

---

# Tema 2

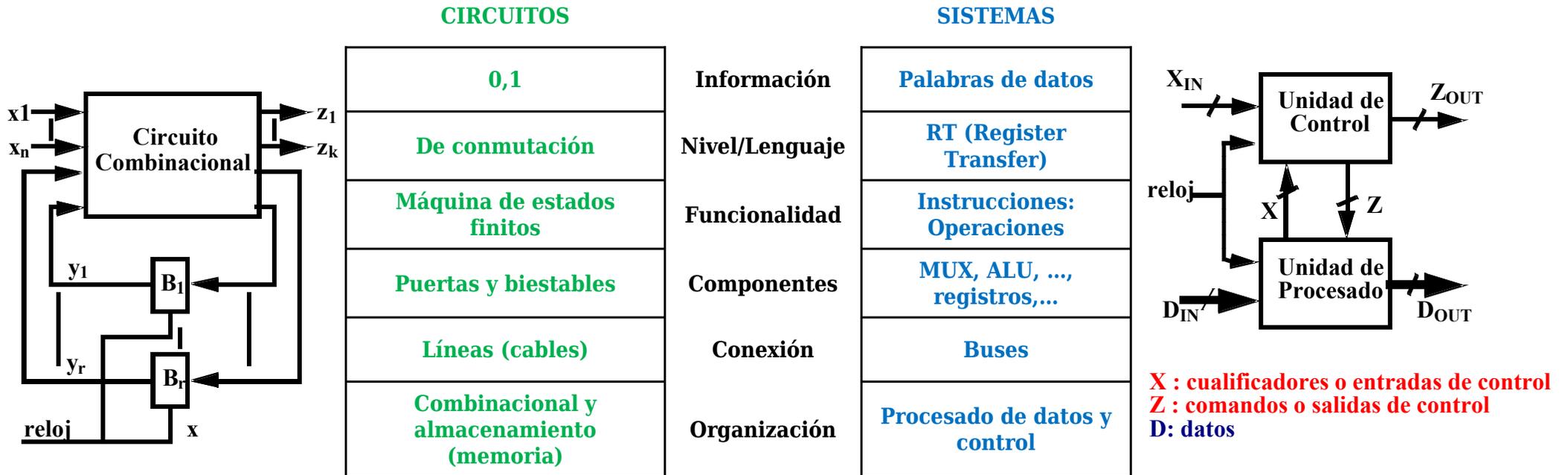
## *Sistemas Digitales*

---

# Contenidos del tema

- ▶ **El nivel RT**
- ▶ **Diseño de la unidad de datos**
  - ▶ **Interconexión mediante buses**
  - ▶ **Ejemplo: diseño de una calculadora simple**
- ▶ **Diseño de la unidad de control:**
  - ▶ **Descripción mediante cartas ASM**
  - ▶ **Descripción mediante Verilog**
- ▶ **Otros ejemplos**

# Nivel RT: circuitos versus sistemas

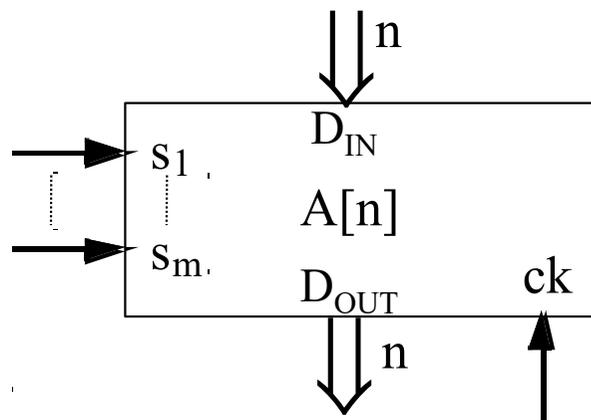


- ▶ Los sistemas que trataremos serán síncronos y sus biestables serán todos disparados por el mismo flanco de la misma señal de reloj.
- ▶ Con frecuencia omitiremos la representación de la señal de reloj.

# Nivel RT: Descripción de componentes

▶ Registro: unidad básica de almacenamiento de datos

▶ Representación estructural



▶ Representación funcional

Control $S_1 S_2 \dots S_m$	Operación a Nivel RT
0 0 ... 0	$A \leftarrow D_{IN}$
0 0 ... 1	$A \leftarrow 0$
...	...
1 1 ... 1	otras

# Nivel RT: Ejemplos de operación

► De escritura  
(secuencial)

► De lectura  
(combinacional)

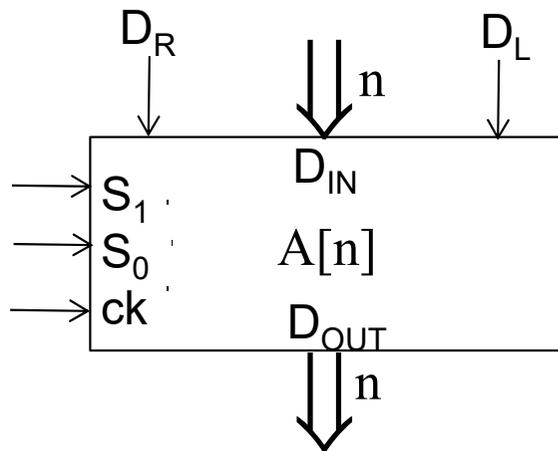
Operación	Notación RT
Carga en paralelo	$A \leftarrow D_{IN}$
Despl. a izquierda	$A \leftarrow SHL(A, D_L)$
Despl. a derecha	$A \leftarrow SHR(A, D_R)$
Incremento	$A \leftarrow A + 1$
Decremento	$A \leftarrow A - 1$
Puesta a 0	$A \leftarrow 0$
Puesta a 1	$A \leftarrow 1...1$
NOP/Inhibición	$A \leftarrow A$

Operación	Notación RT
Lect. incondicional	$Dout = [A]$
Lect. condicional	$R: Dout = [A]$
Función del dato	$Z=1$ si $[A]=0$

# Nivel RT: Ejemplos de descripción

- ▶ Registro universal de  $n$  bits

- ▶ Representación estructural



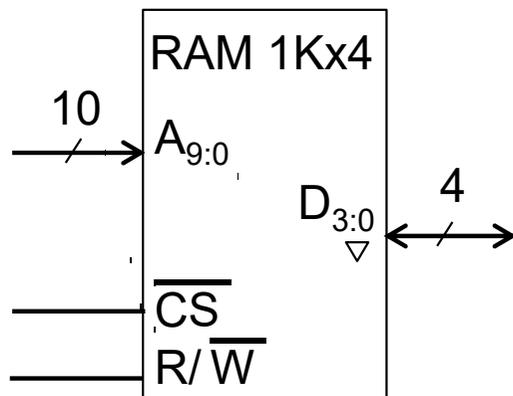
- ▶ Representación funcional

Control $S_1 S_0$	Escritura $A \leftarrow$	Lectura $D_{OUT} =$
0 0	$A \leftarrow A$	
0 1	$A \leftarrow D_{IN}$	
1 0	$A \leftarrow SHR(A, D_R)$	$D_{OUT} = [A]$
1 1	$A \leftarrow SHL(A, D_L)$	

# Nivel RT: Ejemplos de descripción

▶ Memoria RAM comercial: RAM 2114

▶ Representación estructural



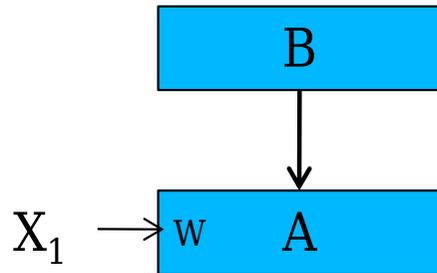
▶ Representación funcional

Control CS' R/W'	Escritura RAM ←	Lectura D <sub>3-0</sub> =
1 -	RAM ← RAM	HI
0 1	RAM ← RAM	[RAM(A)]
0 0	RAM(A) ← D <sub>3-0</sub>	D <sub>3-0</sub>

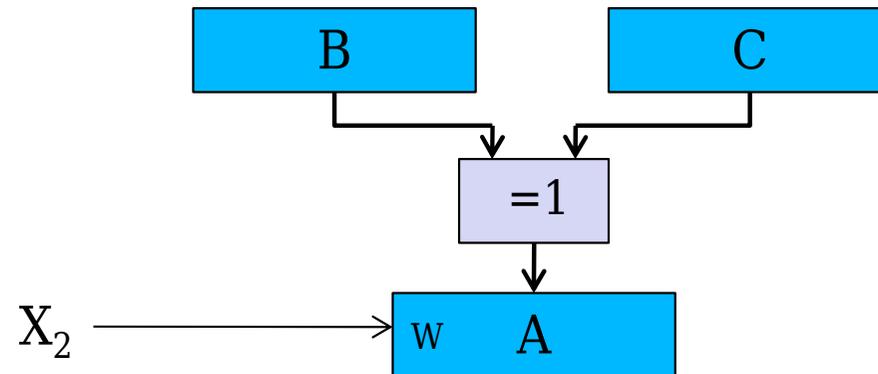
# Nivel RT: Operaciones entre varios registros

► Ejemplos:

$$X_1: A \leftarrow B$$

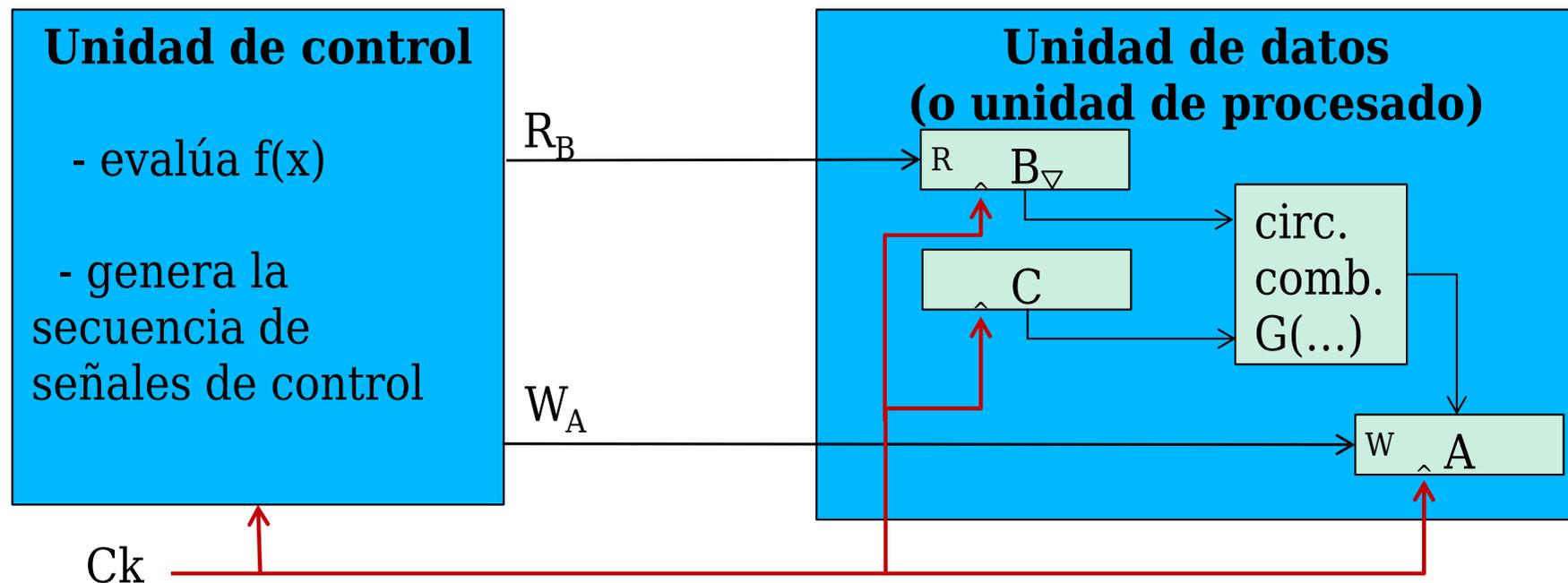


$$X_2: A \leftarrow B \oplus C$$



# Nivel RT: Estructura general del sistema digital

► Generalización:  $f(x): A \leftarrow G(B, C, \dots)$



---

# Nivel RT: Macro y micro -operaciones

## ▶ Macrooperación (o instrucción):

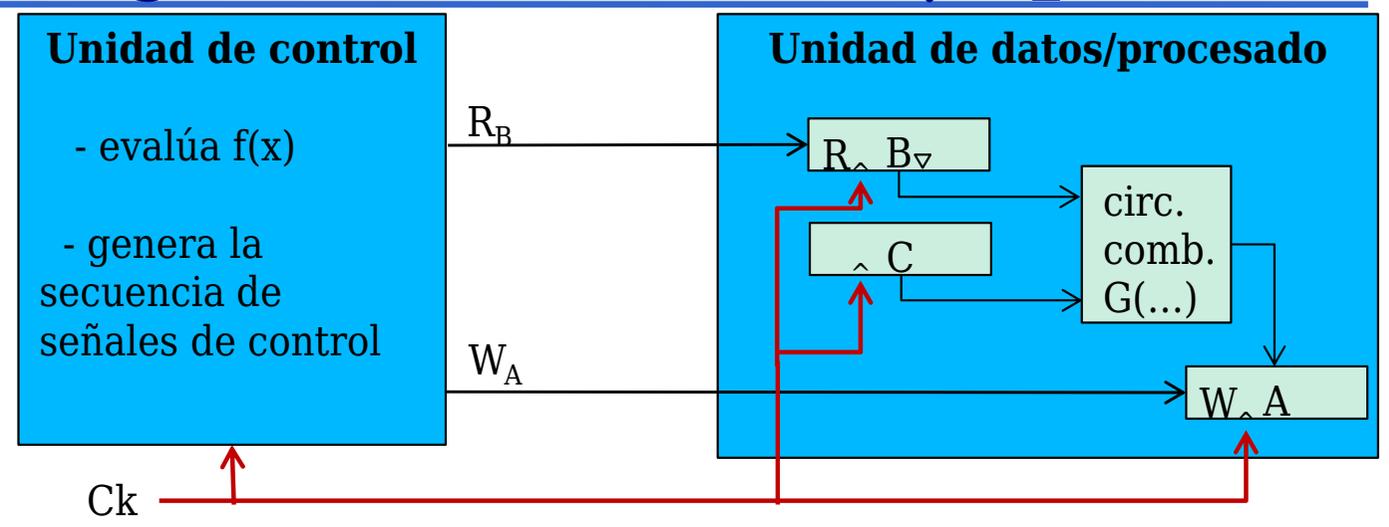
- ▶ Es cada tarea que especifica el usuario y que el sistema realiza automáticamente
- ▶ En general, el sistema emplea **varios ciclos** en su ejecución.
- ▶ La unidad de control “dirige/supervisa” la tarea realizada

## ▶ Microoperación ( $\mu\text{op}$ ):

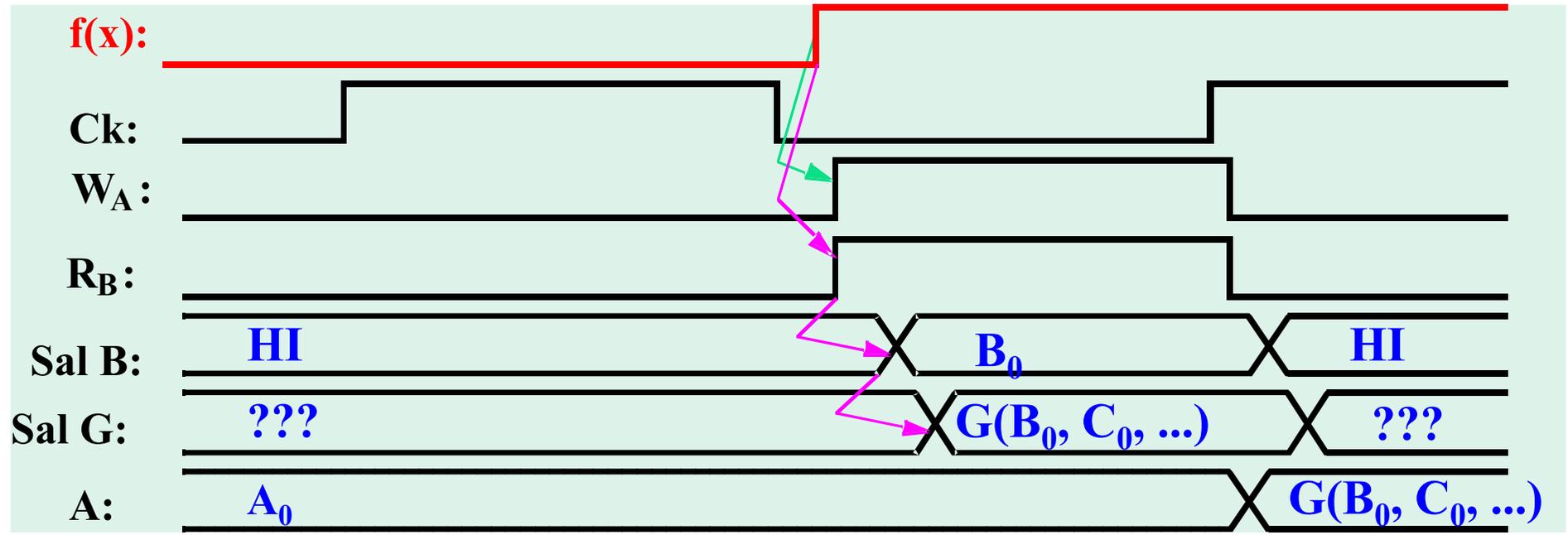
- ▶ Es cada tarea que el sistema realiza en un **único ciclo** de reloj
- ▶ En general, consiste en una o varias transferencias entre registros

# Nivel RT: Ejecución de una $\mu op$

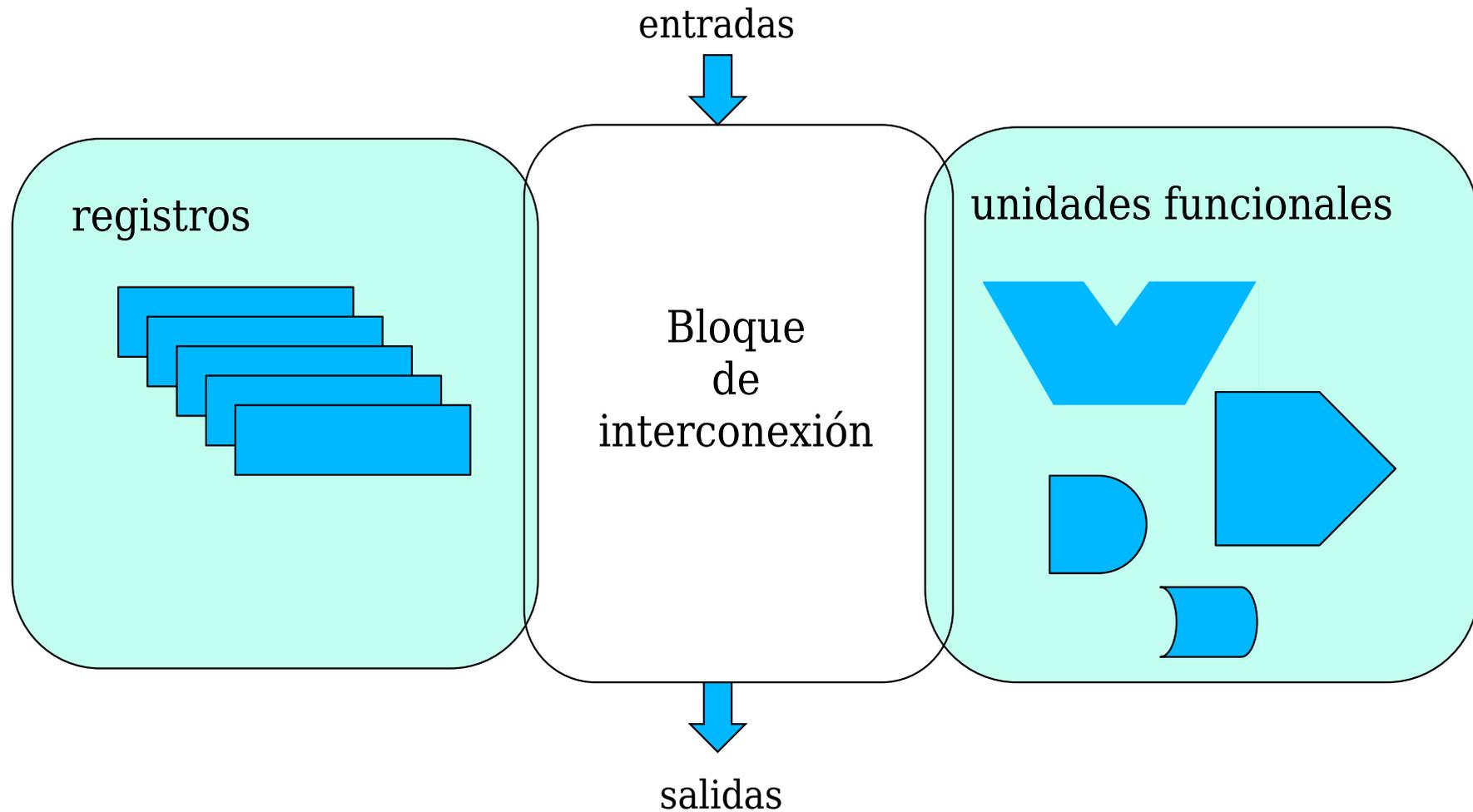
Ciclo K  
 $f(x): A \leftarrow G(B, C, \dots)$



← Ciclo K →



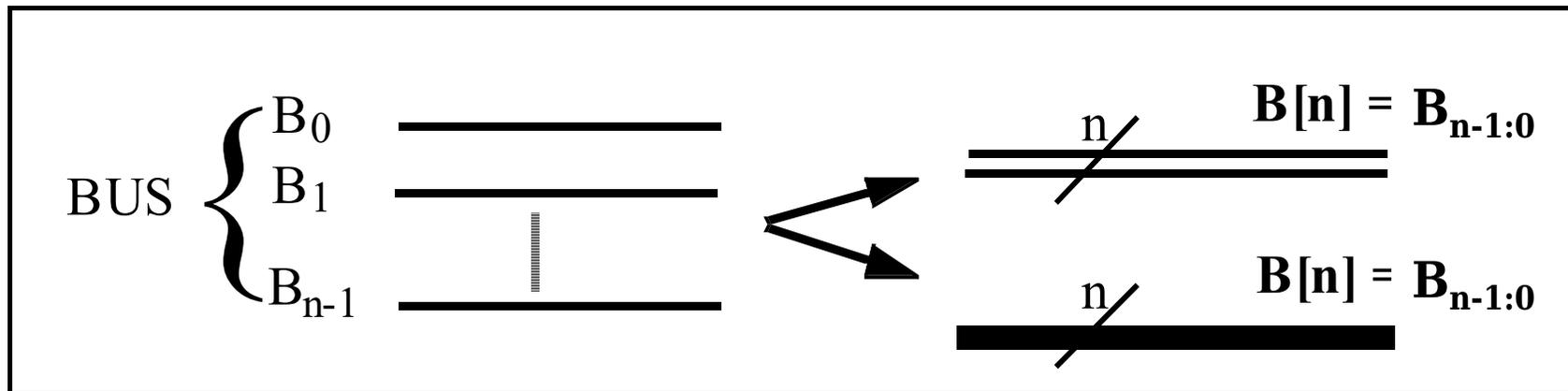
# Componentes de la Unidad de Datos



# Bloque de interconexión: buses

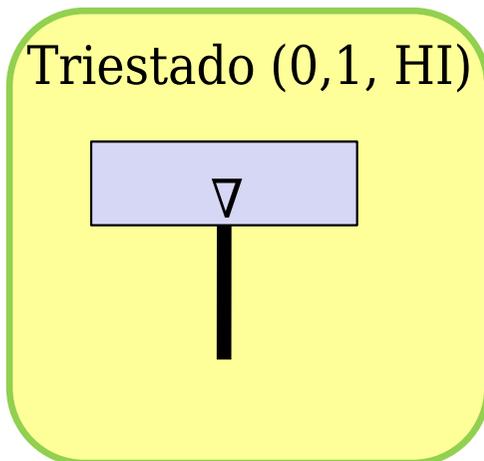
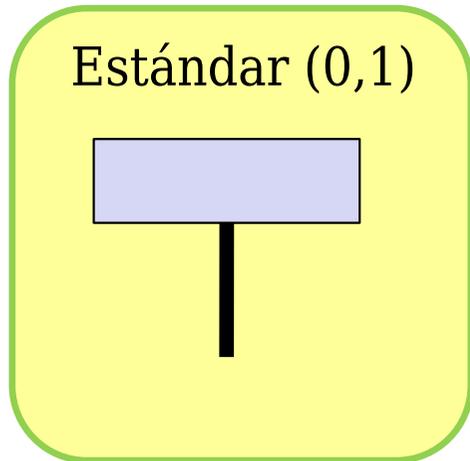
► Bus:

en un sistema digital, un bus es un conjunto de  $n$  líneas ordenadas que discurren en paralelo y transportan información (palabras)

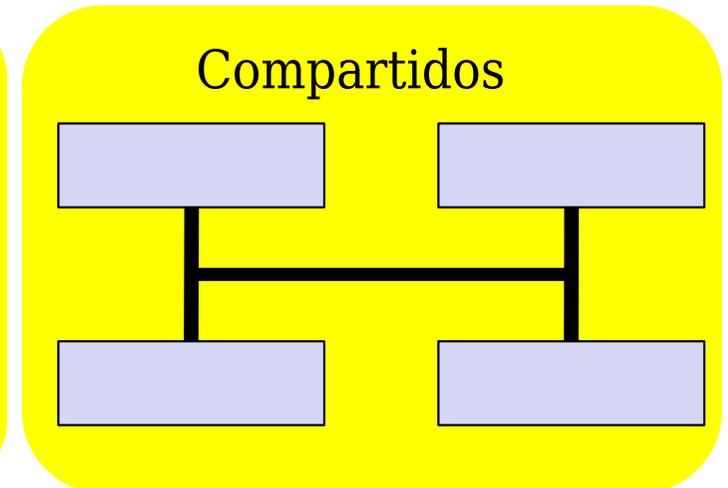
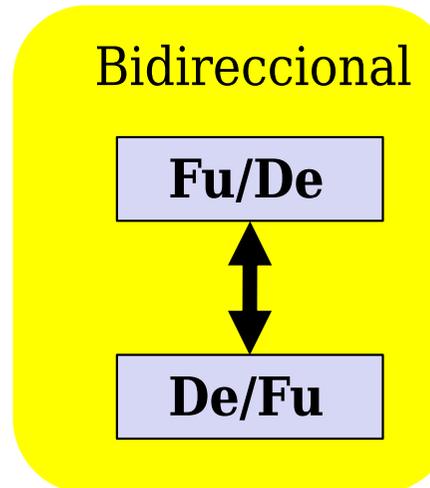
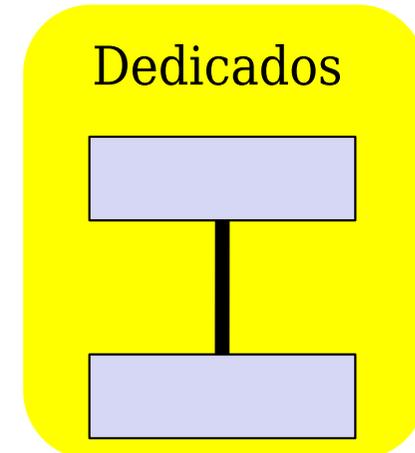
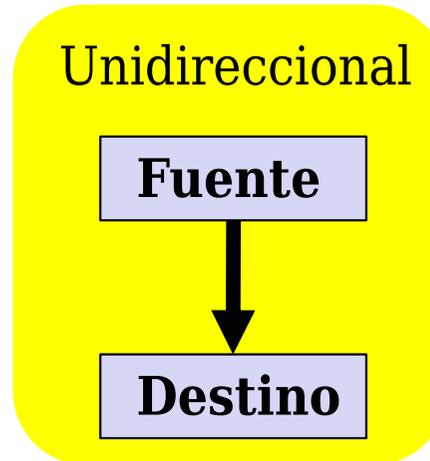


# Bloque de interconexión: buses

► Tipos de salida:



► Tipos de interconexión:



---

# Bloque de interconexión: ejemplo

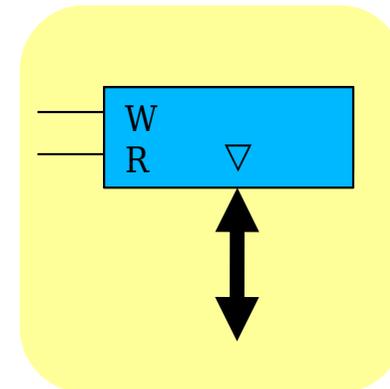
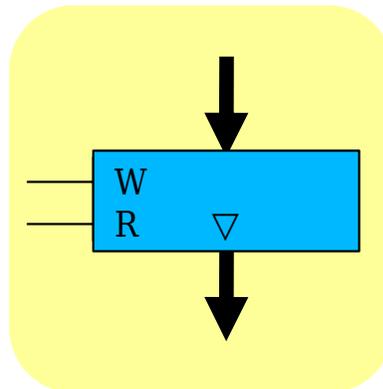
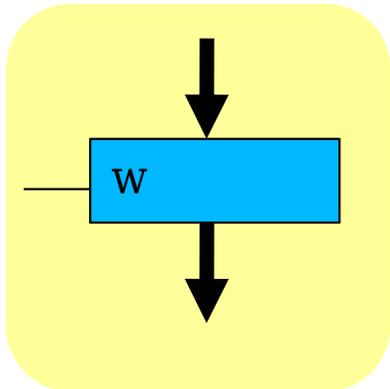
- ▶ Se dispone de 4 registros  $A_3, A_2, A_1, A_0$  con carga en paralelo.



- ▶ Hay que realizar la conexión para la transferencia  $A_D \leftarrow A_F$ , con  $F, D \in \{0, 1, 2, 3\}$
- ▶ Selección de fuente:  $F_1 F_0$
- ▶ Selección de destino:  $D_1 D_0$

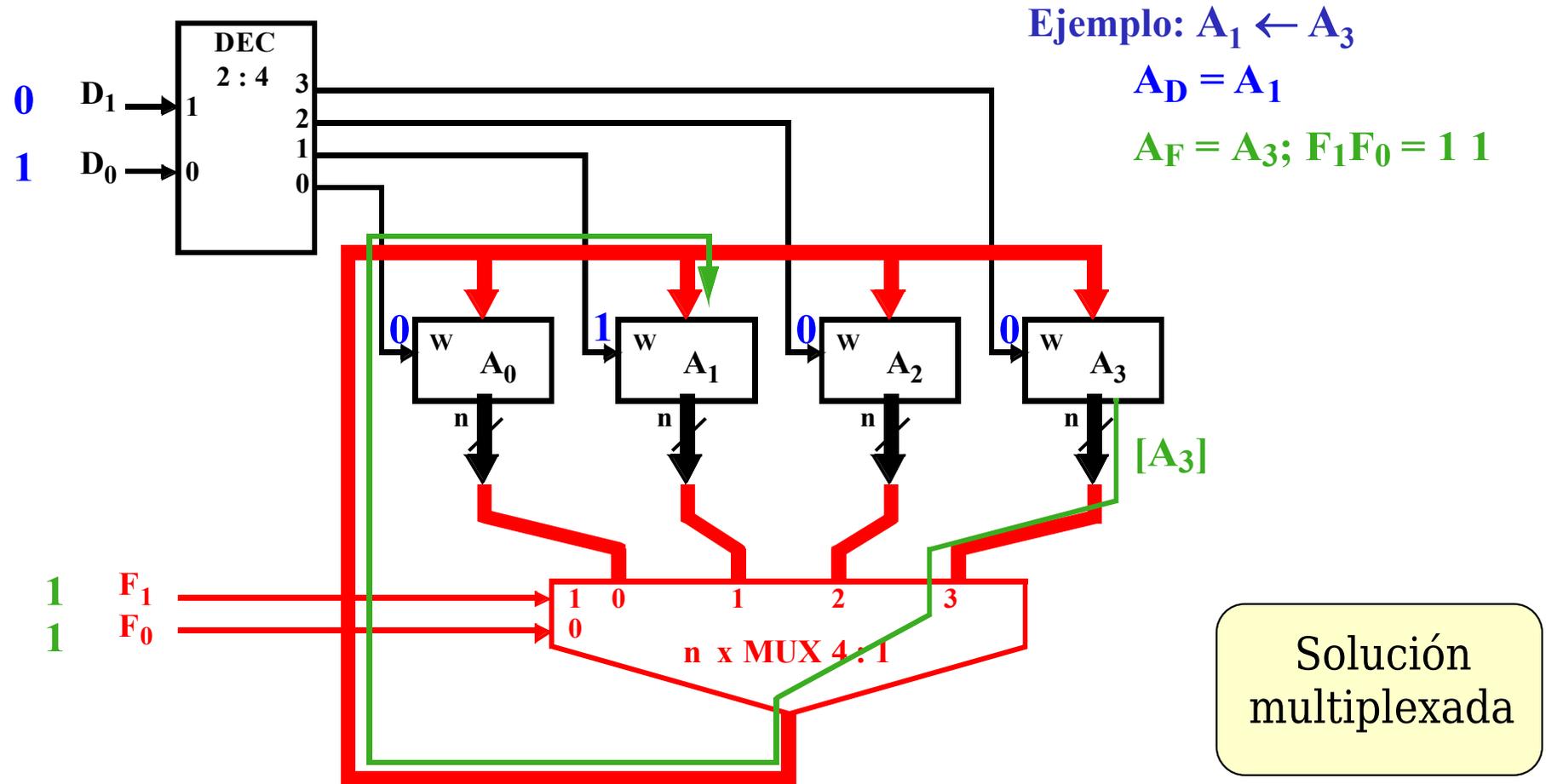
# Bloque de interconexión: ejemplo

- ▶ Caso 1: registros con salida y entrada separadas
- ▶ Caso 2: registros con salida y entrada separadas, salida triestado
- ▶ Caso 3: registros con un único bus bidireccional de salida y entrada



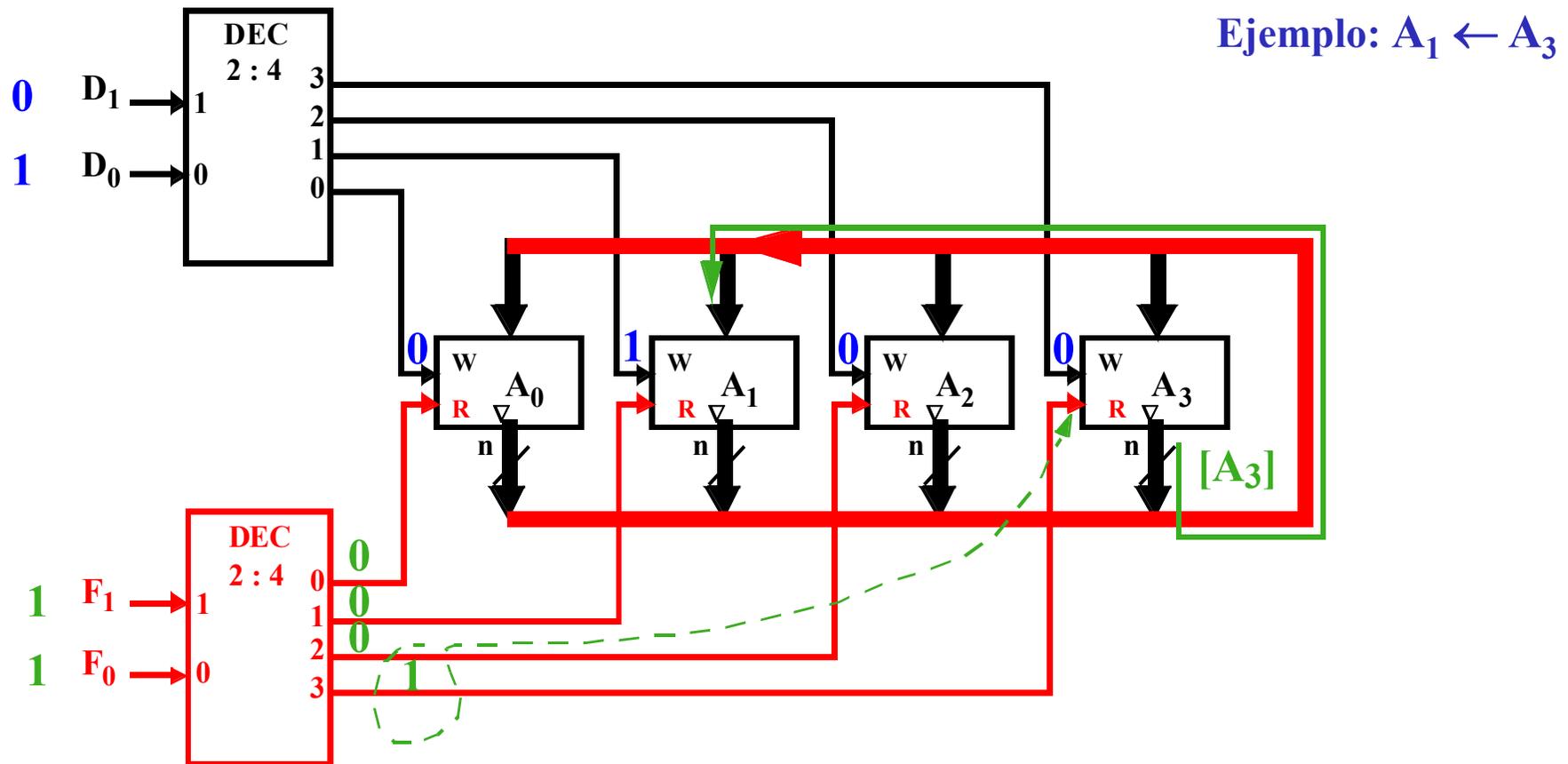
# Bloque de interconexión: ejemplo

- ▶ Caso 1: registros con salida y entrada separadas



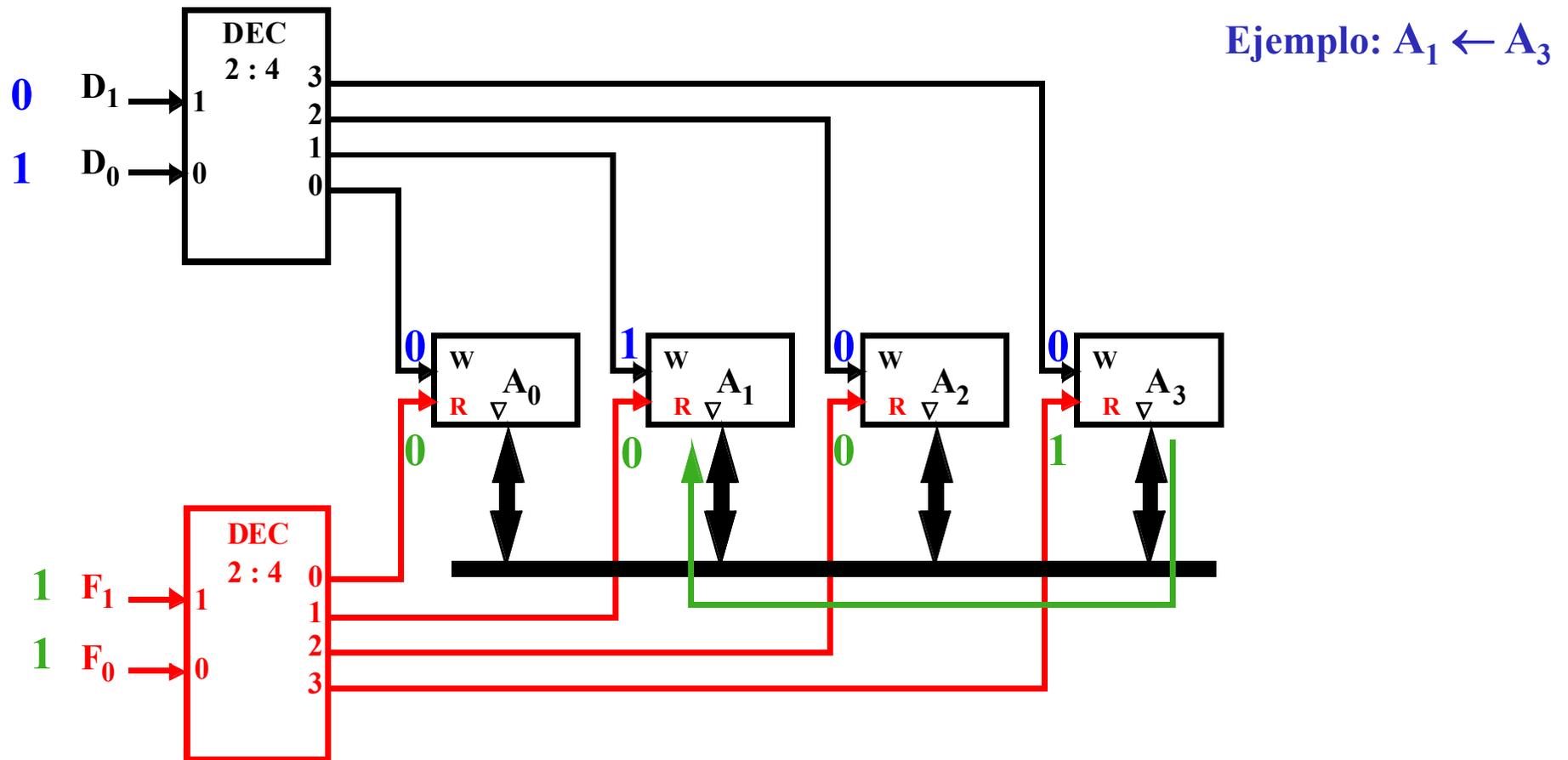
# Bloque de interconexión: ejemplo

- ▶ Caso 2: registros con salida y entrada separadas, salida triestado



# Bloque de interconexión: ejemplo

- ▶ Caso 3: registros con un único bus bidireccional de salida y entrada



---

# Diseño de un sistema digital

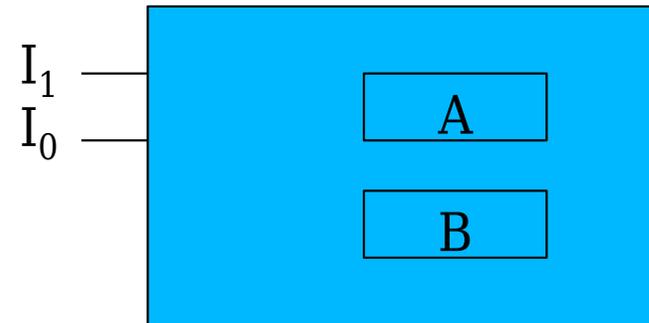
## ▶ Metodología

- ▶ Paso 1: Comprender claramente las **especificaciones** del sistema a diseñar y definir el conjunto de instrucciones/operaciones. Los registros que aparecen en la descripción de las macrooperaciones son los **registros visibles**.
- ▶ Paso 2: Proponer una **unidad de datos** capaz de ejecutar todas las operaciones especificadas. Debe incluir los registros visibles.
- ▶ Paso 3: Describir todos los **componentes a nivel RT** estructural y funcional.
- ▶ Paso 4: Descomponer las macrooperaciones en **microoperaciones** para la arquitectura propuesta.
- ▶ Paso 5: Desarrollar la **unidad de control**

# Diseño de una calculadora simple

- ▶ Paso 1- **Especificaciones** del sistema a diseñar:
  - ▶ Se dispone de 2 registros, A y B y se desea poder realizar cualquiera de las siguientes operaciones:

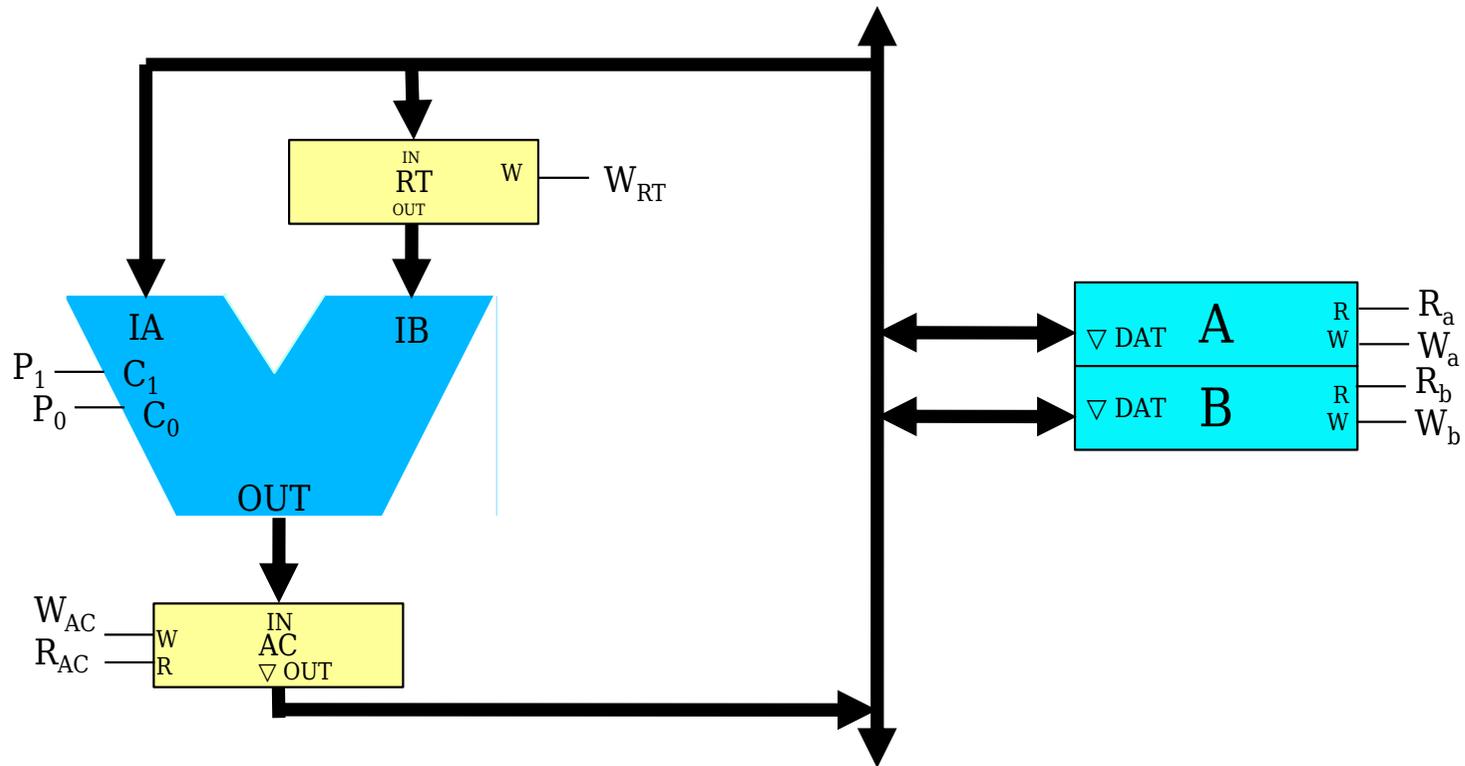
$I_1 I_0$	operación
0 0	$A \leftarrow A + B$
0 1	$B \leftarrow A + B$
1 0	$A \leftarrow A - B$
1 1	$B \leftarrow A - B$



- ▶ Se han asignado los códigos de modo que el registro destino se identifica con  $I_0$  y la operación con  $I_1$ .

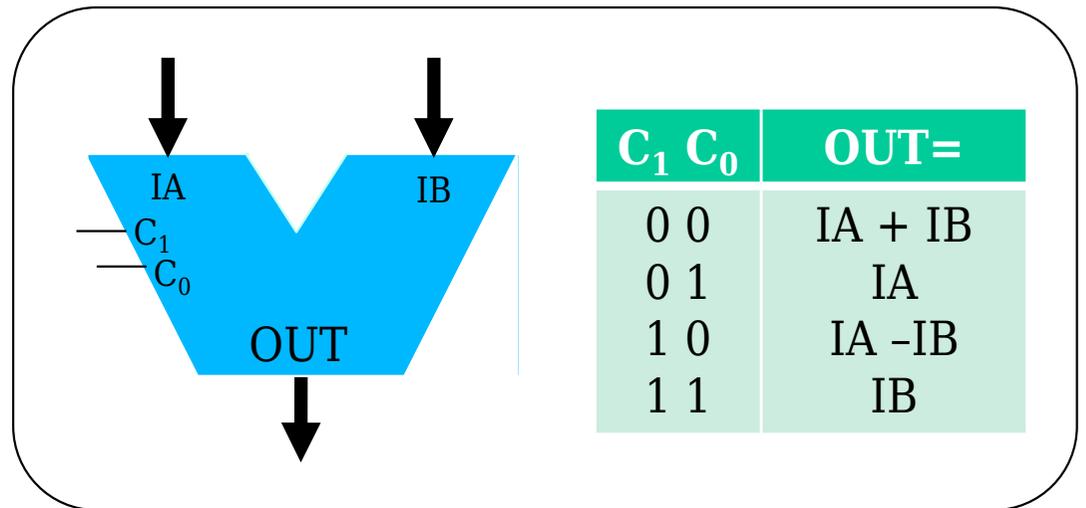
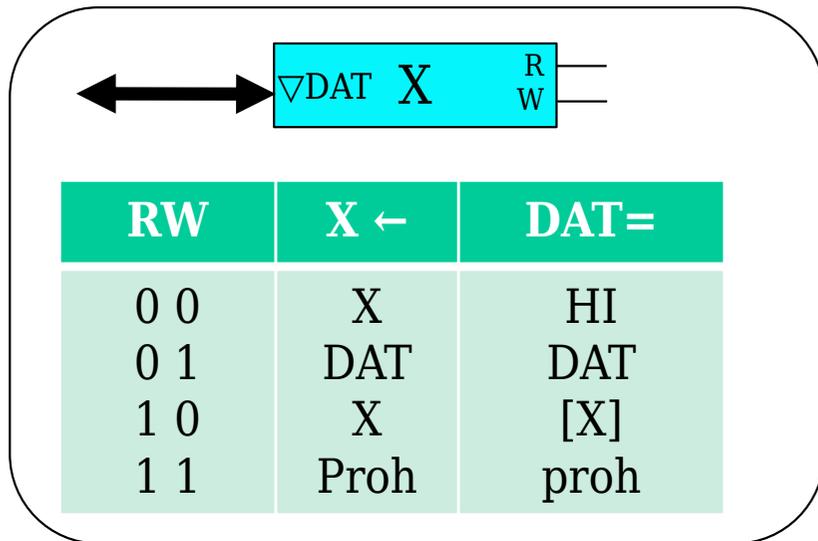
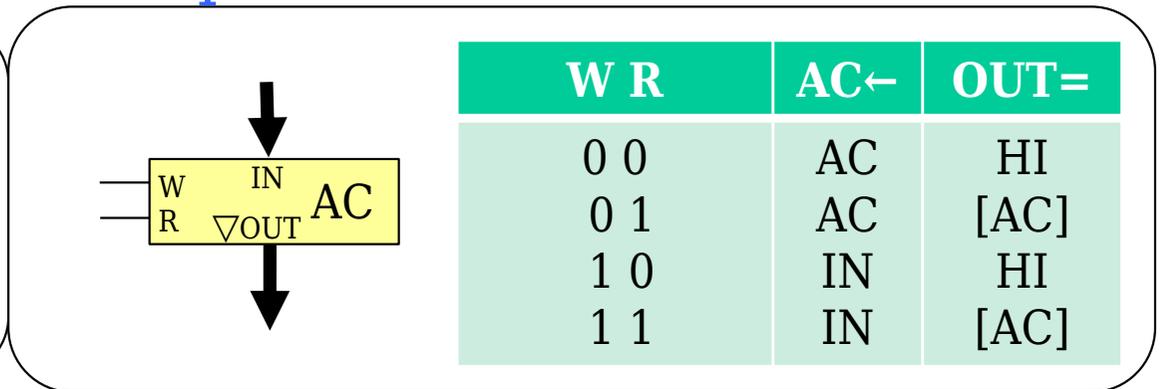
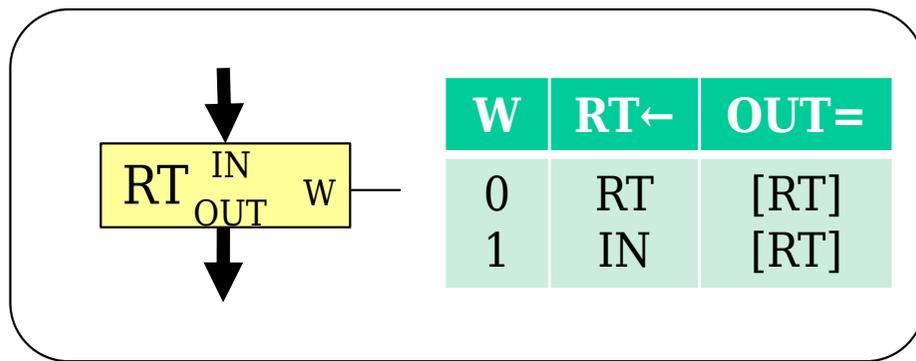
# Diseño de la unidad de datos de una calculadora

- ▶ Paso 2 - Proponemos una **arquitectura genérica** de un bus capaz de ejecutar las operaciones especificadas.



# Diseño de la unidad de datos de una calculadora

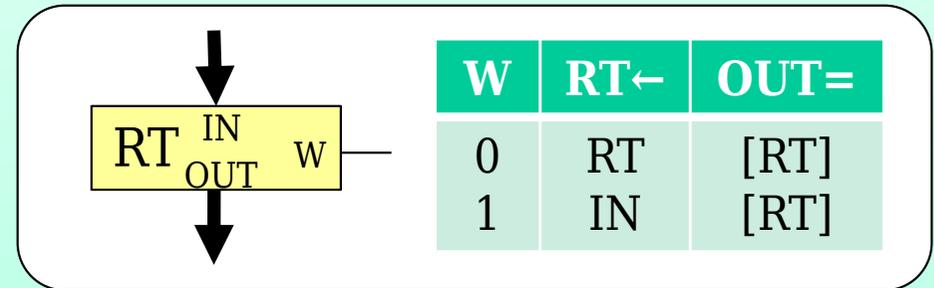
▶ Paso 3 - Describimos los componentes a nivel RT



# Descripción Verilog de la unidad de datos de la calculadora

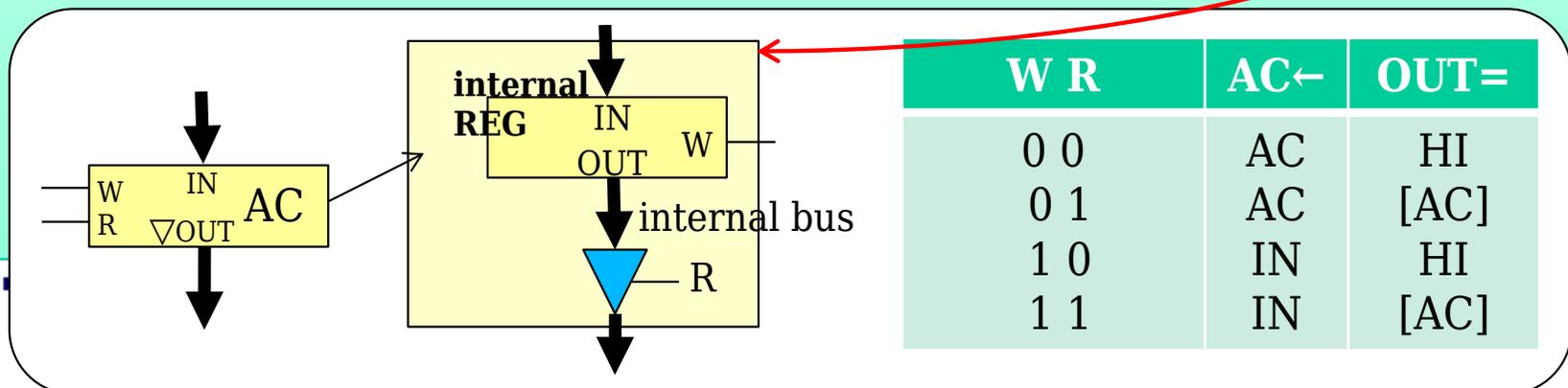
//declaración del tipo módulo correspondiente a RT

```
module type1 (input W, ck, input [7:0] IN, output reg [7:0] OUT);
    always@(posedge ck)
        if(W)
            OUT<=IN;
endmodule
```



//declaración del tipo módulo correspondiente a AC (instanciaremos a un módulo type1 y añadiremos lectura condicional)

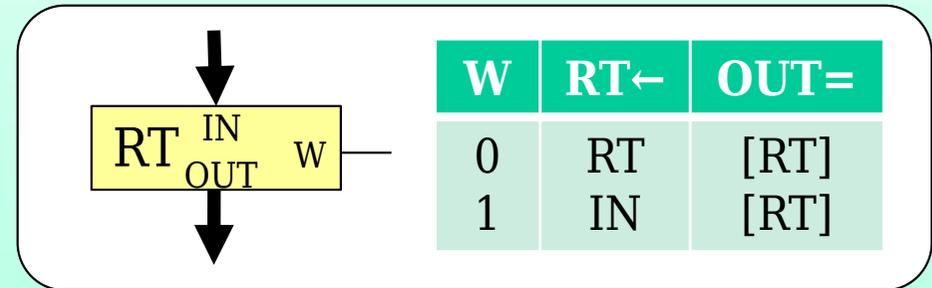
```
module type2 (input W, R, ck, input [7:0] IN, output [7:0] OUT);
    wire [7:0] internal_bus;
    type1 internal_reg(W, ck, IN, internal_bus);
    assign OUT = R ? internal_bus : 'bz;
endmodule
```



# Descripción Verilog de la unidad de datos de la calculadora (con parámetros)

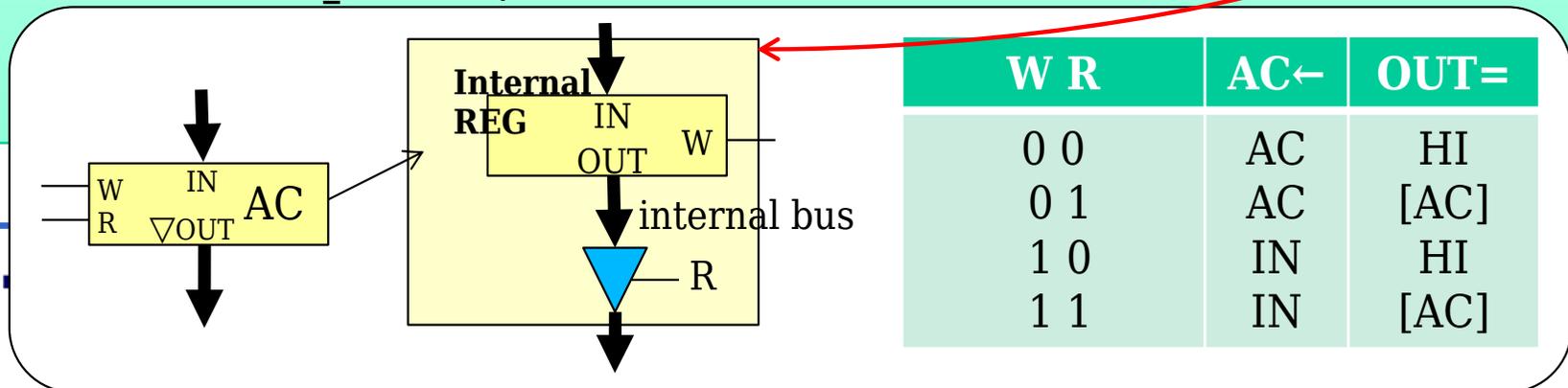
//declaración del tipo módulo correspondiente a RT

```
module type1 #(parameter N=8)
    (input W, ck, input [N-1:0] IN, output reg [N-1:0] OUT);
    always@(posedge ck)
        if(W)
            OUT<=IN;
endmodule
```



//declaración del tipo módulo correspondiente a AC (instanciaremos a un módulo type1 y añadiremos lectura condicional)

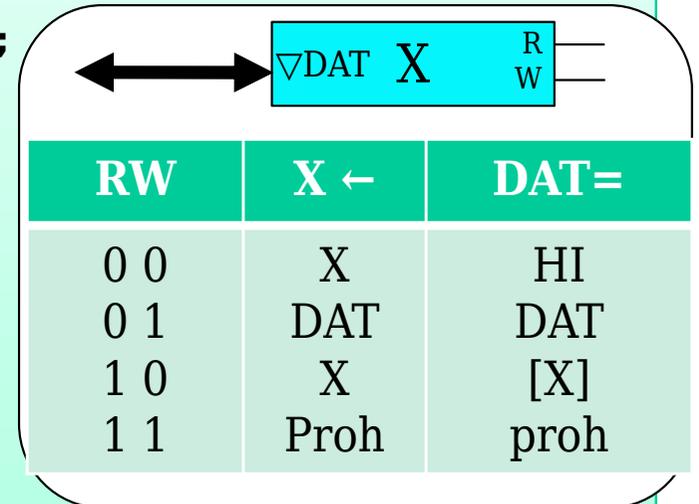
```
module type2 #(parameter N=8)
    (input W, R, ck, input [N-1:0] IN, output [N-1:0] OUT);
    wire [N-1:0] internal_bus;
    type1 #(N) internal_reg(W, ck, IN, internal_bus);
    assign OUT = R ? internal_bus : 'bz;
endmodule
```



# Descripción Verilog de la unidad de datos de la calculadora

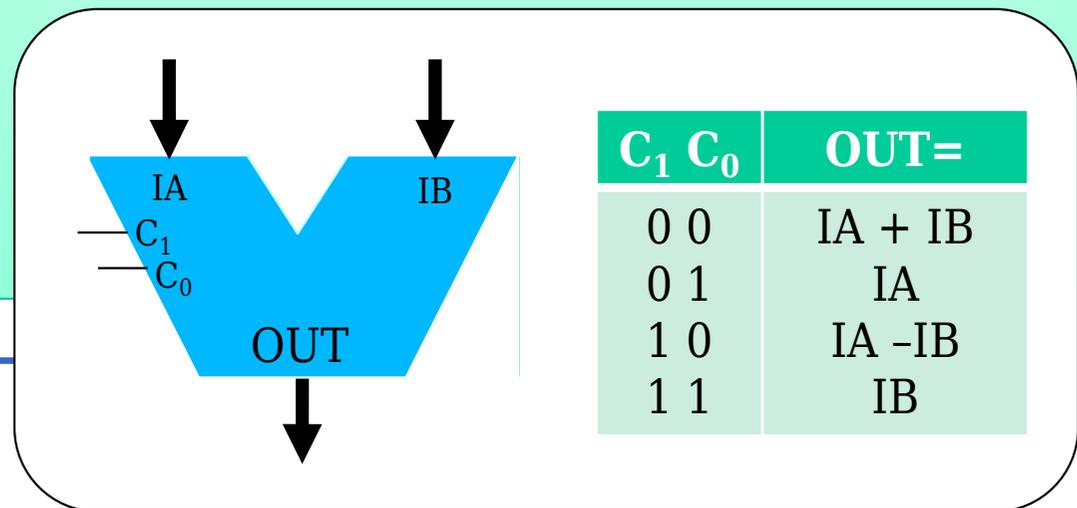
//declaración del tipo módulo correspondiente a RA y RB, (instanciamos al módulo type2)

```
module type3 #(parameter N=8) (input W, R, ck, inout [N-1:0] DAT);
    type2 #(N) internal_register(W,R,ck,DAT,DAT);
endmodule
```



//declaración del tipo módulo correspondiente a la ALU

```
module ALU_type #(parameter N=8)
    (input C1, C0, input [N-1:0] IA,IB, output reg [N-1:0] OUT);
    always@(*)
        case({C1,C0})
            2'b00: OUT=IA+IB;
            2'b01: OUT=IA;
            2'b10: OUT=IA-IB;
            2'b11: OUT=IB;
        endcase
endmodule
```



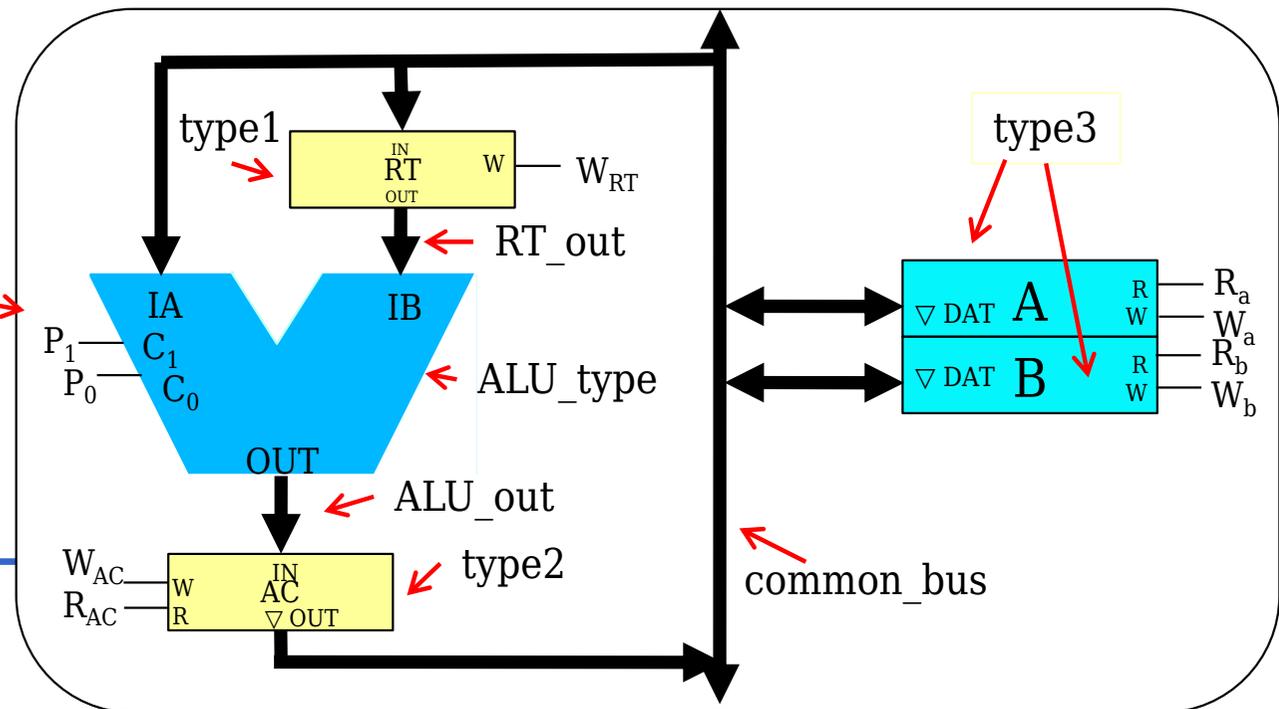
# Descripción Verilog de la unidad de datos de la calculadora

//declaración de la unidad de procesamiento de datos

```
module unidad_datos #(parameter N=8) (input ck,WAC,RAC,WRT,Ra,Rb,Wa,Wb,P0,P1);  
  wire [N-1:0] common_bus, ALU_out, RT_out;  
  type1 #(N) RT(WRT,ck,common_bus,RT_out);  
  type2 #(N) AC(WAC,RAC,ck,ALU_out,common_bus);  
  type3 #(N) A(Wa,Ra,ck,common_bus);  
  type3 #(N) B(Wb,Rb,ck,common_bus);  
  ALU_type #(N) ALU(P1,P0,common_bus,RT_out,ALU_out);  
endmodule
```

Se han utilizado instancias de los módulos definidos anteriormente: type1, type2, type3 y alu\_type

Se han nombrado los buses internos de la unidad: RT\_out, ALU\_out y common\_bus



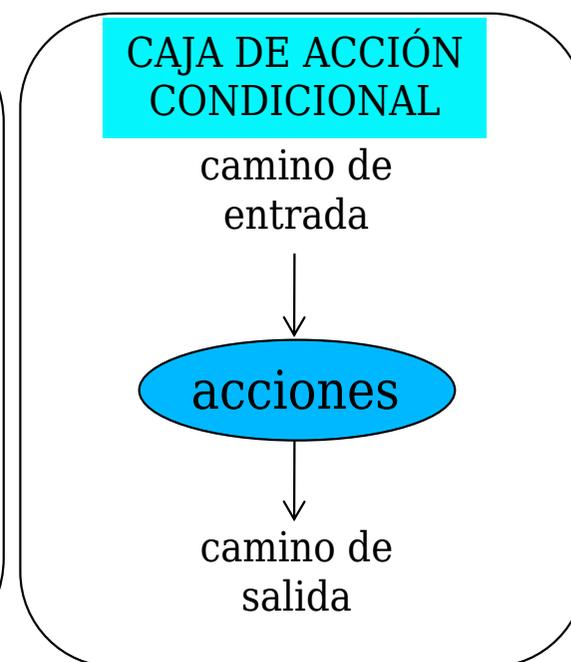
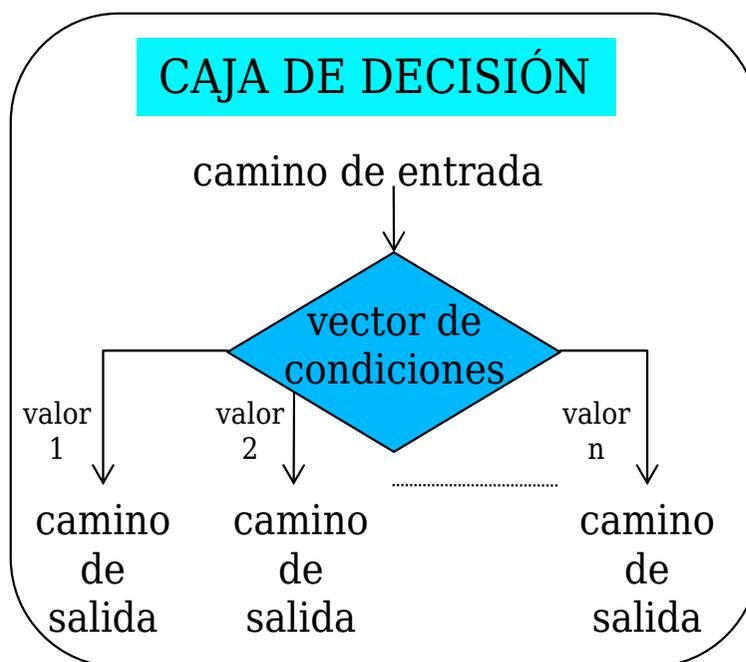
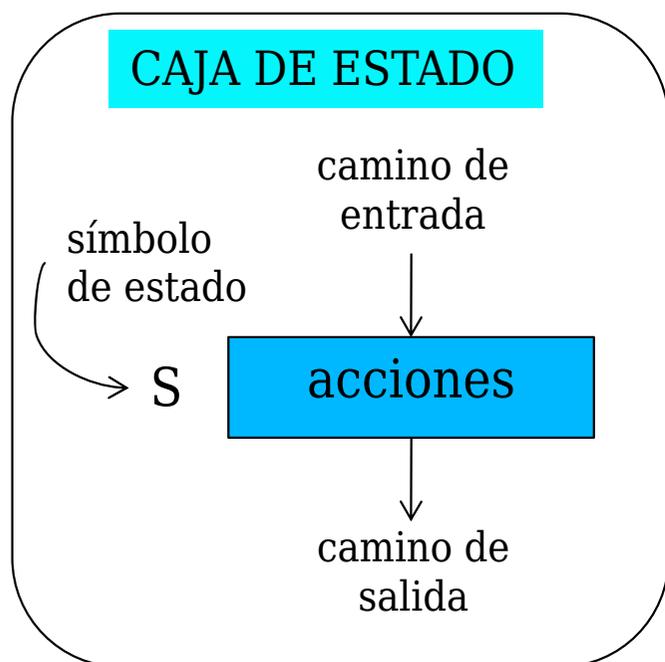
# Descomposición en microoperaciones

- ▶ Paso 4 -Descomponemos las macrooperaciones en **microoperaciones**.
- ▶ Durante la ejecución de una macrooperación pueden modificarse los registros ocultos y solo los registros visibles que aparezcan como destino en la descripción de la macrooperación.

	<b>A ← A + B</b>	<b>B ← A + B</b>	<b>A ← A - B</b>	<b>B ← A - B</b>
μop 1	RT ← B			
μop 2	AC ← A + RT		AC ← A - RT	
μop 3	A ← AC	B ← AC	A ← AC	B ← AC

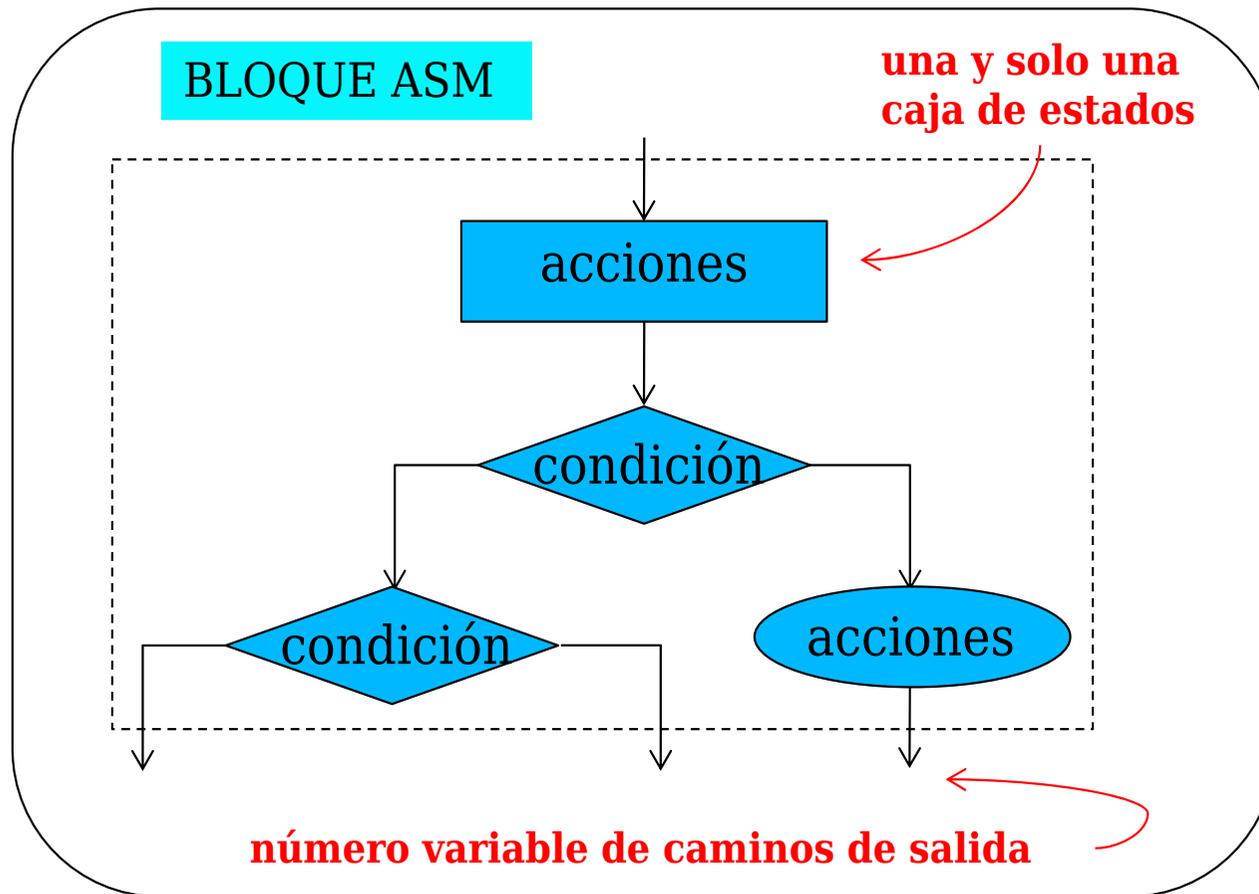
# Descripción mediante cartas ASM

## ► Definiciones



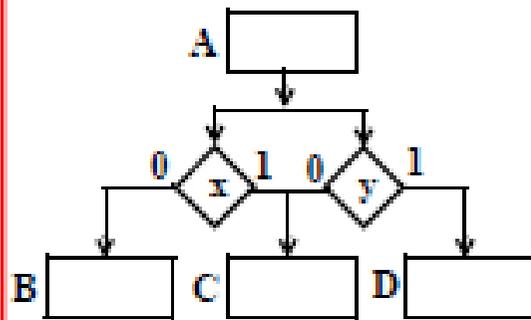
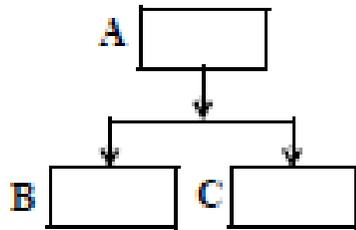
# Descripción mediante cartas ASM

## ► Definiciones

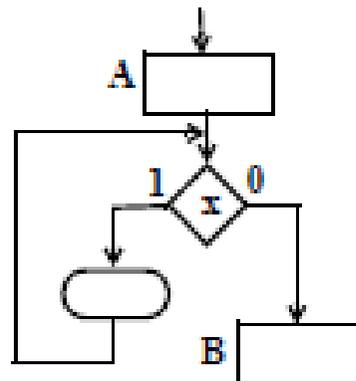
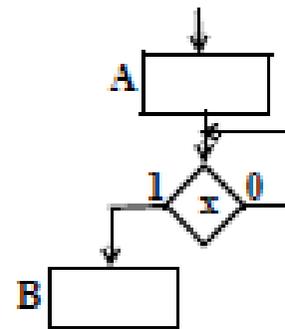


# Errores comunes en cartas ASM

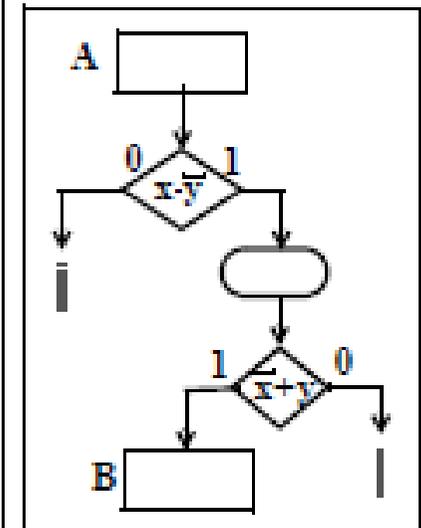
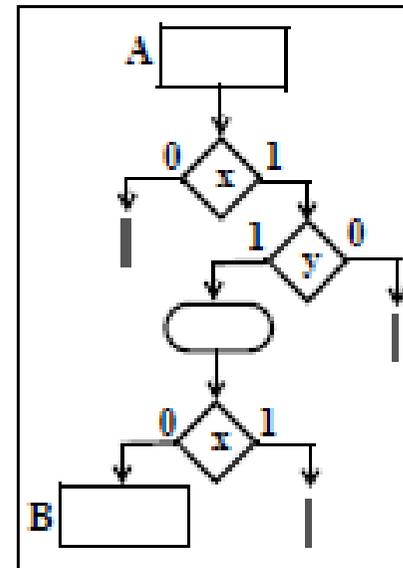
Próximo estado sin determinar



Cerrar lazos sin cajas de estado

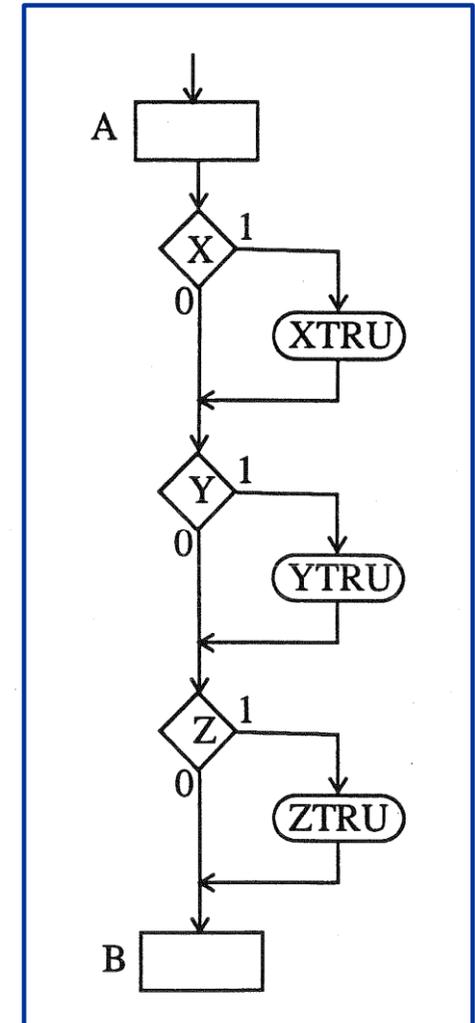
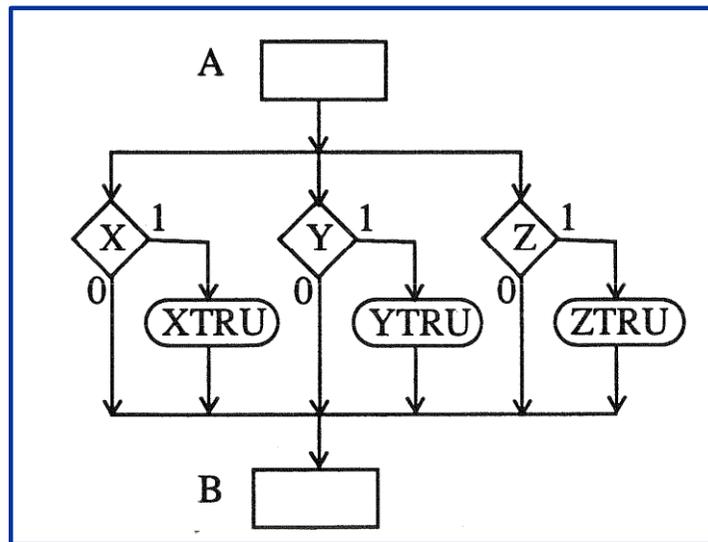


No garantizar la posibilidad lógica de **todos** los caminos



# Consideraciones temporales

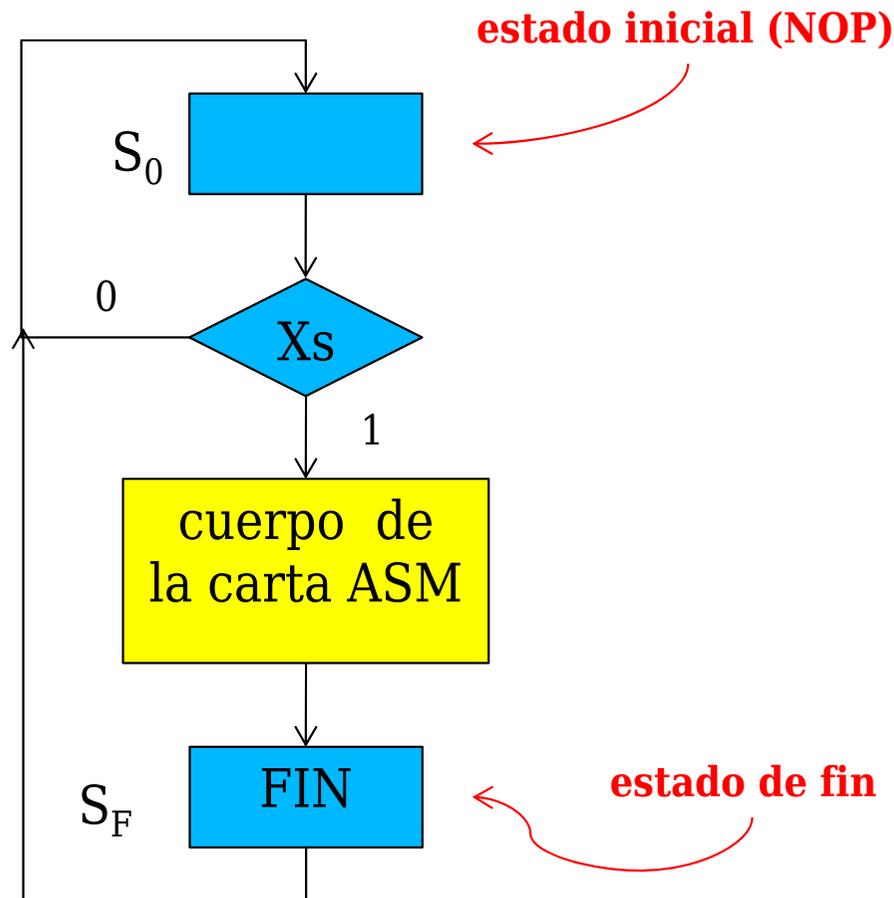
- ▶ El orden de las cajas en un bloque ASM **no implica** orden temporal.
- ▶ Todas las tareas de un bloque ASM se hacen en un ciclo de reloj



- ▶ Igual significado lógico

# Descripción mediante cartas ASM

## ► Inicio y fin de operación

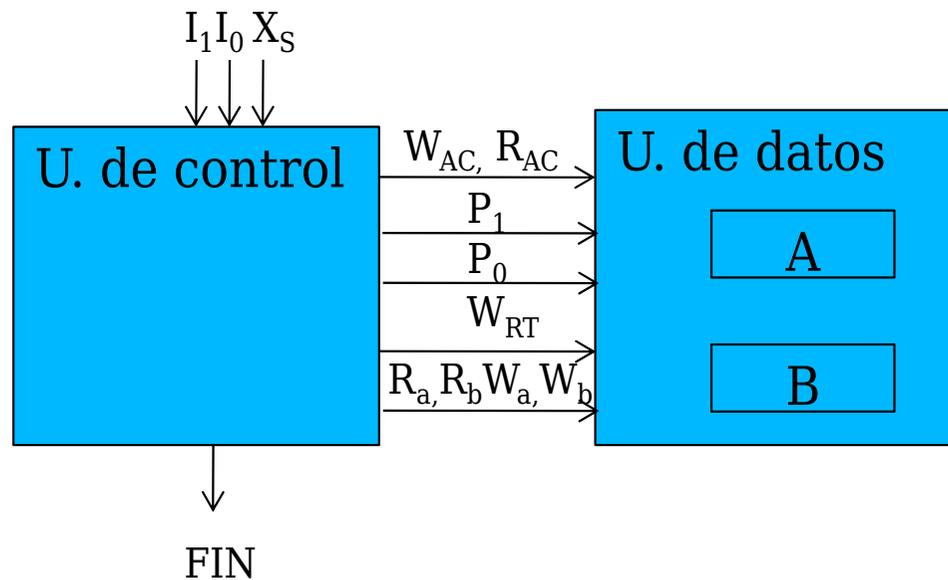


$X_s$ : **entrada** con la que se inicia la operación (Xstart)

FIN: **salida** que indica que la operación ha terminado

# Diseño de la calculadora simple

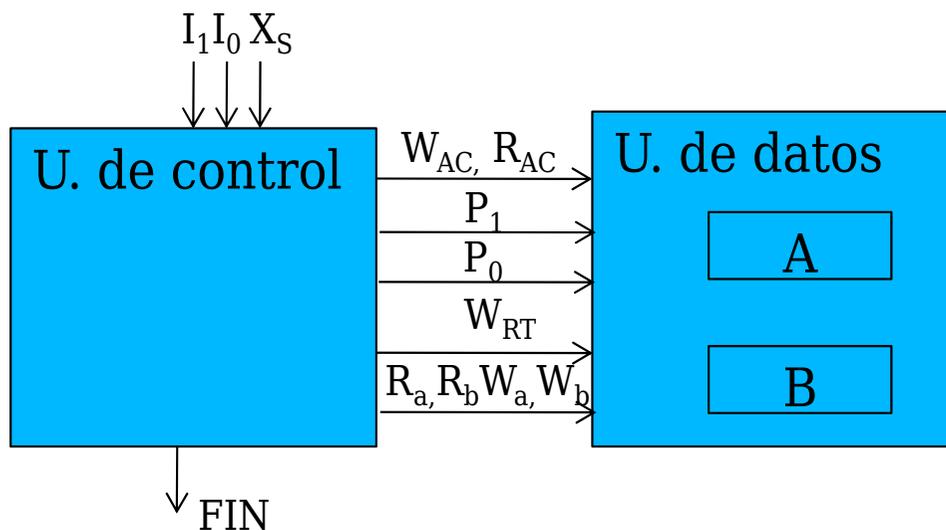
- Organización del sistema digital:



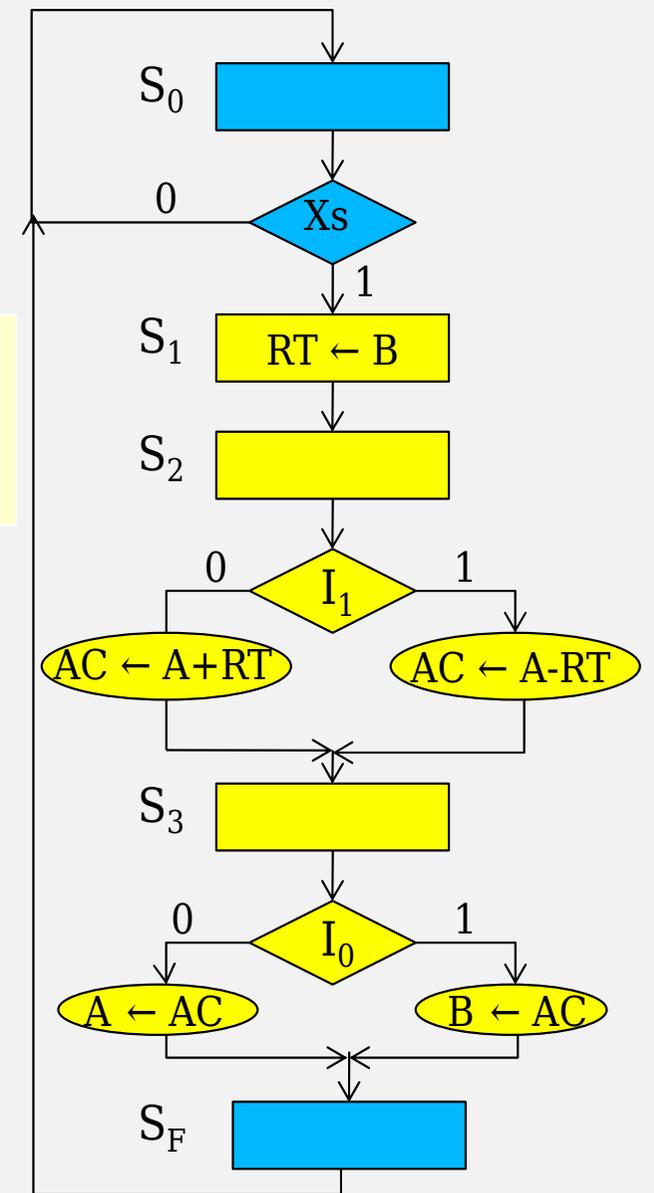
- El usuario especifica la operación proporcionando el valor de  $I_1, I_0$  y genera la orden de comienzo con  $X_S$

# Carta ASM de la calculadora

	$A \leftarrow A + B$ $I_1 I_0 = 00$	$B \leftarrow A + B$ $I_1 I_0 = 01$	$A \leftarrow A - B$ $I_1 I_0 = 10$	$B \leftarrow A - B$ $I_1 I_0 = 11$
1	$RT \leftarrow B$			
2	$AC \leftarrow A + RT$		$AC \leftarrow A - RT$	
3	$A \leftarrow AC$	$B \leftarrow AC$	$A \leftarrow AC$	$B \leftarrow AC$

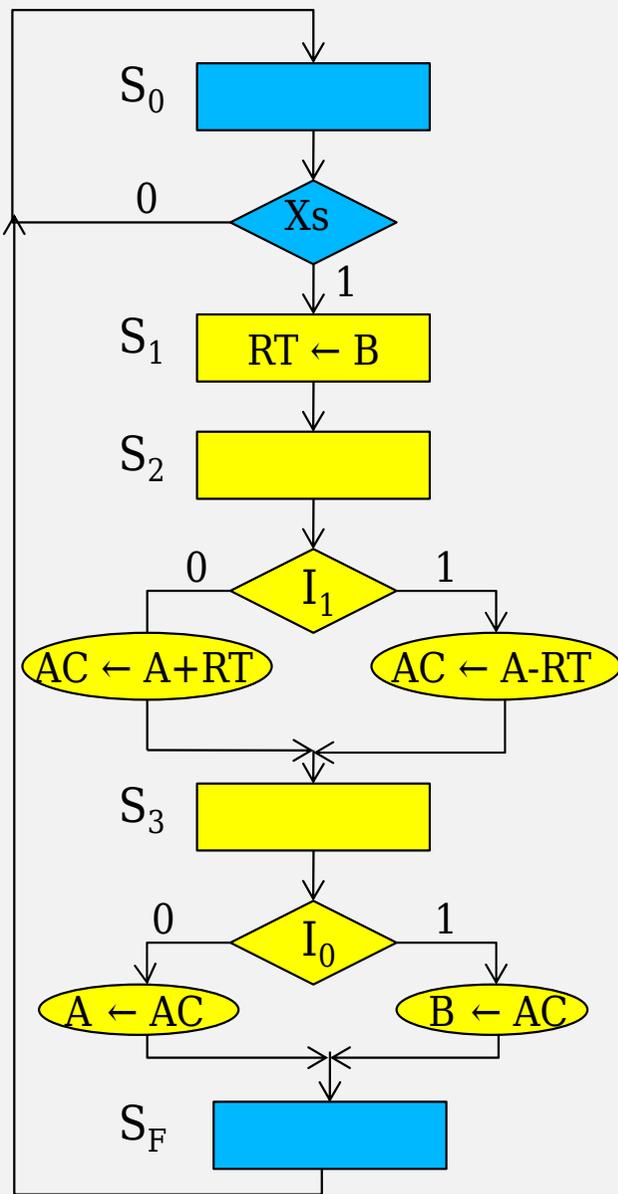


unidad de datos

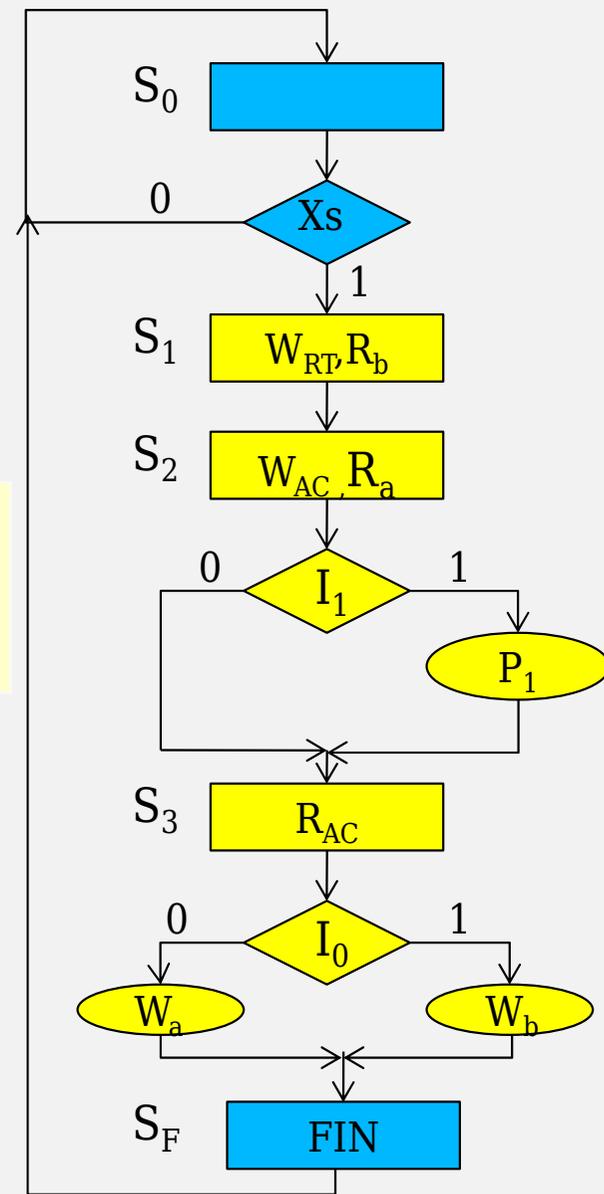


# Carta ASM de la calculadora

unidad de datos



unidad de control



---

# Descripción Verilog de la u. de control de la calculadora

- ▶ La descripción canónica de máquinas de estado en HDL Verilog es un proceso sistemático
- ▶ Se utilizará una estructura general del código en la que hay 2 procesos
  - ▶ Uno de asignación de siguientes estados
  - ▶ Otro de calculo de siguiente estado y salidas

---

# Descripción Verilog de la u. de control de la calculadora, estructura general.

```
module mi_carta_asm(  
    input LISTA_DE_ENTRADAS (incluyendo clk y reset),  
    output reg LISTA_DE_SALIDAS);  
  
// DEFINICIÓN Y ASIGNACIÓN DE ESTADOS  
parameter LISTA_DE_ESTADOS  
  
// VARIABLES PARA ALMACENAR EL ESTADO PRESENTE Y SIGUIENTE  
reg [N:0] current_state, next_state;  
  
// PROCESO DE CAMBIO DE ESTADO  
always @(posedge ck or posedge reset)  
    .....  
// PROCESO SIGUIENTE ESTADO Y SALIDA  
always @(current_state, LISTA_DE_ENTRADAS)  
    .....  
endmodule
```

---

# Descripción Verilog de la u. de control de la calculadora, procedimiento.

En la estructura general hay que completar 4 partes de código:

1. Definición y asignación de estados, según el número de estados utilizaremos más o menos bits.
2. Definición de registros para almacenar el estado actual y el siguiente. Deben ser del mismo tamaño en bits que el utilizado en el punto anterior.
3. Proceso de cambio de estado: siempre es el mismo código
4. Proceso de cálculo de siguiente estado y salida: Hay que rellenar el código correspondiente a la carta ASM

# Descripción Verilog de la u. de control de la calculadora

```

module unidad_control(
  input ck, XS, I0, I1, reset,
  output reg RAC, Rb, Ra, WRT, WAC, Wa, Wb, P1, P0, FIN
);

```

```

parameter S0 = 3'b000,
  S1 = 3'b001,
  S2 = 3'b010,
  S3 = 3'b011,
  SF = 3'b100;

```

Asignación de estados

```

reg [2:0] current_state,next_state;

```

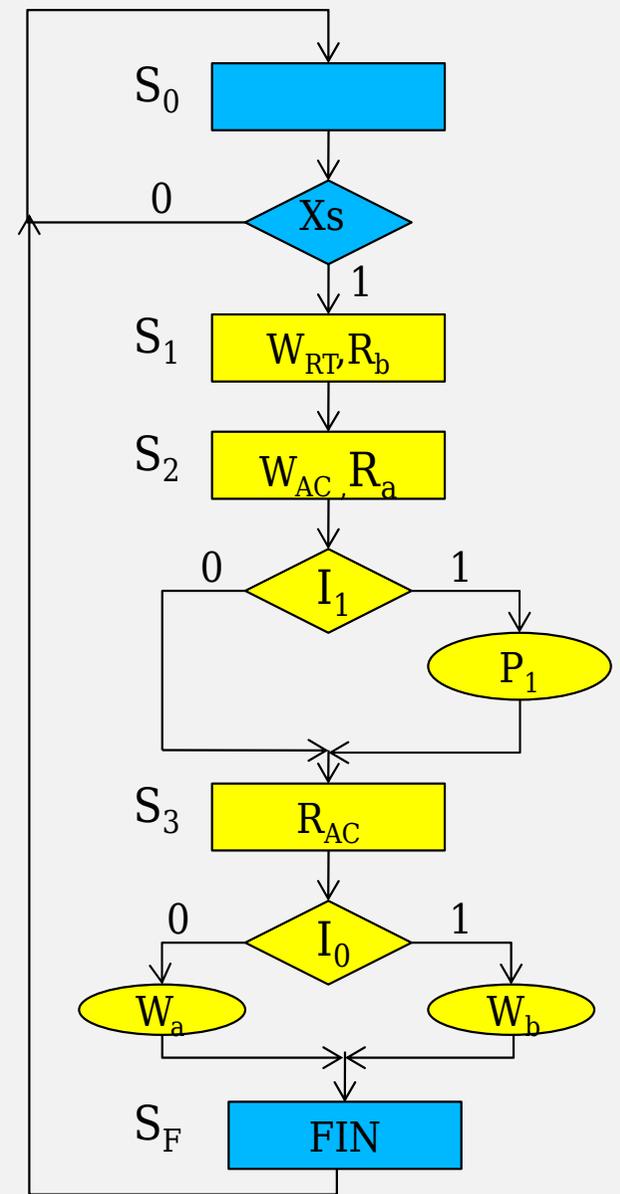
```

always @(posedge ck or posedge reset)
begin
  if(reset)
    current_state <= S0;
  else
    current_state <= next_state;
end

```

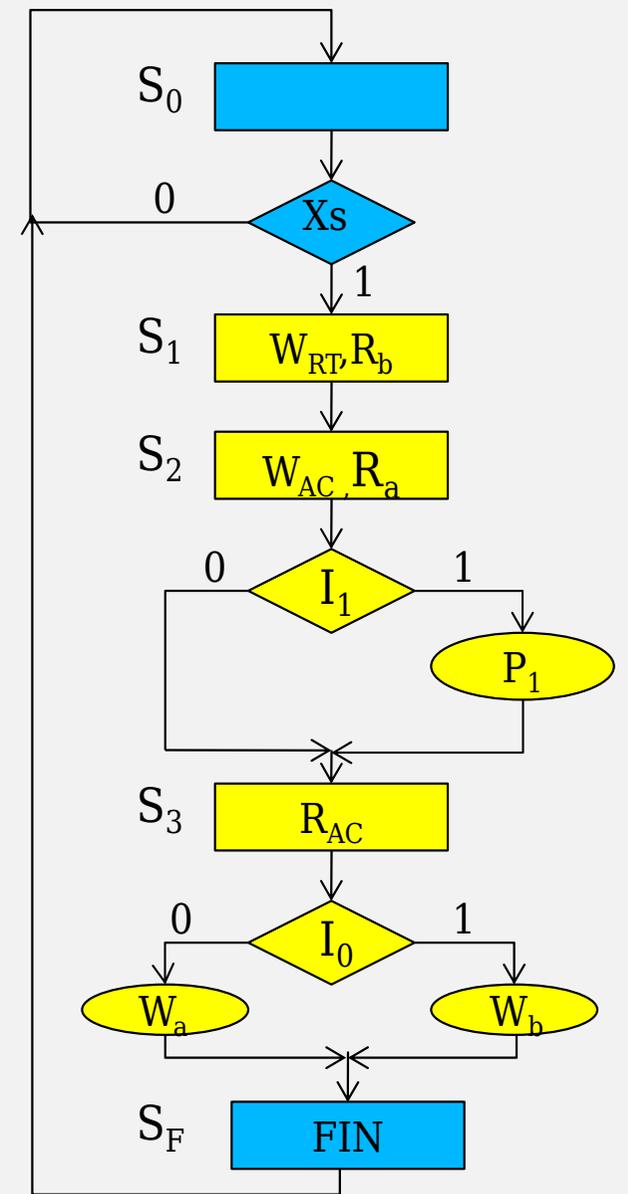
Proceso siguiente estado

SIGUE ->



# Descripción Verilog de la u. de control de la calculadora

- ▶ El proceso de cálculo del siguiente estado y salida se realiza con una única sentencia “CASE”
- ▶ La sentencia “CASE” debe contemplar todos los estados de la carta ASM
- ▶ Antes de la sentencia “CASE” se recomienda establecer por defecto a cero todas las salidas y next\_state a S0



# Descripción Verilog de la u. de control de la calculadora

```

always @(current_state,I0,I1,XS)
begin
  RAC = 0;
  WAC = 0;
  Ra = 0;
  Rb = 0;
  P0 = 0;
  P1 = 0;
  WRT = 0;
  Wa = 0;
  Wb = 0;
  FIN = 0;
  next_state = S0;
  case(current_state)
    S0:
      if(XS) next_state = S1;
    S1:
      begin
        WRT = 1;
        Rb = 1;
        next_state = S2;
      end
  end

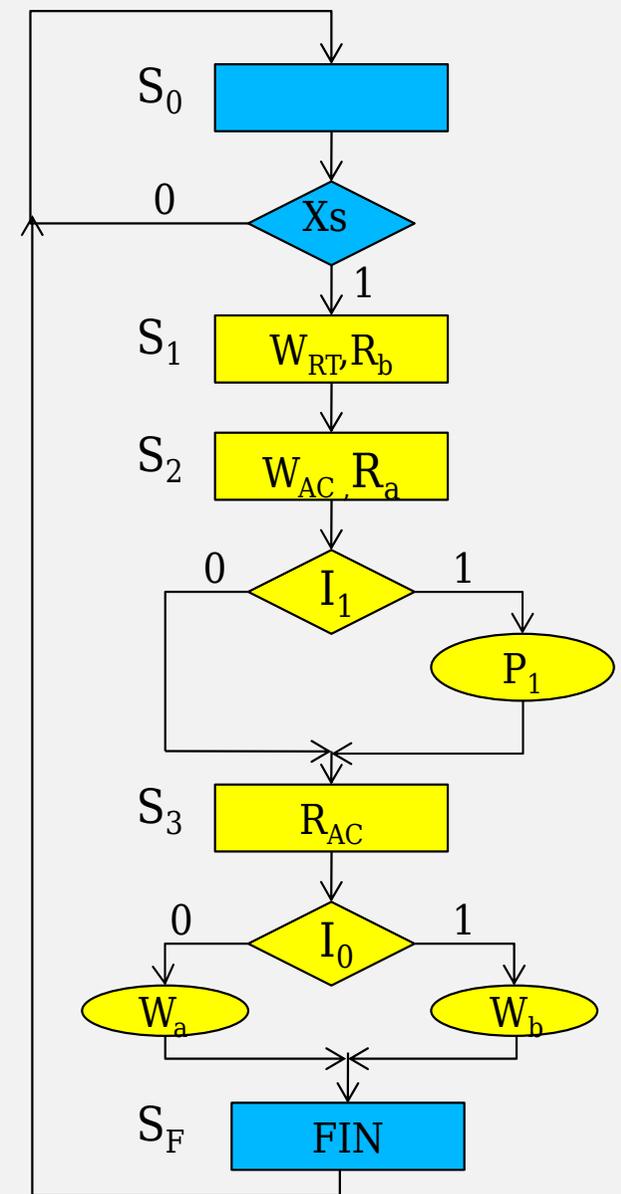
```

Valor por defecto de las salidas establecido a cero

Valor por defecto del estado: S0

Estado S0

Estado S1



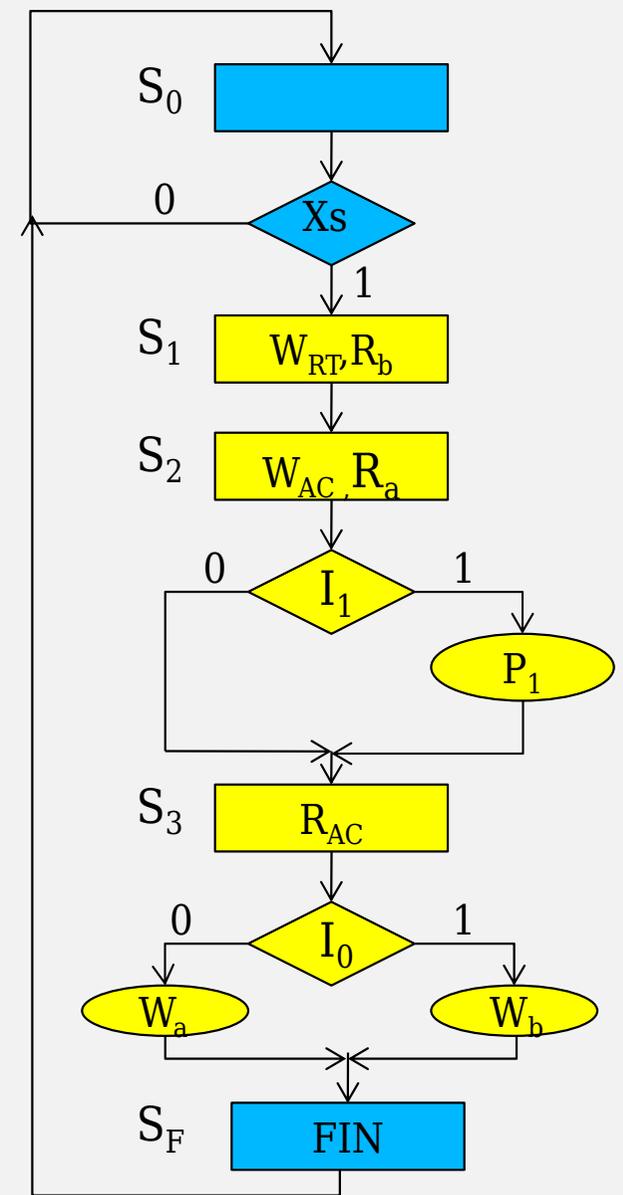
# Descripción Verilog de la u. de control de la calculadora

```
S2:  
begin  
  WAC = 1;  
  Ra = 1;  
  if(I1)  
    P1 = 1;  
  next_state = S3;  
end  
S3:  
begin  
  RAC = 1;  
  if(I0)  
    Wb = 1;  
  else  
    Wa = 1;  
  next_state = SF;  
end  
SF:  
  FIN = 1;  
endcase  
end  
endmodule
```

Estado S2

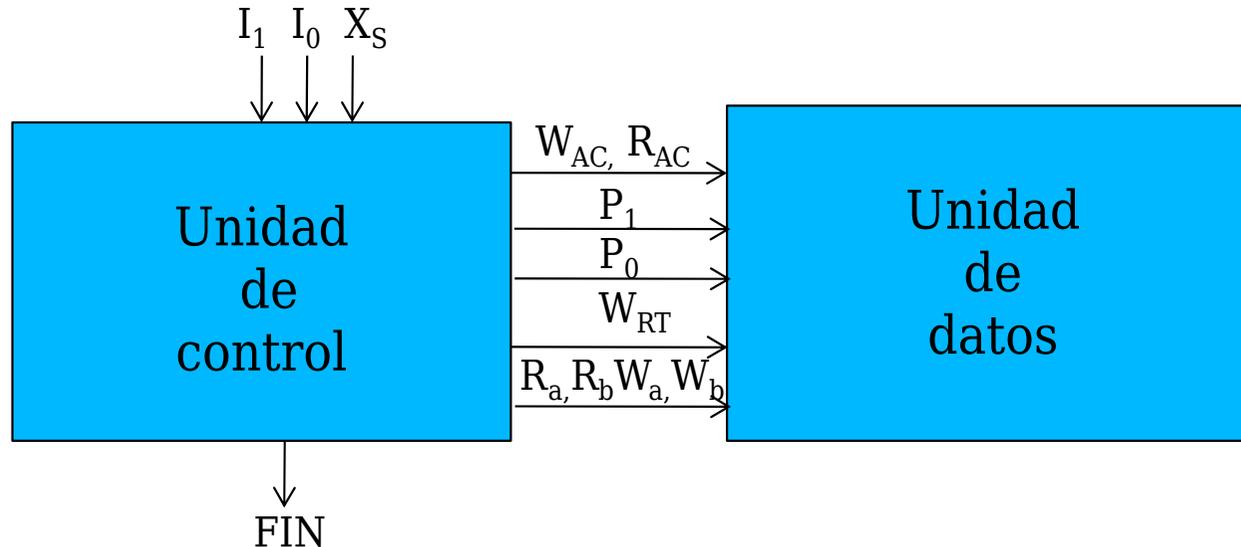
Estado S3

Estado SF



# Diseño de la calculadora simple

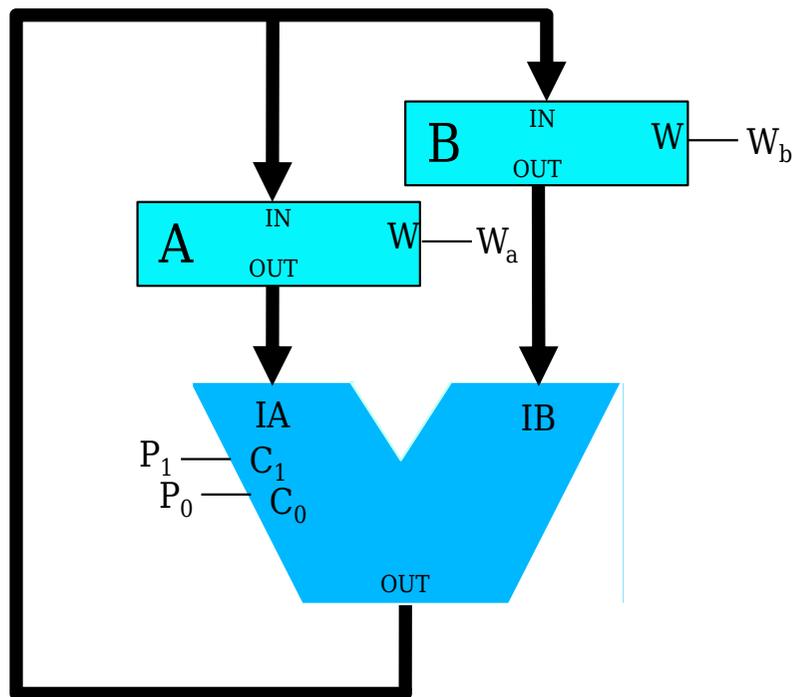
- Conexión de unidades de datos y de control:



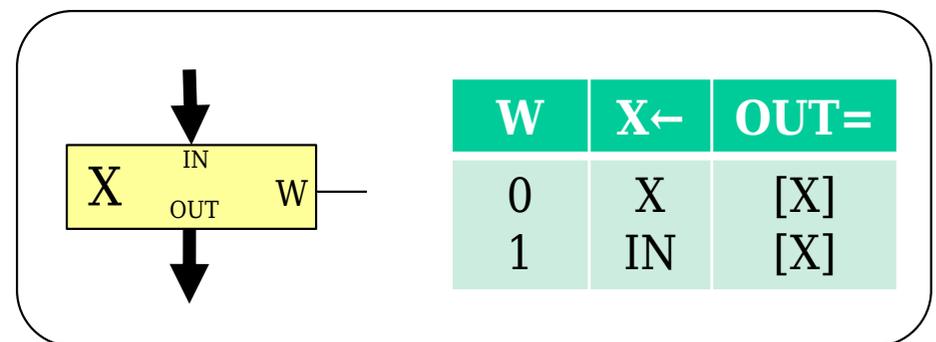
```
module calculadora #(parameter N=8) ( input ck, XS, I0, I1, reset, output FIN);  
  
    wire w1,w2,w3,w4,w5,w6,w7,w8,w9;  
  
    unidad_datos #(N) ud_calc (.ck(ck),.WAC(w1),.RAC(w2),.WRT(w3),  
        .Ra(w4),.Rb(w5),.Wa(w6),.Wb(w7),.P0(w8),.P1(w9));  
  
    unidad_control uc_calc(.ck(ck),.reset(reset),.XS(XS),.FIN(FIN),.I0(I0),.I1(I1),  
        .WAC(w1),.RAC(w2),.WRT(w3),.Ra(w4),.Rb(w5),  
        .Wa(w6),.Wb(w7),.P0(w8),.P1(w9));  
  
endmodule
```

# Diseño de la unidad de datos de una calculadora: solución con 3 buses

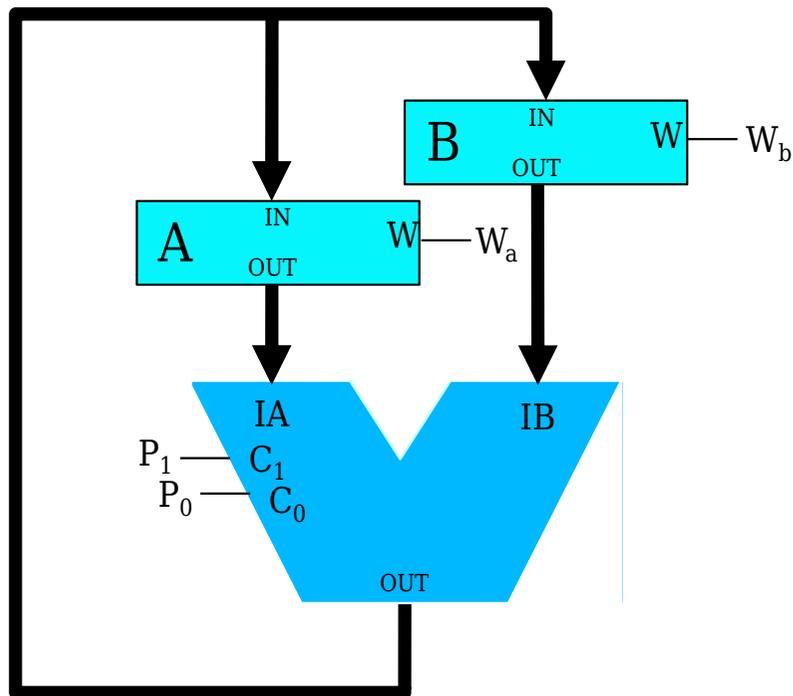
- ▶ Para las mismas especificaciones del ejemplo anterior proponemos una unidad de datos diferente.



- ▶ Arquitectura específica.
- ▶ Con esta arquitectura se necesitan menos registros.



# Descripción Verilog de la u. datos de la calculadora : solución con 3 buses



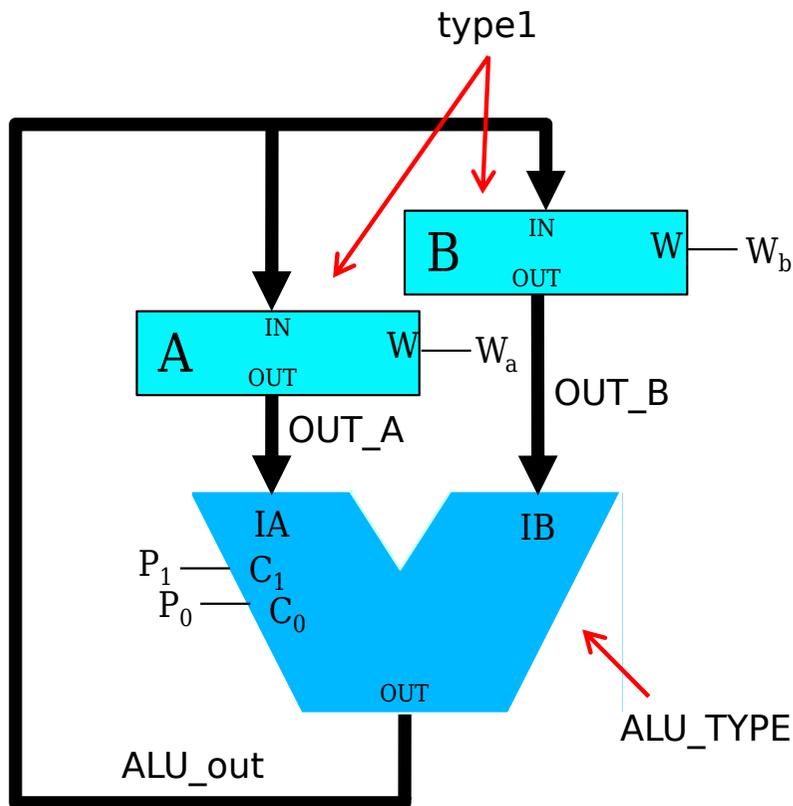
```
//declaración del tipo módulo correspondiente a RA y RB
```

```
module type1 #(parameter N=8)
  (input W, ck, input [N-1:0] IN,
  output reg [N-1:0] OUT);
  always@(posedge ck)
    if(W)
      OUT<=IN;
endmodule
```

```
//declaración del tipo módulo correspondiente a la ALU
```

```
module ALU_type #(parameter N=8)
  ( input C1, C0, input [N-1:0] IA,IB,
  output reg [N-1:0] OUT);
  always@(*)
    case({C1,C0})
      2'b00: OUT=IA+IB;
      2'b01: OUT=IA;
      2'b10: OUT=IA-IB;
      2'b11: OUT=IB;
    endcase
endmodule
```

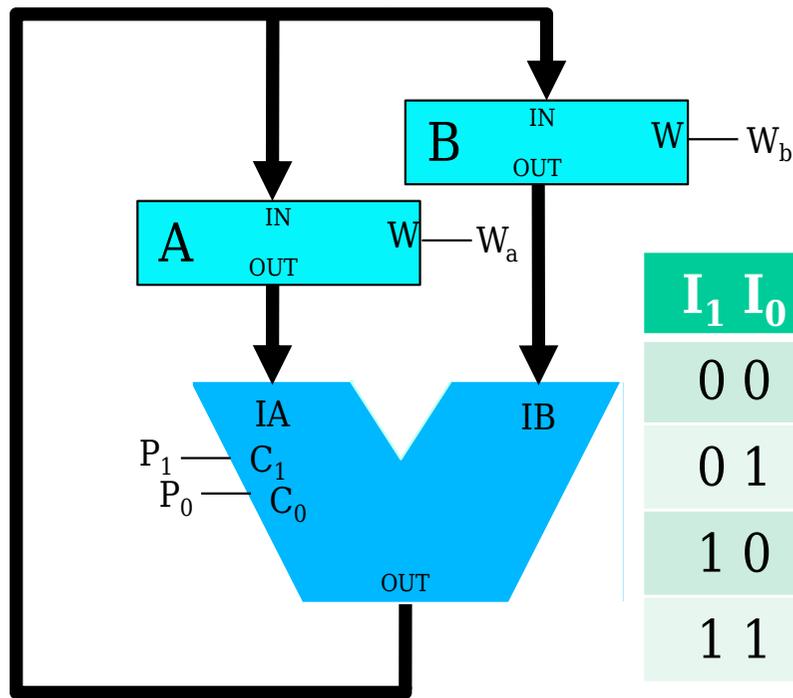
# Descripción Verilog de la u. datos de la calculadora : solución con 3 buses



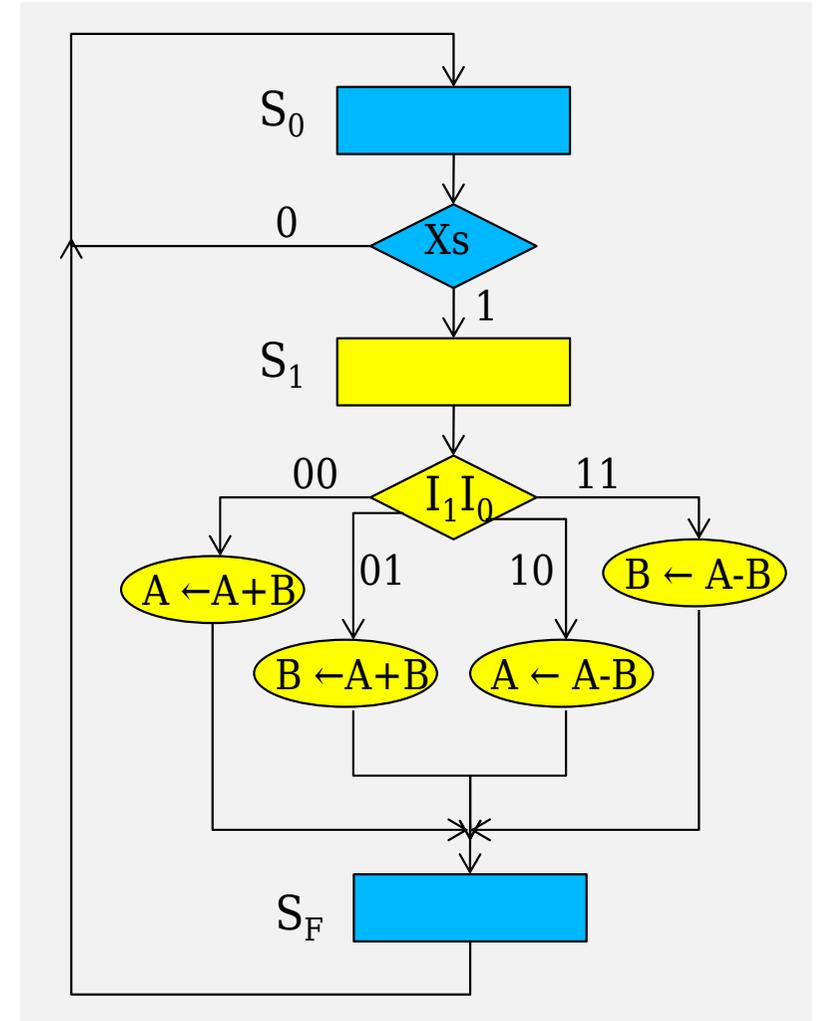
```
//declaración de la unidad de procesado de datos  
module unidad_datos2 #(parameter N=8)  
    (input ck,Wa,Wb,P0,P1);  
  
    wire [N-1:0] ALU_out, OUT_A, OUT_B;  
  
    type1 #(N) A(Wa,ck,ALU_out,OUT_A);  
    type1 #(N) B(Wb,ck,ALU_out,OUT_B);  
    ALU_type #(N) ALU(P1,P0,OUT_A,OUT_B,ALU_out);  
  
endmodule
```

# Carta ASM: solución con 3 buses

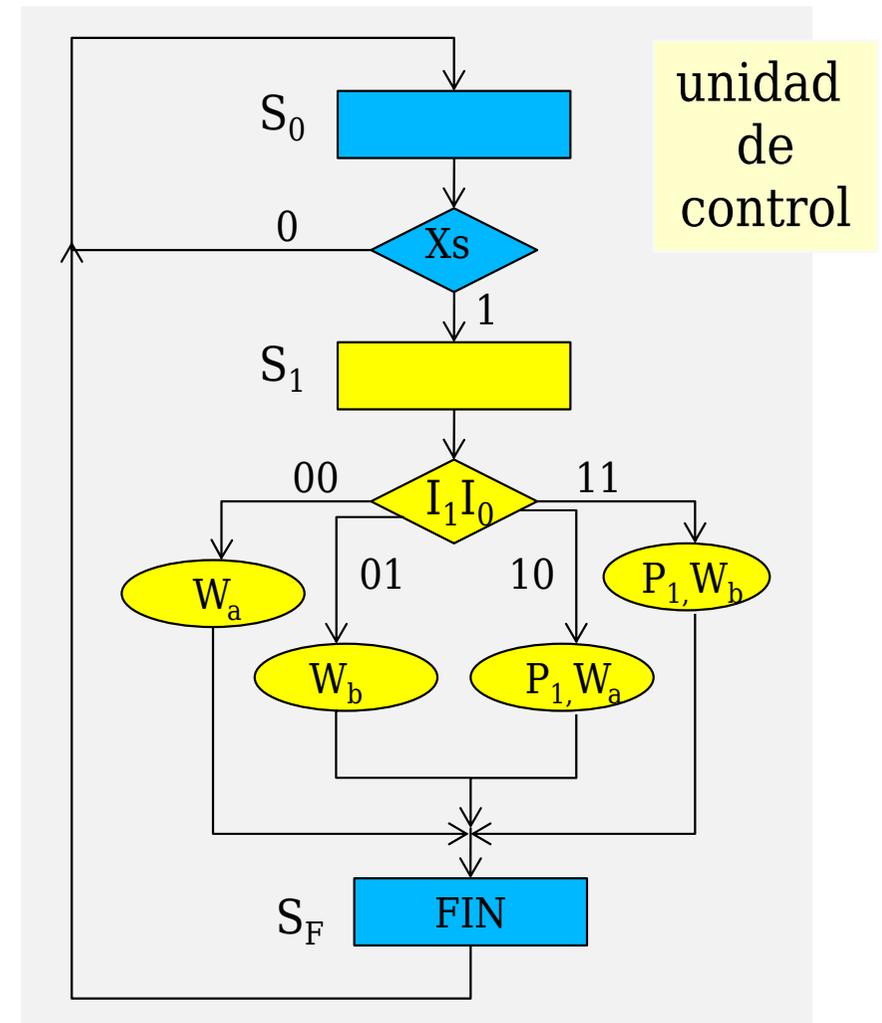
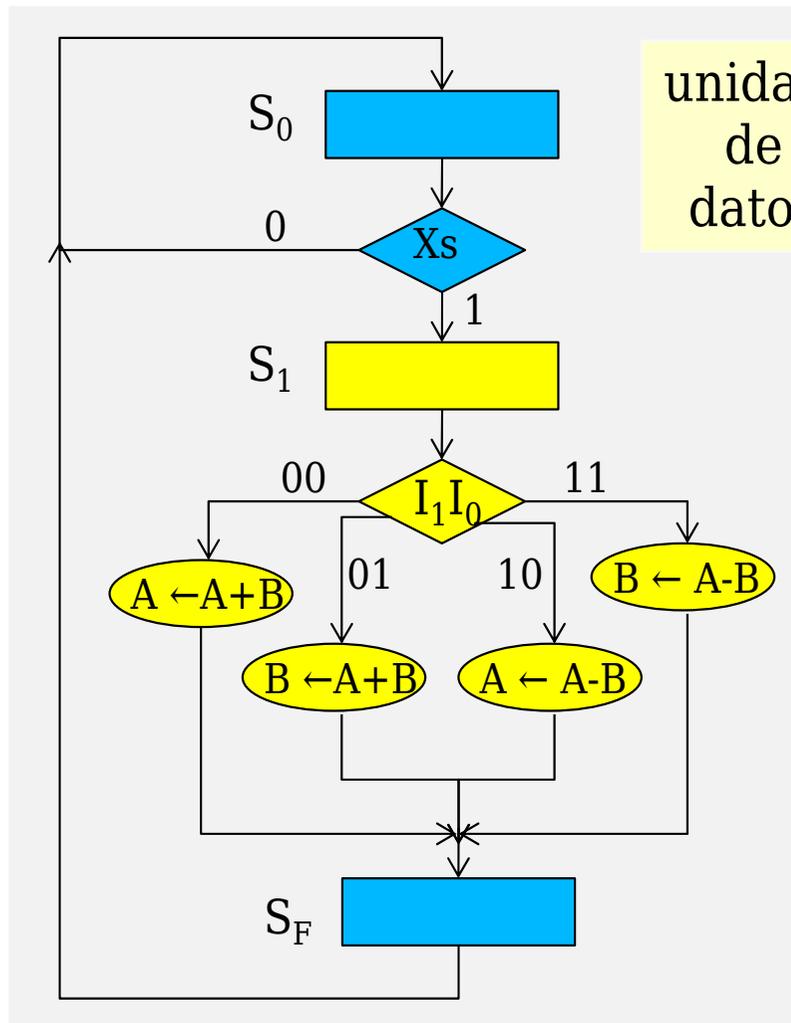
- Las macrooperaciones se realizan en un único ciclo de reloj.



$I_1$	$I_0$	operación
0	0	$A \leftarrow A + B$
0	1	$B \leftarrow A + B$
1	0	$A \leftarrow A - B$
1	1	$B \leftarrow A - B$



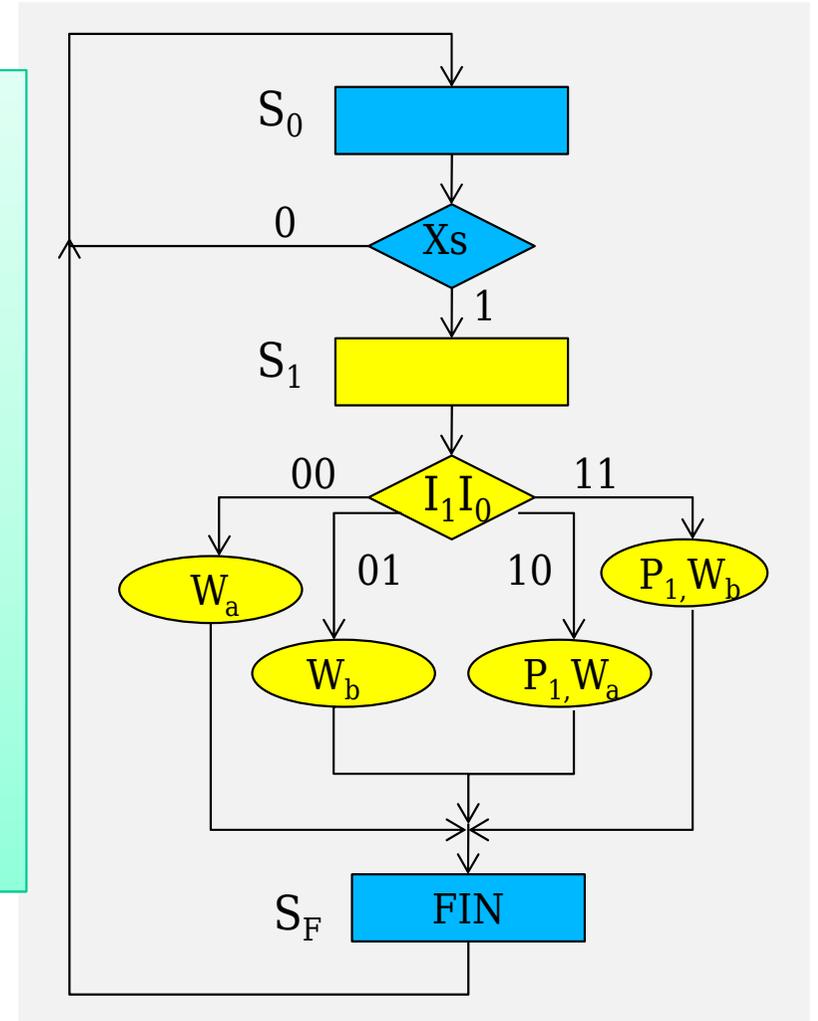
# Carta ASM: solución con 3 buses



# Descripción Verilog de la u. de control de la calculadora: solución con 3 buses

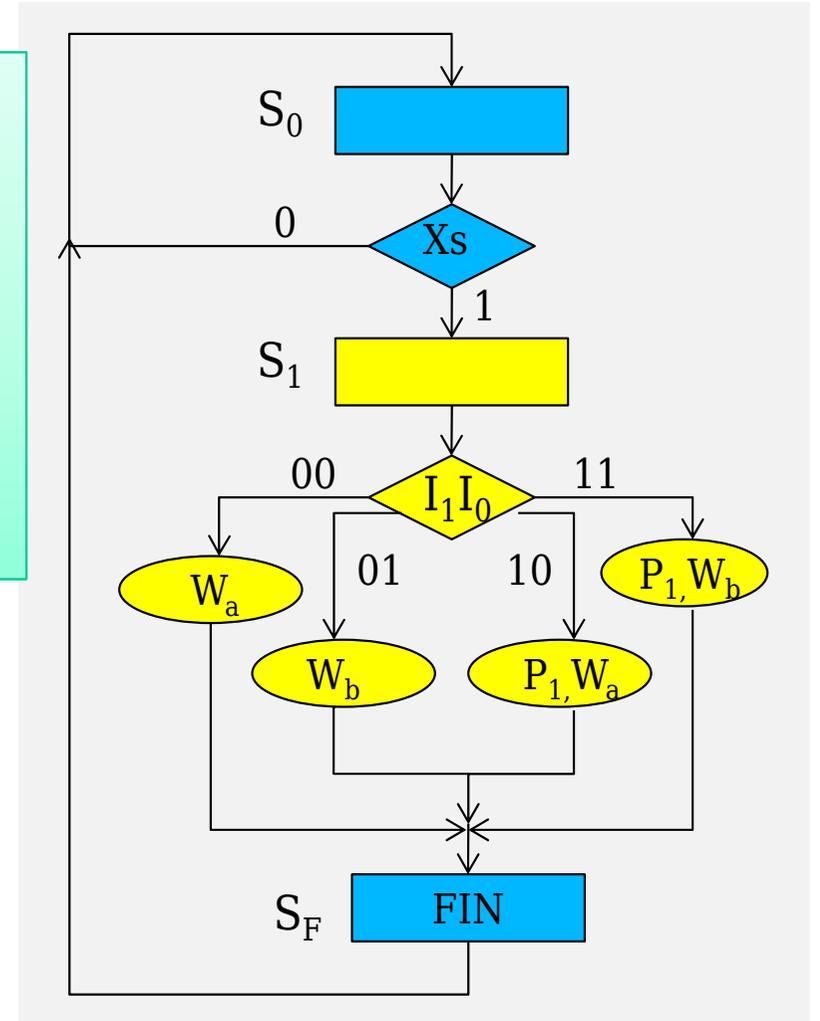
```
module unidad_control2(  
  input ck, XS, I0, I1, reset,  
  output reg Wa, Wb, P1, P0, FIN  
);  
parameter S0 = 2'b00,  
        S1 = 2'b01,  
        SF = 2'b10;  
  
reg [1:0] current_state,next_state;  
  
always @(posedge ck or posedge reset)  
begin  
  if(reset)  
    current_state <= S0;  
  else  
    current_state <= next_state;  
end
```

**SIGUE ->**



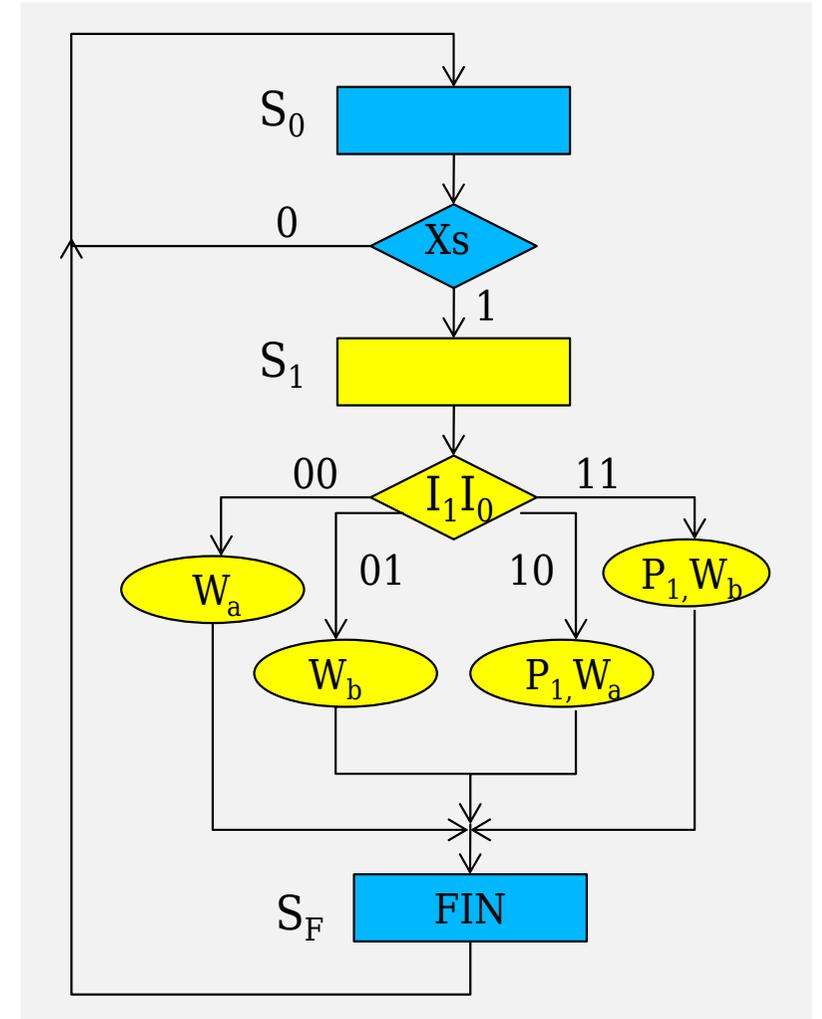
# Descripción Verilog de la u. de control de la calculadora: solución con 3 buses

```
always @(current_state,I0,I1,XS)
begin
  P1 = 0;
  P0 = 0;
  Wa = 0;
  Wb = 0;
  FIN = 0;
  next_state = S0;
  case(current_state)
  S0:
    if (XS) next_state = S1;
    SIGUE ->
```



# Descripción Verilog de la u. de control de la calculadora: solución con 3 buses

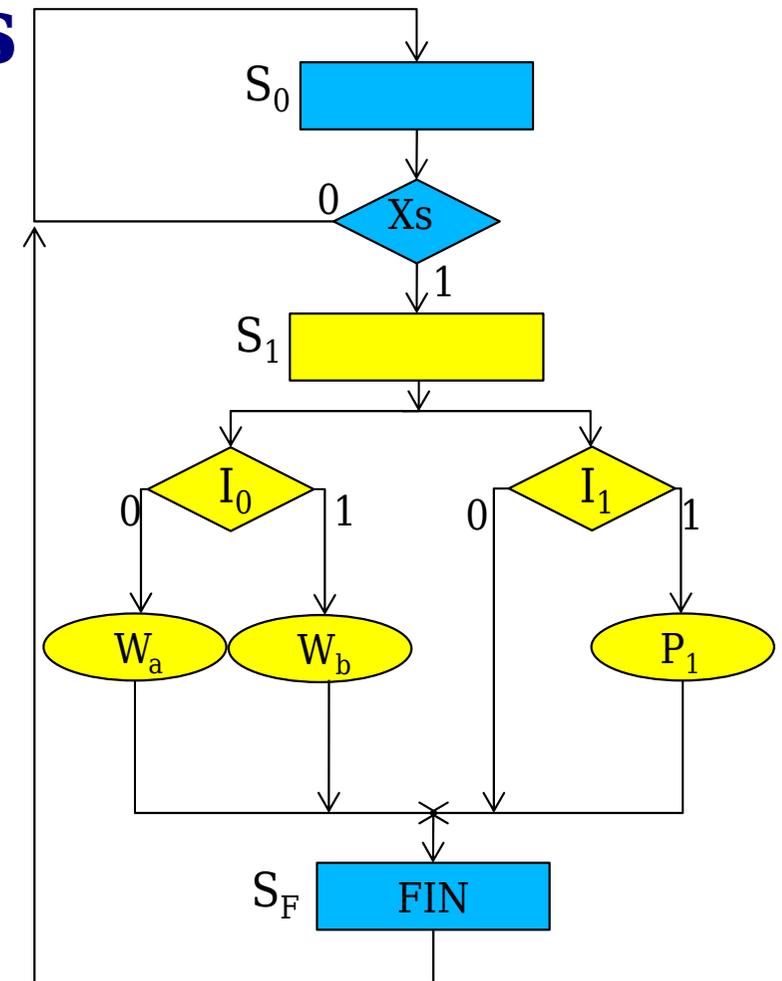
```
S1:  
begin  
if(I1==0 && I0==0)  
  Wa = 1;  
else if(I1==0 && I0==1)  
  Wb = 1;  
  else if(I1==1 && I0==0)  
    begin  
      P1 = 1;  
      Wa = 1;  
    end  
  else  
    begin  
      P1 = 1;  
      Wb = 1;  
    end  
next_state = SF;  
end  
SF:  
FIN = 1;  
endcase  
end  
endmodule
```



# Descripción Verilog (compacta) de la u. de control de la calculadora: solución con 3 buses

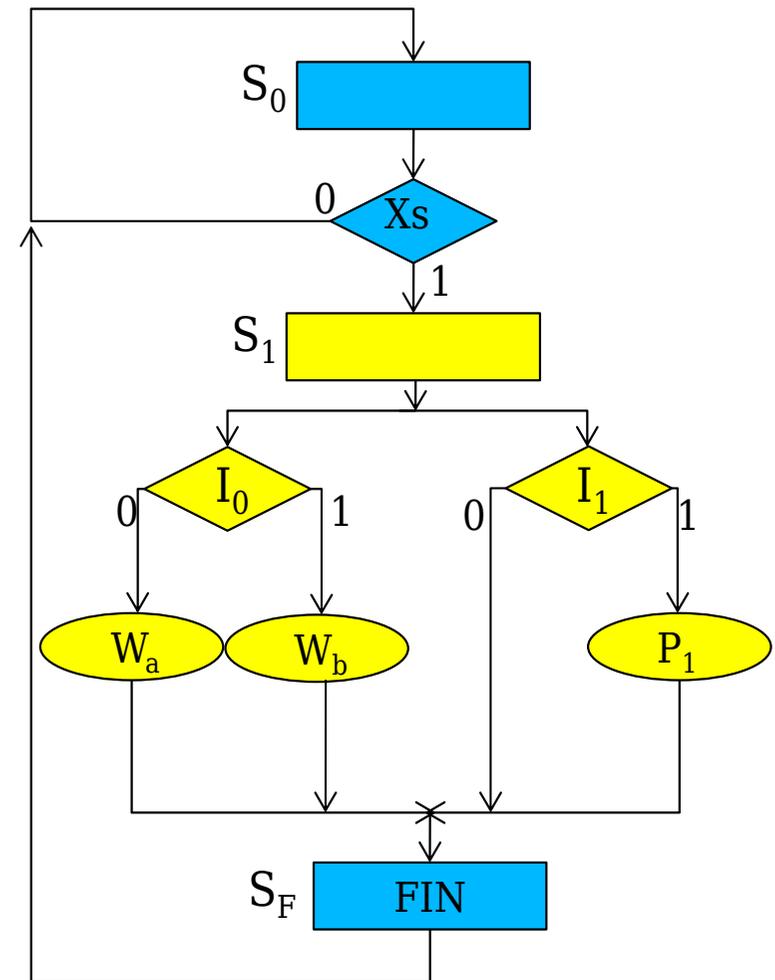
```
module unidad_control2(  
  input ck, XS, I0, I1, reset,  
  output reg P0,P1,Wa,Wb,FIN  
);  
parameter S0 = 2'b00,  
        S1 = 2'b01,  
        SF = 2'b10;  
  
reg [1:0] current_state,next_state;  
  
always @(posedge ck or posedge reset)  
begin  
  if(reset)  
    current_state <= S0;  
  else  
    current_state <= next_state;  
end
```

SIGUE ->



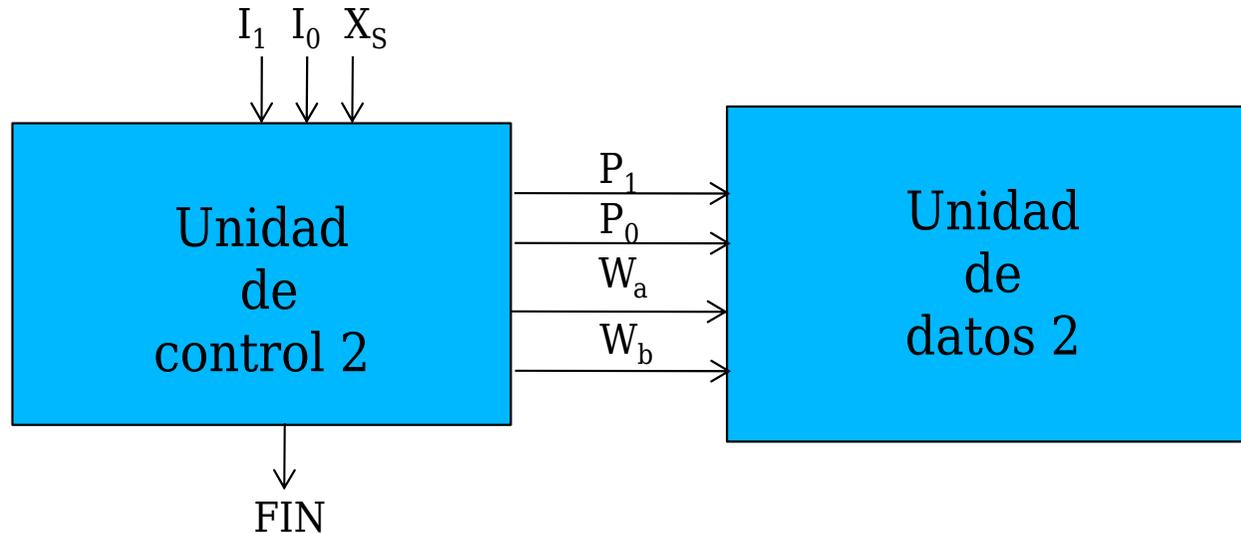
# Descripción Verilog (compacta) ...

```
always @(current_state,I0,I1,XS)
begin
  P0 = 0;
  P1 = 0;
  FIN = 0;
  Wa = 0;
  Wb = 0;
  next_state = S0;
  case(current_state)
  S0:
    if(XS) next_state = S1;
  S1:
    begin
      if(I0)
        Wb = 1;
      else
        Wa = 1;
      if(i1)
        P1 = 1;
      next_state = SF;
    end
  SF:
    FIN = 1;
  endcase
end
endmodule
```



# Diseño de la calculadora simple: solución con 3 buses

- ▶ Conexión de unidades de datos y de control:



```
module calculadora #(parameter N=8) ( input ck, XS, I0, I1, reset, output FIN);  
  
    wire w1,w2,w3,w4;  
  
    unidad_datos2 #(N) ud_calc (.ck(ck),.P1(w1),.P0(w2),.Wa(w3),.Wb(w4));  
  
    unidad_control2 uc_calc(.ck(ck),.reset(reset),.XS(XS),.FIN(FIN),.I0(I0),.I1(I1),  
        .P1(w1),.P0(w2),.Wa(w3),.Wb(w4));  
  
endmodule
```

# Diseño de una calculadora con 8 registros

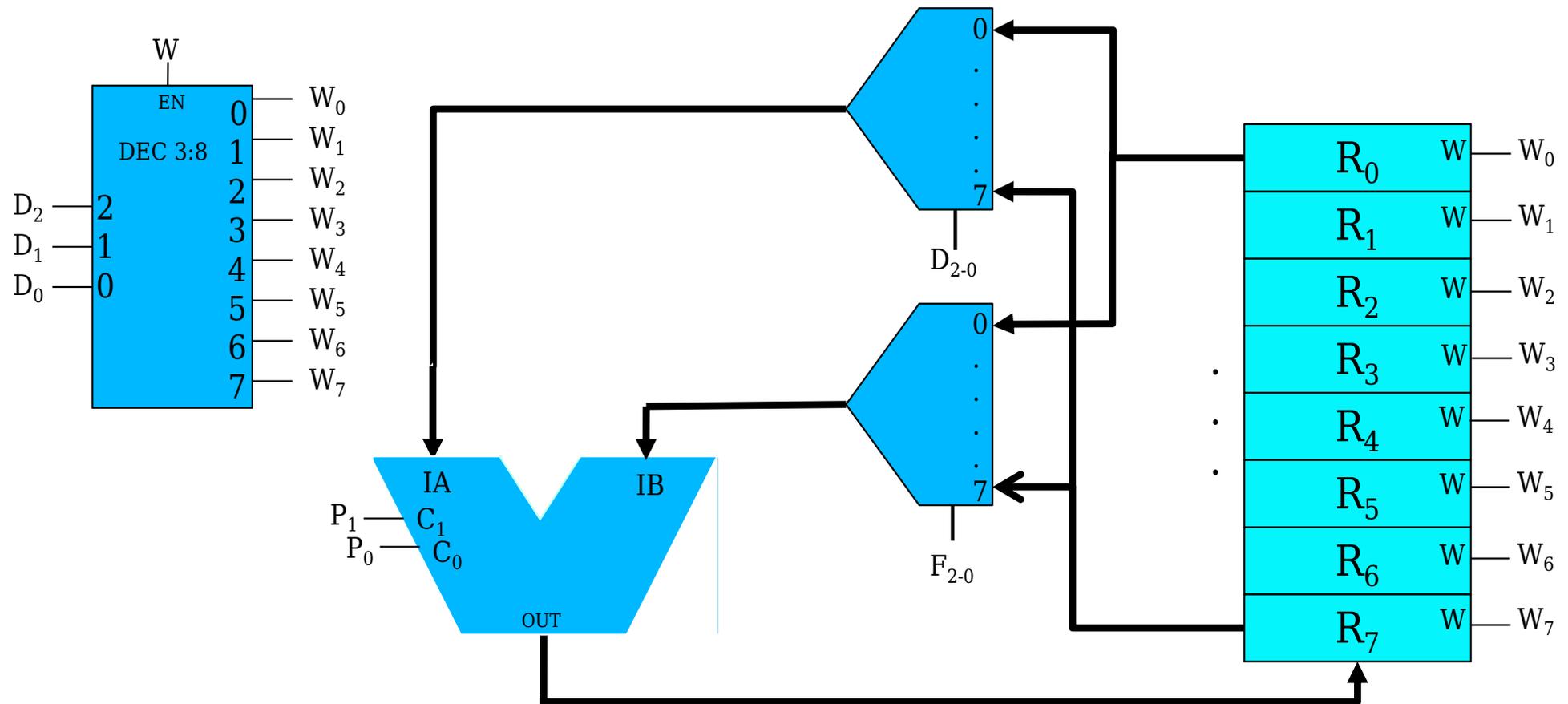
- ▶ **Especificaciones** del sistema a diseñar:
  - ▶ Se dispone de 8 registros ( $R_0, R_1, \dots, R_7$ ) y se desea poder realizar cualquiera de las siguientes operaciones:

$I_1I_0$	operación
0 0	$R_D \leftarrow R_D + R_F$
1 0	$R_D \leftarrow R_D - R_F$
0 1	$R_D \leftarrow R_F$

- ▶  $D, F \in \{0, 1, 2, \dots, 6, 7\}$
- ▶ D y F vienen determinados por  $(D_2D_1D_0)$  y  $(F_2F_1F_0)$

# Diseño de una calculadora con 8 registros

► Arquitectura de la u. de datos:



# Diseño de una calculadora con 8 registros

## ► Descripción Verilog de la unidad de procesado

```
//declaración procedimental de la unidad de procesado de datos

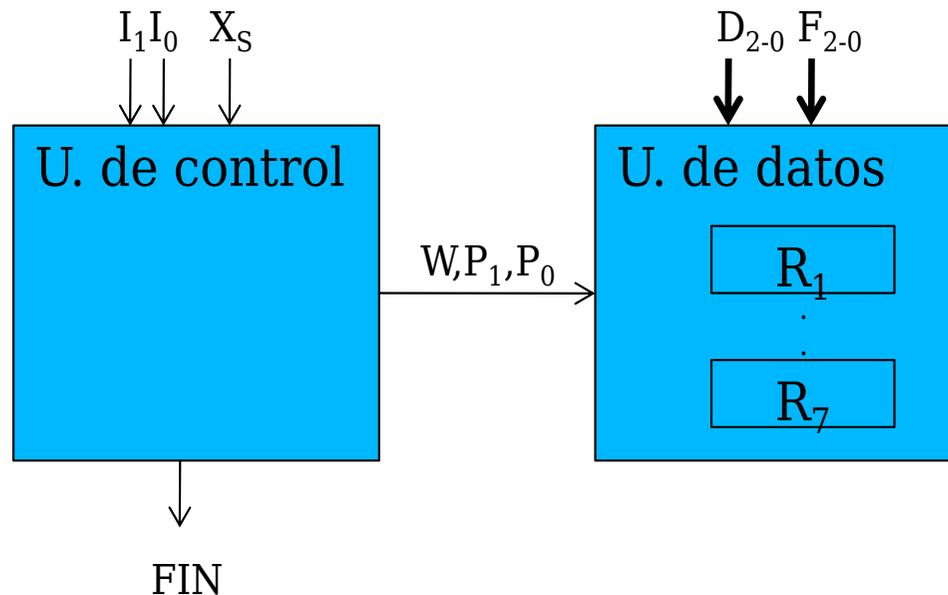
module unidad_datos3 #(parameter N=8) (input ck,W,P0,P1, input [2:0] F,D);

    // "R" es un array (una tabla) formada por 8 "vectores" de "N" bits cada uno
    reg [N-1:0] R [7:0];

    always@(posedge ck)
        if(W)
            case({P1,P0}):
                2'b00: R[D] <= R[D] + R[F];
                2'b01: R[D] <= R[F];
                2'b10: R[D] <= R[D] - R[F];
                2'b11: R[D] <= R[D];
            endcase
    endmodule
```

# Diseño de una calculadora con 8 registros

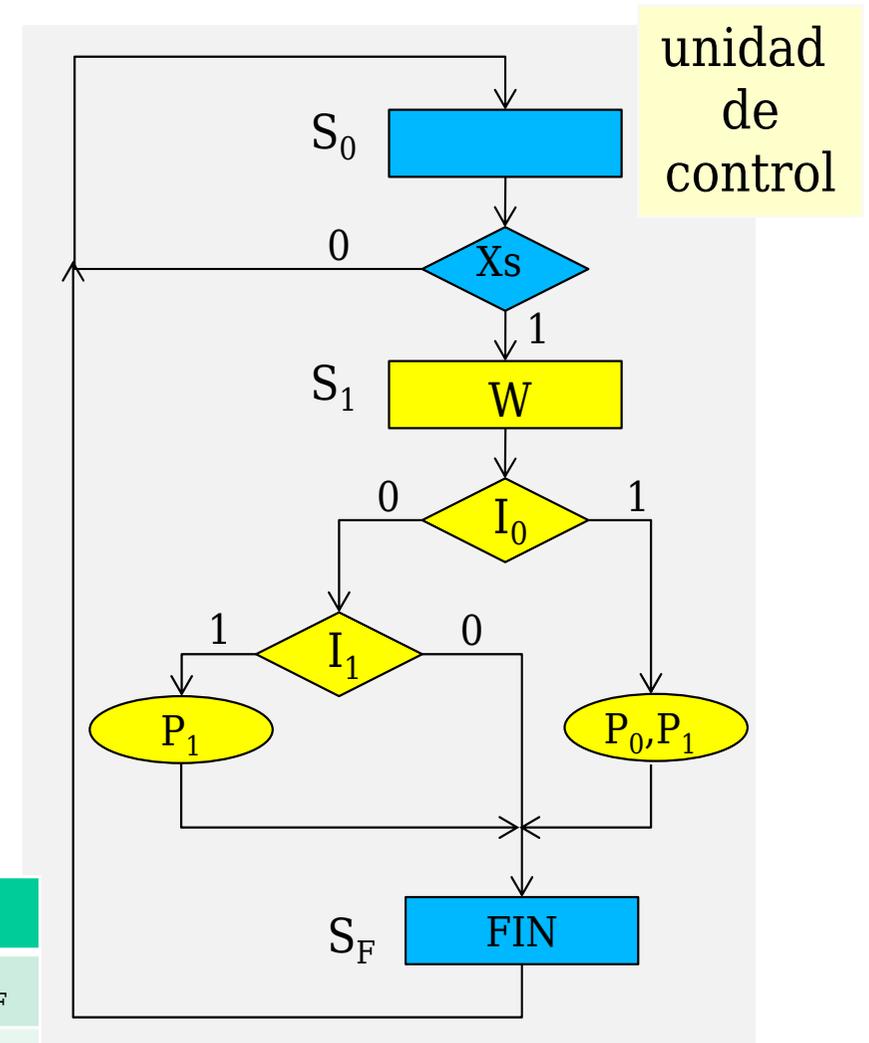
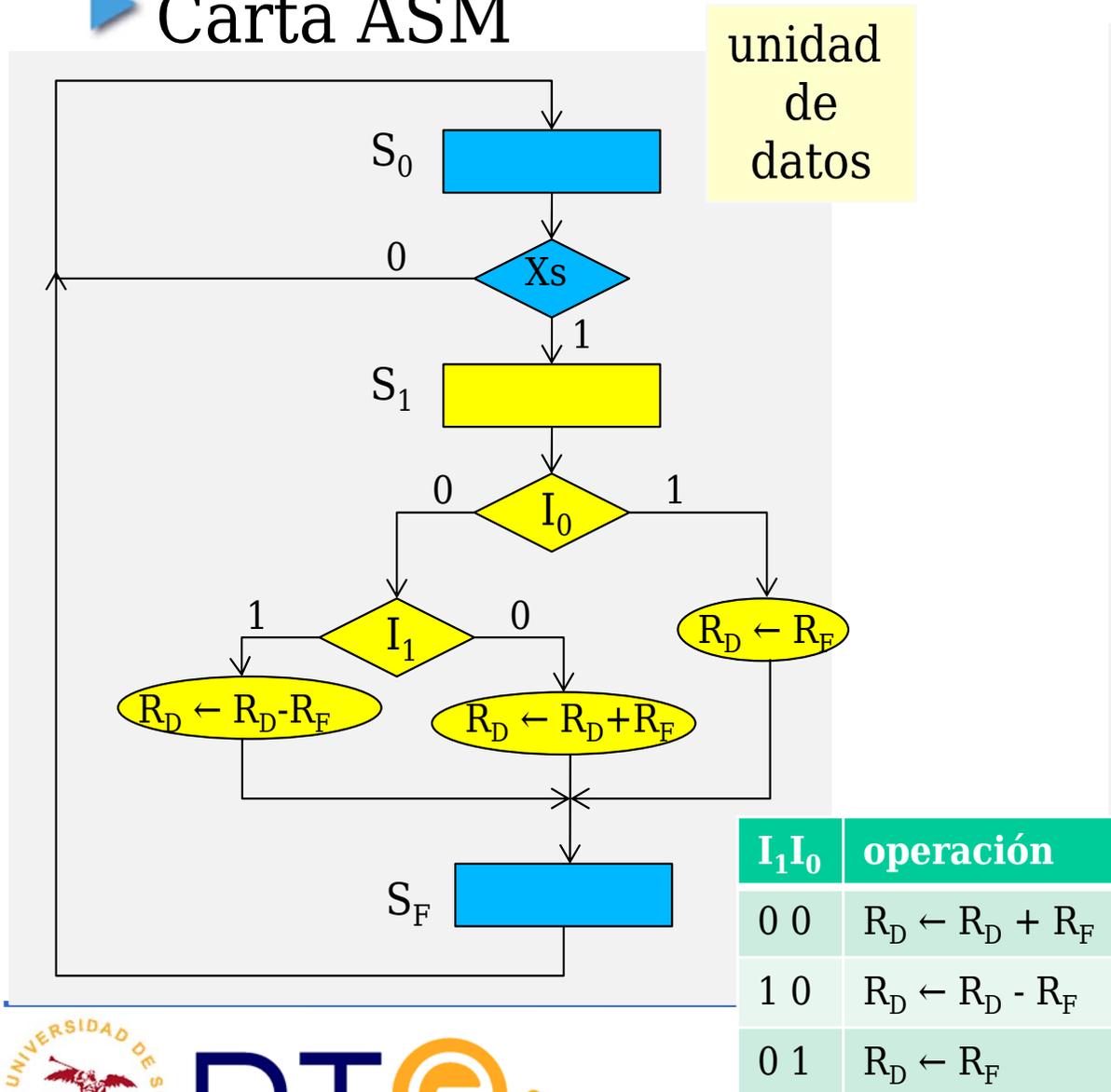
- Organización del sistema digital:



- El usuario especifica la operación proporcionando el valor de  $I_1, I_0, D_{2-0}, F_{2-0}$ , y genera la orden de comienzo con  $X_S$

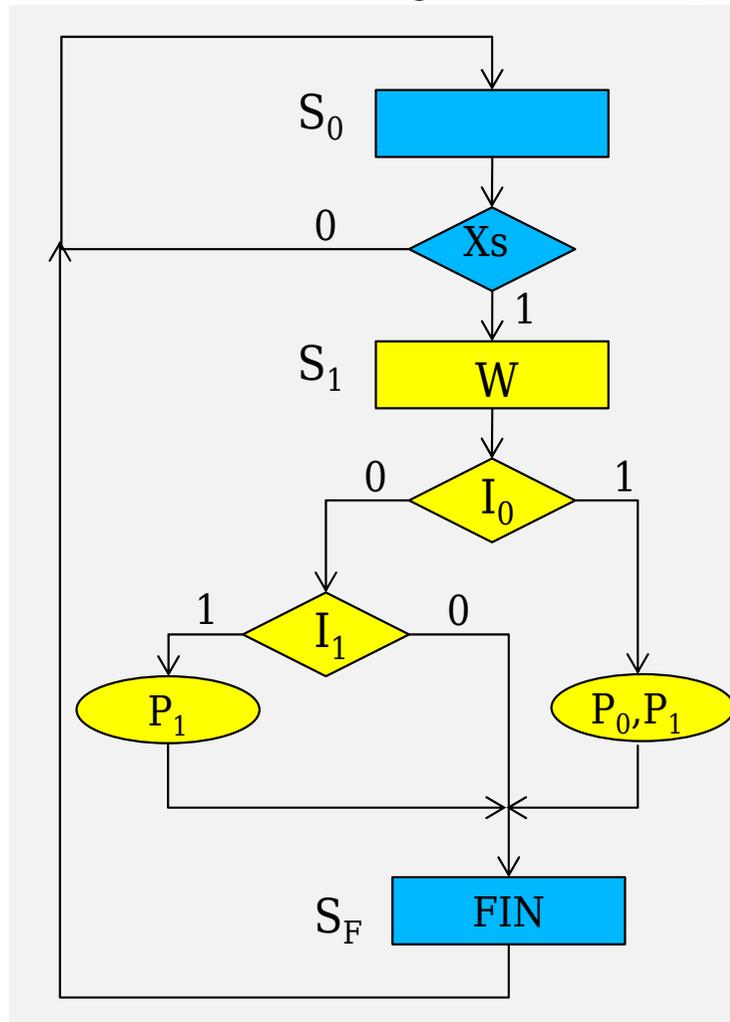
# Diseño de una calculadora con 8 registros

## ▶ Carta ASM



# Diseño de una calculadora con 8 registros

- ▶ Carta ASM y descripción Verilog del controlador



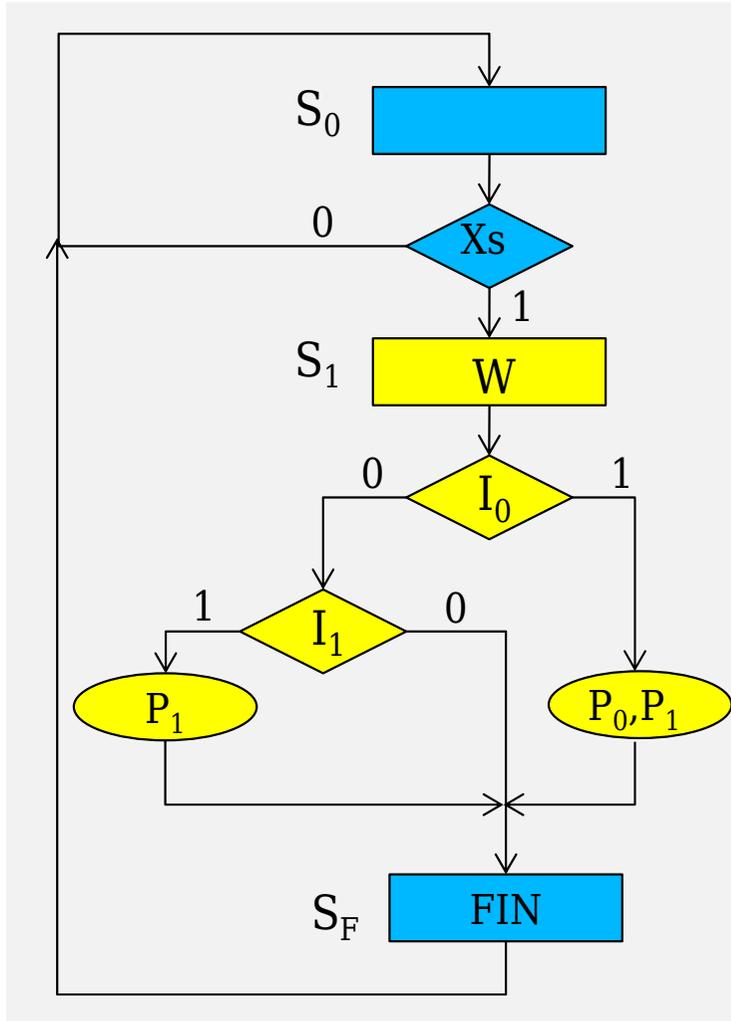
```
module unidad_control3(
    input ck, XS, I0, I1, reset,
    output reg P0,P1,W,FIN
);
parameter S0 = 2'b00,
          S1 = 2'b01,
          SF = 2'b10;

reg [1:0] current_state,next_state;

always @(posedge ck or posedge reset)
begin
    if(reset)
        current_state <= S0;
    else
        current_state <= next_state;
end
```

**SIGUE ->**

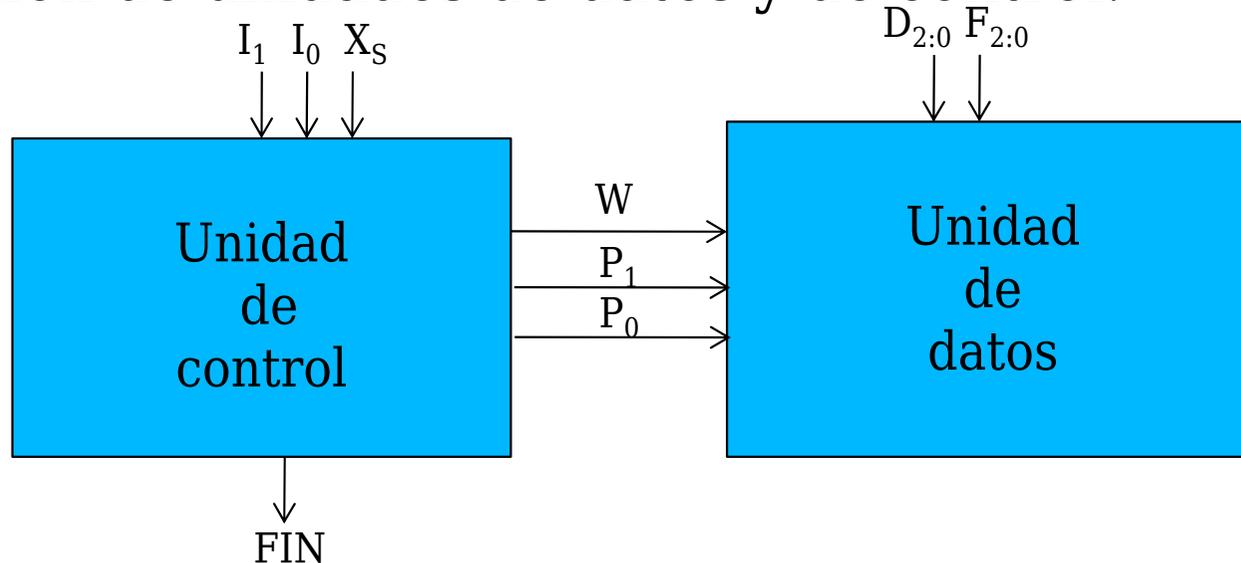
# ▶ Carta ASM y descripción Verilog del controlador



```
always @(current_state,I0,I1,XS)
begin
  P0 = 0;
  P1 = 0;
  FIN = 0;
  W = 0;
  next_state = S0;
  case(current_state)
  S0:
    if(XS) next_state = S1;
  S1:
    begin
      W = 1;
      if(I0)
        begin
          P0 = 1;
          P1 = 1;
        end
      else if (I1)
        P1 = 1;
      next_state = SF;
    end
  SF:
    FIN = 1;
  endcase
end
endmodule
```

# Diseño de una calculadora con 8 registros

- Conexión de unidades de datos y de control:



```
module calculadora8reg #(parameter N=8) ( input ck, XS, IO, I1, reset, input [2:0] D, F,
                                         output FIN);

    wire w1,w2,w3;

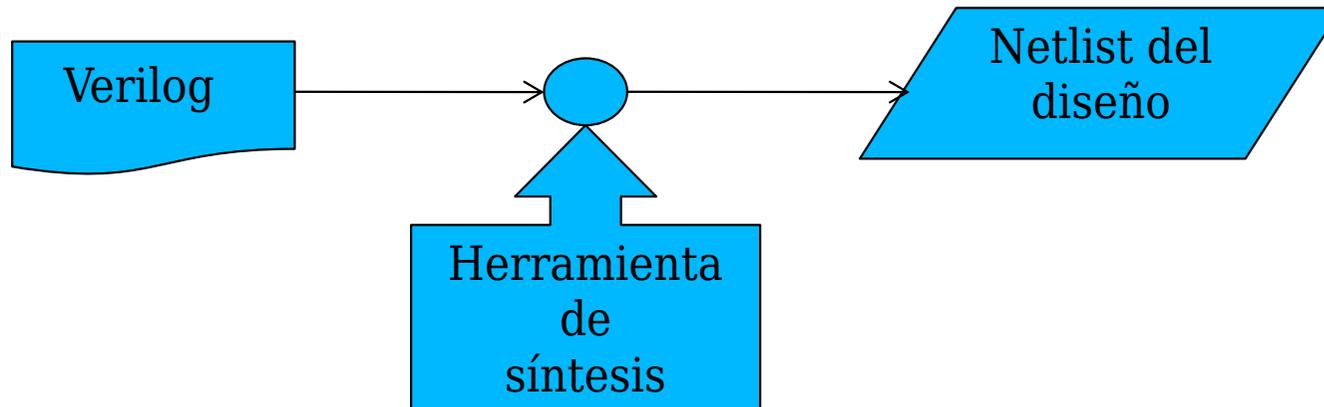
    unidad_datos3 #(N) ud_calc (.ck(ck),.W(w1),.P0(w2),.P1(w3),.D(D),.F(F));

    unidad_control3 uc_calc(.ck(ck),.reset(reset),.XS(XS),.FIN(FIN),.IO(IO),.I1(I1),
                           .W(w1),.P0(w2),.P1(w3));

endmodule
```

---

# Técnicas de realización de u. de control



- ▶ Estrategias:
  - ▶ Cableada (como circuito secuencial síncrono)
  - ▶ Un biestable por estado
  - ▶ Microprogramado

---

# Ejemplo de uso de la calculadora

- ▶ Realización de la operación  $R_0 \leftarrow 3R_1 - R_2$ 
  - ▶ Se trata de una operación más compleja no incluida en la tabla de operación del sistema.
  - ▶ Se puede realizar mediante una secuencia de instrucciones (nivel ISP)
    - ▶ Instrucción 1:  $R_0 \leftarrow R_1$
    - ▶ Instrucción 2:  $R_0 \leftarrow R_0 - R_2$
    - ▶ Instrucción 3:  $R_0 \leftarrow R_0 + R_1$
    - ▶ Instrucción 4:  $R_0 \leftarrow R_0 + R_1$

---

# Calculadora frente a computador

## ▶ Similitudes

- ▶ Podemos resolver problemas complejos a partir de las instrucciones del sistema mediante programación (software).
- ▶ El usuario no necesita ser especialista en la electrónica del sistema (hardware).

## ▶ Deficiencias

- ▶ No hay **automatización en la ejecución** del programa: cada vez que se ejecuta una instrucción el usuario debe activar Xs, esperar la señal de FIN y suministrar la siguiente.
- ▶ No hay **programa almacenado**: para ejecutar cada instrucción el usuario debe proporcionar los valores  $D_{2:0}$  y  $F_{2:0}$  para cada una de las tres instrucciones.