P1.1 PROCESSOR: SPECIFICATION

Jean-Pierre Deschamps

University Rovira i Virgili, Tarragona, Spain

1 SPECIFICATION OF TWO DIGITAL SYSTEMS

UAB
Universitat Autònoma de Barcelona

- Temperature controller (lesson 1.1):

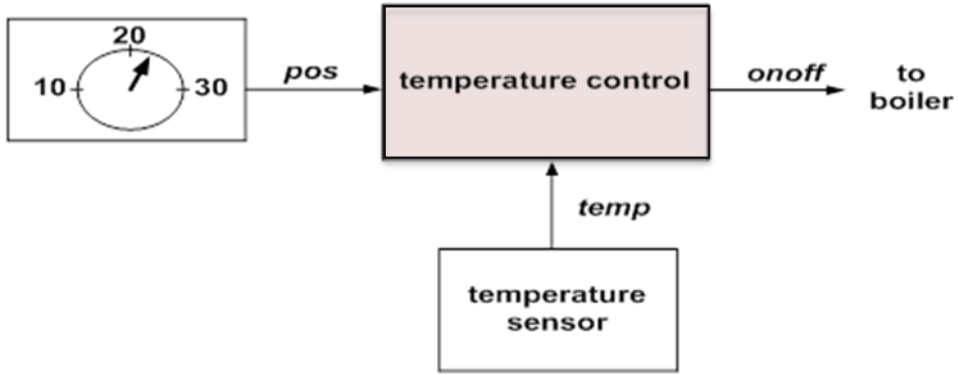
```
loop
  if temp < pos – half_degree then onoff := on;
  elsif temp > pos + half_degree then onoff := off;
  end if;
  wait for 10 s;
end loop;
```

- Chronometer (lesson 1.1)

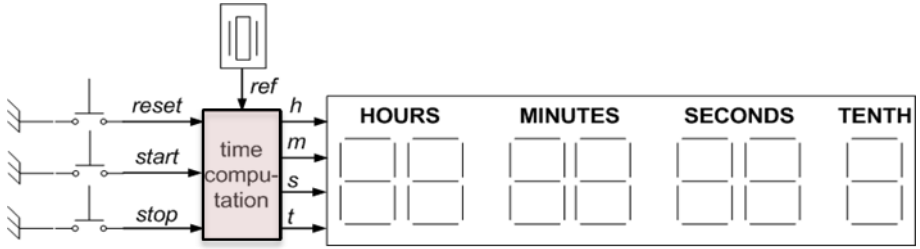
```
loop
    if reset = ON then time := 0;
    elsif start = ON then
        while stop = OFF loop
            if ref_positive_edge = TRUE then time := update(time);
            end if;
        end loop;
    end if;
end loop;
```

To each of them => digital system (lesson 1.1):

Temperature controller:



Chronometer:



2 DESIGN STRATEGIES

Option 1: associate to each algorithm a **completely new system** that executes the corresponding algorithm, and could not execute any other algorithm.

NEVERTHELESS

Both algorithms have some **common characteristics**.

UAB
Universitat Autònoma de Barcelona

```
loop
    if temp < pos – half_degree then onoff := on;
    elsif temp > pos + half_degree then onoff := off;
    end if;
    wait for 10 s;
end loop;
```

```
loop
     if reset = ON then time := 0;
     elsif start = ON then
         while stop = OFF loop
             if ref_positive_edge = TRUE then
             time := update(time);
             end if;
         end loop;
     end if;
 end loop;
```

- sequentially executed instructions

```
.............

n:  if temp < pos – half_degree then onoff := on;
 elsif temp > pos + half_degree then onoff := off;
 end if;

n+1: wait for 10 s;

n+2: if temp < pos – half_degree then onoff := on;
 elsif temp > pos + half_degree then onoff := off;
 end if;

n+3: wait for 10 s;

.................
```

```
loop
    if temp < pos – half_degree then onoff := on;
    elsif temp > pos + half_degree then onoff := off;
    end if;
    wait for 10 s;
 end loop;
```

```
loop
     if reset = ON then time := 0;
     elsif start = ON then
         while stop = OFF loop
             if ref_positive_edge = TRUE then
             time := update(time);
             end if;
         end loop;
     end if;
 end loop;
```

- conditional branches and jumps:

```
if temp < pos – half_degree then onoff := on;
elsif temp > pos + half_degree then onoff := off;

while stop = OFF loop
     if ref_positive_edge = TRUE then
     time := update(time);
     end if;
 end loop;
```

```
loop
    if temp < pos – half_degree then onoff := on;
    elsif temp > pos + half_degree then onoff := off;
    end if;
    wait for 10 s;
 end loop;
```

```
loop
     if reset = ON then time := 0;
     elsif start = ON then
         while stop = OFF loop
             if ref_positive_edge = TRUE then
             time := update(time);
             end if;
         end loop;
     end if;
 end loop;
```

- some instructions read input values or write output values:

  if temp < pos – half_degree (read *temp* and *pos*);

  onoff := on (write *onoff*);

  while stop = OFF (read stop);

  time := update(time) (write time);

```
loop
    if temp < pos – half_degree then onoff := on;
    elsif temp > pos + half_degree then onoff := off;
    end if;
    wait for 10 s;
  end loop;
```

```
loop
     if reset = ON then time := 0;
     elsif start = ON then
         while stop = OFF loop
             if ref_positive_edge = TRUE then
             time := update(time);
             end if;
         end loop;
     end if;
  end loop;
```

- some instructions execute computations:

```
temp – pos;

update(time);
```

Conclusion (another idea)

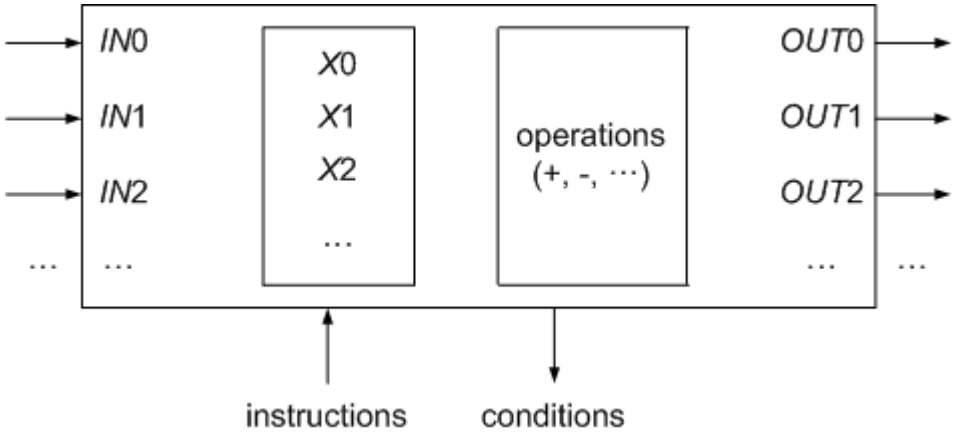
Option 2: define a generic (meta)system that includes

- input ports (*IN*0, *IN*1, *IN*2, ⋯),
- output ports (*OUT*0, *OUT*1, *OUT*2, ⋯),
- memory elements able to store data (*X*0, *X*1, *X*2, ⋯),
- processing resources that can execute computations (+, -, ⋯),

able to interpret instructions such as

- *Xi* := *A* (*A* constant);
- *Xi* := *INj*;
- *OUTi* := *Xj*;
- *OUTi* := *A* (*A* constant);
- *Xi* := *f*(*Xj*, *Xk*) (*f* => a processing resources);
- *goto n*, where *n* is an instruction number;
- *if* some_condition *goto n*, where *n* is an instruction number.

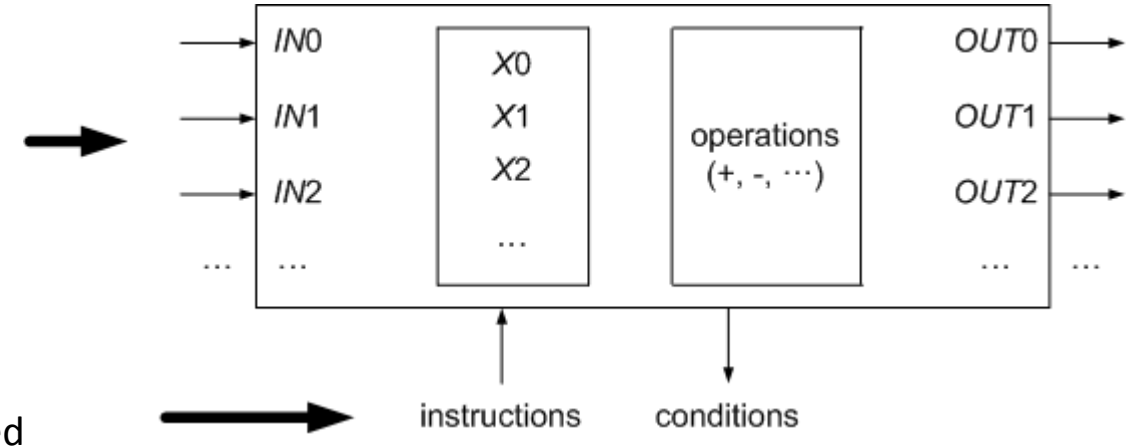
generic (meta)system



SUMMARY

UAB
Universitat Autònoma de Barcelona

Generic system: able to implement any algorithm



List of instructions **(program)**: depends on the particular algorithm to be implemented

Universitat Autònoma de Barcelona

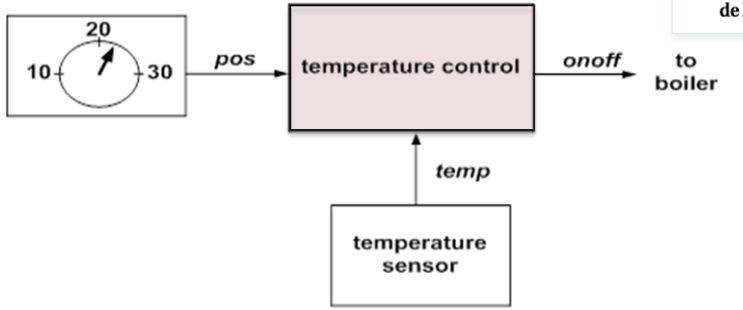
P1.2 EXAMPLES OF PROGRAMS

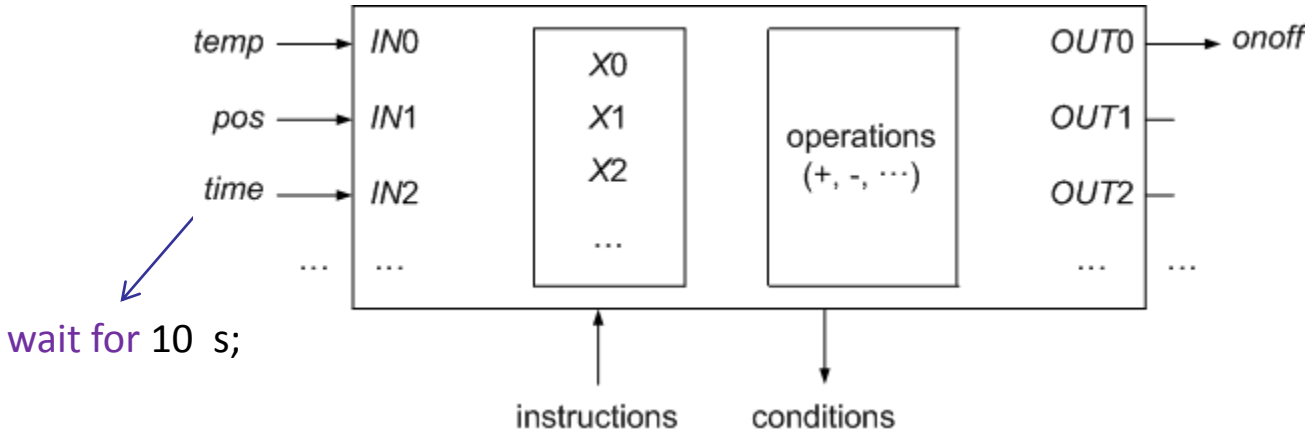
Jean-Pierre Deschamps

University Rovira i Virgili, Tarragona, Spain

1 TEMPERATURE CONTROLLER

```
loop
  if temp < pos then onoff := on;
  elsif temp > pos then onoff := off;
  end if;
  wait for 10 s;
end loop;
```

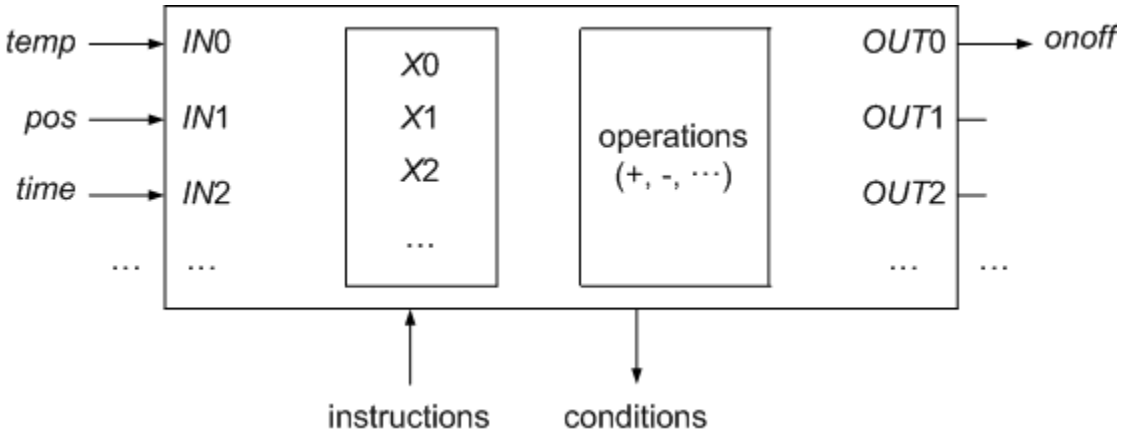




UAB
Universitat Autònoma de Barcelona

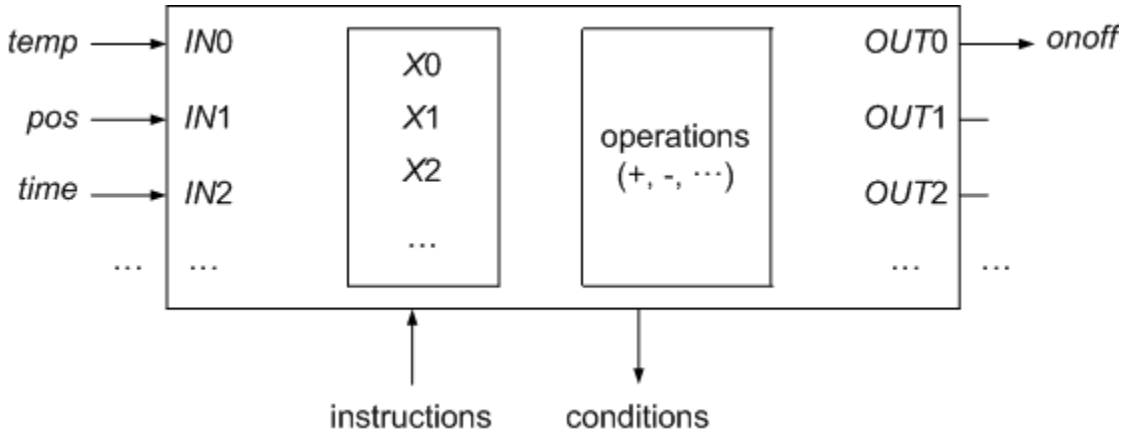
Instruction types:

- *Xi* := *A*;
- *Xi* := *INj*;
- *OUTi* := *Xj*;
- *OUTi* := *A*;
- *Xi* := *Xj* + *Xk*;
- *Xi* := *Xj* - *Xk*;
- *goto n*;
- *if Xi* < 0 *goto n*;
- *if Xi* > 0 *goto n*;



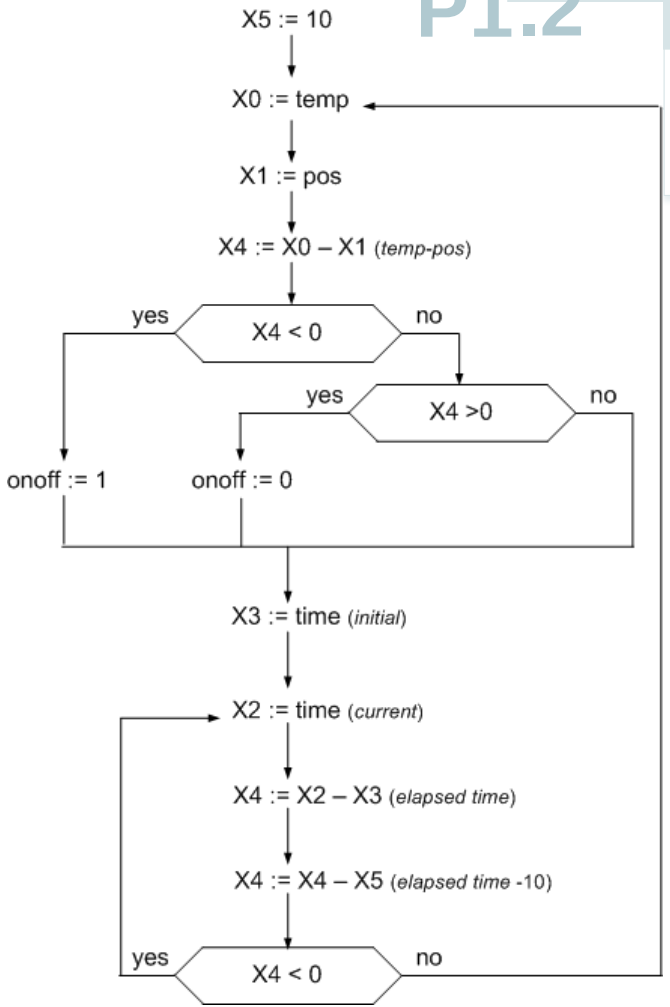
Six memory elements used to store algorithm data:

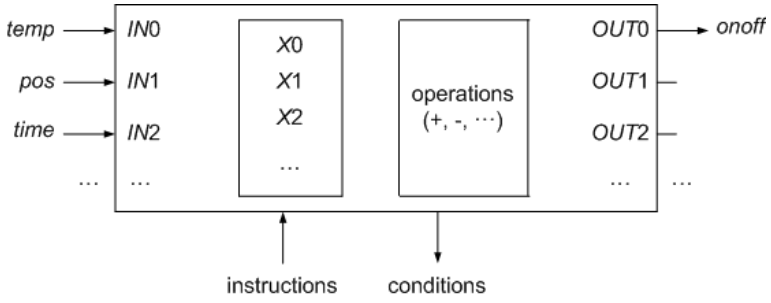
X0: *temp* (read from IN0)
X1: *pos* (read from IN1)
X2: *time* (read from IN2)
X3: *initial time* (read from IN2)
X4: *computation result* (internally generated)
X5: *constant* 10 (internally generated)



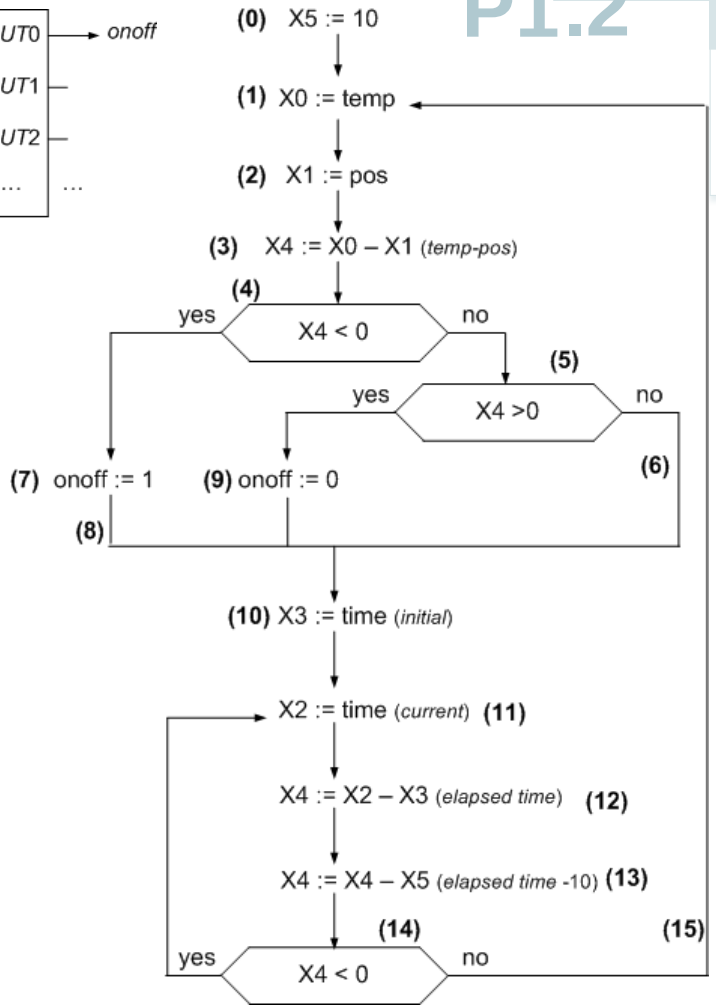
```
loop
  if temp < pos then onoff := on;
  elsif temp > pos then onoff := off;
  end if;
  wait for 10 s;
end loop;
```

X0: *temp* (read from IN0)
X1: *pos* (read from IN1)
X2: *time* (read from IN2)
X3: *initial time* (read from IN2)
X4: *computation result* (internally generated)
X5: *constant 10* (internally generated)



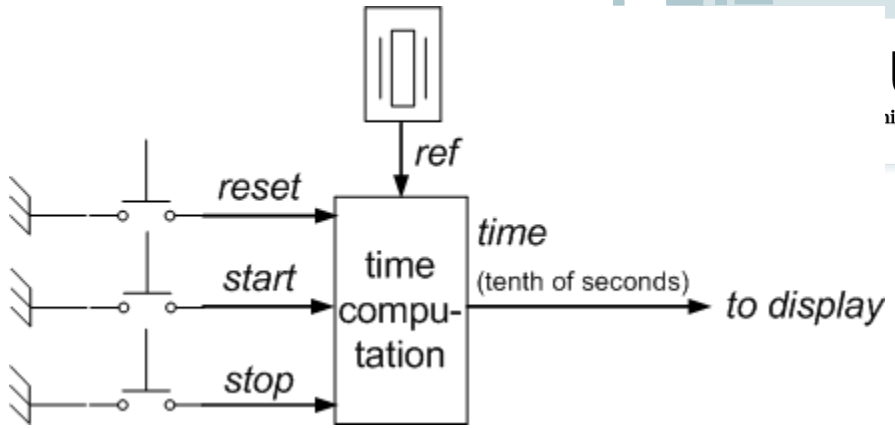


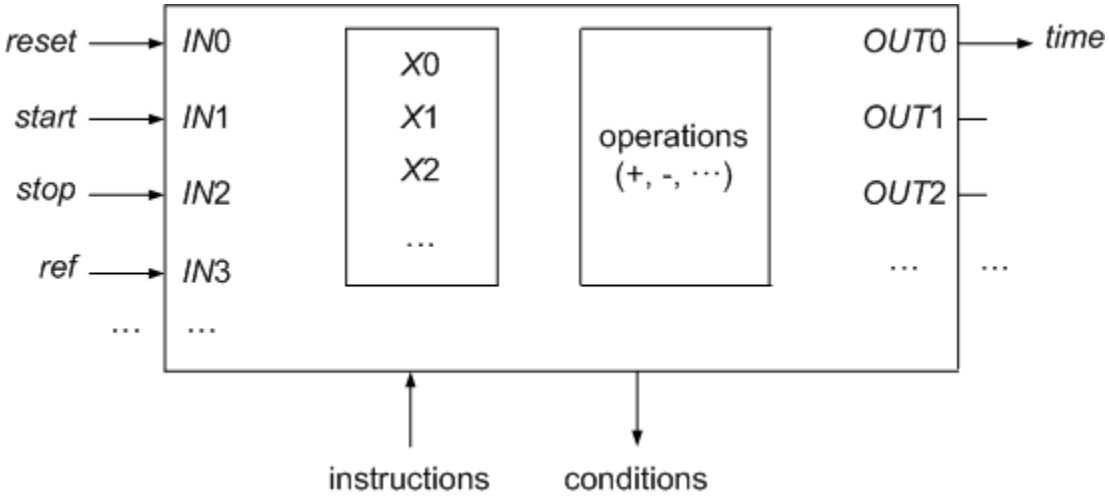
0: *X*5 := 10;
1: *X*0 := *IN*0;
2: *X*1 := *IN*1;
3: *X*4 := *X*0 - *X*1;
4: if *X*4 < 0 then go to 7;
5: if *X*4 > 0 then go to 9;
6: go to 10;
7: *OUT*0 := 1;
8: go to 10;
9: *OUT*0 := 0;
10: *X*3 := *IN*2;
11: *X*2 := *IN*2;
12: *X*4 := *X*2 - *X*3;
13: *X*4 := *X*4 - *X*5;
14: if *X*4 < 0 then go to 11;
15: go to 1;



2 CHRONOMETER

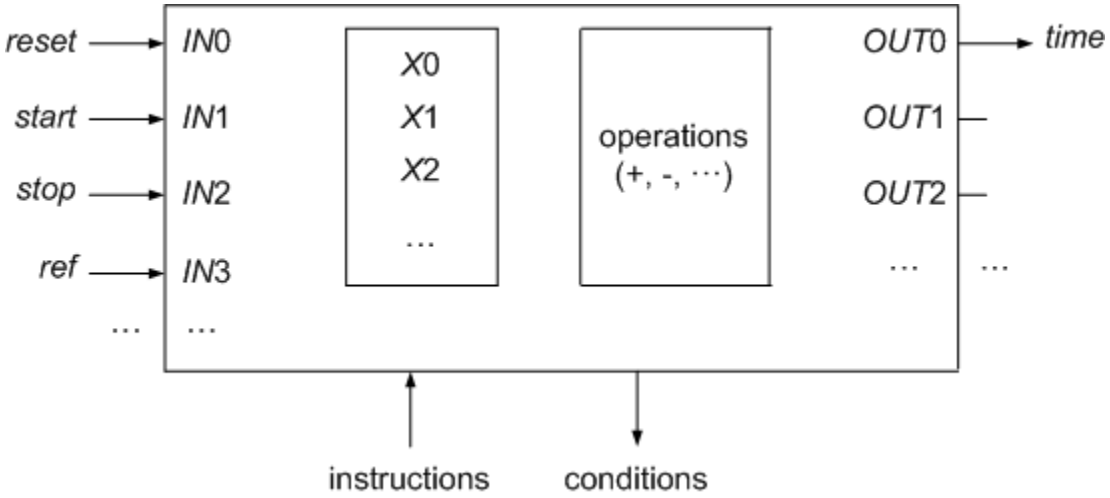
```
loop
    if reset = ON then time := 0;
    elsif start = ON then
        while stop = OFF loop
            if ref_positive_edge = TRUE then
              time := update(time);
            end if;
        end loop;
    end if;
end loop;
```



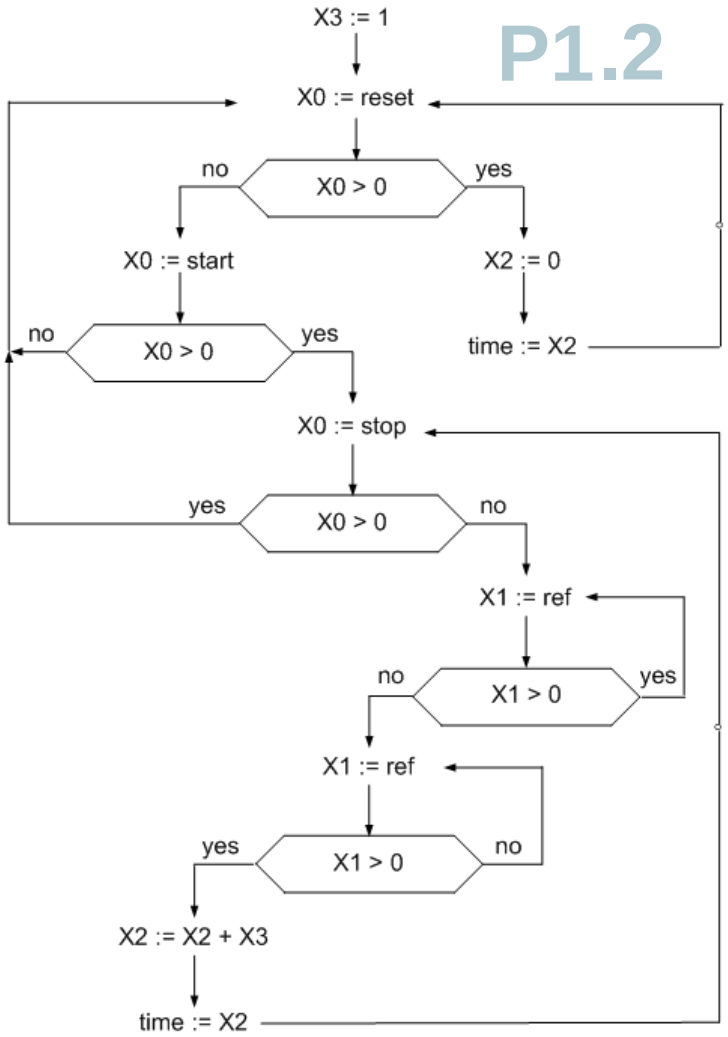


Four memory elements used to store algorithm data:

X0: *reset, start* or *stop* (read from IN0, IN1 or IN2)
X1: *ref* (read from IN3)
X2: *time* (internally generated)
X3: *constant* 1 (internally generated)



```
loop
    if reset = ON then time := 0;
    elsif start = ON then
        while stop = OFF loop
            if ref_positive_edge = TRUE then
              time := update(time);
            end if;
        end loop;
    end if;
end loop;
```



X0: *reset, start* or *stop* (read from IN0, IN1 or IN2)
X1: *ref* (read from IN3)
X2: *time* (internally generated)
X3: *constant* 1 (internally generated)

instruction execution time << T_{ref} = 0.1 s

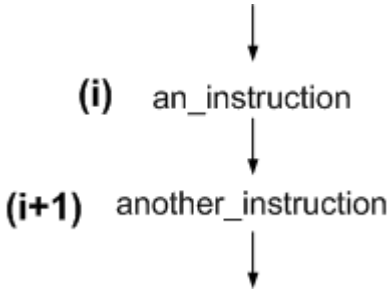
(*Exercise*)

UAB
Universitat Autònoma de Barcelona

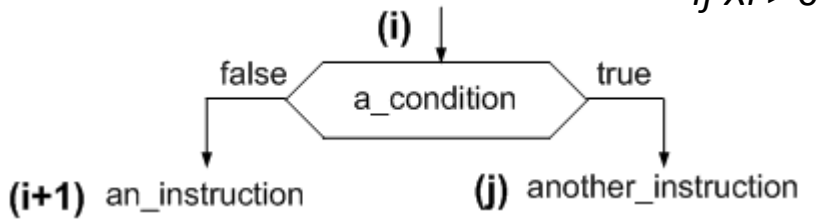
Instruction types:

- *Xi* := *A*;
- *Xi* := *INj*;
- *OUTi* := *Xj*;
- *OUTi* := *A*;
- *Xi* := *Xj* + *Xk*;
- *Xi* := *Xj* - *Xk*;
- *goto n*;
- *if Xi* < 0 *goto n*;
- *if Xi* > 0 *goto n*;

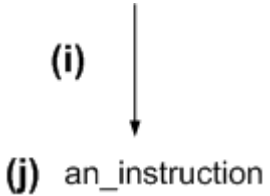
Generate a program corresponding to the previous diagram. For that: **assign numbers to the instructions.**



i: an_instruction;
i+1: another_instruction;



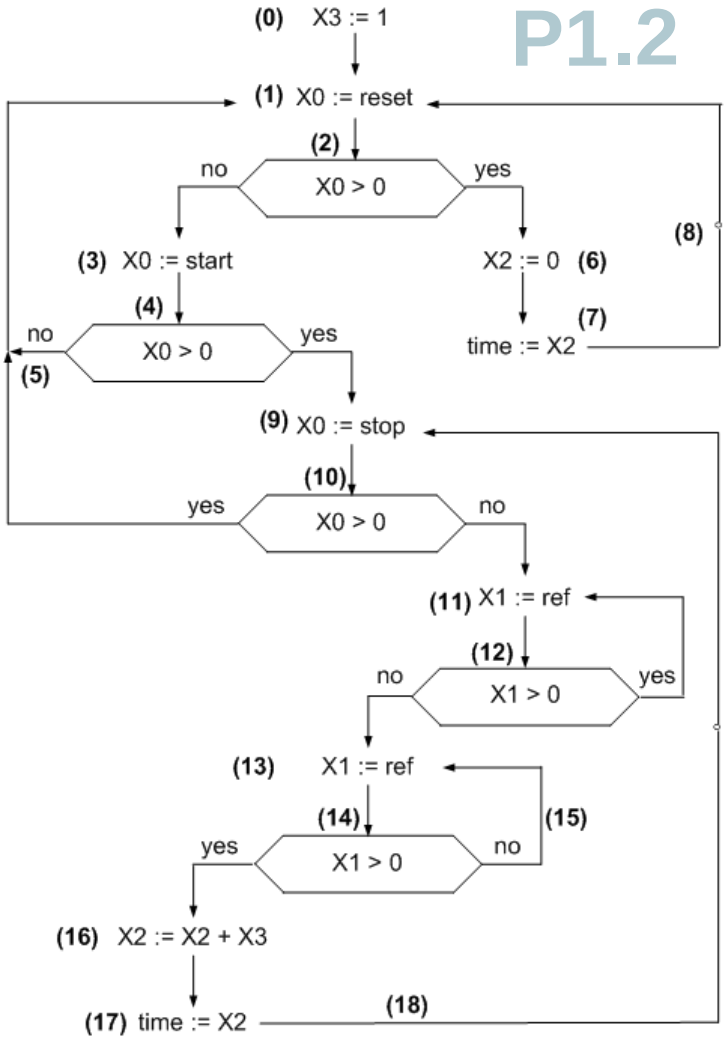
i: if a_condition go to j;
i+1: an_instruction;
.....
J: another_instruction;



i: go to j;
.....
J: an_instruction;

(*Solution*)

0: *X*3 := 1:
1: *X*0 := *IN*0;
2: if *X*0 > 0 then go to 6 ;
3: *X*0 := *IN*1;
4: if *X*0 > 0 then go to 9 ;
5: go to 1;
6: *X*2 := 0;
7: OUT0 := X2;
8: go to 1;
9: X0 ;= *IN*2;
10: if *X*0 > 0 then go to 1 ;
11: *X*1 := *IN*3;
12: if *X*1 > 0 then go to 11 ;
13: *X*1 := *IN3*;
14: if *X*1 > 0 then go to 16;
15: go to 13;
16: *X*2 := *X*2 + *X*3;
17: *OUT*0 := *X*2;
18: go to 9;



SUMMARY

- A generic system, called **PROCESSOR**, has been partially defined.
- It allows implementing many different algorithms.
- Two simple examples have been described.