P2.1 FUNCTIONAL SPECIFICATION

Jean-Pierre Deschamps

University Rovira i Virgili, Tarragona, Spain

1 INSTRUCTION TYPES

UAB
Universitat Autònoma de Barcelona

- 8 instruction types are defined;
- to each instruction => code, list of parameters:

Code and list of parameters	Operation
(ASSIGN_VALUE, k, A)	*Xk* := *A*;
(DATA_INPUT, k, j)	*Xk* := *INj*;
(DATA_OUTPUT, i, j)	*OUTi* := *Xj*;
(OUTPUT_VALUE, i, A)	*OUTi* := *A*,
(OPERATION, i, j, k, f)	*Xk* := *f*(*Xi*, *Xj*);
(JUMP, N)	go to *N*;
(JUMP_POS, i, N)	if *X*i > 0 go to *N*;
(JUMP_NEG, i, N)	if *X*i < 0 go to *N*;

(*OPERATION, i, j, k, f*): *f* is an identifier of an operation (*add*, *subtract*, ···) that one of the processing resources can execute (to be defined later)

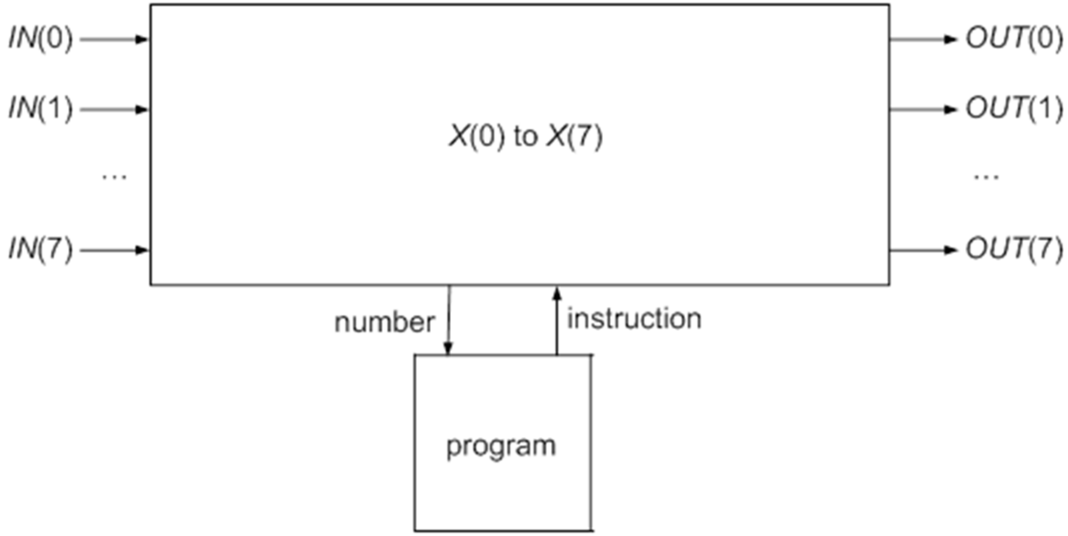
Example: temperature control system.

0: *ASSIGN_VALUE*, 5, 10	(X5 := 10)
1: *DATA_INPUT*, 0, 0	(X0 := *IN*0)
2: *DATA_INPUT*, 1, 1	(*X*1 := *IN*1)
3: *OPERATION*, 4, 0, 1, subtract	(*X*4 := *X*0 - *X*1)
4: *JUMP_NEG*, 4, 7	(if *X*4 < 0 then go to 7)
5: *JUMP_POS*, 4, 9	(if *X*4 > 0 then go to 9)
6: *JUMP*, 10	(go to 10)
7: *OUTPUT_VALUE*, 0, 1	(*OUT*0 := 1)
8: *JUMP*, 10	(go to 10)
9: *OUTPUT_VALUE*, 0, 0	(*OUT*0 := 0)
10: *DATA_INPUT*, 3, 2	(*X*3 := *IN*2)
11: *DATA_INPUT*, 2, 2	(*X*2 := *IN*2)
12: *OPERATION*, 4, 2, 3, subtract	(*X*4 := *X*2 - *X*3)
13: *OPERATION*, 4, 4, 5, subtract	(*X*4 := *X*4 - *X*5)
14: *JUMP_NEG*, 4, 11	(if *X*4 < 0 then go to 11)
15: *JUMP*, 1	(go to 1)

2 FUNCTIONAL SPECIFICATION

Digital system to be implemented:

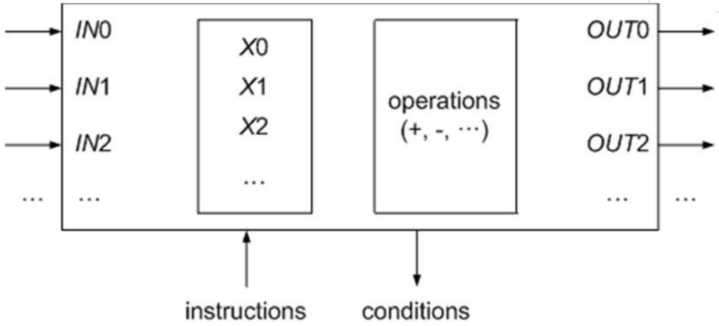
- eigth 8-bit data inputs *IN*(0), *IN*(1), ··· , *IN*(7);
- eigth 8-bit data outputs *OUT*(0), *OUT*(1), ··· , *OUT*(7);
- an input *instruction* whose value corresponds to one of the eight types (*ASSIGN_VALUE*, *i*, *A*), (*DATA_INPUT*, *i*, *j*), ··· ;
- an output *number* that selects the instruction to be executed.



Comment:

Initial specification (week 1):

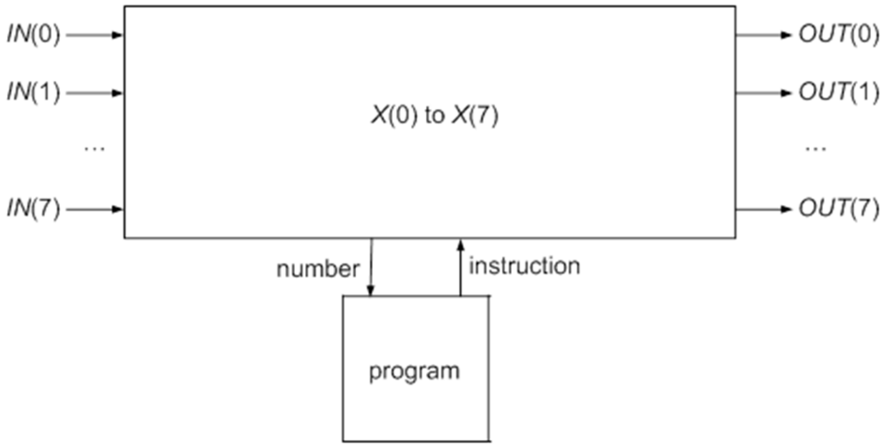
- next instruction defined by an **external** circuit / programmer in function of some internal *conditions*



Modified specification:

- next instruction number **internally** computed by the processor;
- program stored within a **memory**.

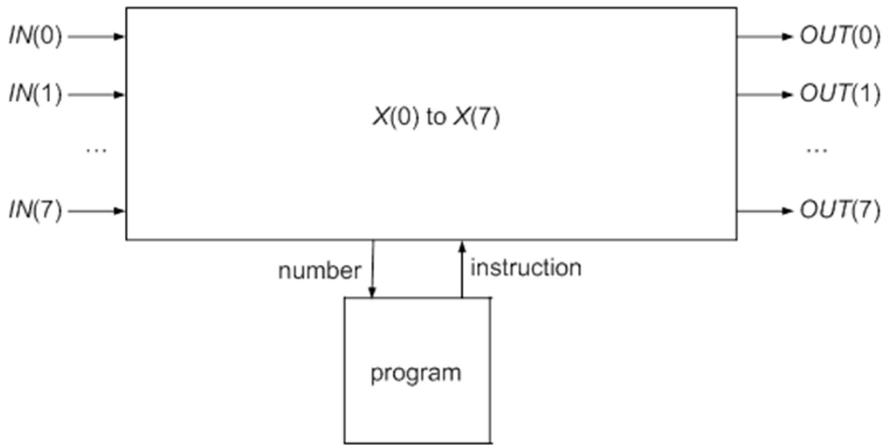
(**= von Neumann architecture**)



UAB
Universitat Autònoma de Barcelona

Algorithm: functional specification

```
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;
```



X: previously declared vector;
number of components: to be defined later;
components: 8-bit numbers.

(*Exercise*)

Assume that

- X is a 16-component vector and each component is an 8-bit number,
- the maximum number of instructions of a program is 256,
- there are only 2 different operations f.

How many different instructions can be defined?

(*Solution*)

P2.1

UAB Universitat Autònoma de Barcelona

(*ASSIGN_VALUE, k, A*)	$Xk := A;$	$16 \cdot 256 = 4096$
(*DATA_INPUT, k, j*)	$Xk := INj;$	$16 \cdot 8 = 128$
(*DATA_OUTPUT, i, j*)	$OUTi := Xj;$	$8 \cdot 16 = 128$
(*OUTPUT_VALUE, i, A*)	$OUTi := A,$	$8 \cdot 256 = 2048$
(*OPERATION, i, j, k, f*)	$Xk := f(Xi, Xj);$	$16 \cdot 16 \cdot 16 \cdot 2 = 8192$
(*JUMP, N*)	go to N;	256
(*JUMP_POS, i, N*)	if Xi > 0 go to N;	$16 \cdot 256 = 4096$
(*JUMP_NEG, i, N*)	if Xi < 0 go to N;	$16 \cdot 256 = 4096$

4096 + 128 + 128 + 2048 + 8192 + 256 + 4096 + 4096 = **23,040** different instructions.

$(2^{14} < 23{,}040 < 2^{15})$

UAB Universitat Autònoma de Barcelona

SUMMARY

- definition of the **instruction set**;
- **von Neumann** architecture;
- **functional** specification.

Universitat Autònoma de Barcelona

P2.2 STRUCTURAL SPECIFICATION

Jean-Pierre Deschamps

University Rovira i Virgili, Tarragona, Spain

1 BLOCK DIAGRAM

P2.2

```
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;
```

external inputs:

*IN*0, *IN*1, ··· , *IN*7;

instruction;

external outputs:

*OUT*0, *OUT*1, ··· , *OUT*7;

number;

internal data: *X*;

data transfers:

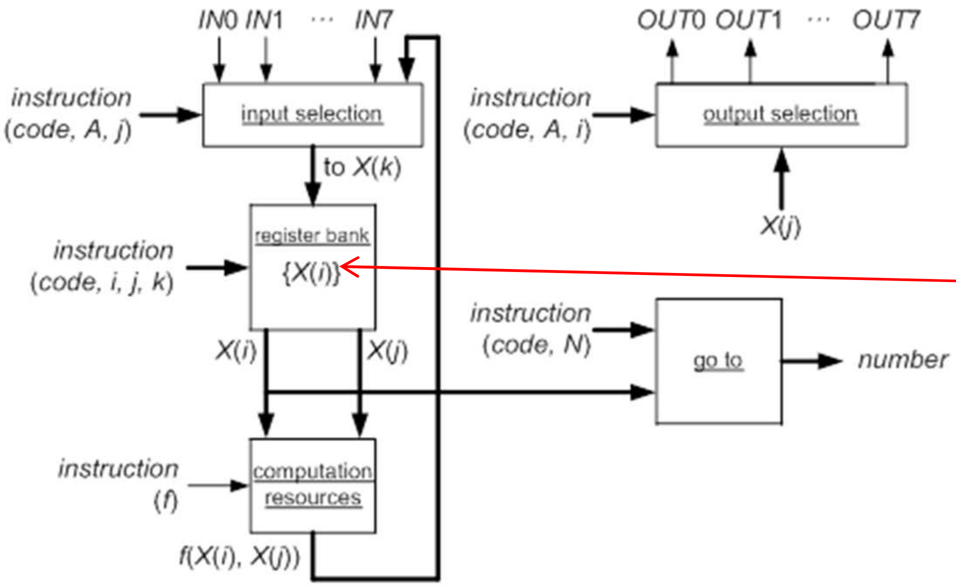
OUT(*i*) <= *X*(*j*) or *A*;

number <= (*number* + 1) or *N*;

X(*k*) <= *A* or *IN(j)* or *f*;

operations: *f*(*X*(*i*), *X*(*j*)).

INTERNAL DATA



external inputs:

*IN*0, *IN*1, ··· , *IN*7;

instruction;

external outputs:

*OUT*0, *OUT*1, ··· , *OUT*7;

number;

internal data: *X*;

data transfers:

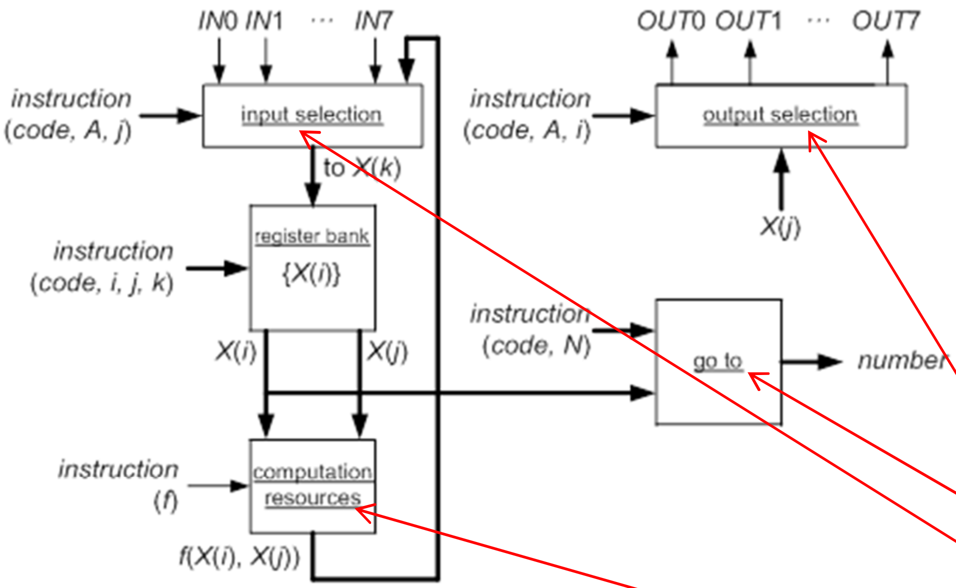
OUT(*i*) <= *X*(*j*) or *A*;

number <= (*number* + 1) or *N*;

X(*k*) <= *A* or *IN(j)* or *f*;

operations: *f*(*X*(*i*), *X*(*j*)).

DATA TRANSFERS



external inputs:
*IN*0, *IN*1, ··· , *IN*7;
instruction;

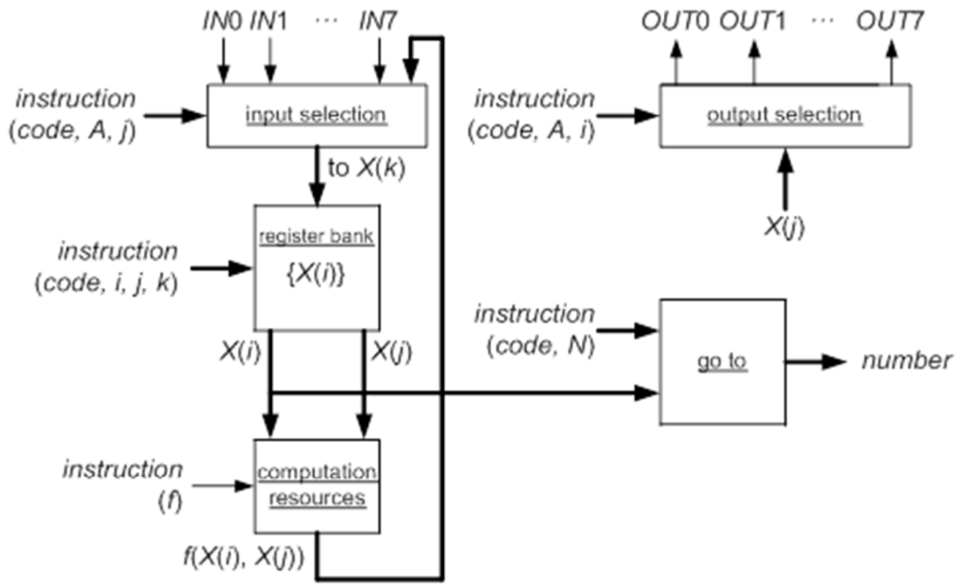
external outputs:
*OUT*0, *OUT*1, ··· , *OUT*7;
number;

internal data: *X*;

data transfers:
OUT(*i*) <= *X*(*j*) or *A*;
number <= (*number* + 1) or *N*;
X(*k*) <= *A* or *IN(j)* or *f*;

operations: *f*(*X*(*i*), *X*(*j*)).

BLOCK DIAGRAM



INSTRUCTIONS (additional specification)

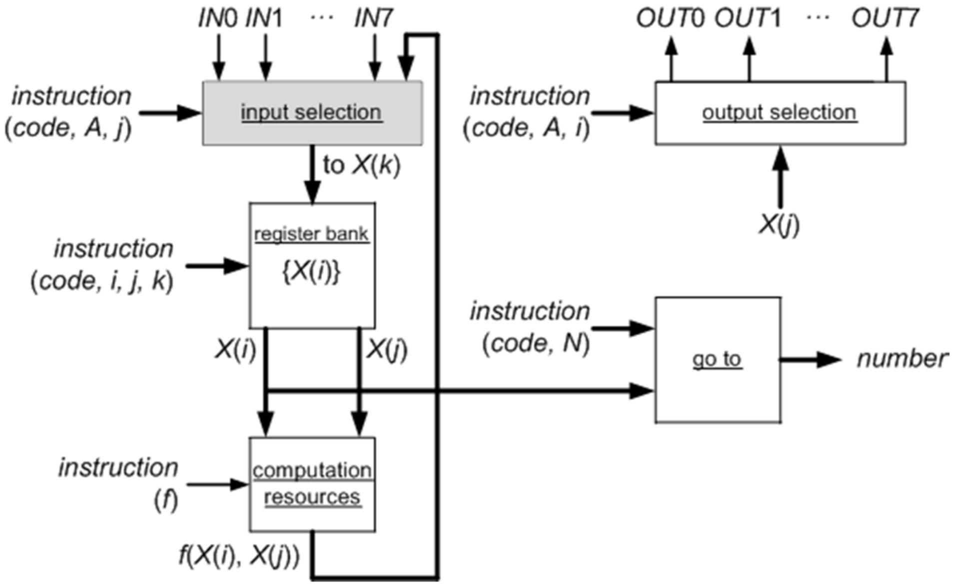
Code and list of parameters	Operation
(ASSIGN_VALUE, k, A)	*Xk* := *A*;
(DATA_INPUT, k, j)	*Xk* := *INj*;
(DATA_OUTPUT, i, j)	*OUTi* := *Xj*;
(OUTPUT_VALUE, i, A)	*OUTi* := *A*,
(OPERATION, i, j, k, f)	*Xk* := *f*(*Xi*, *Xj*);
(JUMP, N)	go to *N*;
(JUMP_POS, i, N)	if *Xi* > 0 go to *N*;
(JUMP_NEG, i, N)	if *Xi* < 0 go to *N*;

- only **two operations** *f*: + and - ;
- register bank: **16 registers** (*i*, *j* and *k* are 4-bit numbers);
- maximum **number of instructions: 256** (*N* is an 8-bit number).

Instruction encoding will be defined later.

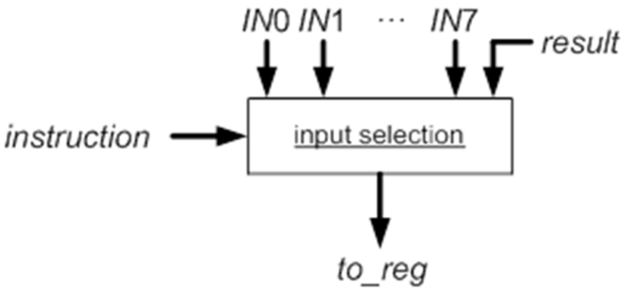
2 BLOCK DESCRIPTION

2.1 INPUT SELECTION




```
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;
```

(input selection)

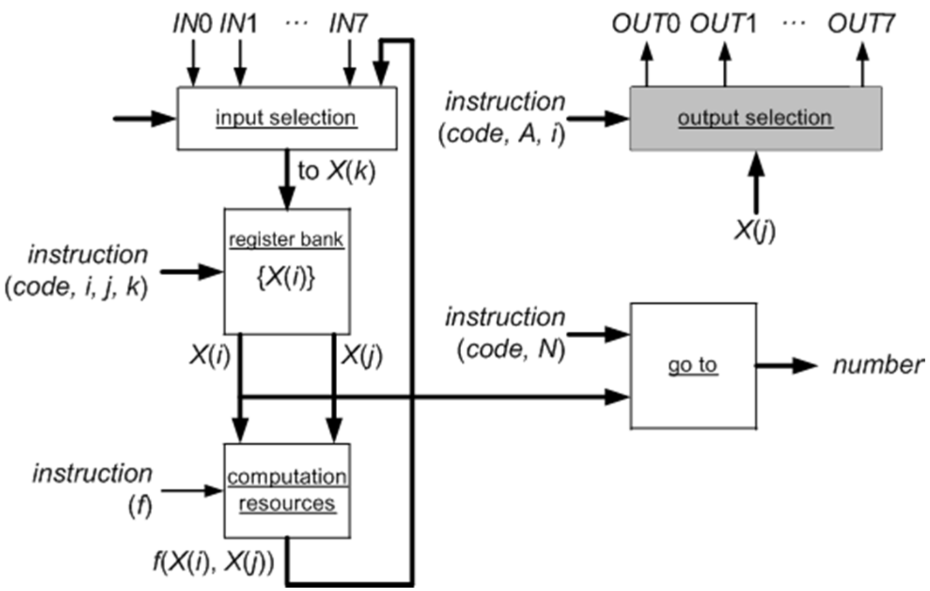


Functional specification

```
loop
  case instruction is
    when (ASSIGN_VALUE, k, A) =>
      to_reg := A;
    when (DATA_INPUT, k, j) =>
      to_reg := IN(j);
    when (OPERATION, i, j, k, f) =>
      to_reg := result;
    when others =>
      to_reg := don't care;
  end case;
end loop;
```

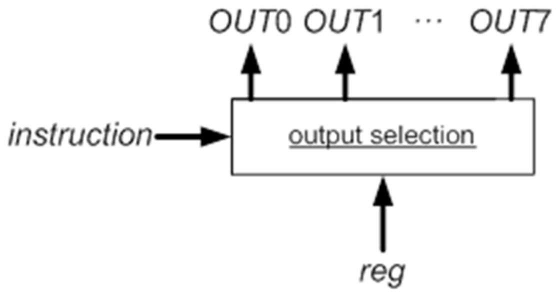
```
number := 0;
loop
  case program(number) is
    when (ASSIGN_VALUE, k, A) =>
      X(k) := A; number := number + 1;
    when (DATA_INPUT, k, j) =>
      X(k) := IN(j); number := number + 1;
    when (DATA_OUTPUT, i, j) =>
      OUT(i) := X(j); number := number + 1;
    when (OUTPUT_VALUE, i, A) =>
      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;
```

2.2 OUTPUT SELECTION




```
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;
```

(output selection)



Functional specification

```
loop
  case program(number) is
    when (DATA_OUTPUT, i, j) =>
      OUT(i) := reg;
    when (OUTPUT_VALUE, i, A) =>
      OUT(i) := A;
     when others =>
      OUT(i) := OUT(i);
  end case;
end loop;
```

registered output
(like a memory element)

```
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;
```

SUMMARY

P2.2

Universitat Autònoma de Barcelona

- **Structural description** = **block diagram**, based on
  - ✓ internal data,
  - ✓ data transfers,
  - ✓ operations.

- **Functional description of blocks**:
  - ✓ input selection,
  - ✓ output selection.