NEXT: DESC: COMPLETE):
NEXT: DESC: FINISHED:
NEXT: DESC: AISIBGE:
OBJECT: OF: TYPE: DESC: (NEXT: DESC: CGASS):
when CODE: DESC =>

MOLE: REFERENCE (NEXT: OBJECT):
when CODE: REF =>

MOLE: RESORT (NEXT: OBJECT):
when CODE: RIO =>

MOLE: AVGDE (NEXT: OBJECT):
when CODE: AVG =>

case NEXT: EOB: OF: IS
region
blockage: DISPATCH (NEXT: CODE: INSTRUCTION): IS

```

when CODE.CALL      =>
    SUBPROGRAM_OP.CALL (NEXT.OBJECT);

when CODE.EXIT_PROC =>
    SUBPROGRAM_OP.EXIT_PROCEDURE (NEXT.PARAM_COUNT);

when CODE.EXIT_FUNC =>
    SUBPROGRAM_OP.EXIT_FUNCTION (NEXT.PARAM_COUNT);

when CODE.EXIT_ACCEPT =>
    SUBPROGRAM_OP.EXIT_ACCEPT (NEXT.PARAM_COUNT);

when CODE.POP_PROC  =>
    SUBPROGRAM_OP.POP_PROC (NEXT.LEVEL_COUNT);

when CODE.POP_FUNC  =>
    SUBPROGRAM_OP.POP_FUNC (NEXT.LEVEL_COUNT);

when CODE.OP        =>
    OBJECT_OP.DISPATCH (NEXT.OPERAND,
                        NEXT.OP,
                        NEXT.FIELD,
                        NEXT.FIELD_KIND);

when CODE.SYS_OP =>
    SYSTEM_OP.DISPATCH (NEXT.SYS_OP);

when CODE.HEADER | CODE.HANDLER | CODE.LOCALS | CODE.PARAMS |
    CODE.LIT_VAL | CODE.LONG_VAL | CODE.CASE_LABEL =>
    ERROR.ILLEGAL_INSTRUCTION;
end case;

end DISPATCH;

```

```

package body MOVE          is separate;
package body PUSH_LITERAL is separate;
package body JUMP          is separate;
package body OBJECT_OP     is separate;
package body SUBPROGRAM_OP is separate;
package body MODULE_OP     is separate;
package body TASKING_OP    is separate;
package body SYSTEM_OP     is separate;
package body ERROR         is separate;

```

```

end EXECUTION_UNIT;

```

END EXECUTION PHASE

```

backends procn ERROR          ts ebbelgcei
backends procn SYSTEM_OB      ts ebbelgcei
backends procn LOGGING_OB     ts ebbelgcei
backends procn MODULE_OB      ts ebbelgcei
backends procn SUBSYSTEM_OB   ts ebbelgcei
backends procn OBJECT_OB      ts ebbelgcei
backends procn QMB           ts ebbelgcei
backends procn BUSH_TIERVAT ts ebbelgcei
backends procn MDAE          ts ebbelgcei

```

END DISPATCH

END CASE

ERROR: ILLEGAL INSTRUCTION!

CODE: G11_AUG : CODE: G100_AUG : CODE: CASE_TABEL =>

WHEN CODE: HEADER : CODE: HANDLER : CODE: GOSUBS : CODE: BAKING :

SYSTEM_OB DISPATCH (NEXT SYS_OB):

WHEN CODE: SYS_OB =>

NEXT: ETEGD_KIND):

NEXT: ETEGD:

NEXT: OB:

OBJECT_OB DISPATCH (NEXT OBJECT_OB):

WHEN CODE: OB =>

SUBSYSTEM_OB SUB_EOIS (NEXT GELER_CODE):

WHEN CODE: SUB_EOIS =>

SUBSYSTEM_OB SUB_EBOS (NEXT GELER_CODE):

WHEN CODE: SUB_EBOS =>

SUBSYSTEM_OB EXIT_ACCESS (NEXT BAKING_CODE):

WHEN CODE: EXIT_ACCESS =>

SUBSYSTEM_OB EXIT_EXCEPTION (NEXT BAKING_CODE):

WHEN CODE: EXIT_EXCEPTION =>

SUBSYSTEM_OB EXIT_PROCESSED (NEXT BAKING_CODE):

WHEN CODE: EXIT_PROCESSED =>

SUBSYSTEM_OB CASE (NEXT OBJECT):

WHEN CODE: CASE =>