

P3.1 STRUCTURAL SPECIFICATION

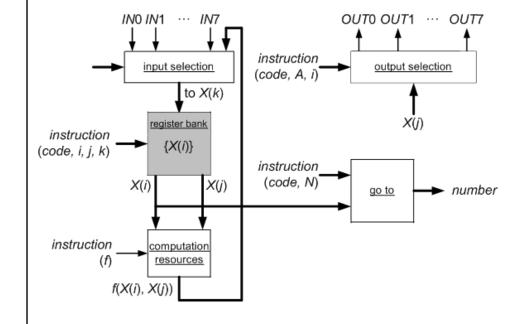
(continuation)

Jean-Pierre Deschamps

University Rovira i Virgili, Tarragona, Spain

2 BLOCK DESCRIPTION

2.3 REGISTER BANK



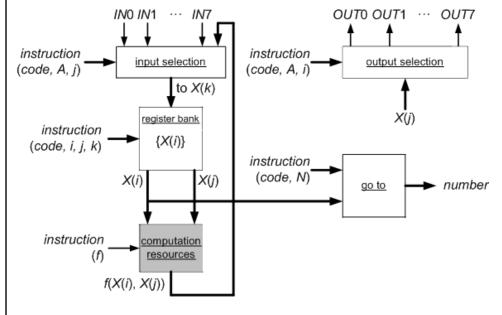
```
number := 0;
loop
case program(number) is
```

```
loop
 case program(number) is
   when (ASSIGN VALUE, k, A) =>
    X(k) := A; number := number + 1;
   when (DATA_INPUT, k, j) =>
    X(k) := IN(j); number := number + 1;
   when (DATA OUTPUT, i, j) =>
    OUT(i) := X(j); number := number + 1;
   when (OUTPUT VALUE, i, A) =>
    OUT(i) := A; number := number + 1;
   when (OPERATION, i, j, k, f) =>
    X(k) := f(X(i), X(j)); number := number + 1;
   when (JUMP, N) =>
    number := N;
   when (JUMP POS, i, N) =>
    if X(i) > 0 then number := N; else number := number + 1; end if;
   when (JUMP_NEG, i, N) =>
    if X(i) < 0 then number := N; else number := number + 1; end if;
 end case;
end loop;
```

P3.1 (register bank) reg in register bank number := 0: instruction — $\{X(i)\}$ loop de Barcelona case program(number) is **Functional specification** when (ASSIGN VALUE, k, A) => loop left out right out X(k) := A; number := number + 1; case program(number) is when (DATA INPUT, k, j) => when (ASSIGN VALUE, k, A) => X(k) := IN(j); number := number + 1; X(k) := reg in; left out := don't care; right out := don't care; when (DATA OUTPUT, i, j) => when (DATA INPUT, k, j) => OUT(i) := X(j); number := number + 1; X(k) := reg in; left out := don't care; right out := don't care; when (OUTPUT VALUE, i, A) => when (DATA OUTPUT, i, j) => OUT(i) := A; number := number + 1; left out := don't care; right out := X(j); when (OPERATION, i, j, k, f) => when (OUTPUT VALUE, i, A) => X(k) := f(X(i), X(j)); number := number + 1; left out := don't care; right out := don't care; when (JUMP, N) => when (OPERATION, i, j, k, f) =>number := N; X(k) := reg in; left out := X(i); right out := X(j);when (JUMP POS, i, N) => when (JUMP, N) => if X(i) > 0 then number := N; else number := number + 1; end if; left out := don't care; right out := don't care; when (JUMP NEG, i, N) => when (JUMP POS, i, N) => if X(i) < 0 then number := N; else number := number + 1; end if; left out := X(i); right out := don't care; end case; when (JUMP NEG, i, N) => end loop; left out := X(i); right out := don't care; end case: 3 end loop;

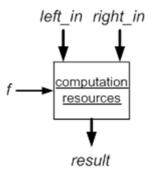
P3.1

2.4 COMPUTATION RESOURCES



```
number := 0;
                                                    Universitat Autònoma
loop
                                                       de Barcelona
 case program(number) is
   when (ASSIGN VALUE, k, A) =>
    X(k) := A; number := number + 1;
   when (DATA INPUT, k, j) =>
    X(k) := IN(j); number := number + 1;
   when (DATA OUTPUT, i, j) =>
     OUT(i) := X(j); number := number + 1;
   when (OUTPUT VALUE, i, A) =>
     OUT(i) := A; number := number + 1;
   when (OPERATION, i, j, k, f) =>
    X(k) := f(X(i), X(j)); number := number + 1;
   when (JUMP, N) =>
     number := N;
   when (JUMP POS, i, N) =>
     if X(i) > 0 then number := N; else number := number + 1; end if;
   when (JUMP_NEG, i, N) =>
    if X(i) < 0 then number := N; else number := number + 1; end if;
 end case;
end loop;
```

(computation resources)



Functional specification

```
if f = 0 then result := left_in + right_in;
else result := left_in - right_in;
end if;
```

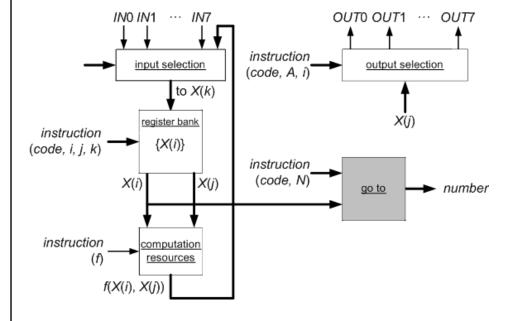
```
P3.1
```

```
de Barcelona
```

```
number := 0;
loop
 case program(number) is
   when (ASSIGN VALUE, k, A) =>
    X(k) := A; number := number + 1;
   when (DATA INPUT, k, j) =>
    X(k) := IN(j); number := number + 1;
   when (DATA OUTPUT, i, j) =>
    OUT(i) := X(j); number := number + 1;
   when (OUTPUT VALUE, i, A) =>
    OUT(i) := A; number := number + 1;
   when (OPERATION, i, j, k, f) =>
    X(k) := f(X(i), X(j)); number := number + 1;
   when (JUMP, N) =>
    number := N;
   when (JUMP POS, i, N) =>
    if X(i) > 0 then number := N; else number := number + 1; end if;
   when (JUMP_NEG, i, N) =>
    if X(i) < 0 then number := N; else number := number + 1; end if;
 end case;
end loop;
```

P3.1

2.5 GO TO



```
number := 0;
                                                    Universitat Autònoma
loop
                                                      de Barcelona
 case program(number) is
   when (ASSIGN VALUE, k, A) =>
    X(k) := A; number := number + 1;
   when (DATA_INPUT, k, j) =>
     X(k) := IN(j); number := number + 1;
   when (DATA OUTPUT, i, j) =>
     OUT(i) := X(j); number := number + 1;
   when (OUTPUT_VALUE, i, A) =>
     OUT(i) := A; number := number + 1;
   when (OPERATION, i, j, k, f) =>
    X(k) := f(X(i), X(j)); number := number + 1;
   when (JUMP, N) =>
     number := N;
   when (JUMP_POS, i, N) =>
     if X(i) > 0 then number := N; else number := number + 1; end if;
   when (JUMP_NEG, i, N) =>
     if X(i) < 0 then number := N; else number := number + 1; end if;
 end case;
end loop;
```

(go to)

```
instruction → go to number data → number = 0;
```

```
loop
  case program(number) is
  when (JUMP, N) =>
    number := N;
  when (JUMP_POS, i, N) =>
    if data > 0 then number := N;
    else number := number + 1; end if;
  when (JUMP_NEG, i, N) =>
    if data < 0 then number := N;
    else number := number + 1; end if;
  when others =>
    number := number + 1;
  end case;
end loop;
```

```
P3.1
```

```
number := 0;
loop
                                                      de Barcelona
 case program(number) is
   when (ASSIGN VALUE, k, A) =>
    X(k) := A; number := number + 1;
   when (DATA_INPUT, k, j) =>
    X(k) := IN(j); number := number + 1;
   when (DATA OUTPUT, i, j) =>
    OUT(i) := X(j); number := number + 1;
   when (OUTPUT_VALUE, i, A) =>
    OUT(i) := A; number := number + 1;
   when (OPERATION, i, j, k, f) =>
    X(k) := f(X(i), X(j)); number := number + 1;
   when (JUMP, N) =>
    number := N;
   when (JUMP POS, i, N) =>
    if X(i) > 0 then number := N; else number := number + 1; end if;
   when (JUMP NEG, i, N) =>
    if X(i) < 0 then number := N; else number := number + 1; end if;
 end case;
end loop;
```

SUMMARY

P3.1

UAB

Universitat Autònoma de Barcelona

- Structural description completed.
- Functional description of blocks:
- √ register bank,
- ✓ computation resources,
- ✓ go to block.