Tempo em Ada

Em STR é necessário controlar o tempo porque:

- **atrasar processos até que aconteça algum evento**
- **Programar timeouts**
- **Especificar proporções de execução**
- **Especificar deadlines**
- **Satisfazer requerimento s de tempo**

Podemos ter necessidade de dispor de

- **Intervalos de tempo**
- **Tempos absolutos**

Os relógios:

→ módulos de HW e SW que permitem medir o tempo

Características

Estáticas

→ Resolução

→ Intervalo de valores

Dinâmicas

→ Granulidade

→ Exactidão

→ Estabilidade

Relógios em ADA

Ada.calendar

Ada.Real_time

Ada.Calendar (1)

```
package Ada.Calendar is

   type Time is private;

   subtype Year_Number    is Integer  range 1901..2099;
   subtype Month_Number   is Integer  range 1..12;
   subtype Day_Number     is Integer  range 1..31;
   subtype Day_Duration   is Duration range 0.0..86_400.0;

   function Clock return Time;

   function Year    (Date : Time) return Year_Number;
   function Month   (Date : Time) return Month_Number;
   function Day     (Date : Time) return Day_Number;
   function Seconds (Date : Time) return Day_Duration;

   procedure Split (Date    : in  Time;
                    Year    : out Year_Number;
                    Month   : out Month_Number;
                    Day     : out Day_Number;
                    Seconds : out Day_Duration);
```

Ada.Calendar (2)

```
   function Time_Of
     (Year    : Year_Number;
      Month   : Month_Number;
      Day     : Day_Number;
      Seconds : Day_Duration := 0.0)
     return    Time;

   function "+" (Left : Time;     Right : Duration) return Time;
   function "+" (Left : Duration; Right : Time)     return Time;
   function "-" (Left : Time;     Right : Duration) return Time;
   function "-" (Left : Time;     Right : Time)     return Duration;

   function "<"  (Left, Right : Time) return Boolean;
   function "<=" (Left, Right : Time) return Boolean;
   function ">"  (Left, Right : Time) return Boolean;
   function ">=" (Left, Right : Time) return Boolean;


   Time_Error : exception;
end Ada.Calendar;
```

```
use Ada.Calendar;
declare
   Old_Time, New_Time : Time;
   Interval           : Duration;
begin
   Old_Time := Clock;
   -- instrucciones cuya duración se mide
   New_Time := Clock;
   Interval := New_Time - Old_Time;
end;
```

Ada.Real_Time (1)

```
package Ada.Real_Time is

   type Time is private;
   Time_First : constant Time;
   Time_Last  : constant Time;
   Time_Unit  : constant := -- real number;

   type Time_Span is private;
   Time_Span_First :  constant Time_Span;
   Time_Span_Last  :  constant Time_Span;
   Time_Span_Zero  :  constant Time_Span;
   Time_Span_Unit  :  constant Time_Span;

   Tick : constant Time_Span;
   function Clock return Time;

   function "+"  (Left : Time;      Right : Time_Span) return Time;
   function "+"  (Left : Time_Span; Right : Time)      return Time;
   function "-"  (Left : Time;      Right : Time_Span) return Time;
   function "-"  (Left : Time;      Right : Time) return Time_Span;

   function "<"  (Left, Right : Time) return Boolean;
   function "<=" (Left, Right : Time) return Boolean;
   function ">"  (Left, Right : Time) return Boolean;
   function ">=" (Left, Right : Time) return Boolean;
```

Ada.Real_Time (2)

```
function "+" (Left, Right : Time_Span)            return Time_Span;
function "-" (Left, Right : Time_Span)            return Time_Span;
function "-" (Right : Time_Span)                  return Time_Span;
function "*" (Left : Time_Span; Right : Integer)  return Time_Span;
function "*" (Left : Integer;   Right : Time_Span)return Time_Span;
function "/" (Left, Right : Time_Span)            return Integer;
function "/" (Left : Time_Span; Right : Integer)  return Time_Span;
function "abs" (Right : Time_Span)                return Time_Span;

function "<"  (Left, Right : Time_Span) return Boolean;
function "<=" (Left, Right : Time_Span) return Boolean;
function ">"  (Left, Right : Time_Span) return Boolean;
function ">=" (Left, Right : Time_Span) return Boolean;

function To_Duration  (TS : Time_Span)  return Duration;
function To_Time_Span (D : Duration)    return Time_Span;

function Nanoseconds  (NS : integer) return Time_Span;
function Microseconds (US : integer) return Time_Span;
function Milliseconds (MS : integer) return Time_Span;
```

Ada.Real_Time (3)

```
   type Seconds_Count is new Integer range ...;

   procedure Split (T : Time;
                    SC : out Seconds_Count;
                    TS : out Time_Span);
   function Time_Of (SC : Seconds_Count; TS : Time_Span) return Time;

end Ada.Real_Time;
```

```
use Ada.Real_Time;
declare
   Old_Time, New_Time : Time;
   Interval           : Time_Span;
begin
   Old_Time := Clock;
   -- instrucciones cuya duración se mide
   New_Time := Clock;
   Interval := New_Time - Old_Time;
end;
```

Relógio calendário

- **proporciona valores de tempo com resolução de 1 s**

Relógios de tempo real

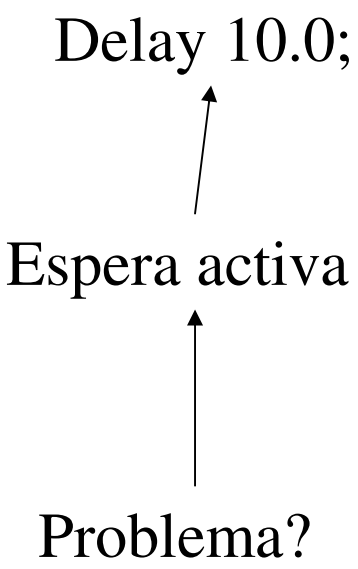
- **podemos definir relógios relógios distintos**
- **pelo menos deve existir um nomeado clock_realtime**
- **a resolução é de 1 ns**
- **a granulidade depende da implementação**

Existem dois tipos de atrasos

Atrasos relativos → nos quais uma execução se suspende durante um intervalo de tempo relativo ao instante actual

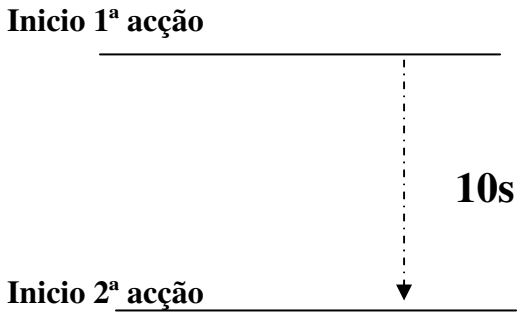
Suponhamos que se pretende abandonar a tarefa A 10 segundos após o instante actual

```
Time_And_Date  : Time;
Year,Month,Day : Integer;
Start, actual  : Day_Duration;
Time_And_Date  : Time;

...
Time_And_Date := Clock;
 Split(Time_And_Date, Year, Month, Day, Start);
loop
      Time_And_Date := Clock;
      Split(Time_And_Date, Year, Month, Day, actual);
      Exit when (actual-start)>10.0;
end loop;
```



Atrasos absolutos → nos quais a execução se suspende até que se chegue a um determinado instante de tempo absoluto

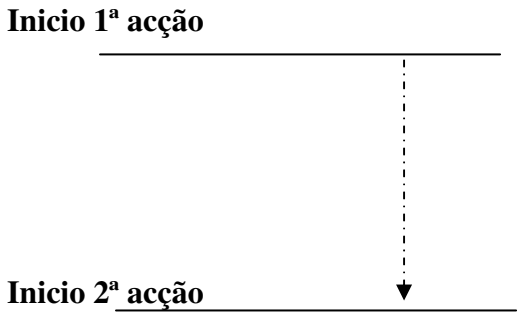
```
Time_And_Date  : Time;
Year,Month,Day : Integer;
Start, actual  : Day_Duration;
Time_And_Date  : Time;


...
Time_And_Date := Clock;
 Split(Time_And_Date, Year, Month, Day, Start);
      primeira_acção;
Time_And_Date := Clock;
 Split(Time_And_Date, Year, Month, Day, actual);
      delay 10.0-(actual-start);
      segunda_acção;
```



O delay until

→ suspende uma execução até que o valor do relógio seja igual ao especificado na expresssão

```
Time_And_Date  : Time;
Year,Month,Day : Integer;
Start, actual  : Day_Duration;
Time_And_Date  : Time;


...
Time_And_Date := Clock;
 Split(Time_And_Date, Year, Month, Day, Start);
      primeira_acção;
     delay until (start+10.0);
      segunda_acção;
```



Notas importantes

→ Se especificarmos um tempo inferior ao valor actual do relógio não se produz qualquer efeito

```
Time_And_Date  : Time;
Year,Month,Day : Integer;
Start, actual  : Day_Duration;
Time_And_Date  : Time;

...
Time_And_Date := Clock;
 Split(Time_And_Date, Year, Month, Day, Start);
      primeira_acção;
     delay until (start+10.0);
      segunda_acção;
```

Start = 5 s

Demora 12 s

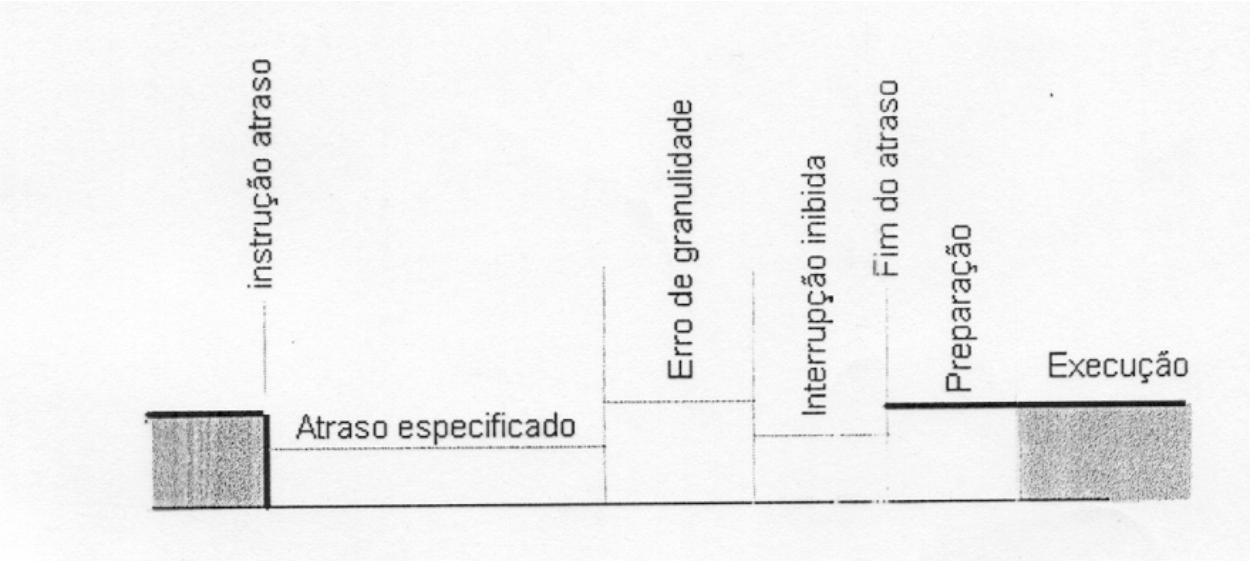
Teria que esperar até ao instante 15 s ...

→ **A instrução delay until T**

Pode ser aproximada por delay (T-Start)

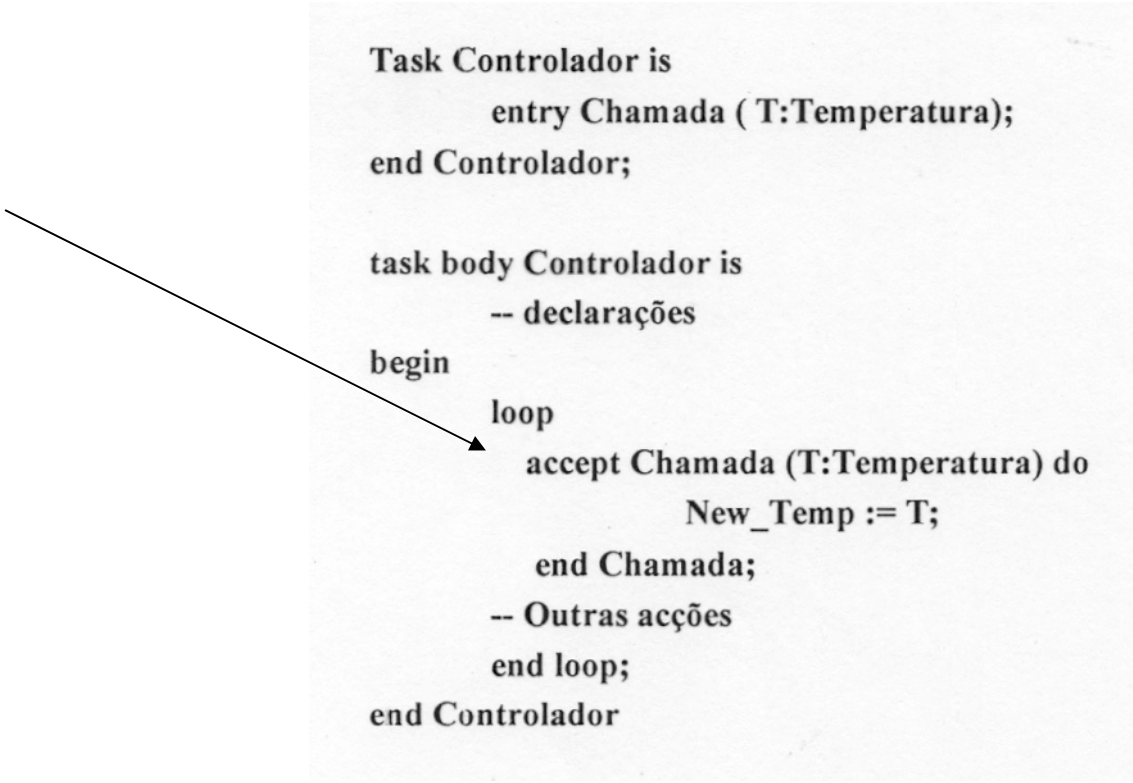
Mas não é exactamente a mesma coisa, pois para que assim fosse a expressão (T-Start) teria de se executar de forma atómica.

→ *EXECUÇÃO DE UM ATRASO*



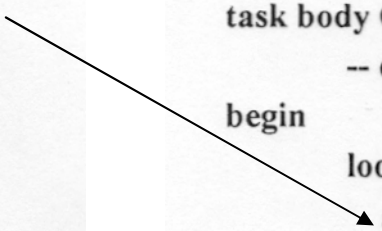
Chamadas condicionais → **utilizam-se quando se pretende executar uma acção alternativa se uma chamada não for atendida imediatamente**

```
select
controlador.chamada(T);
else
-- acção alternativa;
end select;
```



```
Task Controlador is
        entry Chamada ( T:Temperatura);
end Controlador;

task body Controlador is
        -- declarações
begin
        loop
           accept Chamada (T:Temperatura) do
                    New_Temp := T;
            end Chamada;
        -- Outras acções
        end loop;
end Controlador
```

```
loop
-- ler o novo valor de T
                select
                  Controlador. Chamada (T);
                or
                   delay 0.5;
                   -- Acção alternativa
                end select;
end loop
```



```
Task Controlador is
        entry Chamada ( T:Temperatura);
end Controlador;

task body Controlador is
        -- declarações
begin
        loop
          accept Chamada (T:Temperatura) do
                        New_Temp := T;
            end Chamada;
        -- Outras acções
        end loop;
end Controlador
```