



Tema 1:

# De sistema digital a computador

Fundamentos de computadores II

**José Manuel Mendías Cuadros**

*Dpto. Arquitectura de Computadores y Automática*

*Universidad Complutense de Madrid*





# Contenidos

- ✓ Circuitos de propósito específico.
- ✓ Ruta de datos de propósito general + controlador.
- ✓ Circuito de propósito general: computador.
- ✓ Modelo de Von-Neumann.
- ✓ Arquitectura del procesador.
- ✓ Estructura del procesador.
- ✓ Otros conceptos básicos.

Transparencias basadas en los cursos:

- Katzalin Olcoz et al. *Fundamentos de Computadores II*. UCM
- Chris Terman. *Computation Structures*. MIT Open Courseware



FC-1

# Circuitos de propósito específico

versión 15/01/23

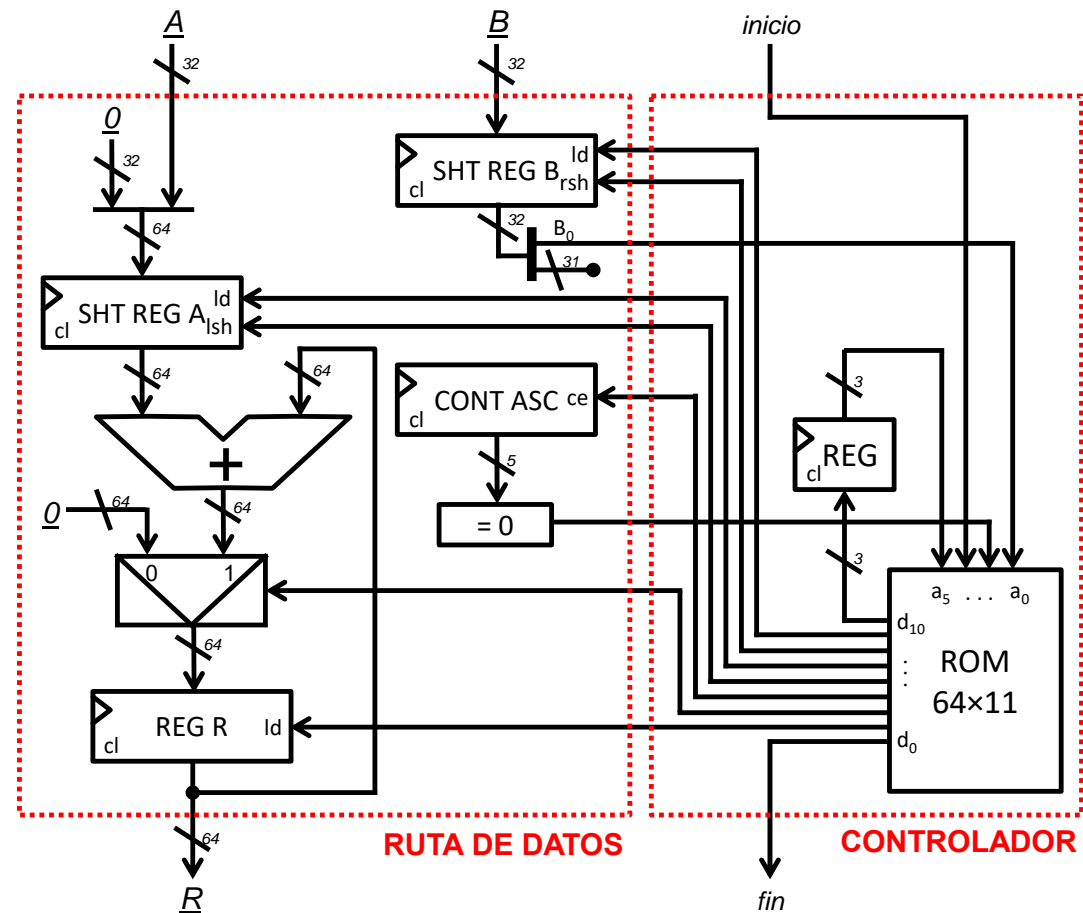
- Un **algoritmo** se **implementa en hardware** conectando:
  - **Ruta de datos**: realiza las operaciones y almacena resultados parciales.
  - **Controlador**: secuencia la realización de las operaciones según lo indicado por el algoritmo.

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<32; C++ )
{
  if( B0==1 )
    R = R+A;
  A = A<<1;
  B = B>>1;
};
Rout = R;

```

Algoritmo para multiplicar dos números de **32 bits**





# Circuitos de propósito específico

versión 15/01/23

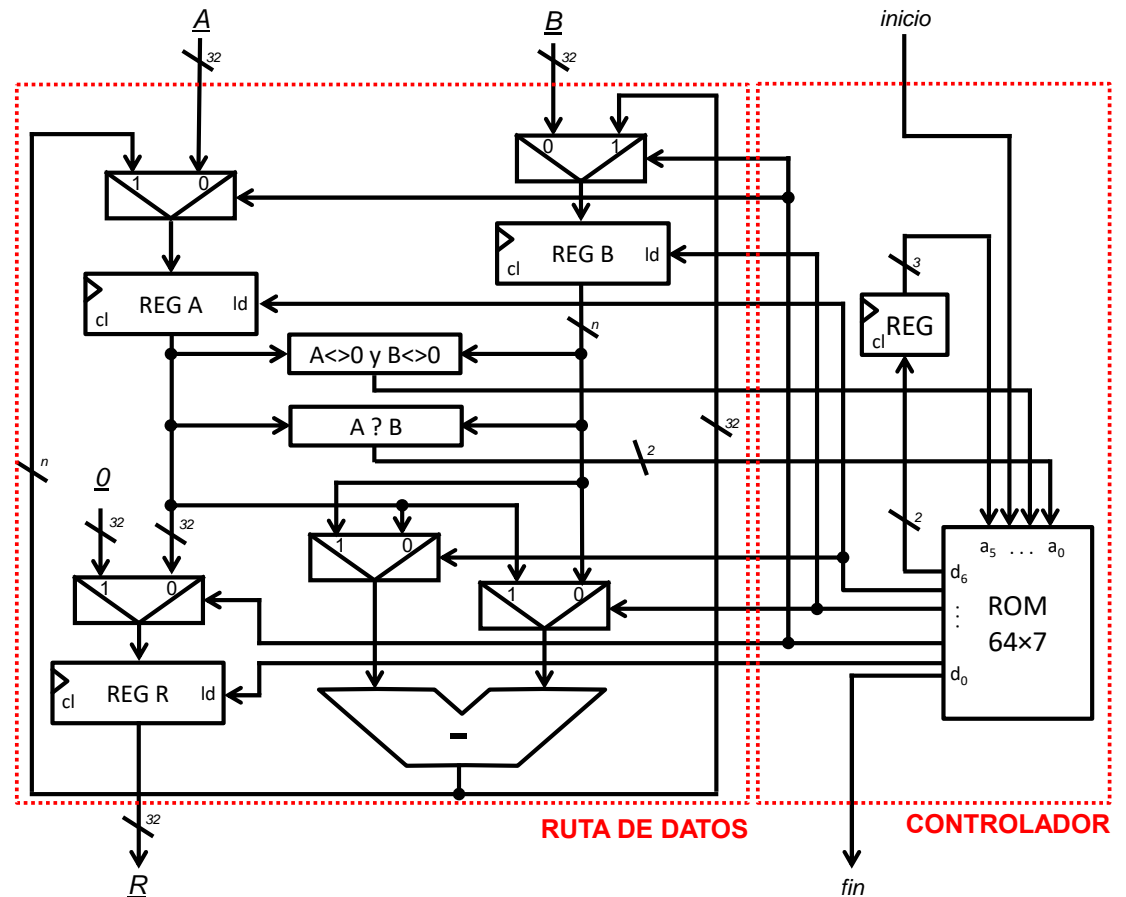
- Los circuitos obtenidos mediante diseño algorítmico **son muy eficientes**, pero tienen **una única funcionalidad**.
  - Algoritmos diferentes** requieren **circuitos distintos** para ejecutarse.
  - Para **cambiar su funcionalidad**, se debe **rediseñar el hardware**.

```

A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
    if( A>B )
      A = A-B;
    else
      B = B-A;
  R = A;
};
Rout = R;

```

Algoritmo para calcular el MCD de 2 números de **32 bits**





# Circuitos de propósito específico



versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

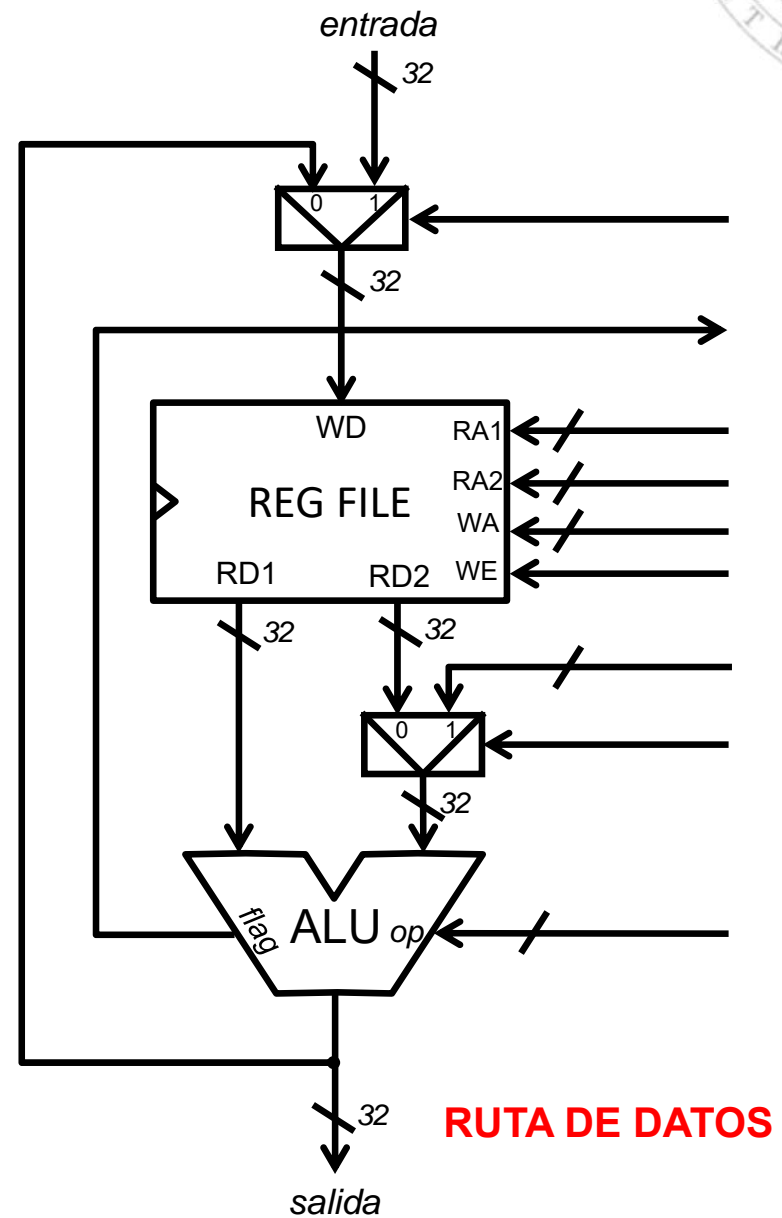
5

- En general, estas **rutas de datos** son **muy específicas** porque para cada algoritmo particular:
  - El **número de registros** se ajusta al número de variables del mismo.
  - El **número y tipo de unidades funcionales** se ajusta al número y tipo de cálculos a realizar.
  - El **número interconexiones** se maximiza para realizar simultáneamente y en paralelo el máximo número de transferencias entre registros posible.
  - La **anchura de cada interconexión** se ajusta a la anchura requerida por cada cálculo.
- Asimismo, estos **controladores** son **muy específicos** porque:
  - El **número de señales de control/estado** de cada ruta de datos es distinto.
  - Siguen una **secuencia de estados** fija almacenada en ROM.



# Ruta de datos de propósito general

- Pero es posible diseñar una ruta de datos más **general**:

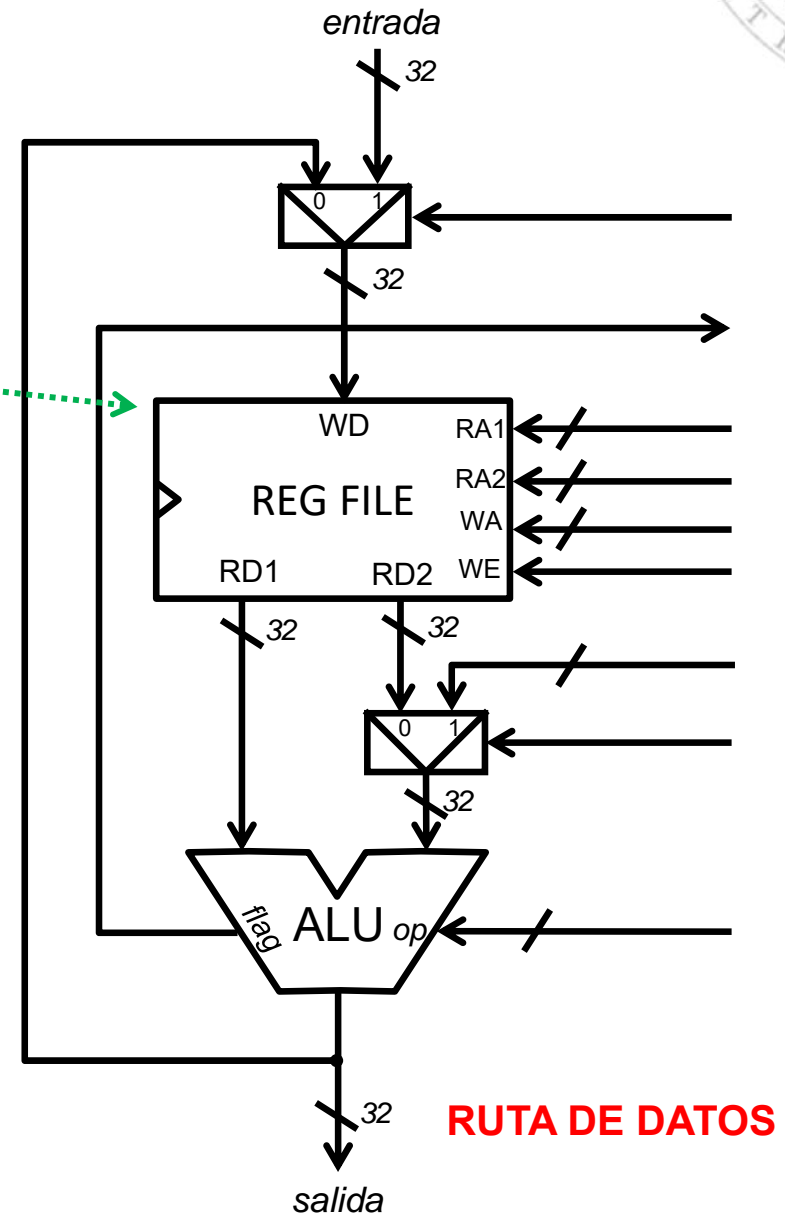




# Ruta de datos de propósito general

- Pero es posible diseñar una ruta de datos más **general**:

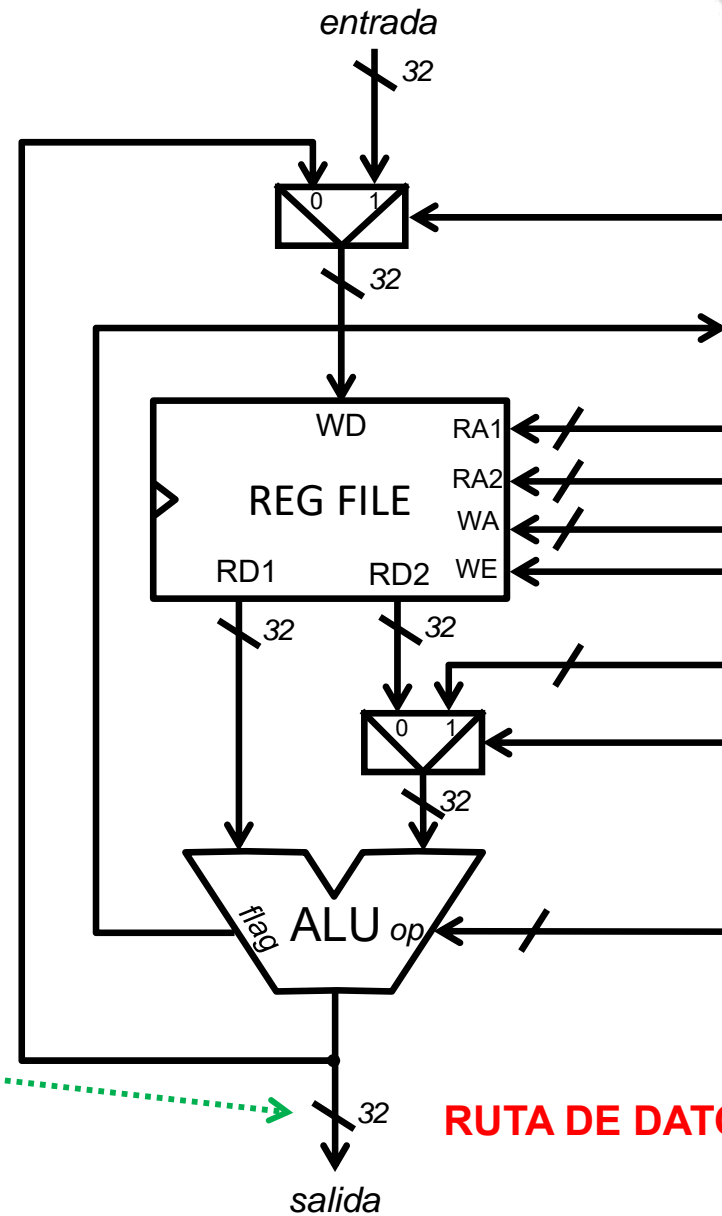
Se usa un *banco de registros* con un número suficientemente grande de ellos





# Ruta de datos de propósito general

- Pero es posible diseñar una ruta de datos más **general**:



Se elige una *anchura de datos homogénea* suficientemente ancha para todas las interconexiones

**RUTA DE DATOS**

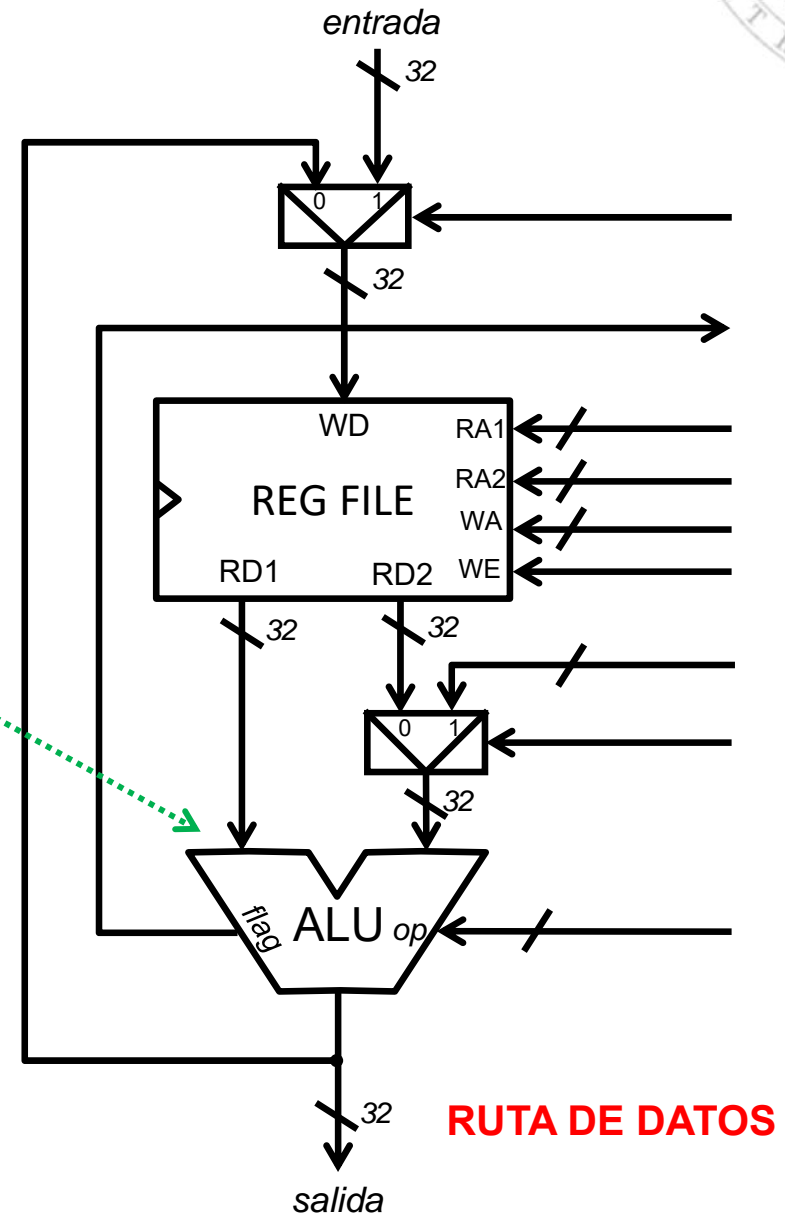




# Ruta de datos de propósito general

- Pero es posible diseñar una ruta de datos más **general**:

Se elige una **ALU genérica** capaz de realizar un rango suficientemente amplio de operaciones aritméticas, lógicas y relacionales

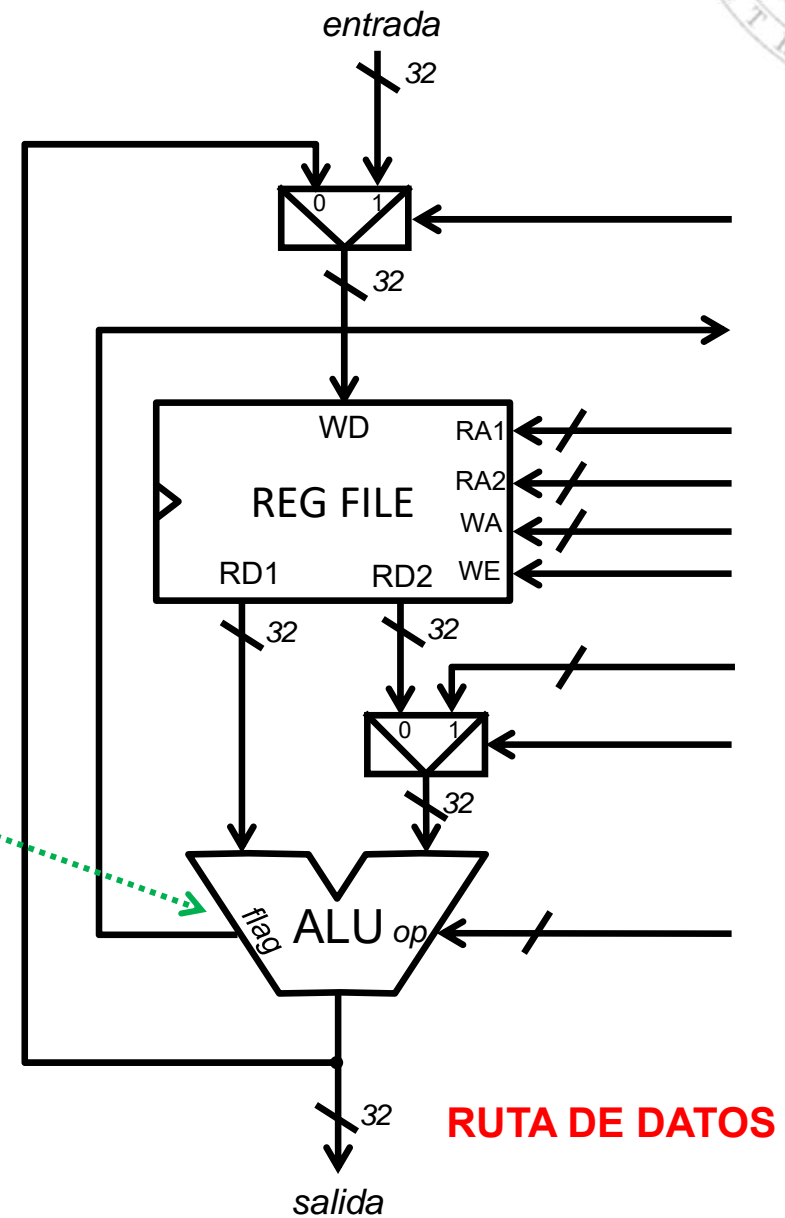




# Ruta de datos de propósito general

- Pero es posible diseñar una ruta de datos más **general**:

La ALU dispone de suficientes flags para indicar si los operandos cumplen cualquier tipo de relación



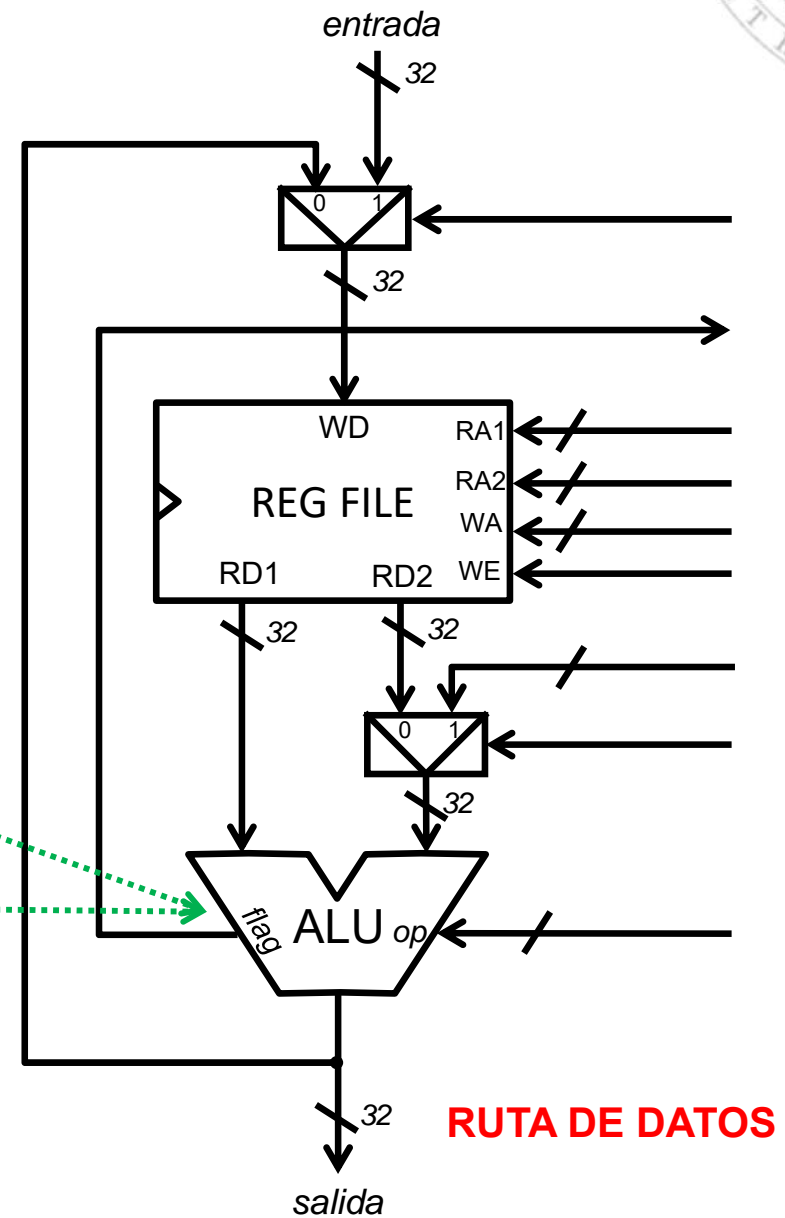


# Ruta de datos de propósito general

- Pero es posible diseñar una ruta de datos más **general**:

La ALU dispone de suficientes flags para indicar si los operandos cumplen cualquier tipo de relación

Para este caso, supongamos que **existe un único flag** que **solo se pone a 1** cuando la ALU realiza una **operación de comparación** que es **cierta**





# Ruta de datos de propósito general

## Multiplicación

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

12

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

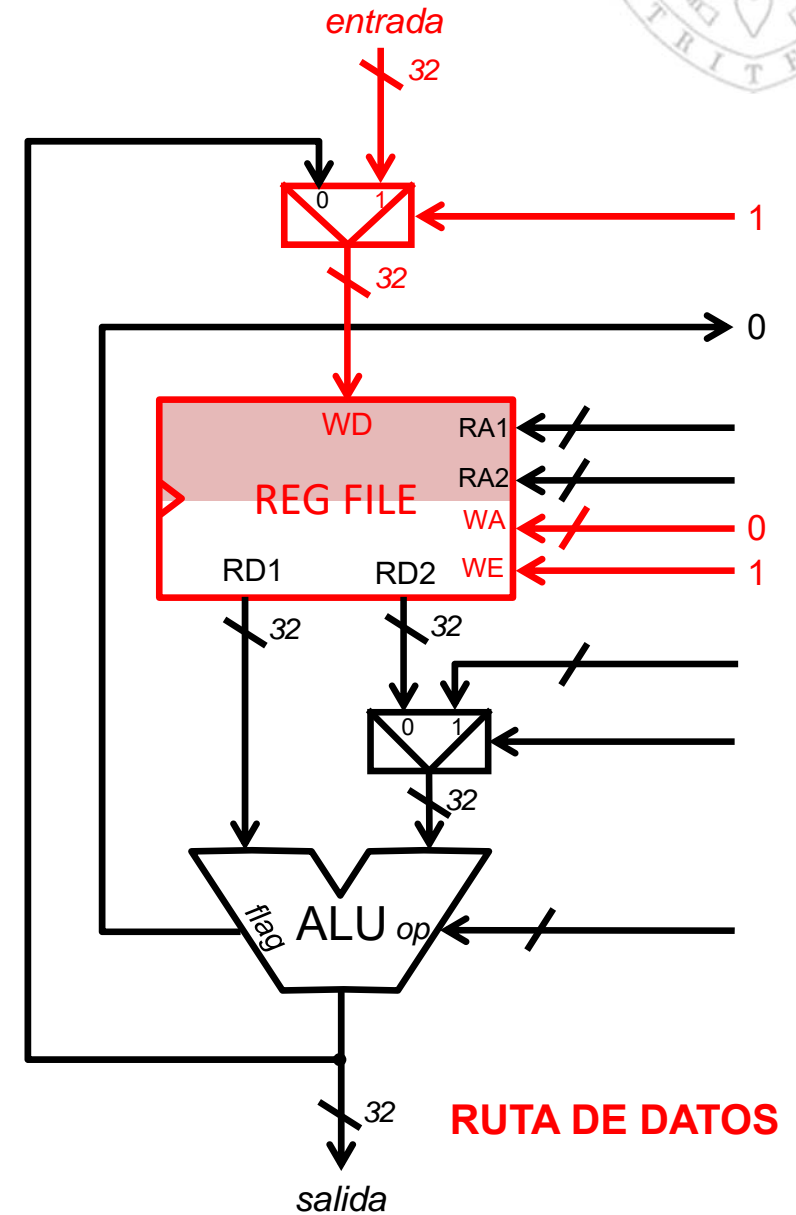
Rout = R;

```

R0 es A

Estados y transferencias entre registros

S0 R0 ← entrada



RUTA DE DATOS



# Ruta de datos de propósito general

## Multiplicación

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

13

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

Rout = R;

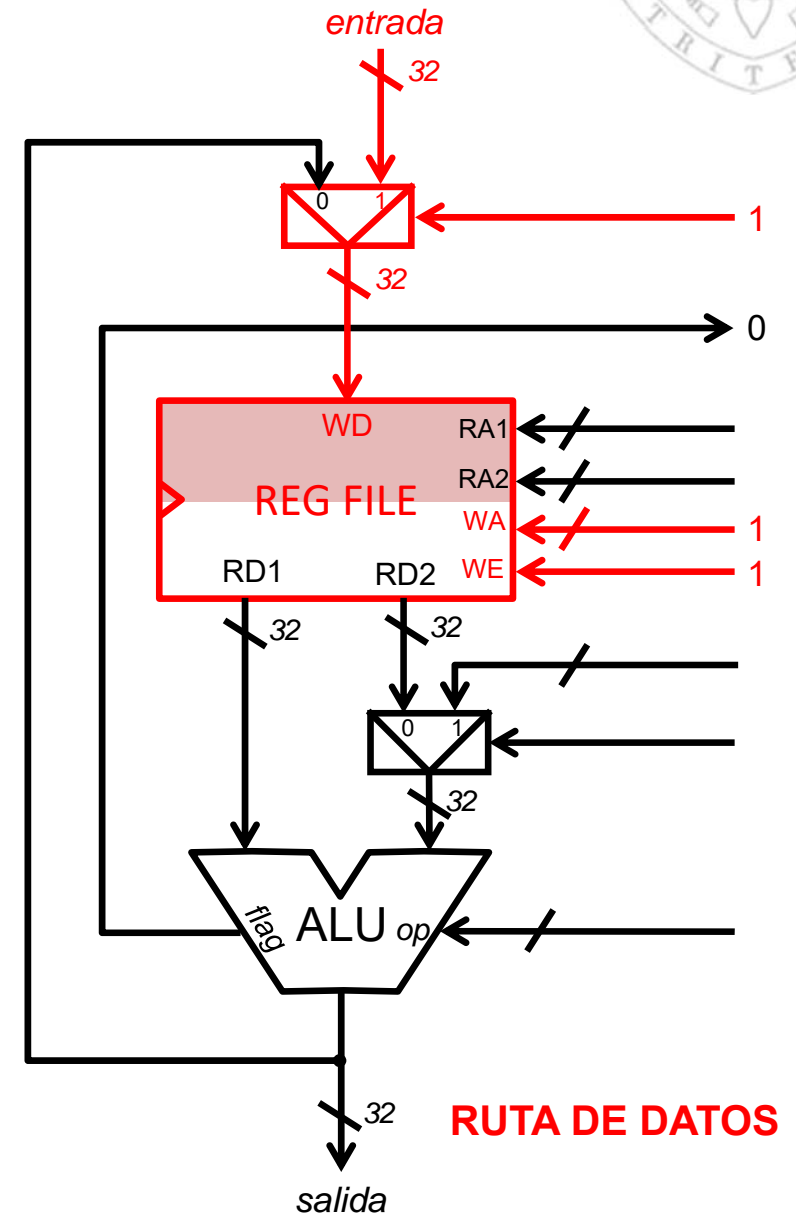
```

Estados y transferencias entre registros

**S0** R0 ← entrada

**S1** R1 ← entrada

R0 es A R1 es B



**RUTA DE DATOS**



# Ruta de datos de propósito general

## Multiplicación

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

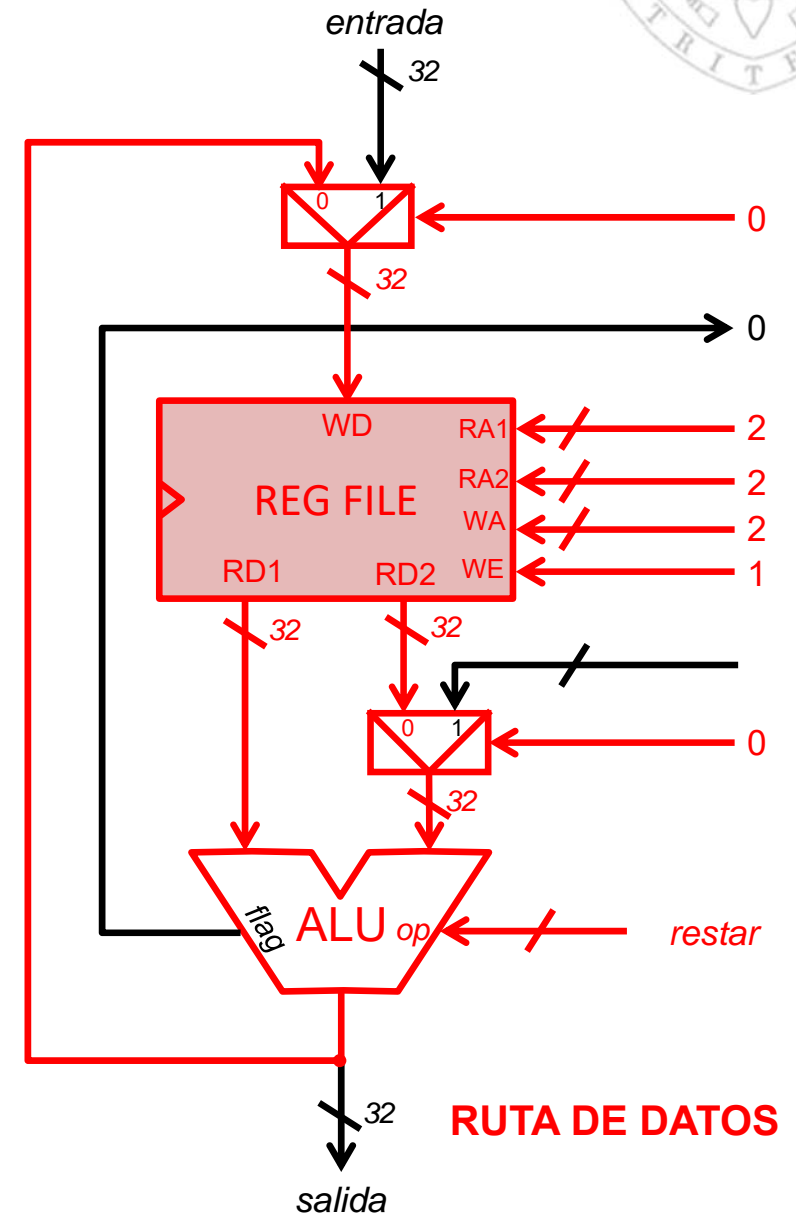
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2

R0 es A   R1 es B   R2 es R





# Ruta de datos de propósito general

## Multiplicación

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

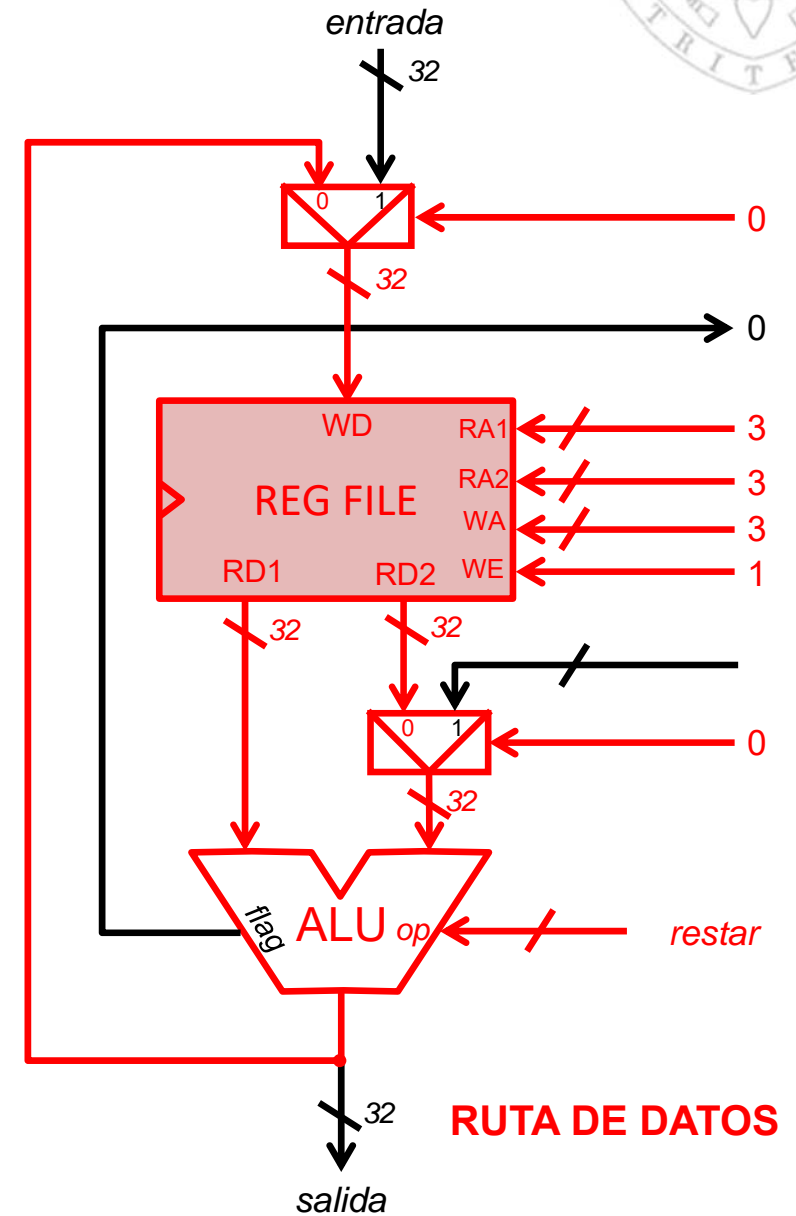
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3

R0 es A   R1 es B   R2 es R   R3 es C



**RUTA DE DATOS**



# Ruta de datos de propósito general

## Multiplicación

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

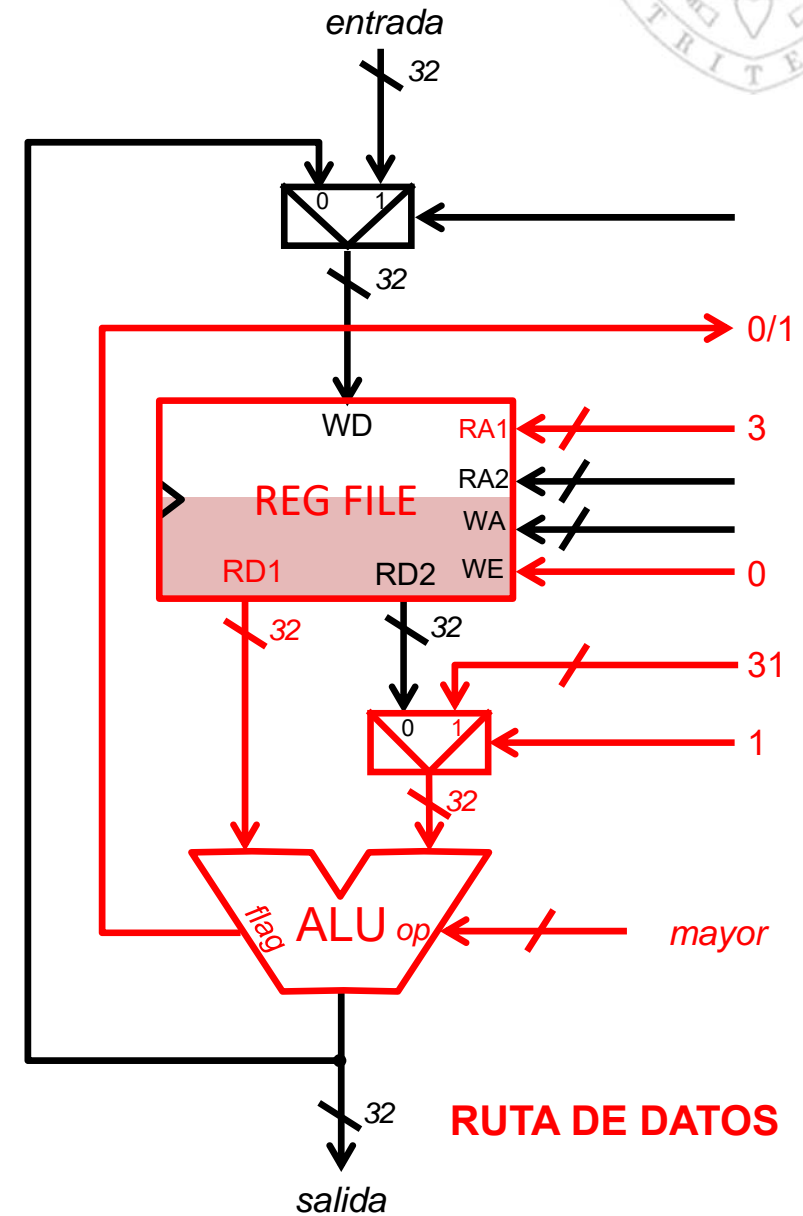
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3
<b>S4</b>	si R3 > 31, ir a S12

R0 es A R1 es B R2 es R R3 es C







# Ruta de datos de propósito general

## Multiplicación

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

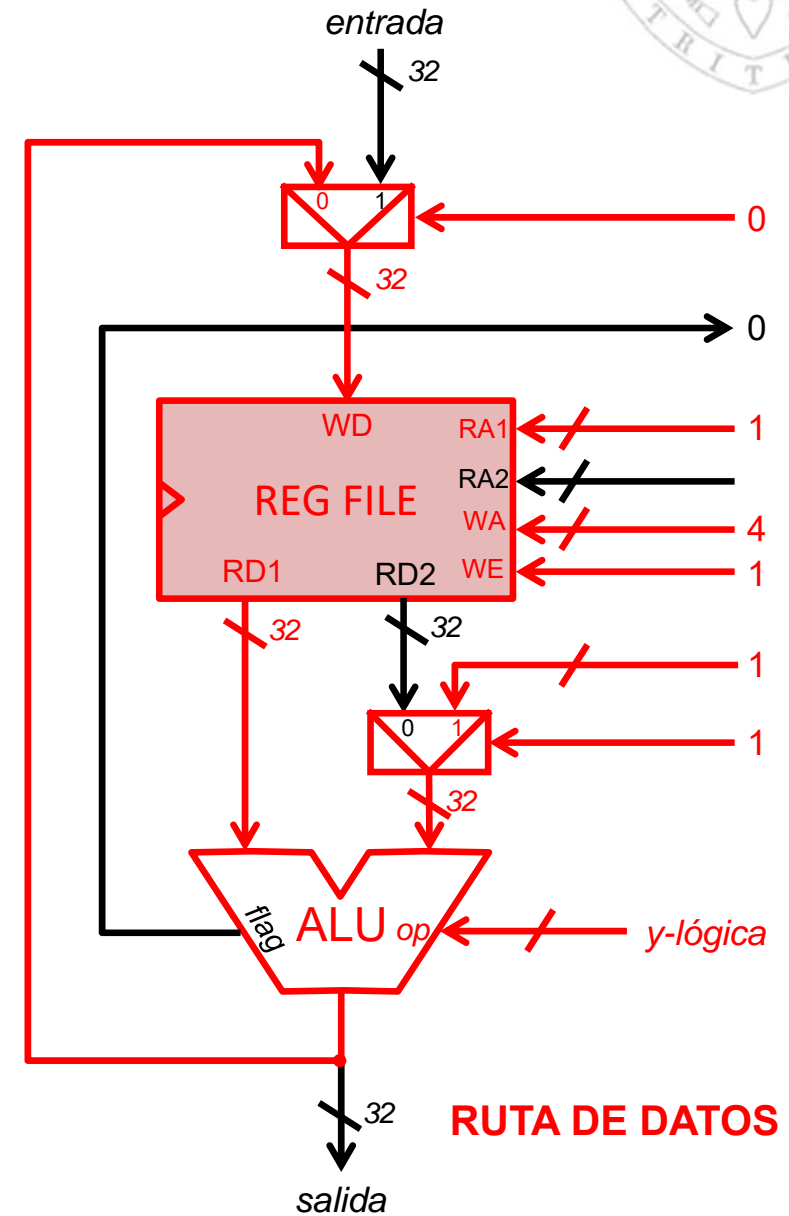
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3
<b>S4</b>	si R3 > 31, ir a S12
<b>S5</b>	R4 ← R1 & 1

R0 es A R1 es B R2 es R R3 es C





# Ruta de datos de propósito general

## Multiplicación

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

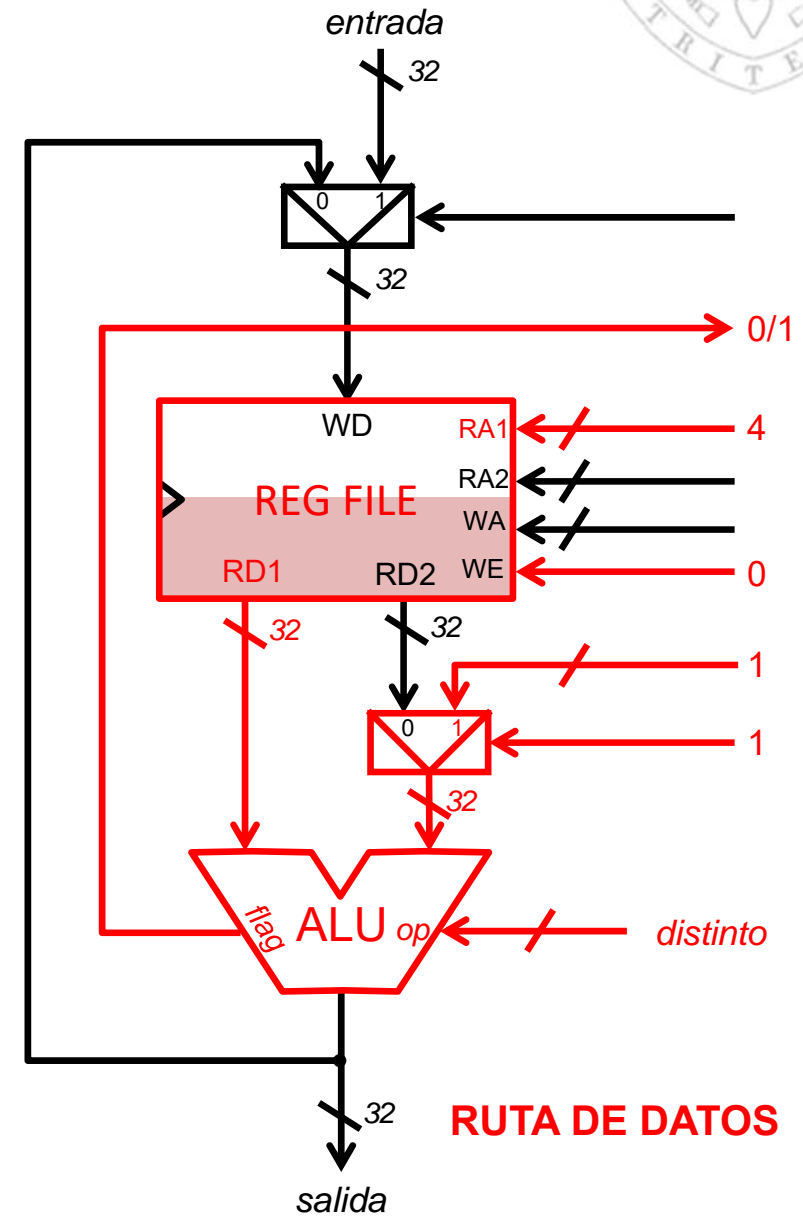
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3
<b>S4</b>	si R3 > 31, ir a S12
<b>S5</b>	R4 ← R1 & 1
<b>S6</b>	si R4 != 1, ir a S8

R0 es A R1 es B R2 es R R3 es C





# Ruta de datos de propósito general

## Multiplicación

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

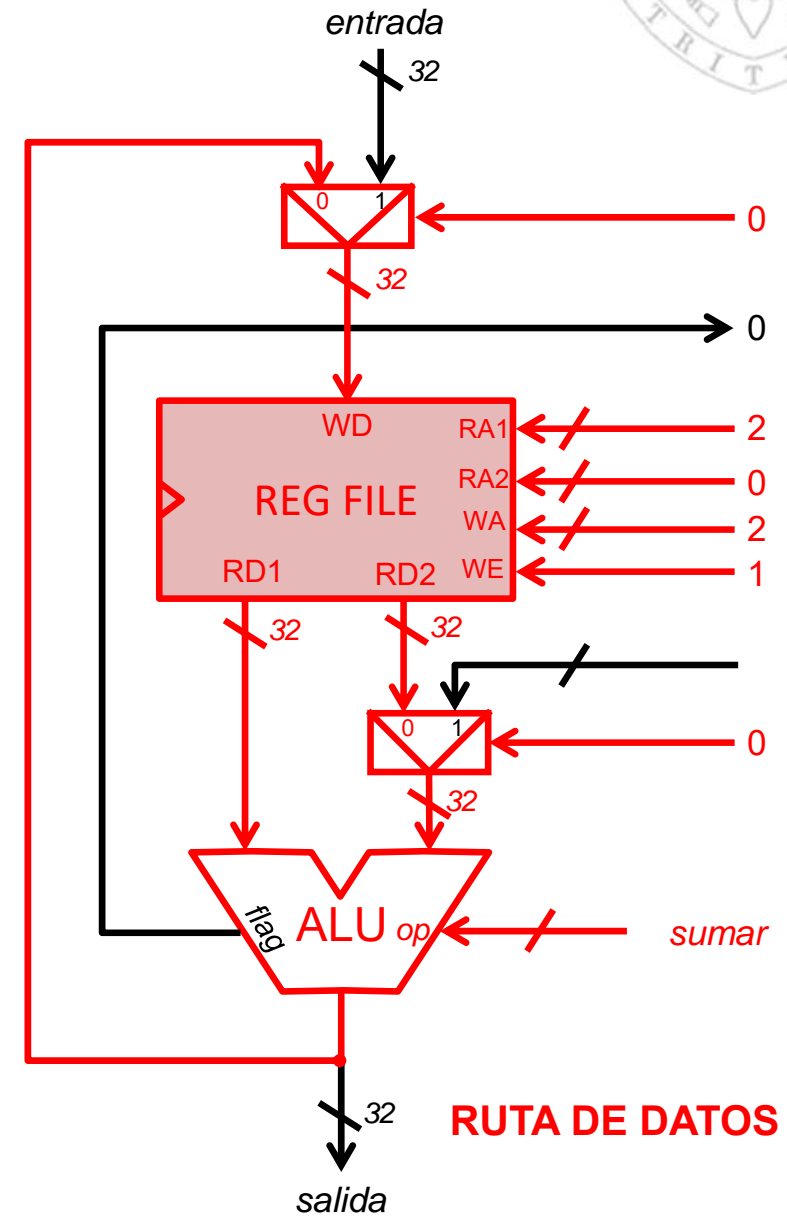
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3
<b>S4</b>	si R3 > 31, ir a S12
<b>S5</b>	R4 ← R1 & 1
<b>S6</b>	si R4 != 1, ir a S8
<b>S7</b>	R2 ← R2 + R0

R0 es A R1 es B R2 es R R3 es C





# Ruta de datos de propósito general

## Multiplicación

versión 15/01/23

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

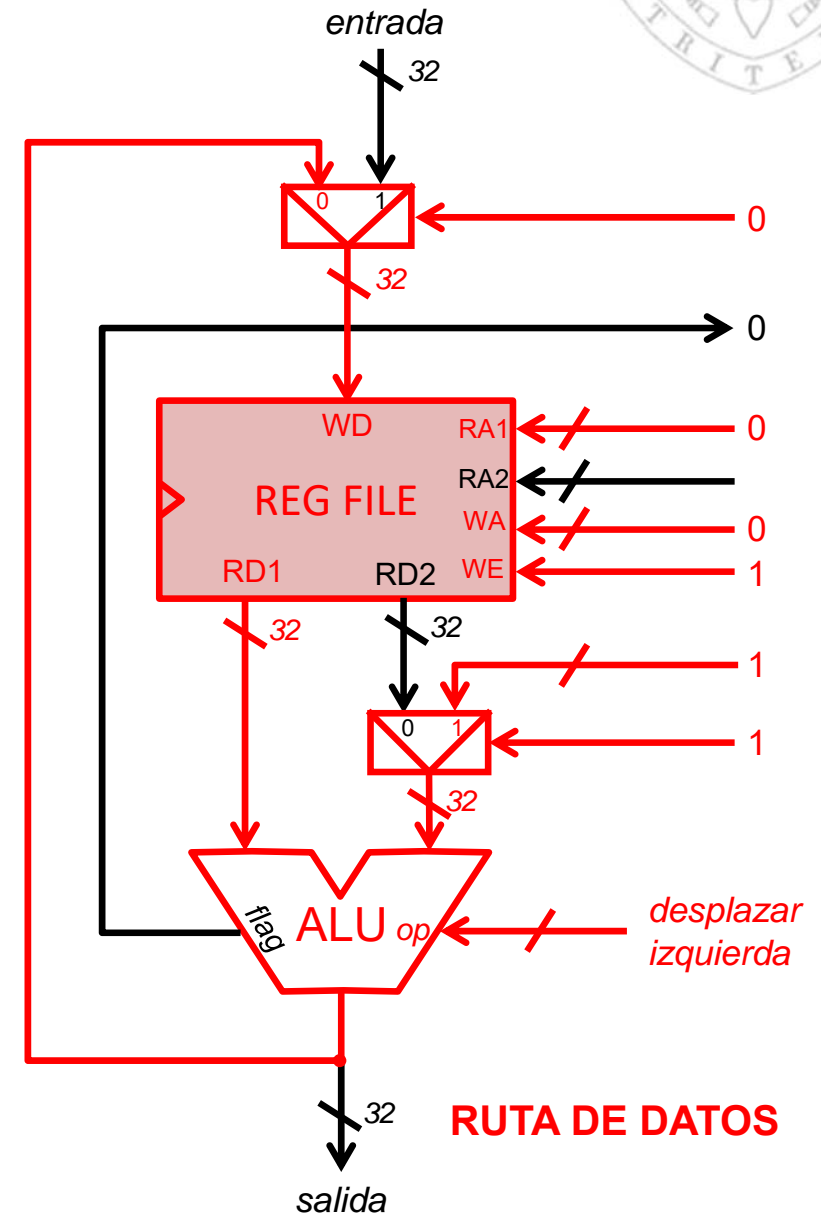
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3
<b>S4</b>	si R3 > 31, ir a S12
<b>S5</b>	R4 ← R1 & 1
<b>S6</b>	si R4 != 1, ir a S8
<b>S7</b>	R2 ← R2 + R0
<b>S8</b>	R0 ← R0 << 1

R0 es A R1 es B R2 es R R3 es C





# Ruta de datos de propósito general

## Multiplicación

versión 15/01/23

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

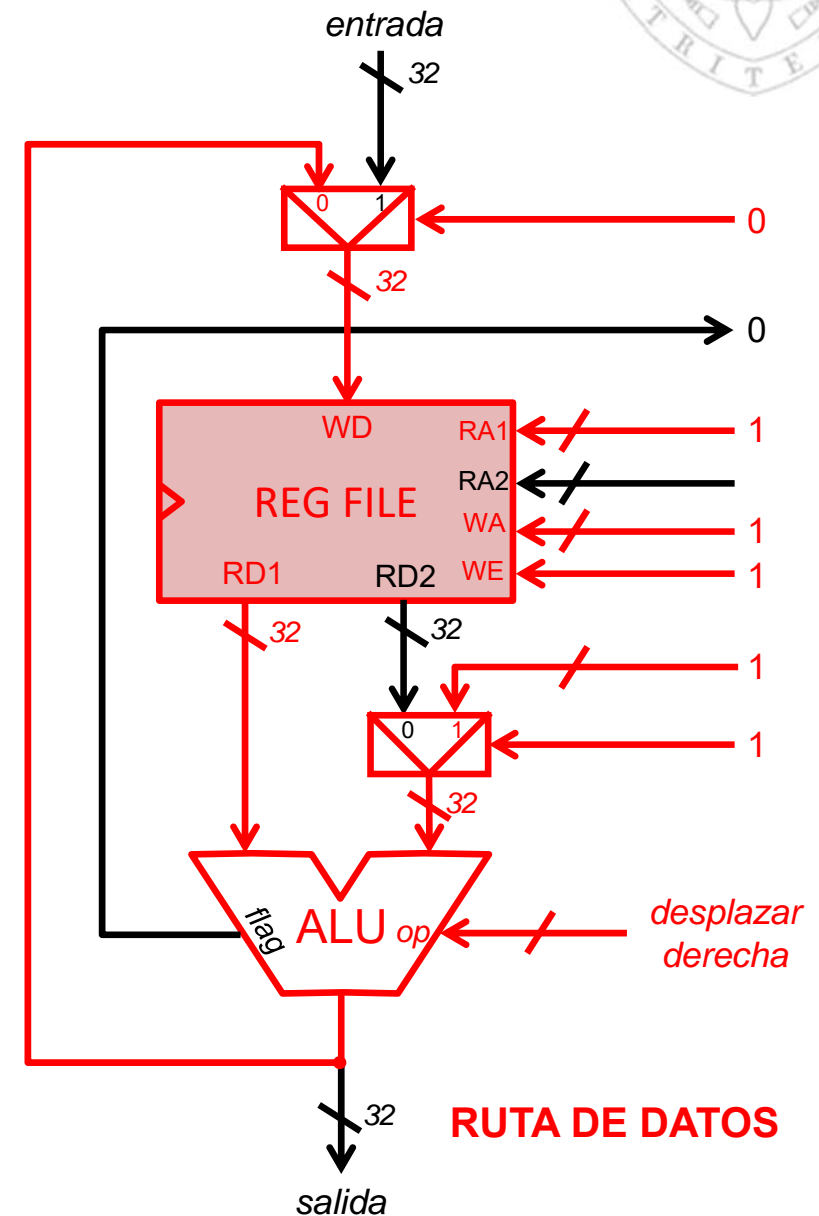
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3
<b>S4</b>	si R3 > 31, ir a S12
<b>S5</b>	R4 ← R1 & 1
<b>S6</b>	si R4 != 1, ir a S8
<b>S7</b>	R2 ← R2 + R0
<b>S8</b>	R0 ← R0 << 1
<b>S9</b>	R1 ← R1 >> 1

R0 es A R1 es B R2 es R R3 es C





# Ruta de datos de propósito general

## Multiplicación

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

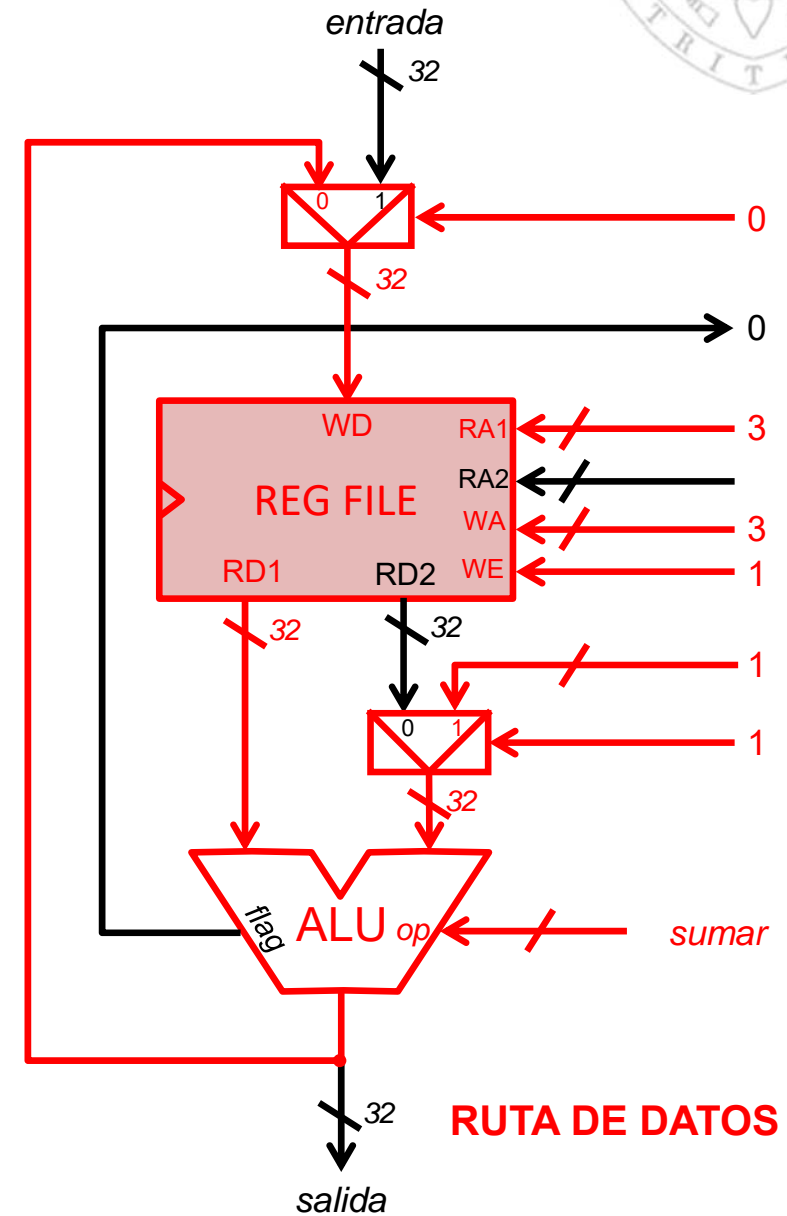
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3
<b>S4</b>	si R3 > 31, ir a S12
<b>S5</b>	R4 ← R1 & 1
<b>S6</b>	si R4 != 1, ir a S8
<b>S7</b>	R2 ← R2 + R0
<b>S8</b>	R0 ← R0 << 1
<b>S9</b>	R1 ← R1 >> 1
<b>S10</b>	R3 ← R3 + 1

R0 es A R1 es B R2 es R R3 es C



**ruta de datos**



# Ruta de datos de propósito general

## Multiplicación

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

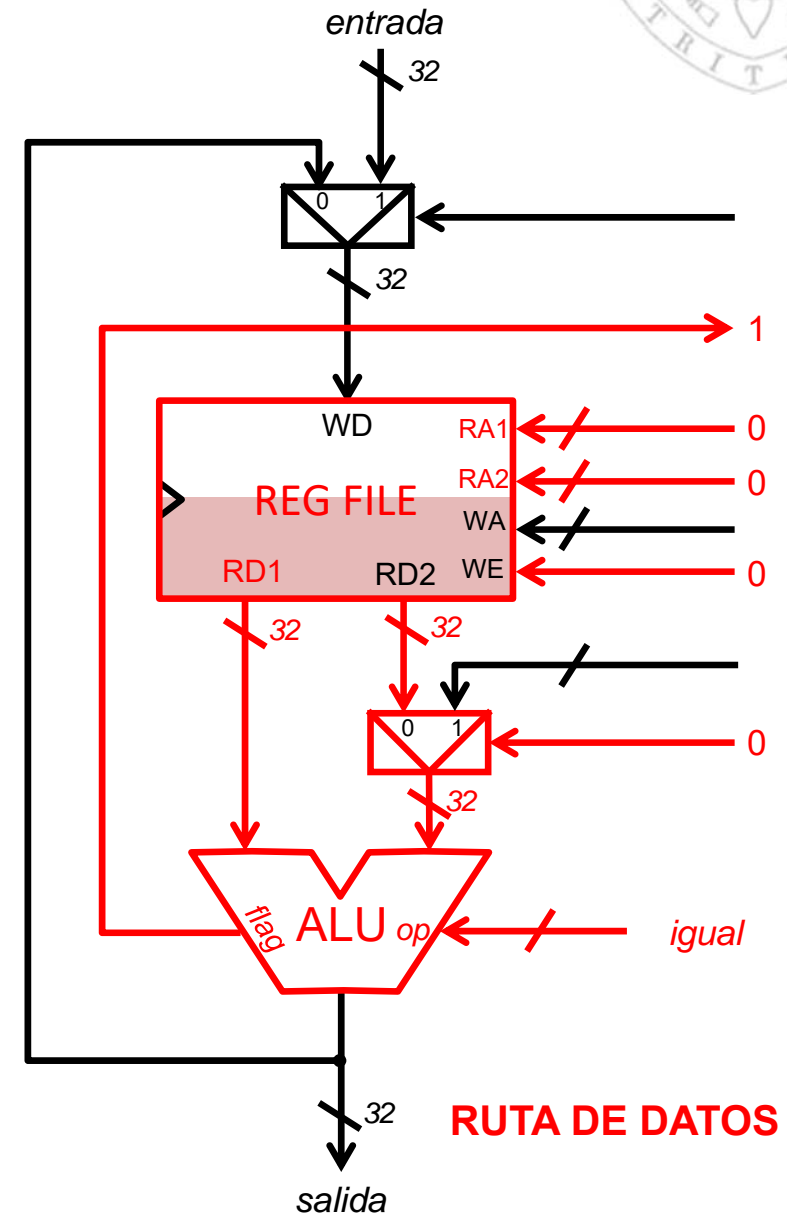
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3
<b>S4</b>	si R3 > 31, ir a S12
<b>S5</b>	R4 ← R1 & 1
<b>S6</b>	si R4 != 1, ir a S8
<b>S7</b>	R2 ← R2 + R0
<b>S8</b>	R0 ← R0 << 1
<b>S9</b>	R1 ← R1 >> 1
<b>S10</b>	R3 ← R3 + 1
<b>S11</b>	si R0 == R0, ir a S4

R0 es A R1 es B R2 es R R3 es C





# Ruta de datos de propósito general

## Multiplicación

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

    R = R+A;
  A = A<<1;
  B = B>>1;
};

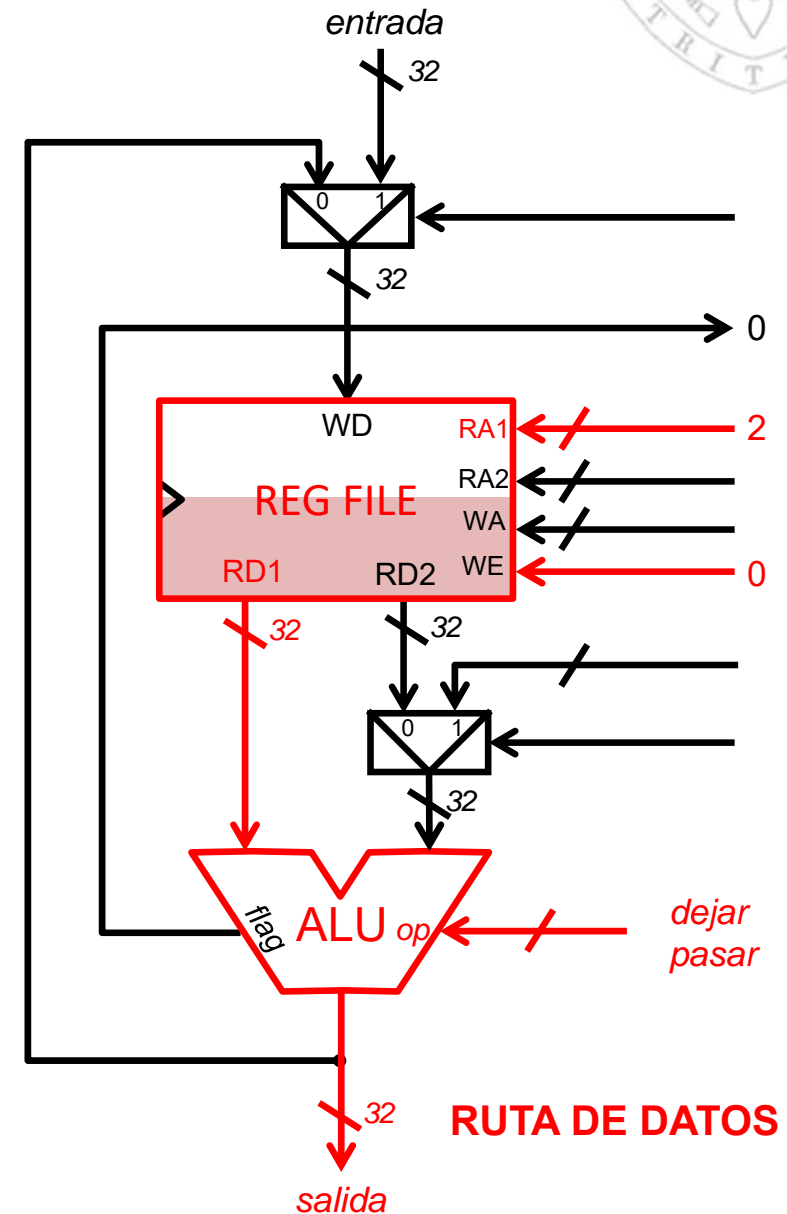
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3
<b>S4</b>	si R3 > 31, ir a S12
<b>S5</b>	R4 ← R1 & 1
<b>S6</b>	si R4 != 1, ir a S8
<b>S7</b>	R2 ← R2 + R0
<b>S8</b>	R0 ← R0 << 1
<b>S9</b>	R1 ← R1 >> 1
<b>S10</b>	R3 ← R3 + 1
<b>S11</b>	si R0 == R0, ir a S4
<b>S12</b>	salida ← R2

R0 es A R1 es B R2 es R R3 es C







# Ruta de datos de propósito general

## Multiplicación

Algoritmo para multiplicar dos números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
for( C=0; C<=31; C++ )
{
  if( B0==1 )

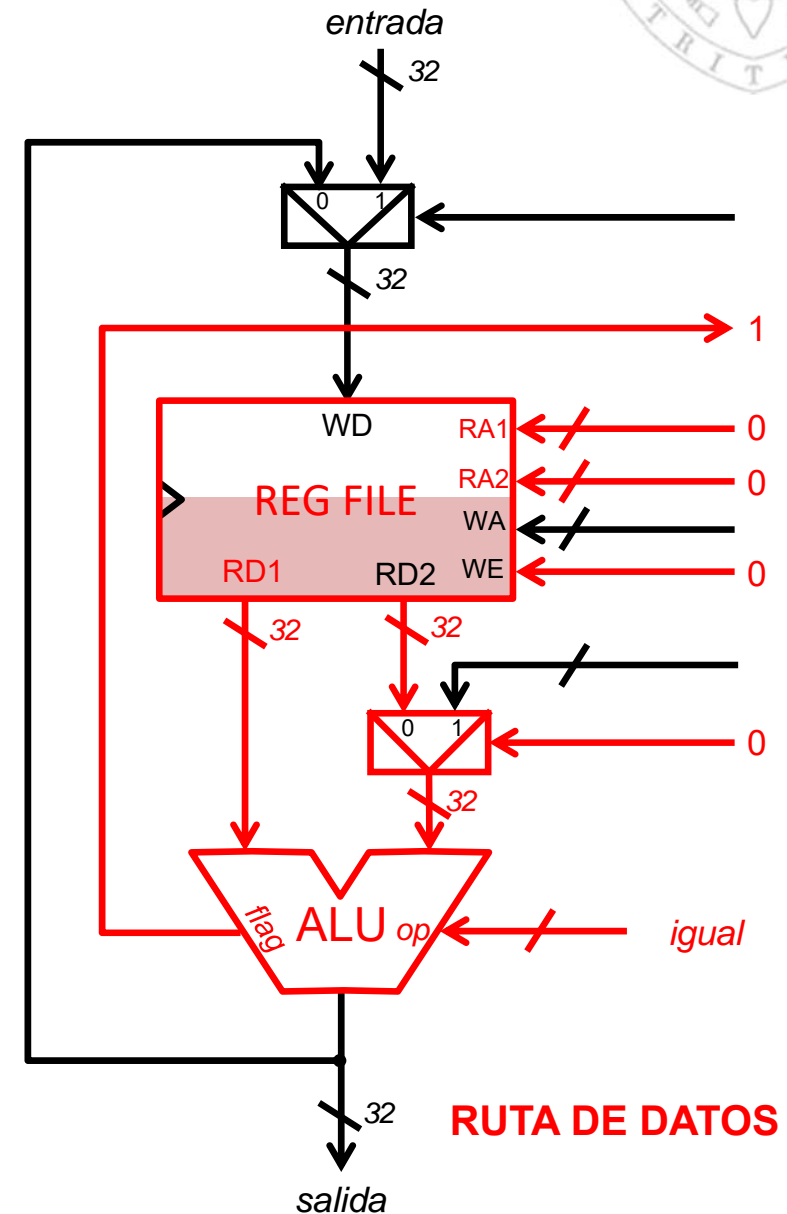
    R = R+A;
  A = A<<1;
  B = B>>1;
};
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3
<b>S4</b>	si R3 > 31, ir a S12
<b>S5</b>	R4 ← R1 & 1
<b>S6</b>	si R4 != 1, ir a S8
<b>S7</b>	R2 ← R2 + R0
<b>S8</b>	R0 ← R0 << 1
<b>S9</b>	R1 ← R1 >> 1
<b>S10</b>	R3 ← R3 + 1
<b>S11</b>	si R0 == R0, ir a S4
<b>S12</b>	salida ← R2
<b>S13</b>	si R0 == R0, ir a S0

R0 es A R1 es B R2 es R R3 es C



**ruta de datos**



# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

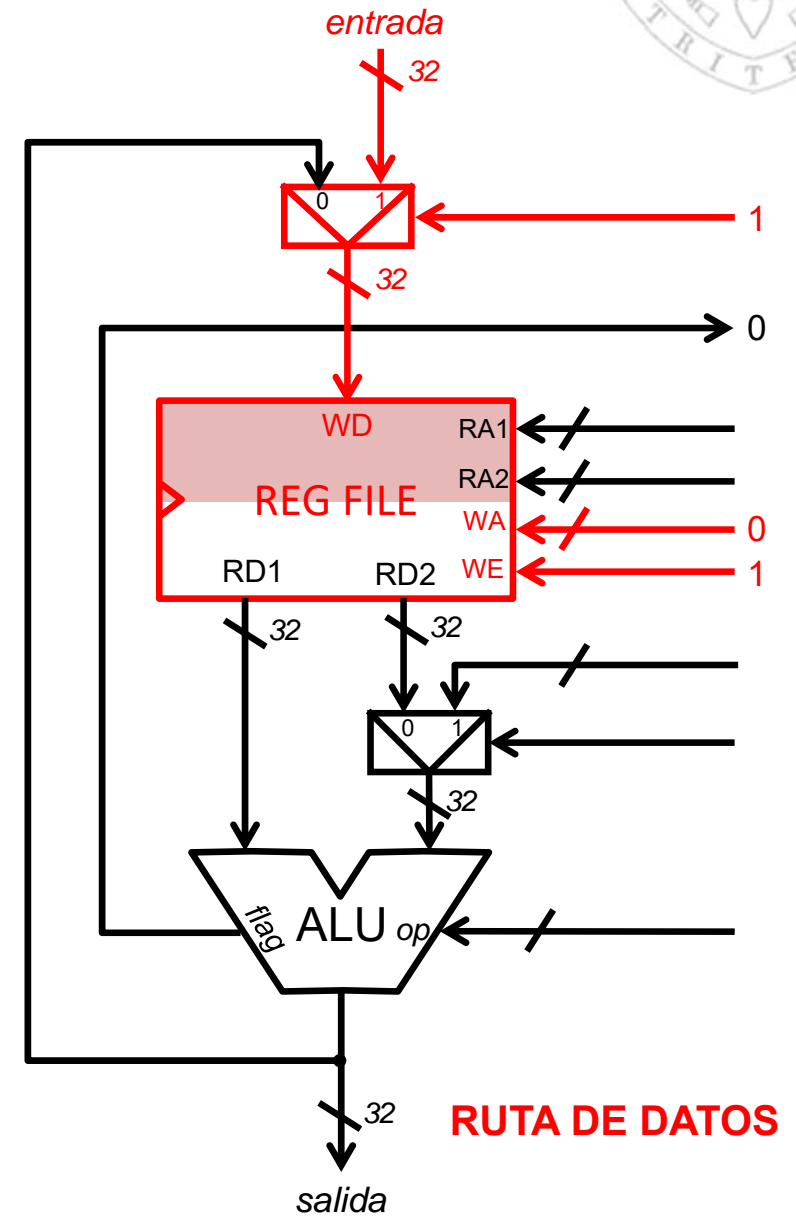
A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
    if( A>B )
      A = A-B;
    else
      B = B-A;

  R = A;
};
Rout = R;

```

S0 R0 ← entrada

R0 es A



RUTA DE DATOS



# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

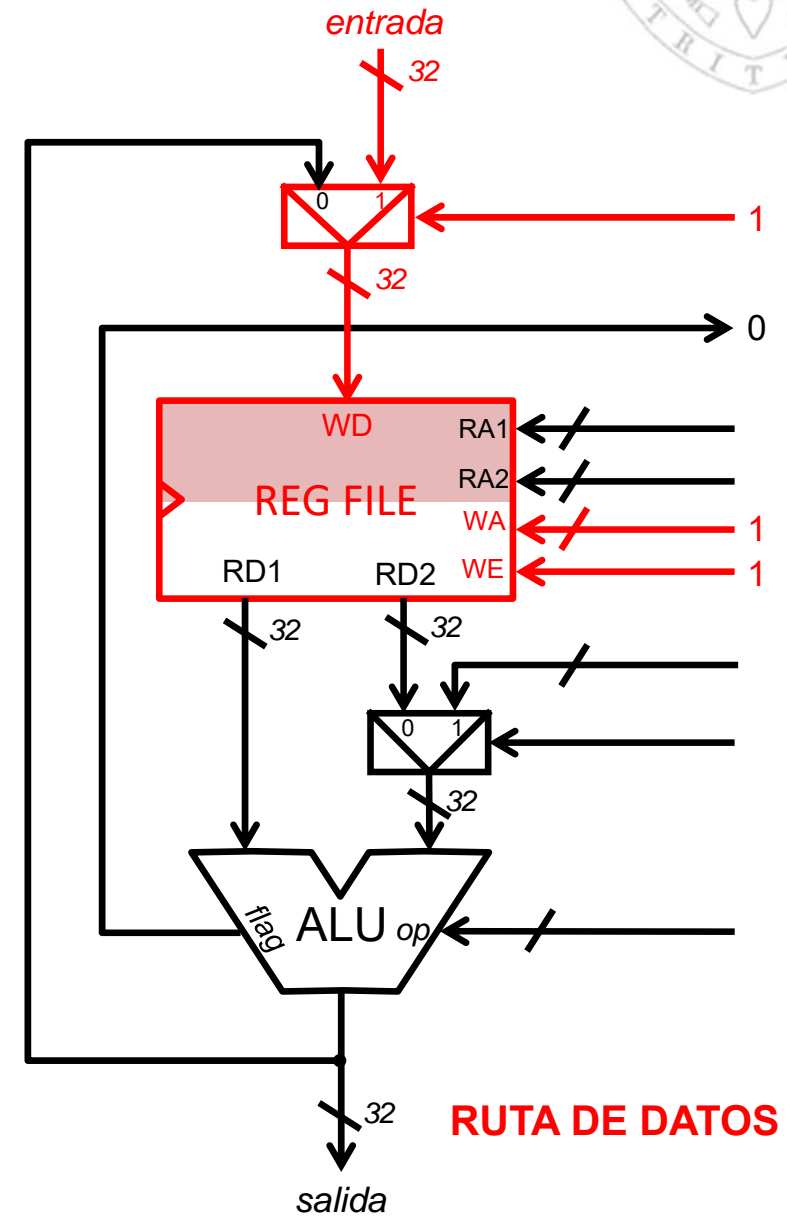
A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
  {
    if( A>B )
      A = A-B;
    else
      B = B-A;
  }
  R = A;
};
Rout = R;

```

Estados y transferencias entre registros

- S0 R0 ← entrada
- S1 R1 ← entrada

R0 es A R1 es B



RUTA DE DATOS



# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
    if( A>B )
      A = A-B;
    else
      B = B-A;

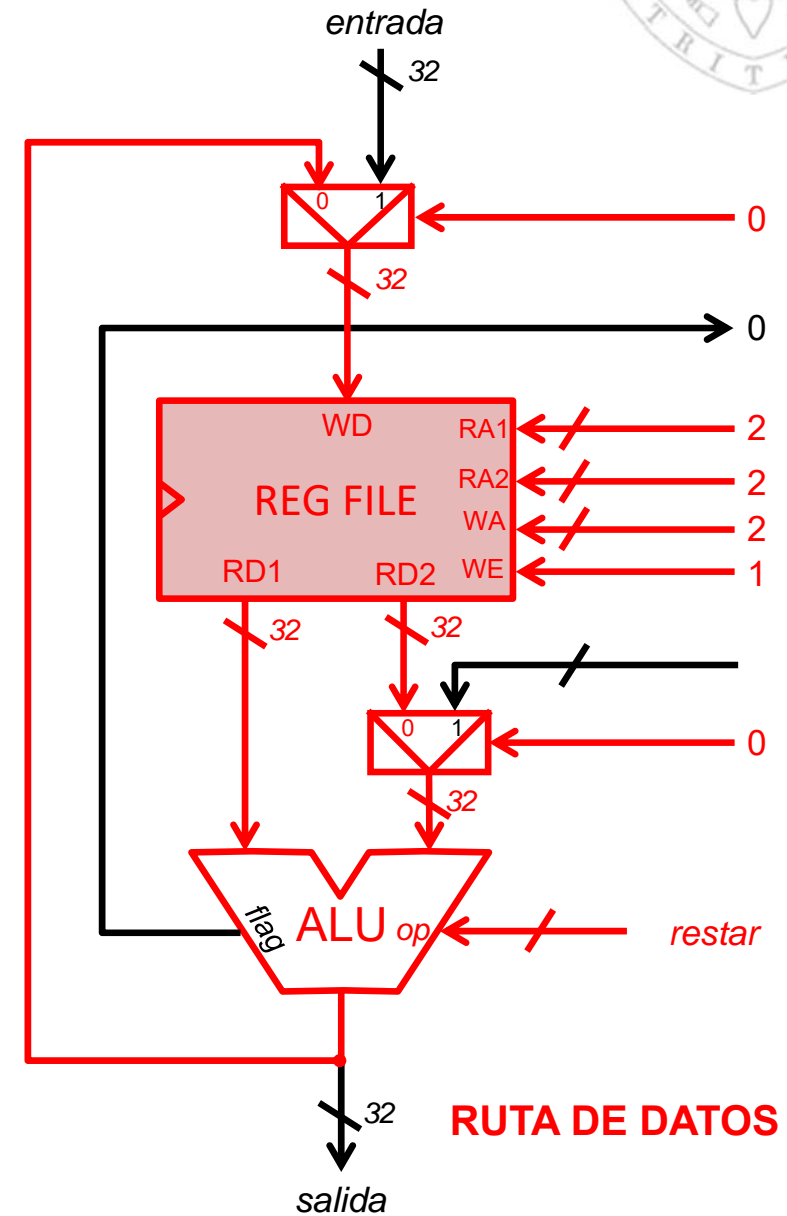
  R = A;
};
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 - R2

R0 es A R1 es B R2 es R



RUTA DE DATOS



# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
    if( A>B )
      A = A-B;
    else
      B = B-A;

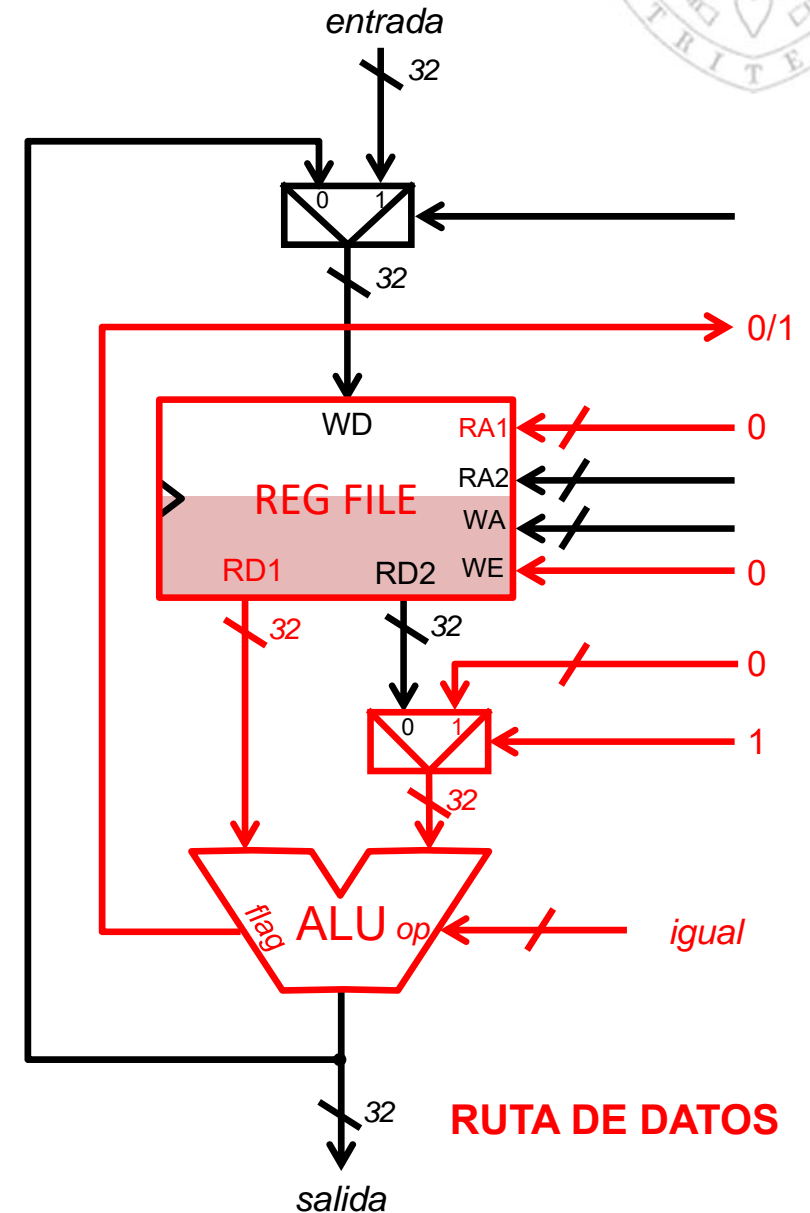
  R = A;
};
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 - R2
<b>S3</b>	si R0 == 0, ir a S12

R0 es A R1 es B R2 es R





# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

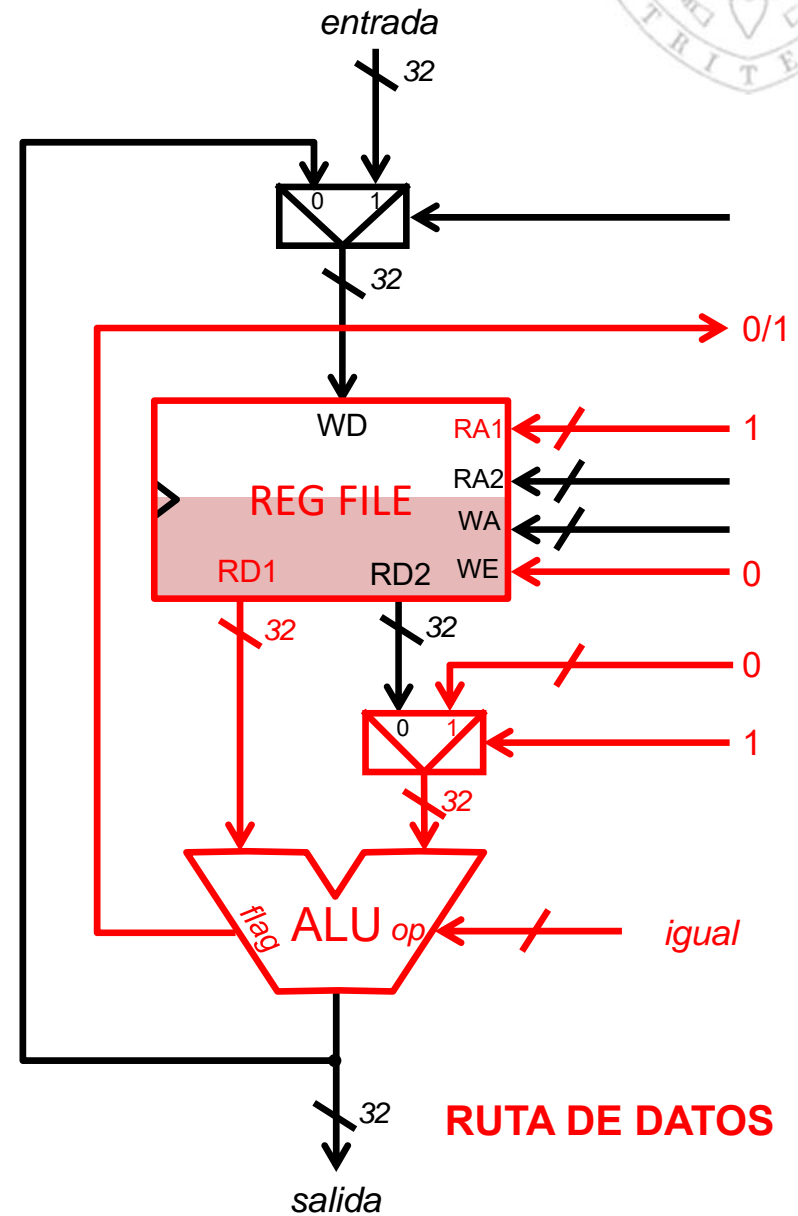
A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
  {
    if( A>B )
      A = A-B;
    else
      B = B-A;
  }
  Rout = R;
}

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 - R2
<b>S3</b>	si R0 == 0, ir a S12
<b>S4</b>	si R1 == 0, ir a S12

R0 es A R1 es B R2 es R





# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

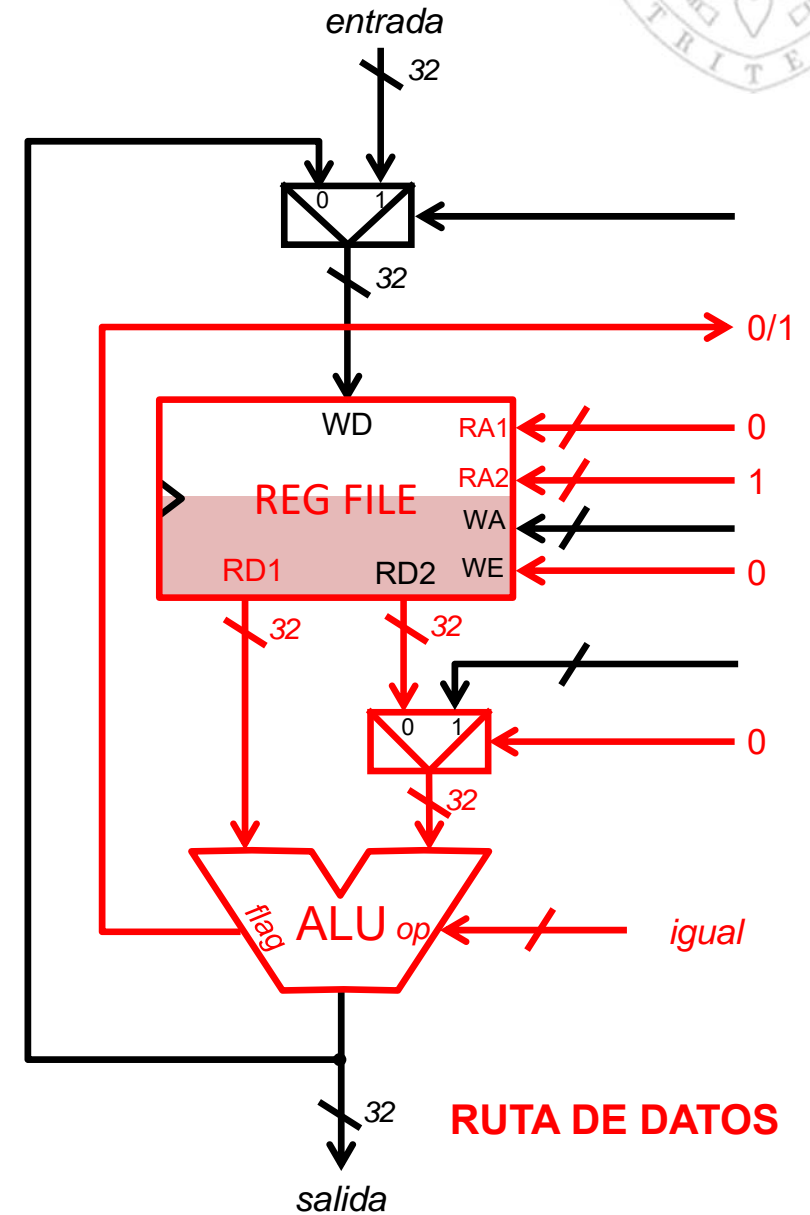
A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
  {
    if( A>B )
      A = A-B;
    else
      B = B-A;
  }
  R = A;
}
Rout = R;

```

Estados y transferencias entre registros

- S0** R0 ← entrada
- S1** R1 ← entrada
- S2** R2 ← R2 - R2
- S3** si R0 == 0, ir a S12
- S4** si R1 == 0, ir a S12
- S5** si R0 == R1, ir a S11

R0 es A R1 es B R2 es R



RUTA DE DATOS



# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
    if( A>B )
      A = A-B;
    else
      B = B-A;

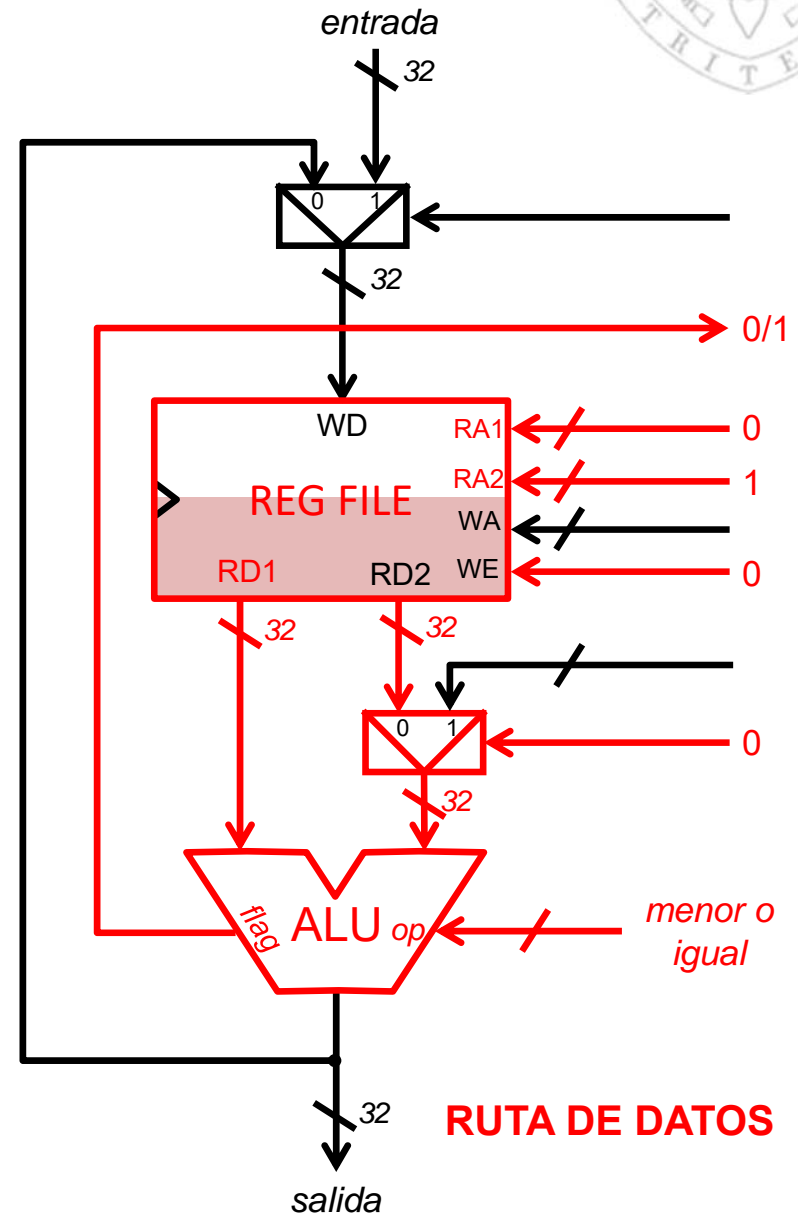
  R = A;
};
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 - R2
<b>S3</b>	si R0 == 0, ir a S12
<b>S4</b>	si R1 == 0, ir a S12
<b>S5</b>	si R0 == R1, ir a S11
<b>S6</b>	si R0 <= R1, ir a S9

R0 es A R1 es B R2 es R







# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

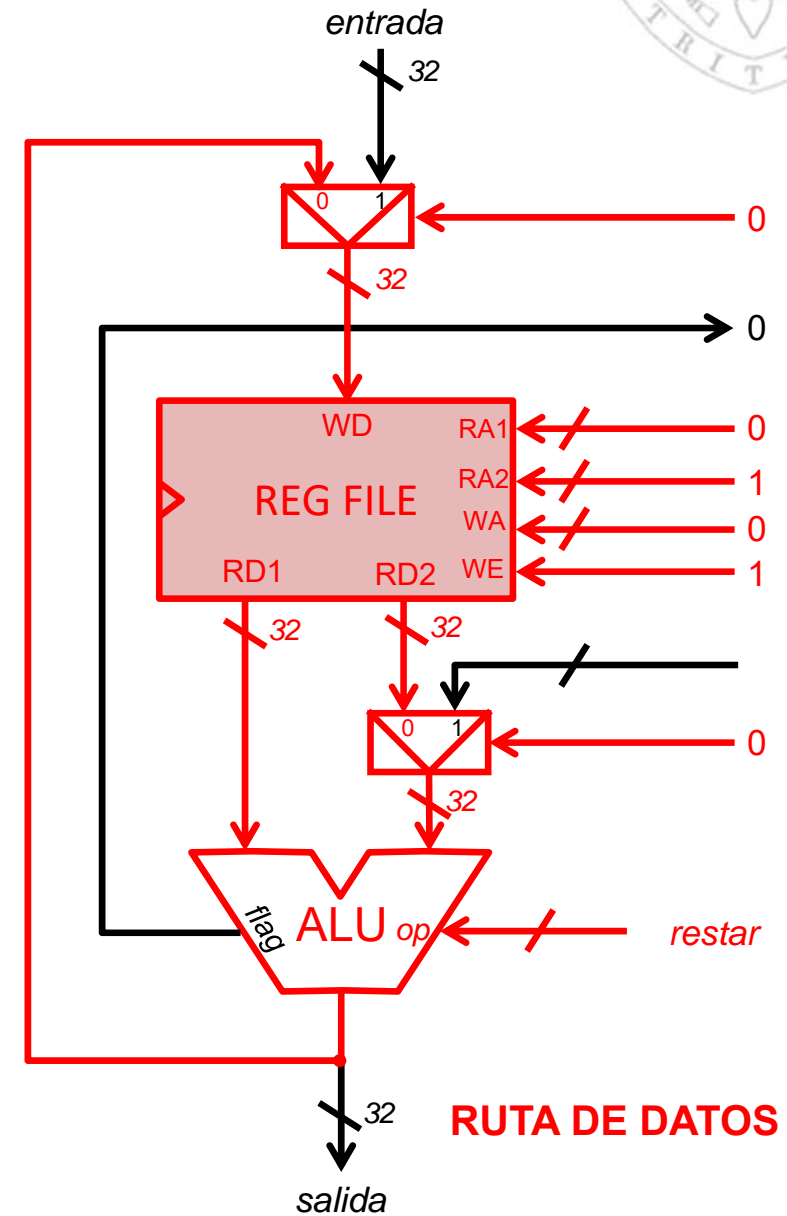
A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
  {
    if( A>B )
      A = A-B;
    else
      B = B-A;
  }
  Rout = R;
}

```

Estados y transferencias entre registros

<b>S0</b>	$R0 \leftarrow \text{entrada}$
<b>S1</b>	$R1 \leftarrow \text{entrada}$
<b>S2</b>	$R2 \leftarrow R2 - R2$
<b>S3</b>	si $R0 == 0$ , ir a S12
<b>S4</b>	si $R1 == 0$ , ir a S12
<b>S5</b>	si $R0 == R1$ , ir a S11
<b>S6</b>	si $R0 \leq R1$ , ir a S9
<b>S7</b>	$R0 \leftarrow R0 - R1$

R0 es A R1 es B R2 es R



RUTA DE DATOS



# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

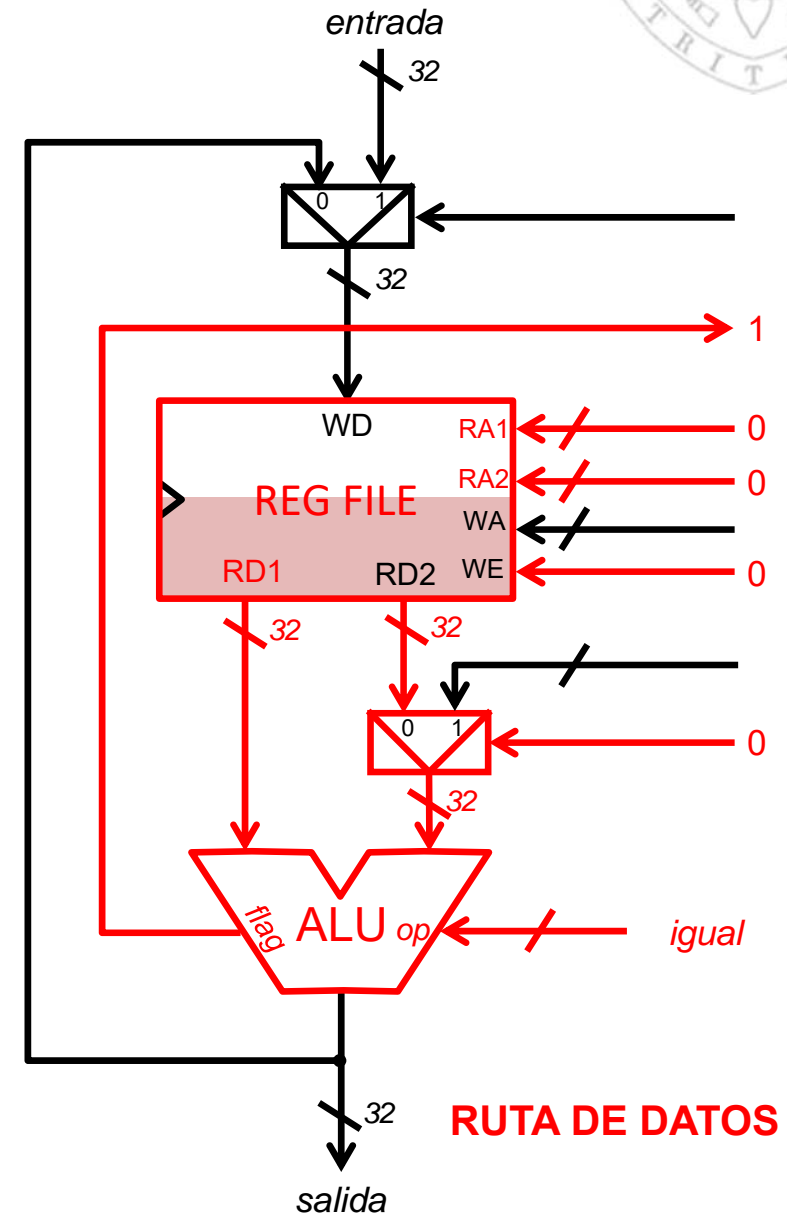
A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
  {
    if( A>B )
      A = A-B;
    else
      B = B-A;
  }
  Rout = R;
}

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 - R2
<b>S3</b>	si R0 == 0, ir a S12
<b>S4</b>	si R1 == 0, ir a S12
<b>S5</b>	si R0 == R1, ir a S11
<b>S6</b>	si R0 <= R1, ir a S9
<b>S7</b>	R0 ← R0 - R1
<b>S8</b>	si R0 == R0, ir a S5

R0 es A R1 es B R2 es R





# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
    if( A>B )
      A = A-B;
    else
      B = B-A;

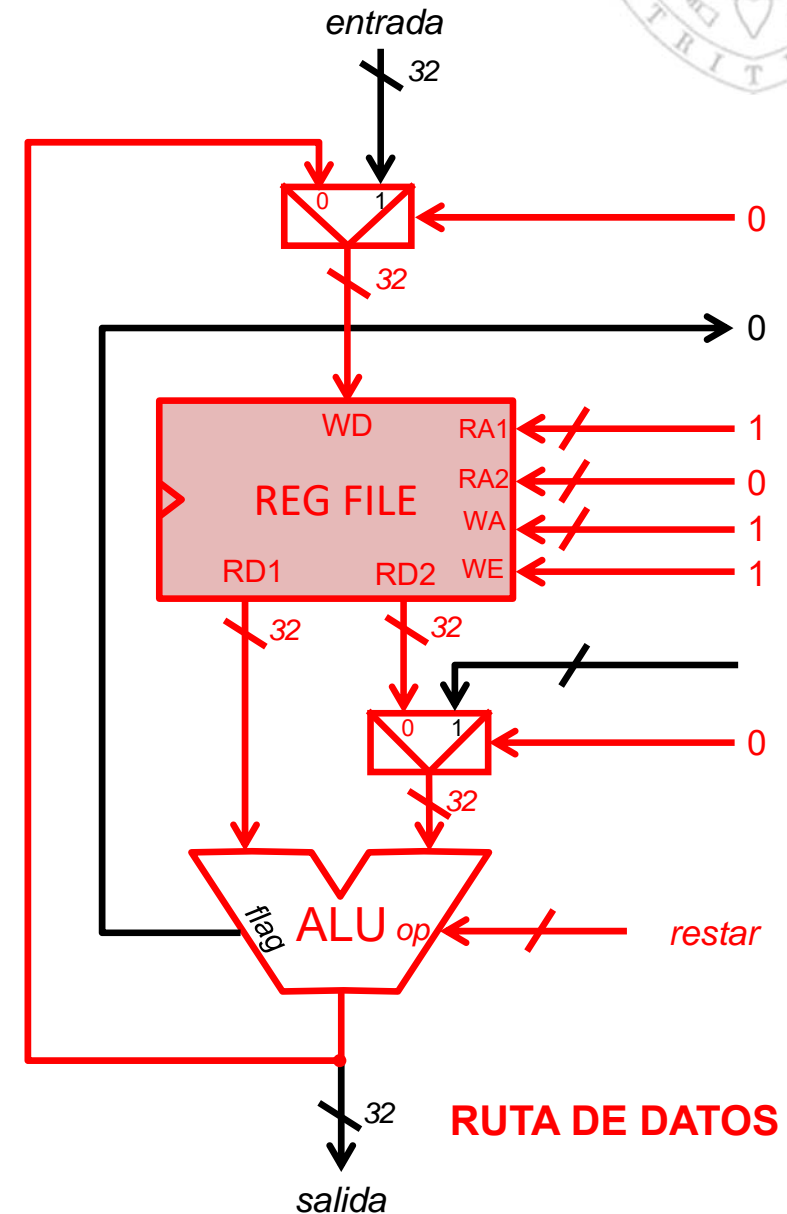
  R = A;
};
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 - R2
<b>S3</b>	si R0 == 0, ir a S12
<b>S4</b>	si R1 == 0, ir a S12
<b>S5</b>	si R0 == R1, ir a S11
<b>S6</b>	si R0 <= R1, ir a S9
<b>S7</b>	R0 ← R0 - R1
<b>S8</b>	si R0 == R0, ir a S5
<b>S9</b>	R1 ← R1 - R0

R0 es A R1 es B R2 es R





# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

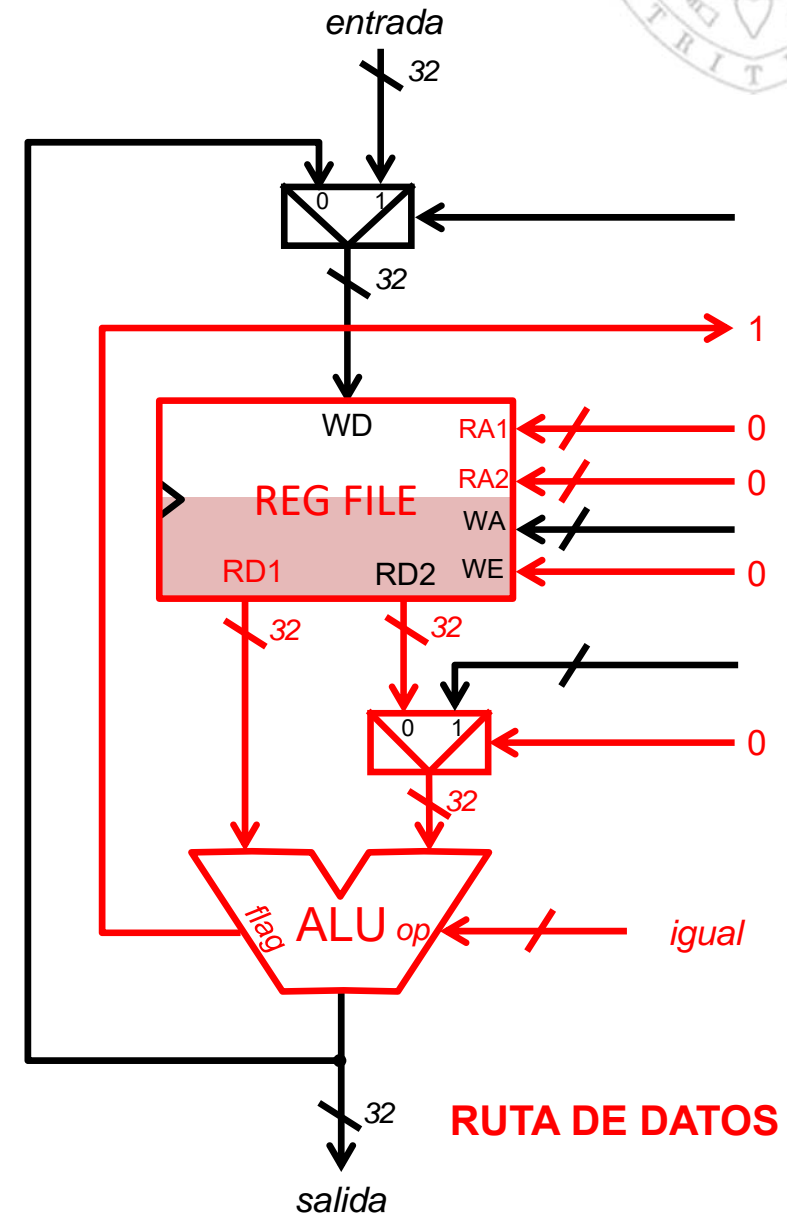
A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
  {
    if( A>B )
      A = A-B;
    else
      B = B-A;
  }
  R = A;
}
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 - R2
<b>S3</b>	si R0 == 0, ir a S12
<b>S4</b>	si R1 == 0, ir a S12
<b>S5</b>	si R0 == R1, ir a S11
<b>S6</b>	si R0 <= R1, ir a S9
<b>S7</b>	R0 ← R0 - R1
<b>S8</b>	si R0 == R0, ir a S5
<b>S9</b>	R1 ← R1 - R0
<b>S10</b>	si R0 == R0, ir a S5

R0 es A R1 es B R2 es R





# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

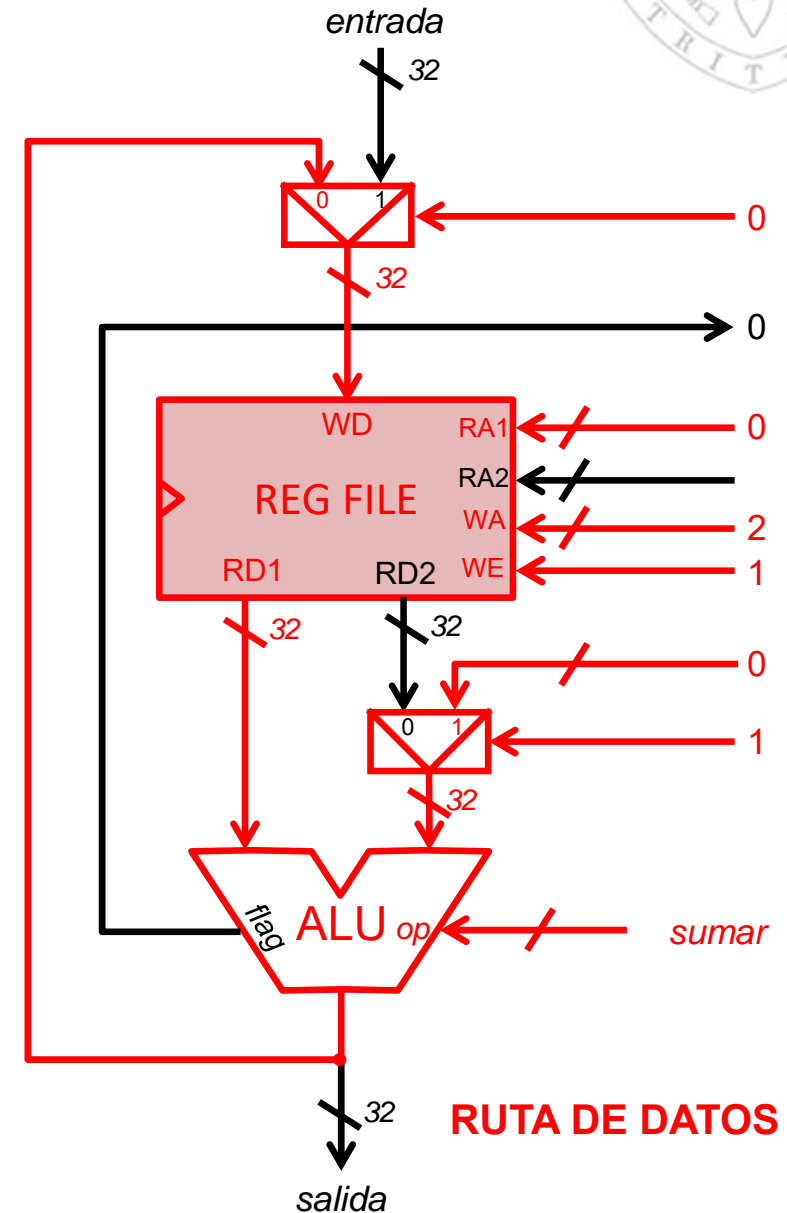
A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
  {
    if( A>B )
      A = A-B;
    else
      B = B-A;
  }
  R = A;
};
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 - R2
<b>S3</b>	si R0 == 0, ir a S12
<b>S4</b>	si R1 == 0, ir a S12
<b>S5</b>	si R0 == R1, ir a S11
<b>S6</b>	si R0 <= R1, ir a S9
<b>S7</b>	R0 ← R0 - R1
<b>S8</b>	si R0 == R0, ir a S5
<b>S9</b>	R1 ← R1 - R0
<b>S10</b>	si R0 == R0, ir a S5
<b>S11</b>	R2 ← R0 + 0

R0 es A R1 es B R2 es R





# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
    if( A>B )
      A = A-B;
    else
      B = B-A;

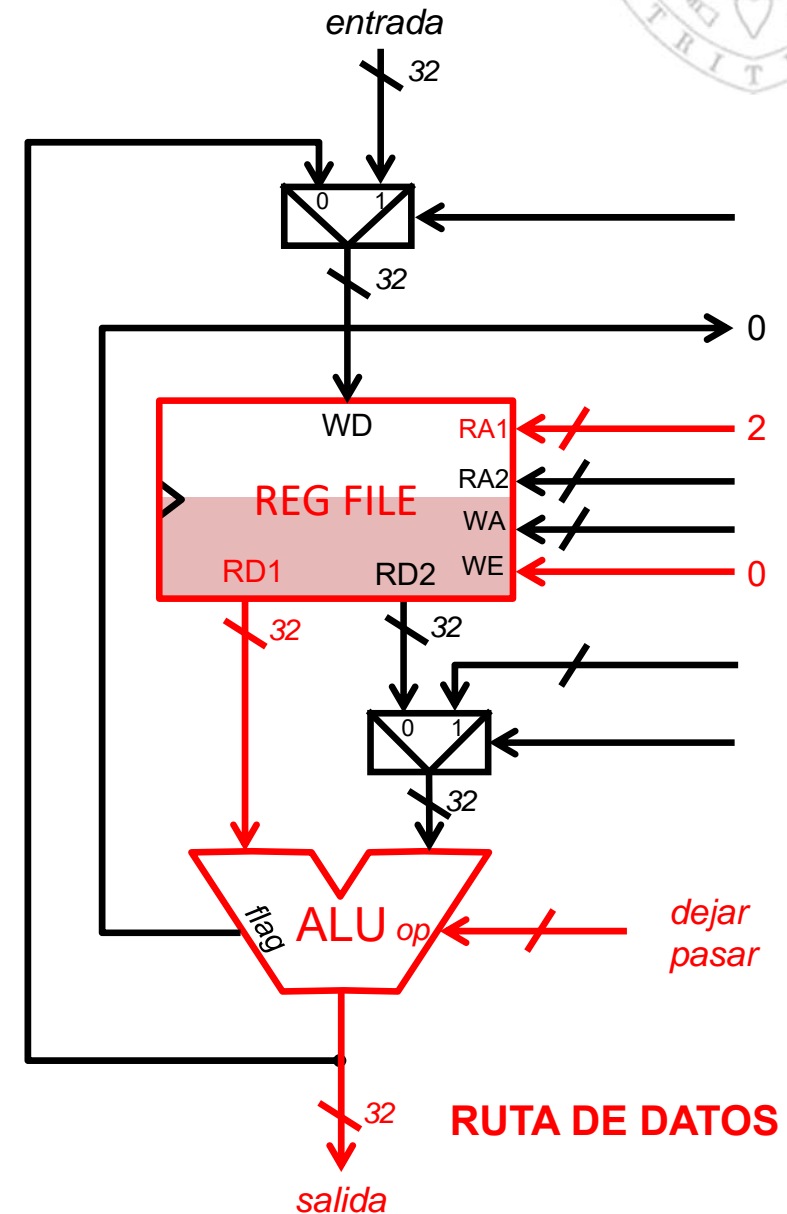
  R = A;
};
Rout = R;

```

Estados y transferencias entre registros

S0	R0 ← entrada
S1	R1 ← entrada
S2	R2 ← R2 - R2
S3	si R0 == 0, ir a S12
S4	si R1 == 0, ir a S12
S5	si R0 == R1, ir a S11
S6	si R0 <= R1, ir a S9
S7	R0 ← R0 - R1
S8	si R0 == R0, ir a S5
S9	R1 ← R1 - R0
S10	si R0 == R0, ir a S5
S11	R2 ← R0 + 0
S12	salida ← R2

R0 es A R1 es B R2 es R





# Ruta de datos de propósito general

## Máximo común divisor

Algoritmo para calcular el MCD de 2 números de 32 bits

```

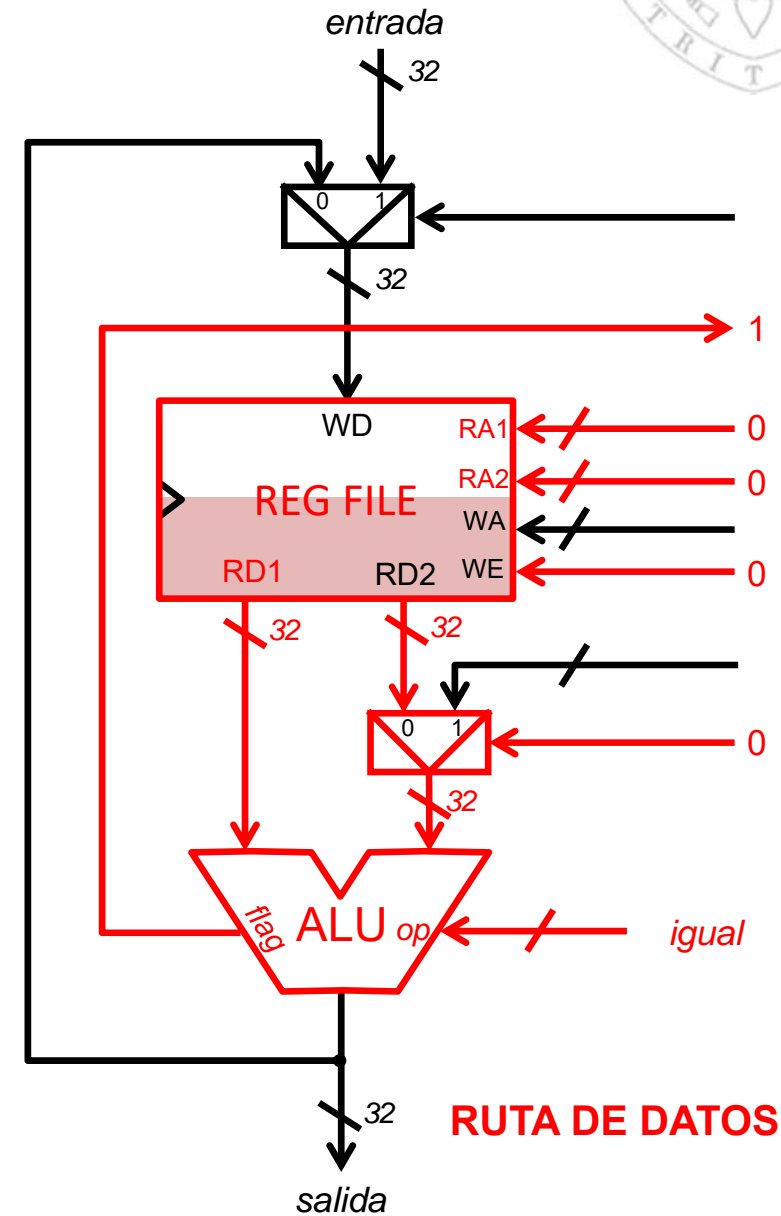
A = Ain;
B = Bin;
R = 0;
if( A!=0 && B!=0 )
{
  while( A!=B )
  {
    if( A>B )
      A = A-B;
    else
      B = B-A;
  }
  R = A;
}
Rout = R;

```

Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 - R2
<b>S3</b>	si R0 == 0, ir a S12
<b>S4</b>	si R1 == 0, ir a S12
<b>S5</b>	si R0 == R1, ir a S11
<b>S6</b>	si R0 <= R1, ir a S9
<b>S7</b>	R0 ← R0 - R1
<b>S8</b>	si R0 == R0, ir a S5
<b>S9</b>	R1 ← R1 - R0
<b>S10</b>	si R0 == R0, ir a S5
<b>S11</b>	R2 ← R0 + 0
<b>S12</b>	salida ← R2
<b>S13</b>	si R0 == R0, ir a S0

R0 es A R1 es B R2 es R



# Controlador implementado en ROM

## Multiplicación



### Estados y transferencias entre registros

<b>S0</b>	R0 ← entrada
<b>S1</b>	R1 ← entrada
<b>S2</b>	R2 ← R2 – R2
<b>S3</b>	R3 ← R3 – R3
<b>S4</b>	si R3 > 31, ir a S12
<b>S5</b>	R4 ← R1 & 1
<b>S6</b>	si R4 != 1, ir a S8
<b>S7</b>	R2 ← R2 + R0
<b>S8</b>	R0 ← R0 << 1
<b>S9</b>	R1 ← R1 >> 1
<b>S10</b>	R3 ← R3 + 1
<b>S11</b>	si R0 == R0, ir a S4
<b>S12</b>	salida ← R2
<b>S13</b>	si R0 == R0, ir a S0

### Contenido de la ROM (sin codificar)

dir	op	rd	rs1	rs2	cte	salto	dir sig.
0	<i>cargar</i>	0	-	-	-	-	<i>dir+1</i>
1	<i>cargar</i>	1	-	-	-	-	<i>dir+1</i>
2	<i>restar</i>	2	2	2	-	-	<i>dir+1</i>
3	<i>restar</i>	3	3	3	-	-	<i>dir+1</i>
4	<i>saltar si mayor cte</i>	-	3	-	31	12	<i>dir+1</i> ó 12
5	<i>y-lógica cte</i>	4	1	-	1	-	<i>dir+1</i>
6	<i>saltar si distinto cte</i>	-	4	-	1	8	<i>dir+1</i> ó 8
7	<i>sumar</i>	2	2	0	-	-	<i>dir+1</i>
8	<i>desplazar izq cte</i>	0	0	-	1	-	<i>dir+1</i>
9	<i>desplazar der cte</i>	1	1	-	1	-	<i>dir+1</i>
10	<i>sumar cte</i>	3	3	-	1	-	<i>dir+1</i>
11	<i>saltar si igual</i>	-	0	0	-	4	4
12	<i>guardar</i>	-	2	-	-	-	<i>dir+1</i>
13	<i>saltar si igual</i>	-	0	0	-	0	0



# Controlador implementado en ROM

## Máximo común divisor



### Estados y transferencias entre registros

<b>S0</b>	$R0 \leftarrow \text{entrada}$
<b>S1</b>	$R1 \leftarrow \text{entrada}$
<b>S2</b>	$R2 \leftarrow R2 - R2$
<b>S3</b>	si $R0 == 0$ , ir a S12
<b>S4</b>	si $R1 == 0$ , ir a S12
<b>S5</b>	si $R0 == R1$ , ir a S11
<b>S6</b>	si $R0 \leq R1$ , ir a S9
<b>S7</b>	$R0 \leftarrow R0 - R1$
<b>S8</b>	si $R0 == R0$ , ir a S5
<b>S9</b>	$R1 \leftarrow R1 - R0$
<b>S10</b>	si $R0 == R0$ , ir a S5
<b>S11</b>	$R2 \leftarrow R0 + 0$
<b>S12</b>	salida $\leftarrow R2$
<b>S13</b>	si $R0 == R0$ , ir a S0

### Contenido de la ROM (sin codificar)

dir	op	rd	rs1	rs2	cte	salto	dir sig.
0	<i>cargar</i>	0	-	-	-	-	<i>dir+1</i>
1	<i>cargar</i>	1	-	-	-	-	<i>dir+1</i>
2	<i>restar</i>	2	2	2	-	-	<i>dir+1</i>
3	<i>saltar si mayor cte</i>	-	0	-	0	12	<i>dir+1</i> ó 12
4	<i>saltar si mayor cte</i>	-	1	-	0	12	<i>dir+1</i> ó 12
5	<i>saltar si igual</i>	-	0	1	-	11	<i>dir+1</i> ó 11
6	<i>saltar si menor o igual</i>	-	0	1	-	9	<i>dir+1</i> ó 9
7	<i>restar</i>	0	0	1	-	-	<i>dir+1</i>
8	<i>saltar si igual</i>	-	0	0	-	5	5
9	<i>restar</i>	1	1	0	-	-	<i>dir+1</i>
10	<i>saltar si igual</i>	-	0	0	-	5	5
11	<i>sumar cte</i>	2	0	-	0	-	<i>dir+1</i>
12	<i>guardar</i>	-	2	-	-	-	<i>dir+1</i>
13	<i>saltar si igual</i>	-	0	0	-	0	0



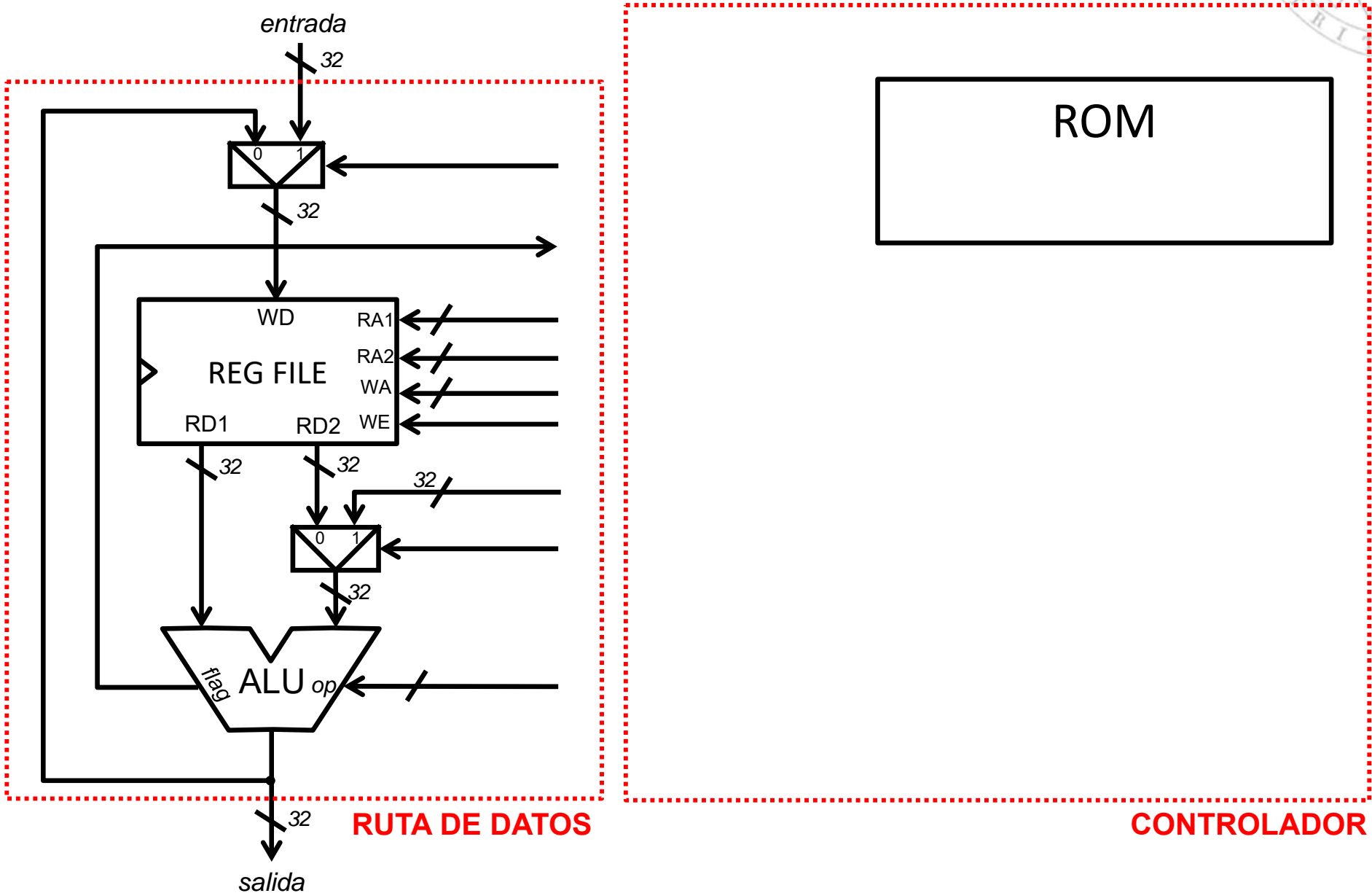
# Ruta de datos general + controlador

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

42



RUTA DE DATOS

CONTROLADOR



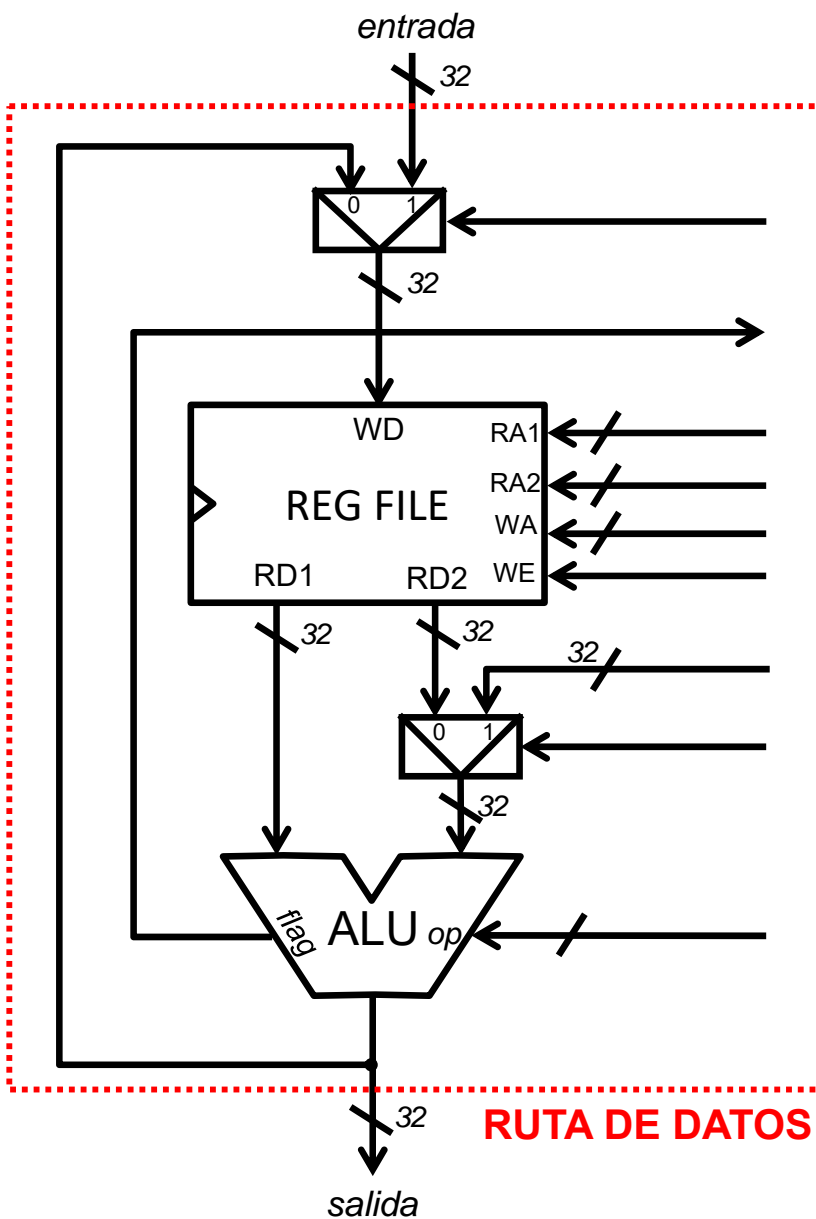
# Ruta de datos general + controlador

versión 15/01/23

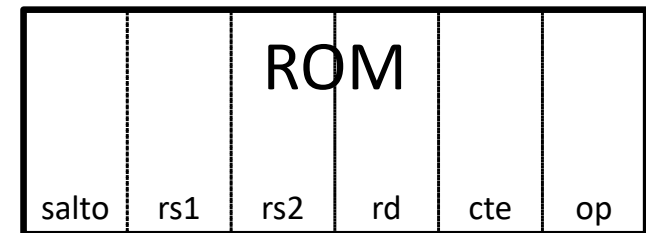
tema 1:  
De sistema digital a computador

FC-2

43



**RUTA DE DATOS**



**CONTROLADOR**



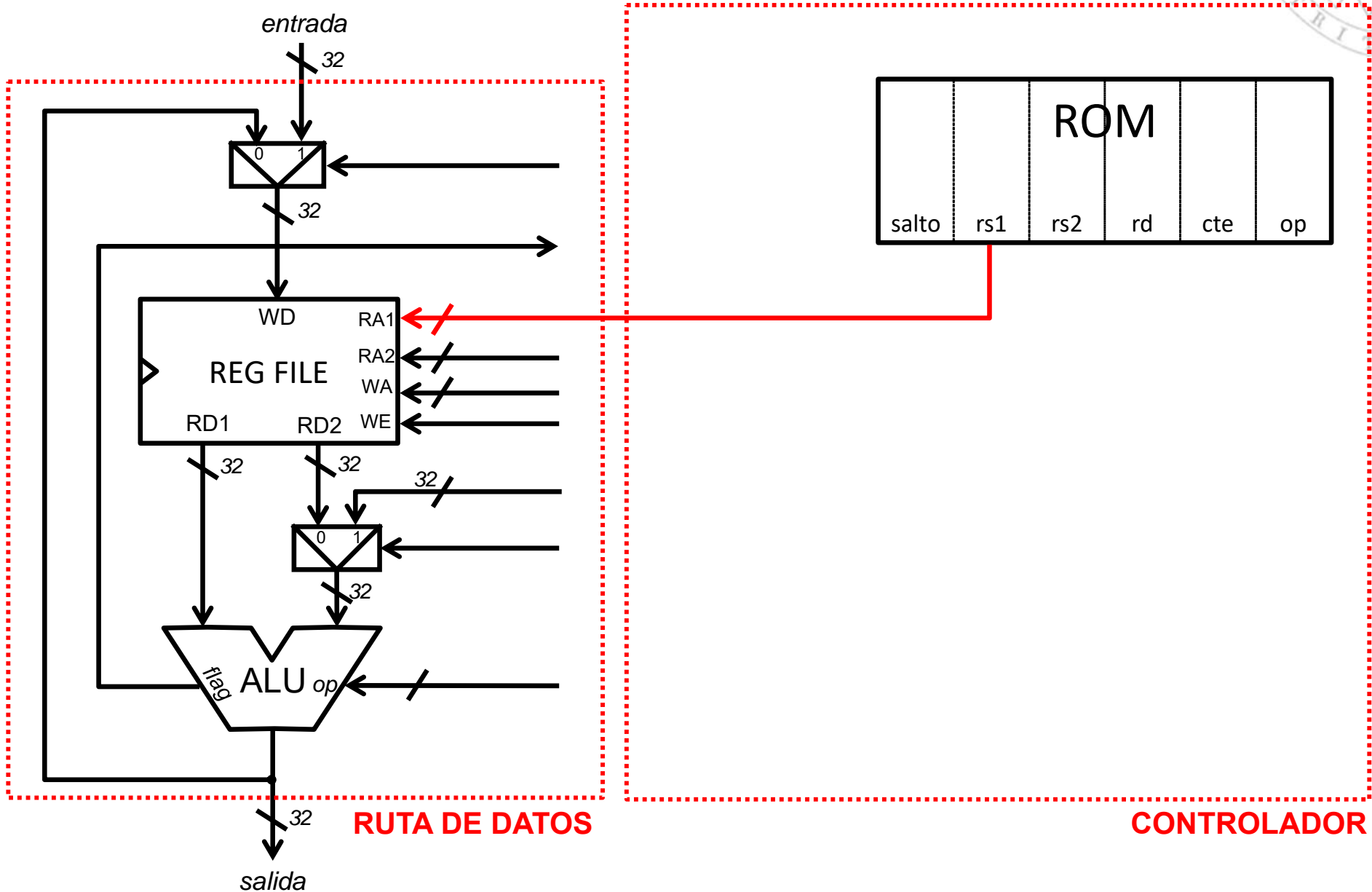
# Ruta de datos general + controlador

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

44



RUTA DE DATOS

CONTROLADOR



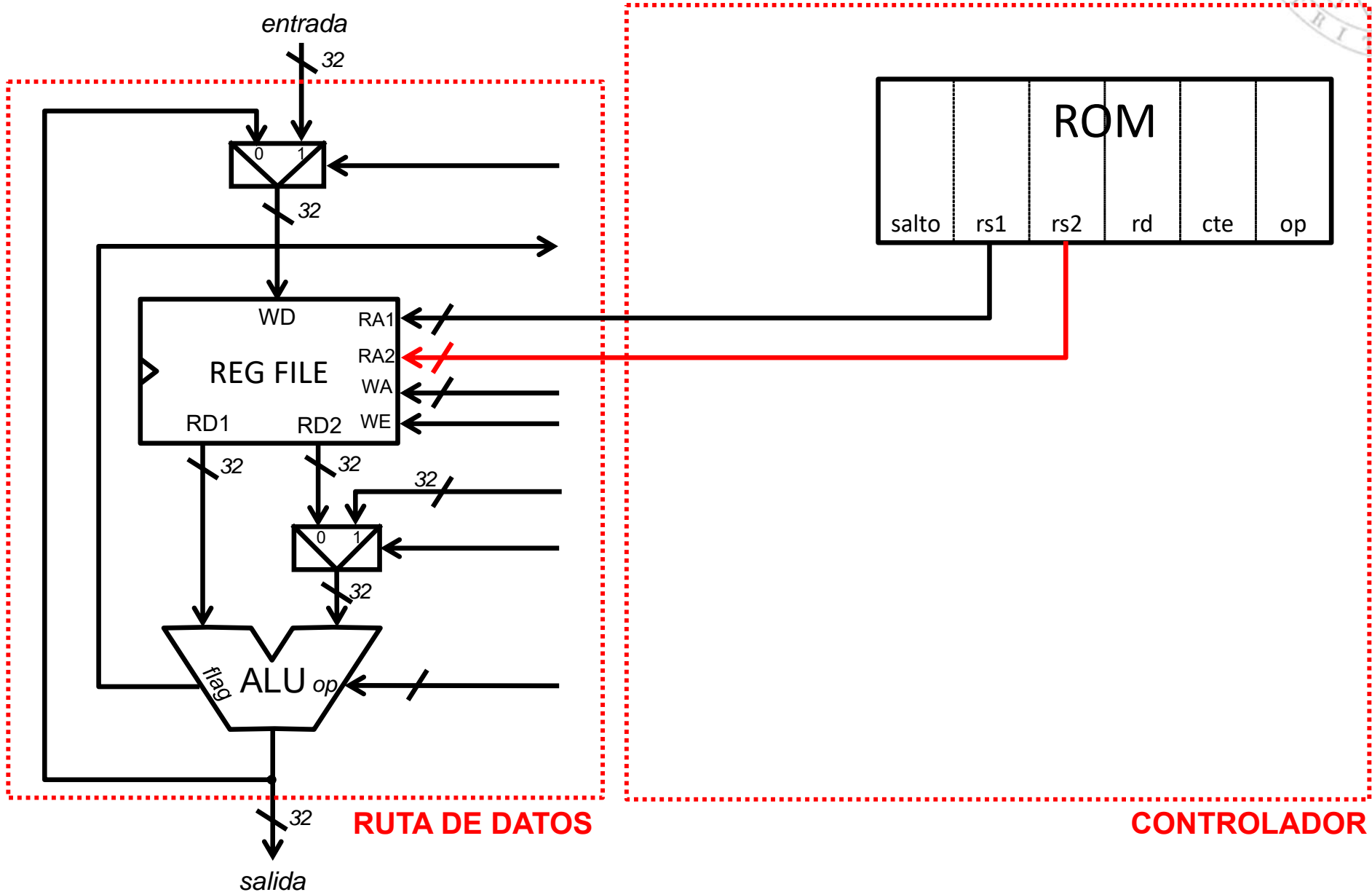
# Ruta de datos general + controlador

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

45





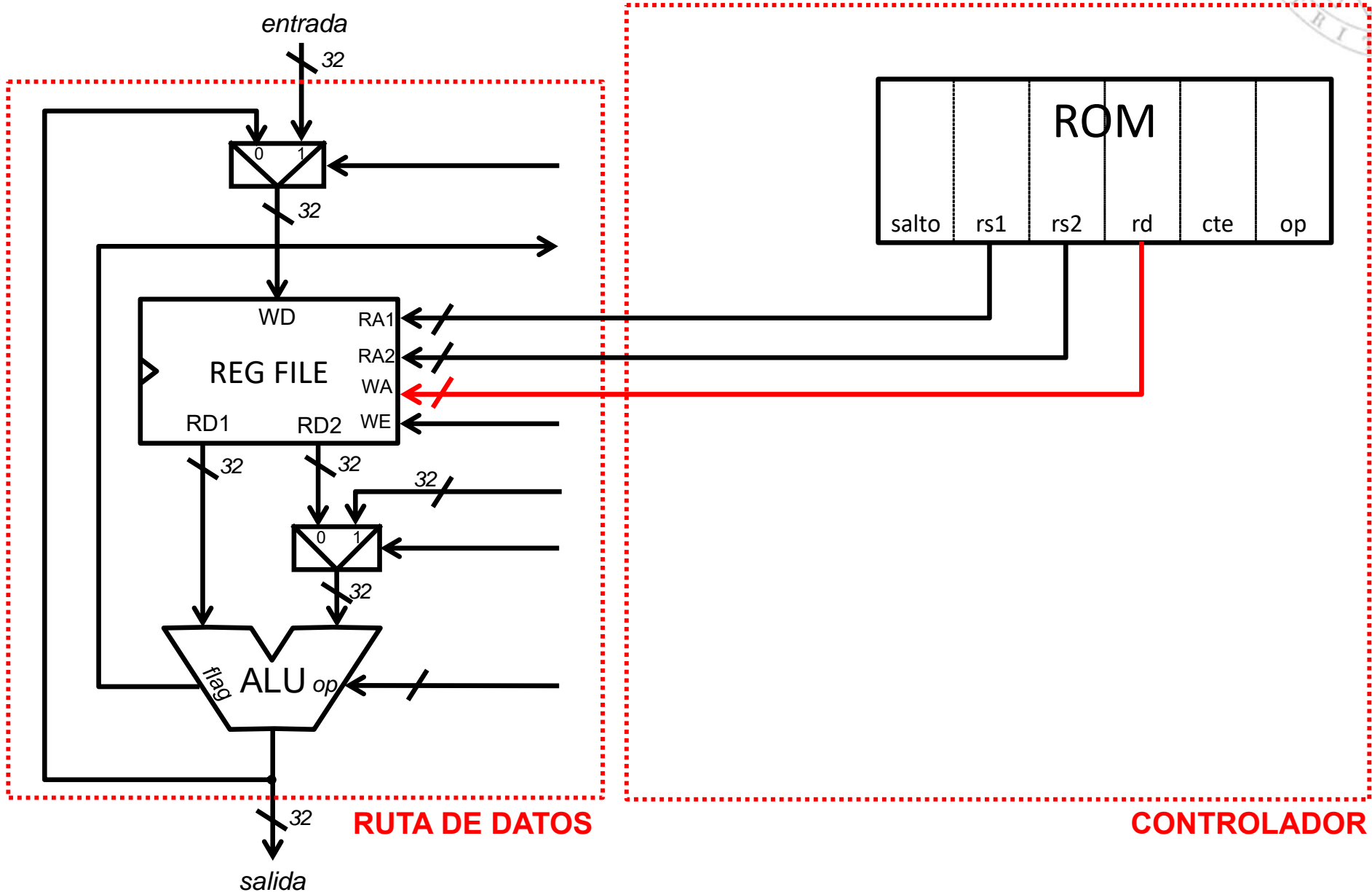
# Ruta de datos general + controlador

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

46



RUTA DE DATOS

CONTROLADOR



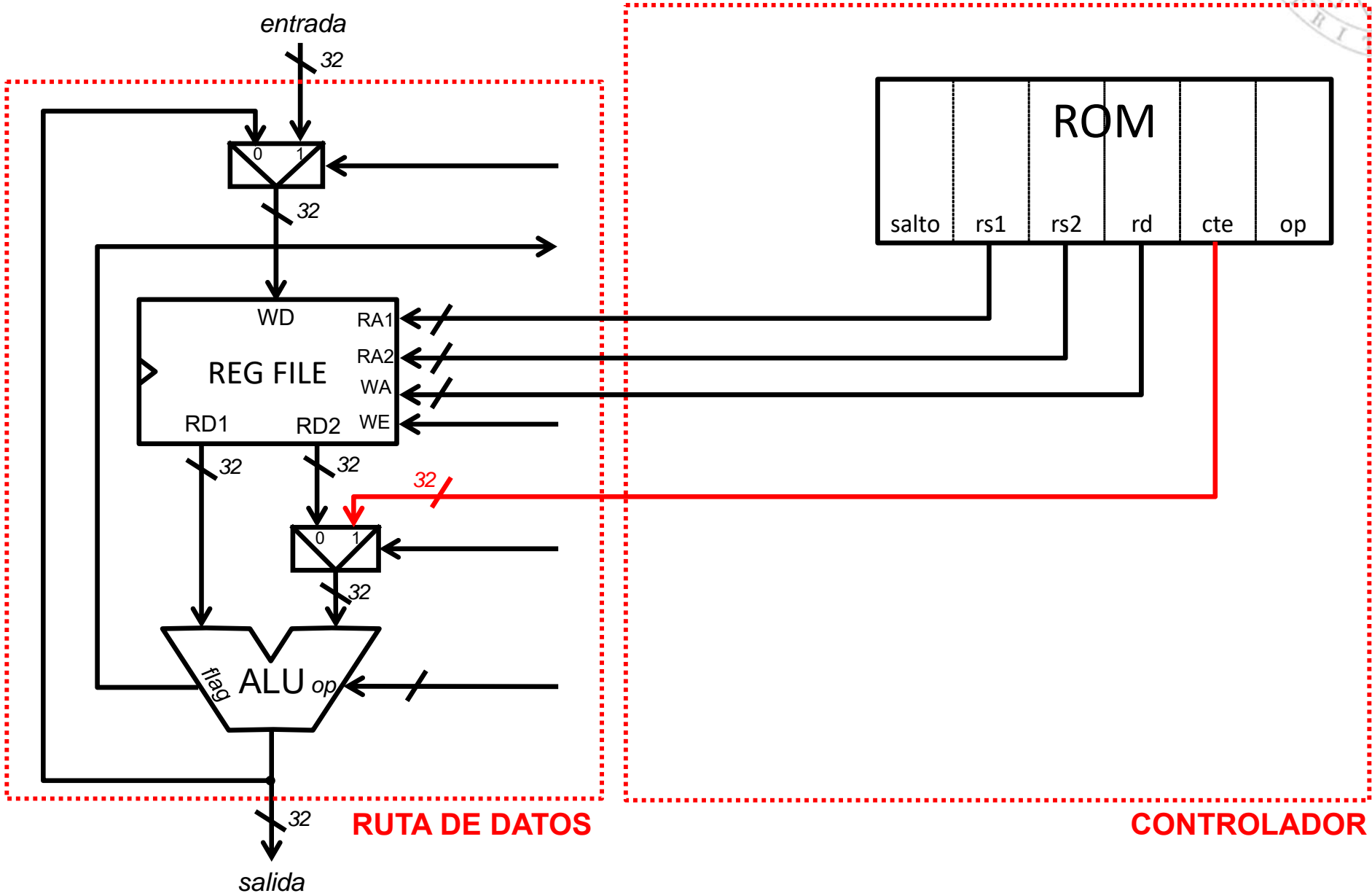
# Ruta de datos general + controlador

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

47



RUTA DE DATOS

CONTROLADOR



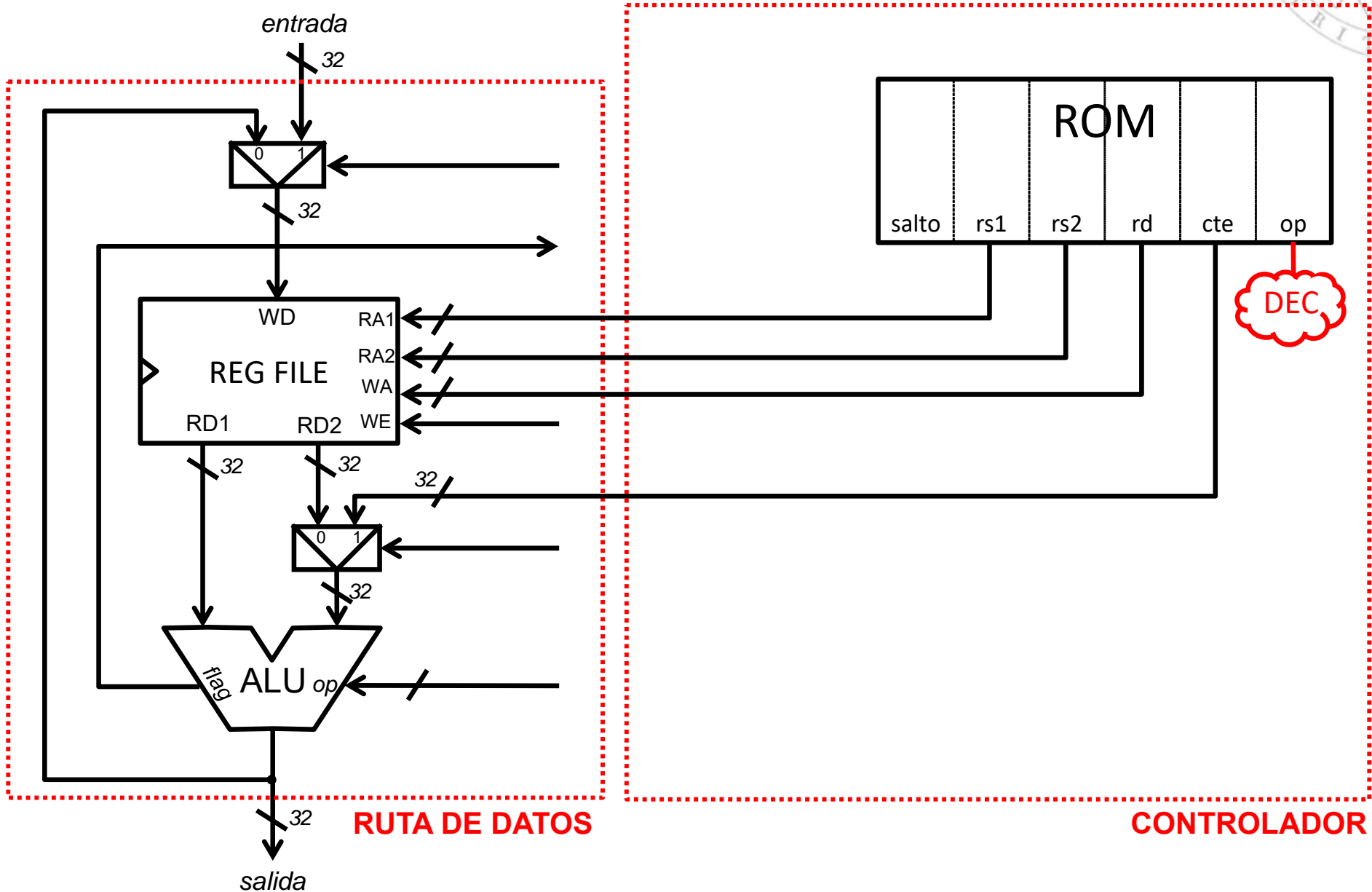
# Ruta de datos general + controlador

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

48







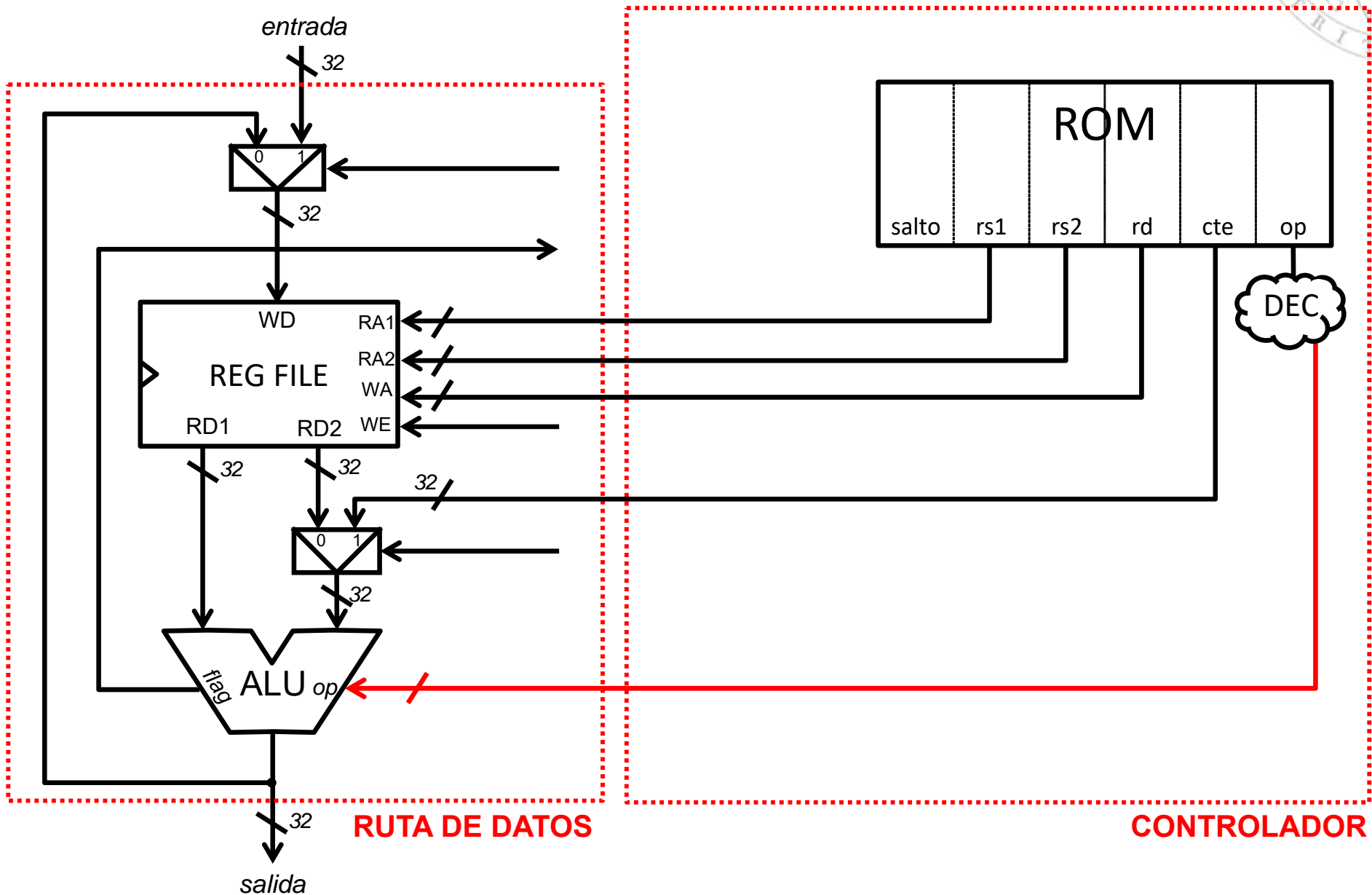
# Ruta de datos general + controlador

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

49



RUTA DE DATOS

CONTROLADOR



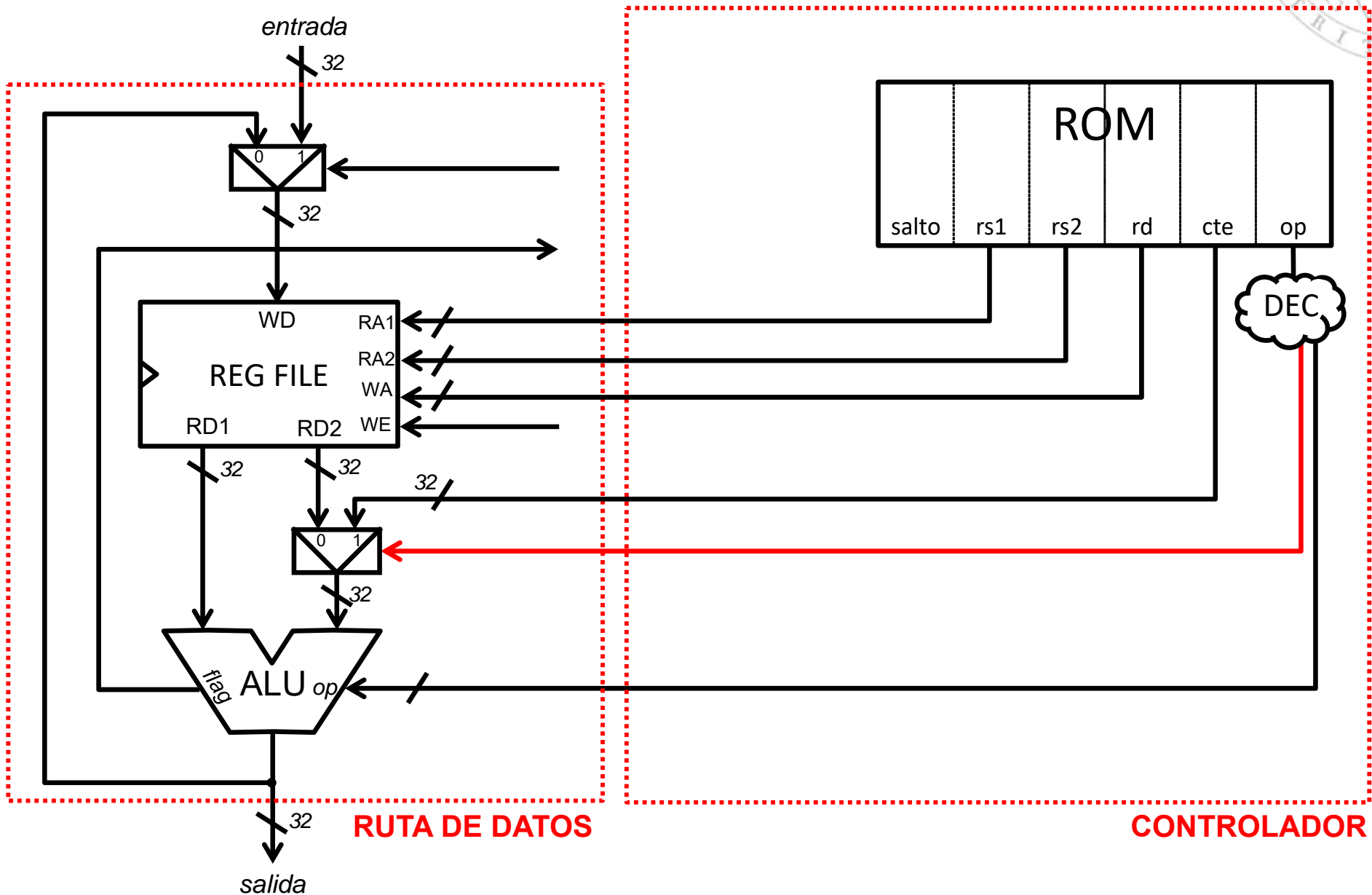
# Ruta de datos general + controlador

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

50



RUTA DE DATOS

CONTROLADOR



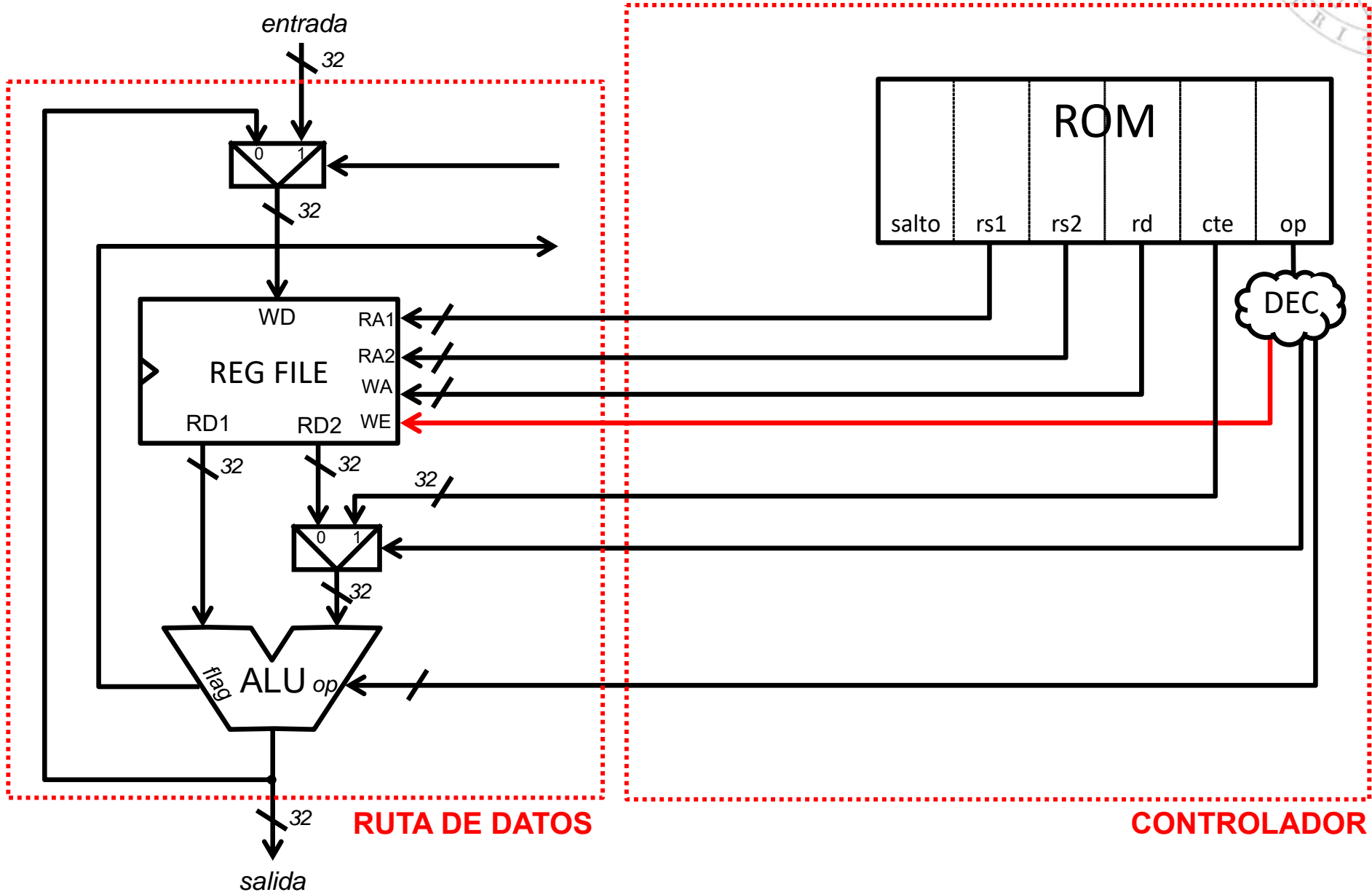
# Ruta de datos general + controlador

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

51



**RUTA DE DATOS**

**CONTROLADOR**



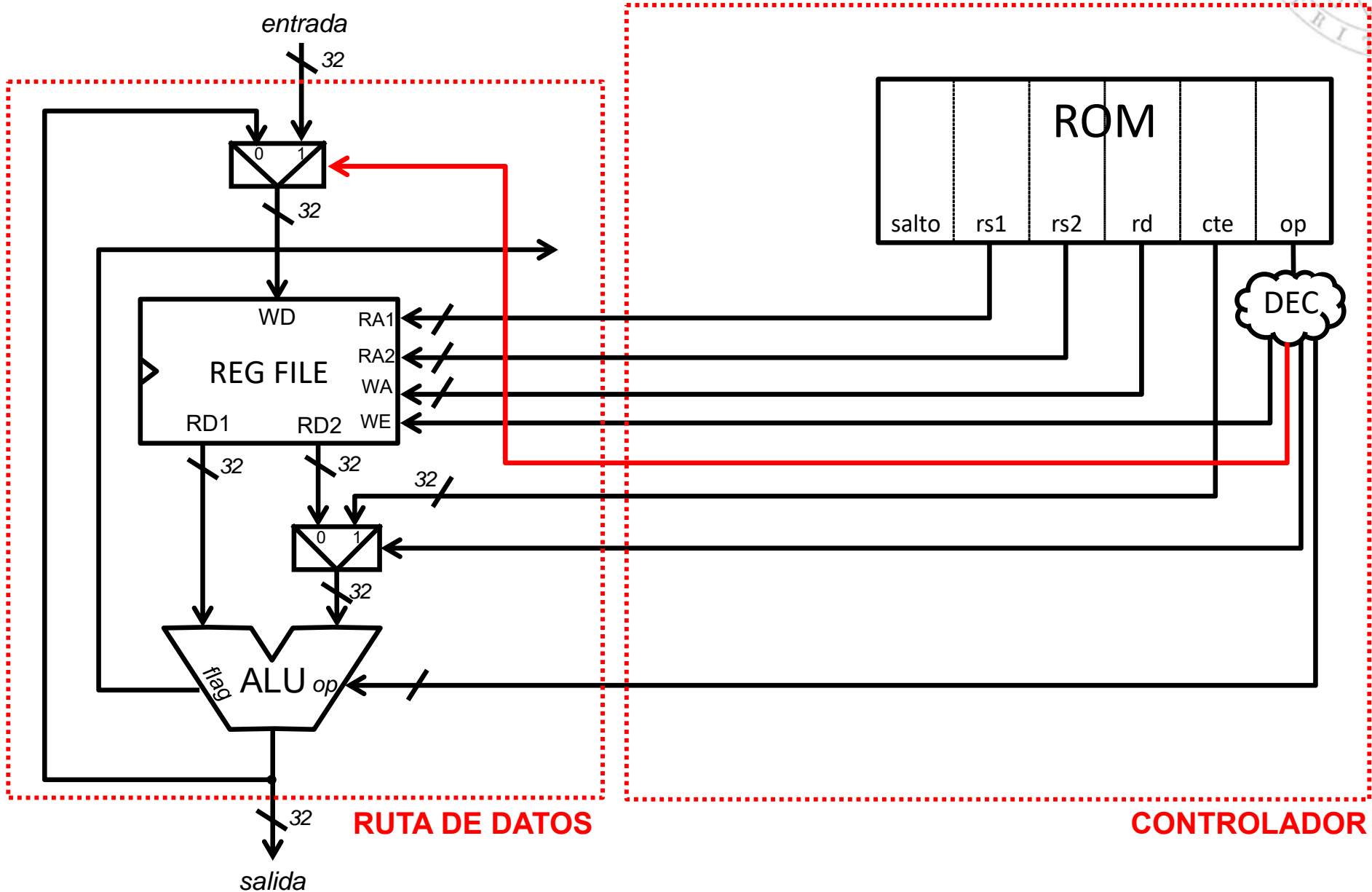
# Ruta de datos general + controlador

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

52





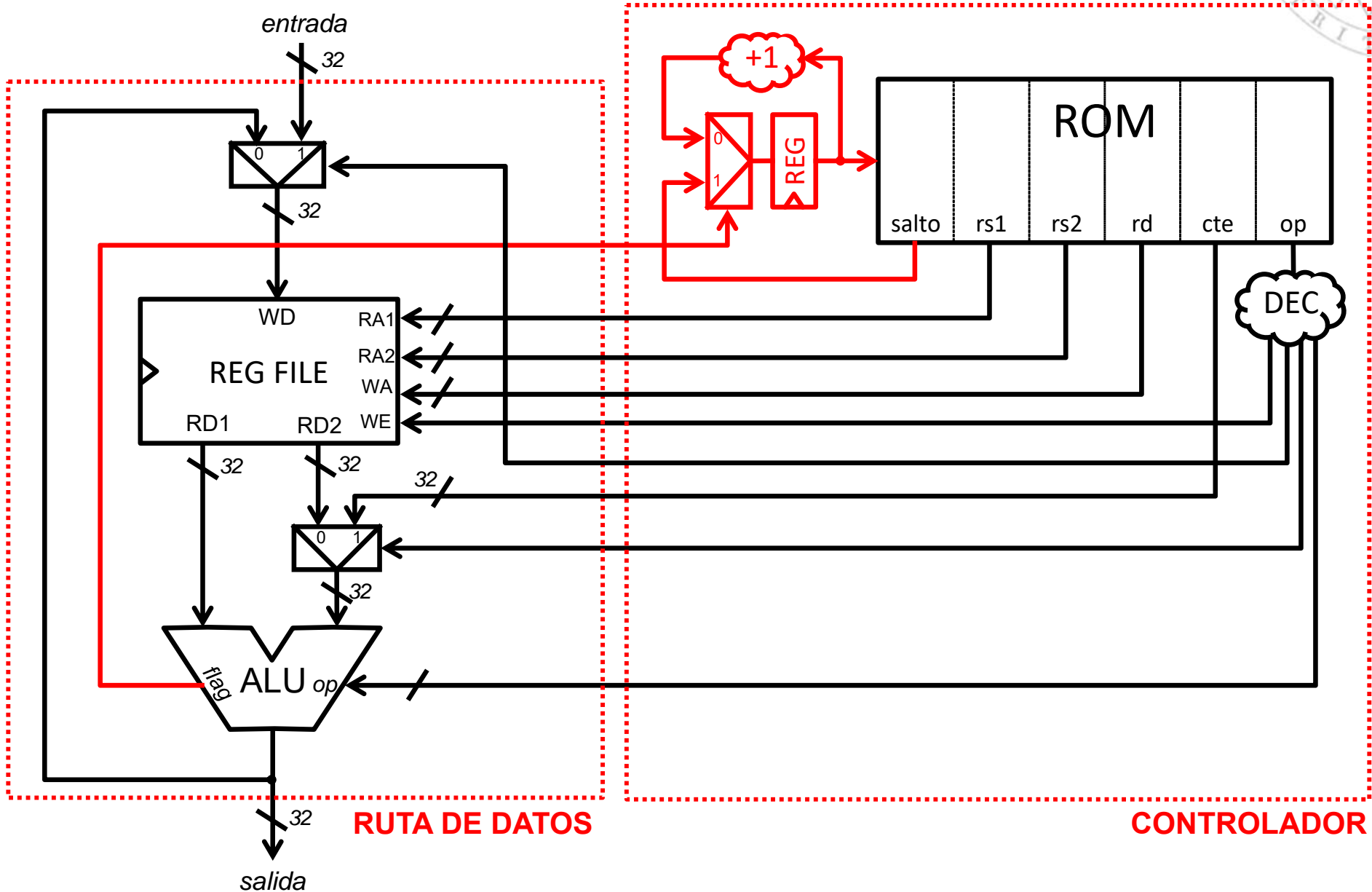
# Ruta de datos general + controlador

versión 15/01/23

tema 1:  
De sistema digital a computador

FC-2

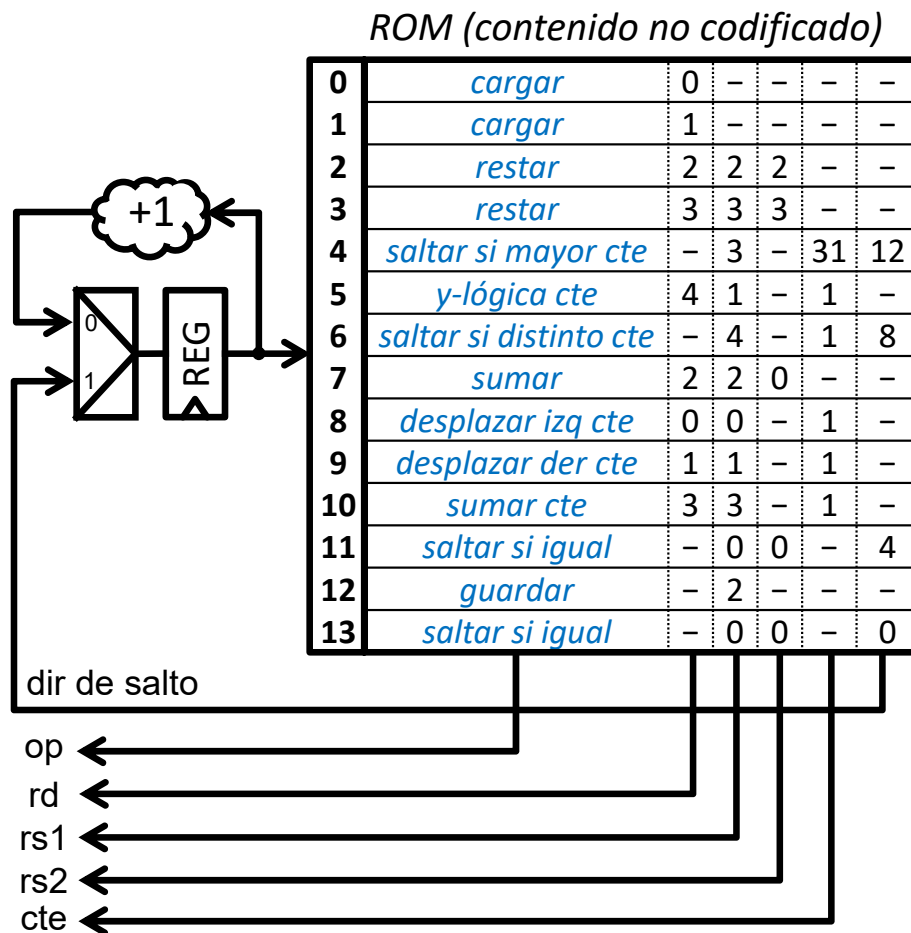
53





# Ruta de datos general + controlador

- En este controlador podemos encontrar muchos elementos que son comunes a un procesador de propósito general:





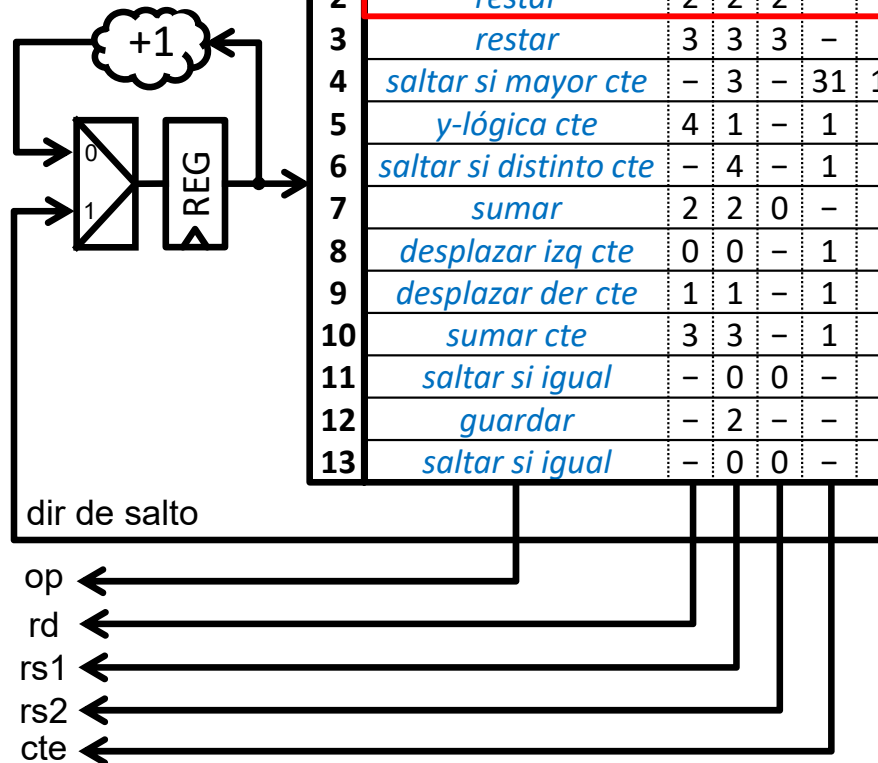
# Ruta de datos general + controlador

- En este controlador podemos encontrar muchos elementos que son comunes a un procesador de propósito general:

ROM (contenido no codificado)

0	cargar	0	-	-	-	-
1	cargar	1	-	-	-	-
2	restar	2	2	2	-	-
3	restar	3	3	3	-	-
4	saltar si mayor cte	-	3	-	31	12
5	y-lógica cte	4	1	-	1	-
6	saltar si distinto cte	-	4	-	1	8
7	sumar	2	2	0	-	-
8	desplazar izq cte	0	0	-	1	-
9	desplazar der cte	1	1	-	1	-
10	sumar cte	3	3	-	1	-
11	saltar si igual	-	0	0	-	4
12	guardar	-	2	-	-	-
13	saltar si igual	-	0	0	-	0

La ROM almacena *instrucciones*: *ordenes más elementales* que puede ejecutar la ruta de datos





# Ruta de datos general + controlador

- En este controlador podemos encontrar muchos elementos que son comunes a un procesador de propósito general:

ROM (contenido no codificado)

0	cargar	0	-	-	-	-
1	cargar	1	-	-	-	-
2	restar	2	2	2	-	-
3	restar	3	3	3	-	-
4	saltar si mayor cte	-	3	-	31	12
5	y-lógica cte	4	1	-	1	-
6	saltar si distinto cte	-	4	-	1	8
7	sumar	2	2	0	-	-
8	desplazar izq cte	0	0	-	1	-
9	desplazar der cte	1	1	-	1	-
10	sumar cte	3	3	-	1	-
11	saltar si igual	-	0	0	-	4
12	guardar	-	2	-	-	-
13	saltar si igual	-	0	0	-	0

← Cada instrucción define la *operación* a realizar, los *operandos fuente* a usar y el *destino* del resultado



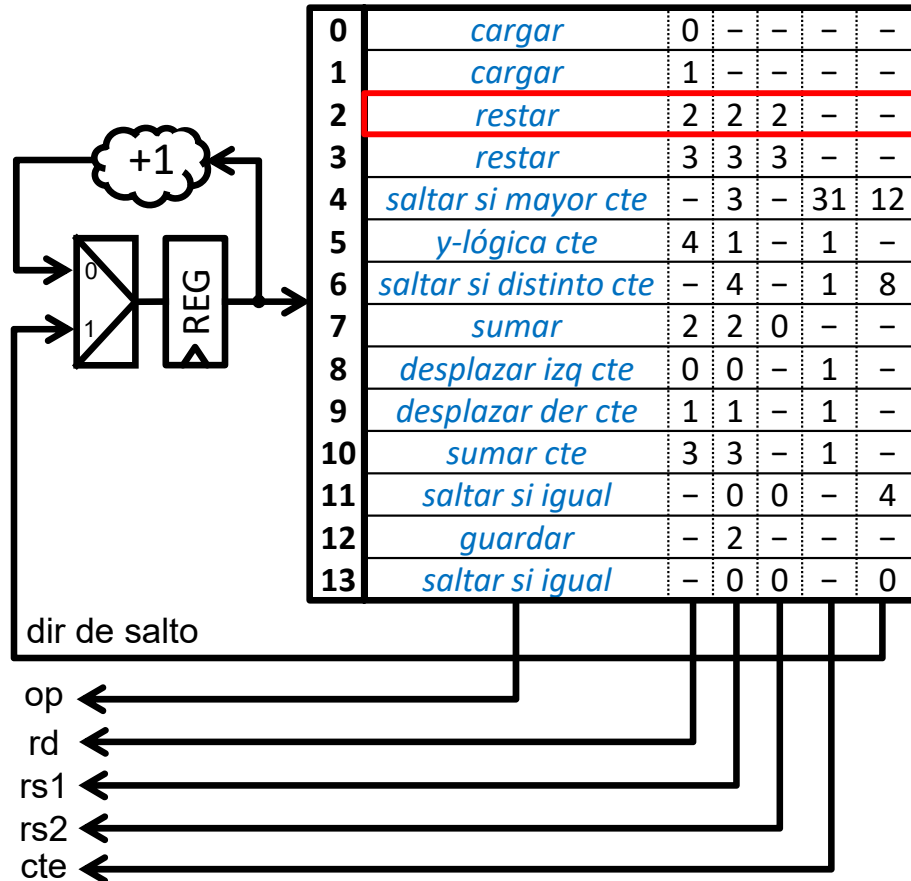




# Ruta de datos general + controlador

- En este controlador podemos encontrar muchos elementos que son comunes a un procesador de propósito general:

ROM (contenido no codificado)



Las instrucciones se almacenan en ROM codificadas (**código máquina**) pero conviene usar una representación simbólica (**ensamblador**)

2 `10010110010001000100000000`

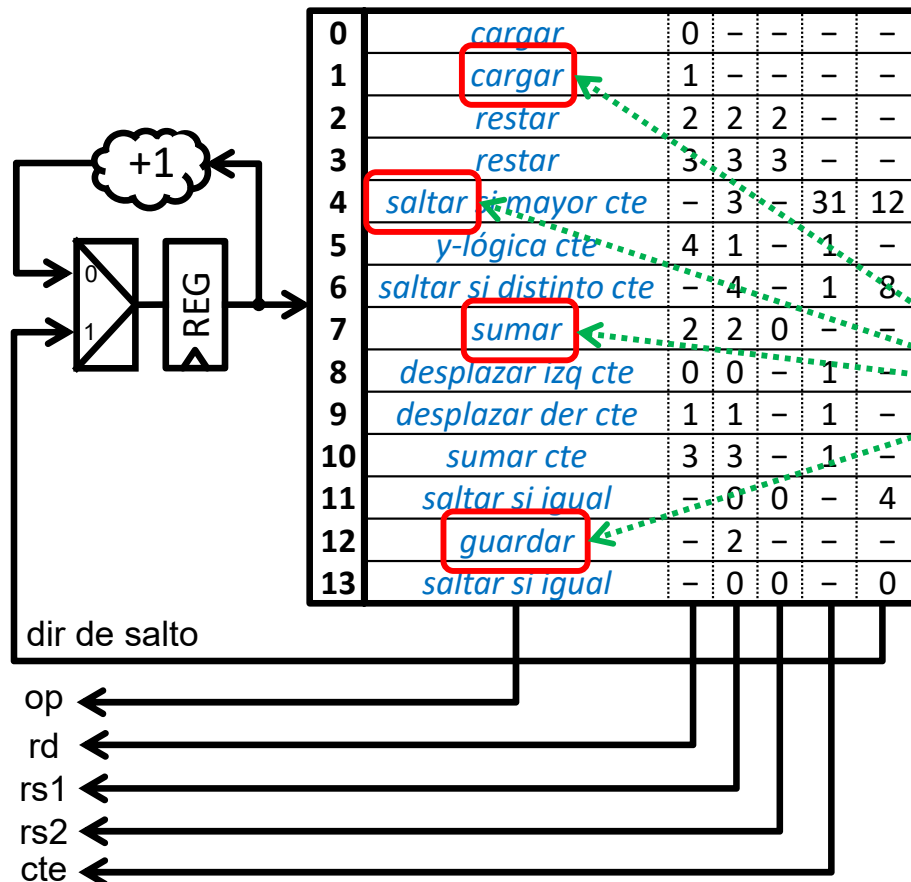
2 *restar* R2, R2, R2



# Ruta de datos general + controlador

- En este controlador podemos encontrar muchos elementos que son comunes a un procesador de propósito general:

ROM (contenido no codificado)

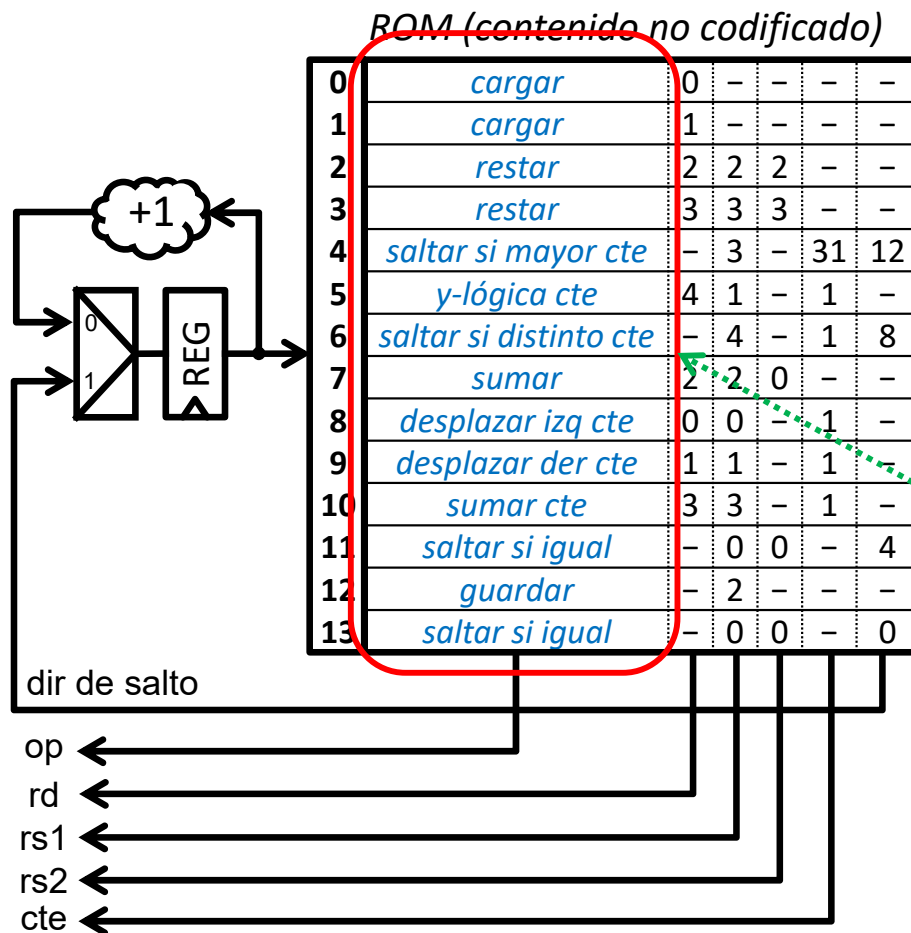


Las instrucciones son de distintos tipos: aritméticas, de salto, de transferencia de datos...



# Ruta de datos general + controlador

- En este controlador podemos encontrar muchos elementos que son comunes a un procesador de propósito general:



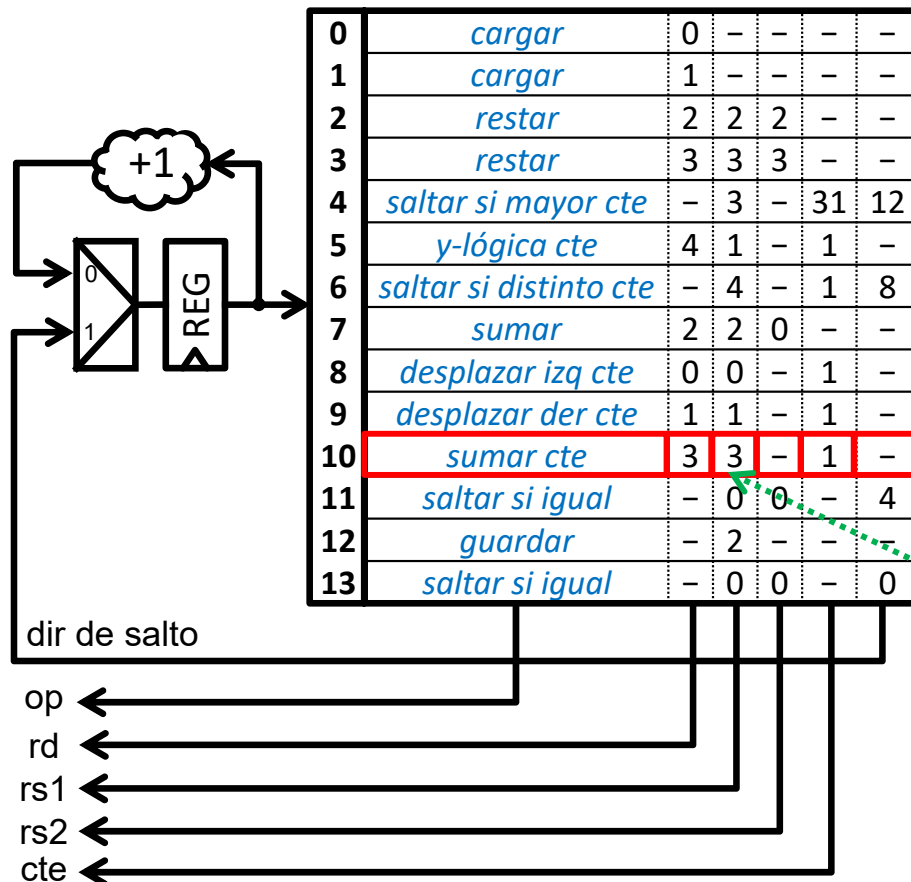
El conjunto de instrucciones diferentes que pueden ejecutarse forma el **repertorio de instrucciones**



# Ruta de datos general + controlador

- En este controlador podemos encontrar muchos elementos que son comunes a un procesador de propósito general:

ROM (contenido no codificado)



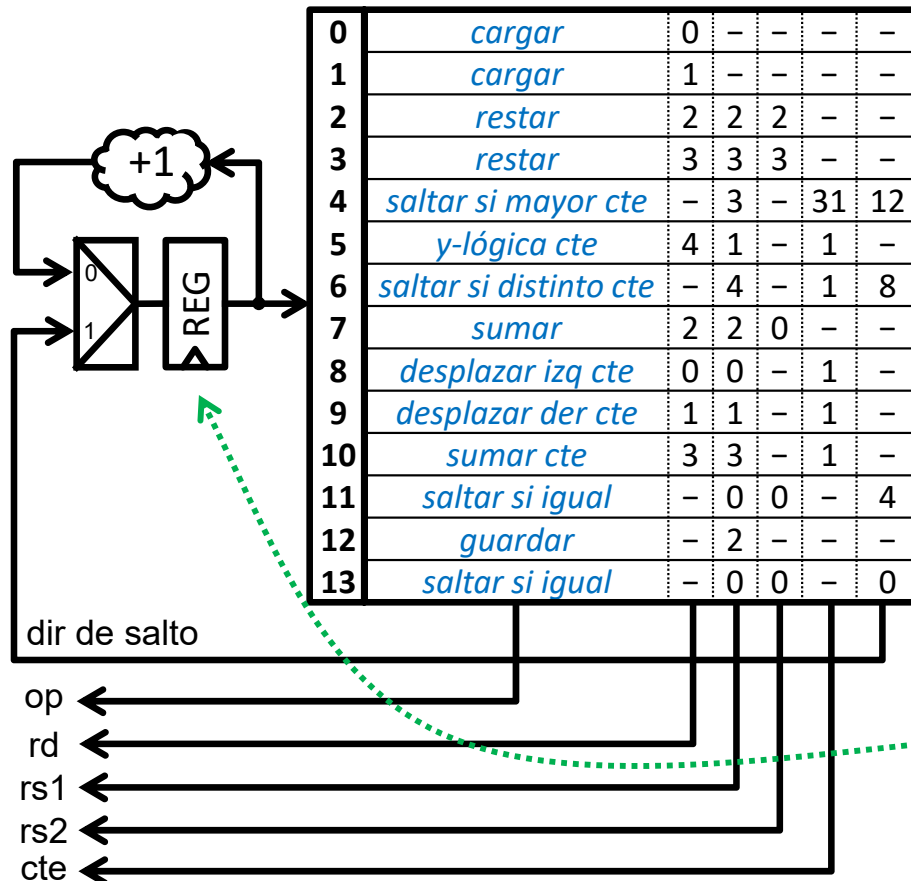
Las instrucciones tienen un **formato** común que ubica y codifica los campos de información



# Ruta de datos general + controlador

- En este controlador podemos encontrar muchos elementos que son comunes a un procesador de propósito general:

ROM (contenido no codificado)



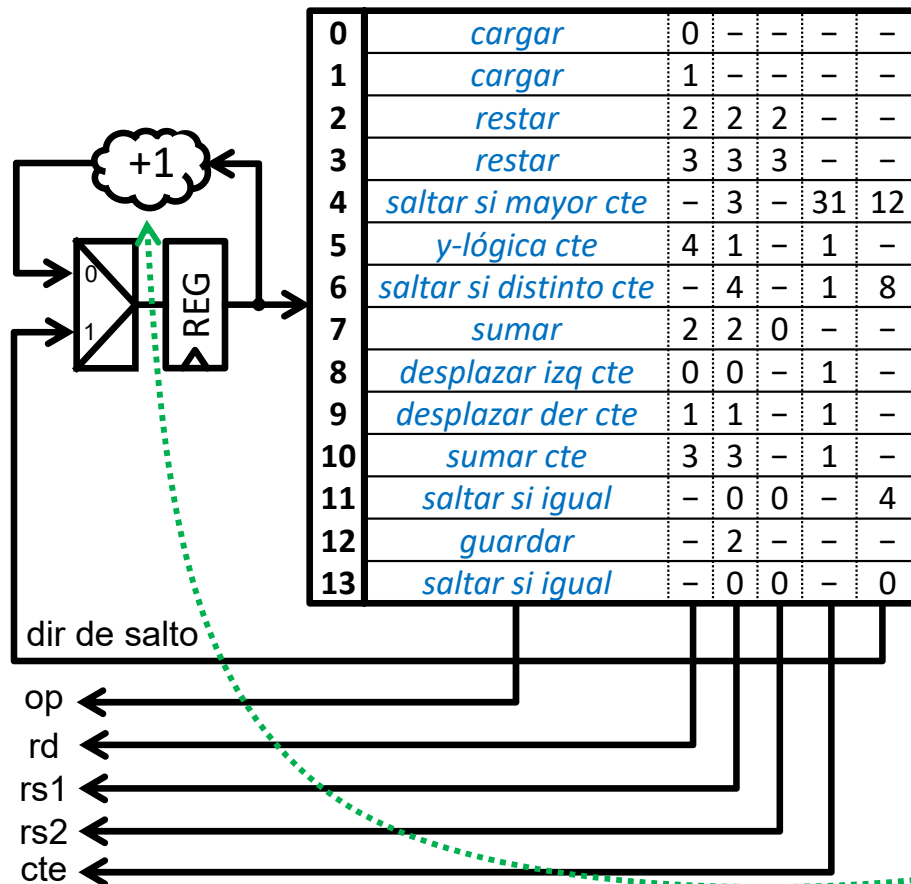
Existe un registro "contador de programa (PC)" que direcciona la memoria



# Ruta de datos general + controlador

- En este controlador podemos encontrar muchos elementos que son comunes a un procesador de propósito general:

ROM (contenido no codificado)

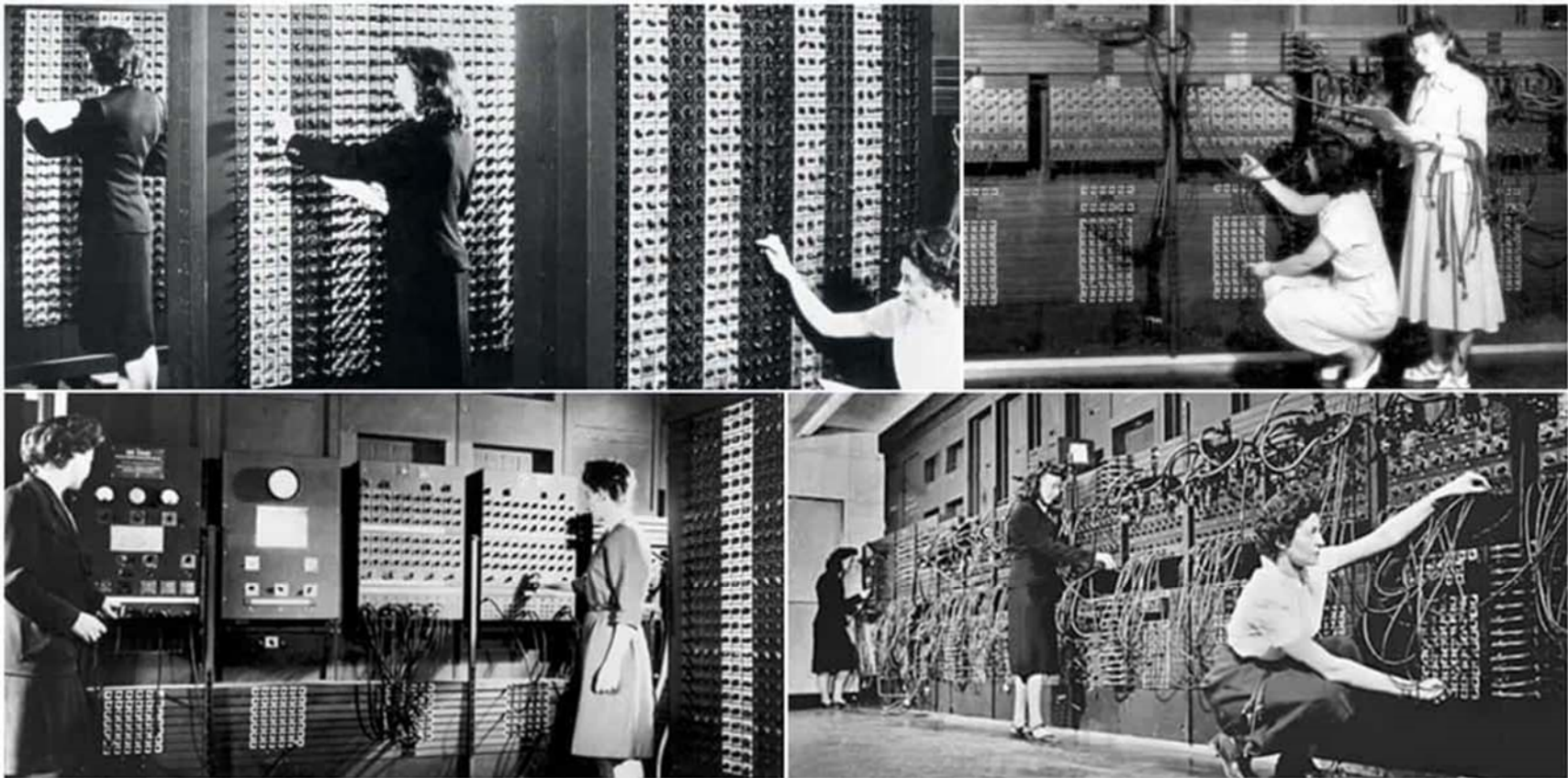


Normalmente, las instrucciones se *ejecutan una tras otra* en el orden en que están almacenadas



# Ruta de datos general + controlador

- Basta con **modificar el programa contenido** de la ROM para que la ruta de datos efectúe un algoritmo distinto.
  - El ENIAC (1946) se programaba recableando el computador.

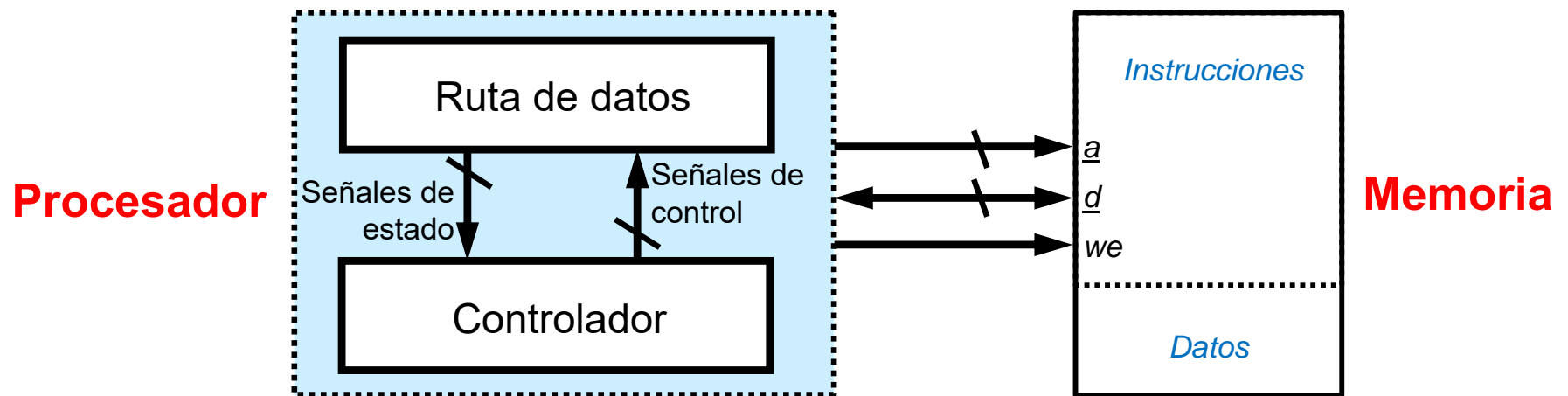


fuelle: <https://alltogether.swe.org>



# Circuito de propósito general: computador

- Pero es posible plantear un **circuito aún más general** que **evite su rediseño** cada vez que se quiera **cambiar el algoritmo** que realiza:
  - Reemplazando la ROM por una RAM para facilitar el cambio del programa.
  - Reemplazando la entrada/salida externa de la ruta de datos por una conexión a la misma RAM para la lectura/escritura de **datos almacenados en ella**.



- Aparecen de manera natural nuevos elementos:
  - **SW**: conjunto de programas que ejecuta el circuito.
  - **Programador**: persona (que no diseña HW) pero que desarrolla SW.
  - **Compilador**: traductor de algoritmos a secuencias de instrucciones.





# Modelo Von Neumann (1945)

## Principios

- Computador con **programa almacenado en memoria**.
  - Un **programa** es una **secuencia de instrucciones y datos**.
    - **Instrucciones**: que gobiernan el funcionamiento del computador.
    - **Datos**: que son procesados por las instrucciones.
  
- La memoria la forman palabras **organizadas linealmente**.
  - Todas del **mismo tamaño**.
  - Cada una **identificada por la dirección** que ocupa en la memoria.
  - Conteniendo indistinguiblemente instrucciones o datos codificados.
  
- Las instrucciones del programa se ejecutan **secuencialmente**:
  - Es decir, en el mismo **orden en que están almacenadas** en memoria.
  - Este orden implícito sólo puede cambiar tras ejecutar una instrucción de salto.
  - Existe un registro **contador de programa** (PC) que almacena la **dirección de memoria que ocupa la instrucción** a ejecutar.



# Modelo Von Neumann (1945)

## Principios

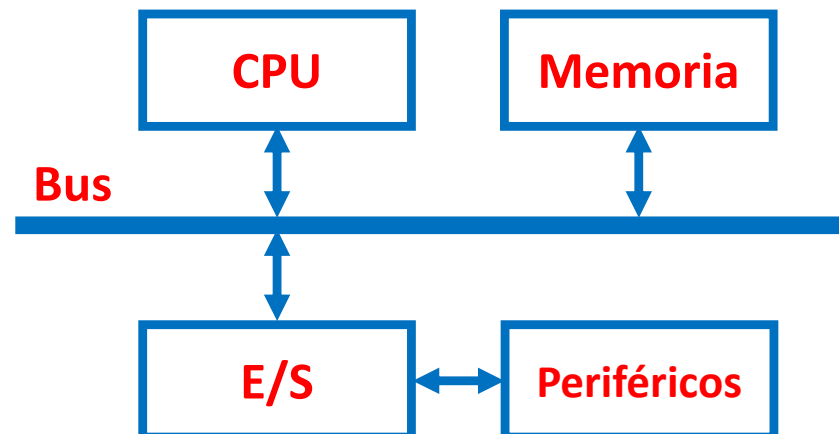
- La **ejecución de toda instrucción** supone:
  - Leer la **instrucción** de memoria (*fetch*) cuya dirección contiene el PC.
  - Descodificar la instrucción.
  - Leer los **operandos fuente** indicados en la instrucción.
  - **Efectuar** la operación indicada en la instrucción sobre los operandos leídos.
  - **Escribir** el resultado de la operación en el destino indicado en la instrucción.
  - **Actualizar el PC con la dirección** de la instrucción siguiente a ejecutar
    - Por defecto, es el PC + el tamaño de la instrucción que se está ejecutando.
    - Si la instrucción es de salto, la dirección será la indicada por la instrucción.
- Esta **sucesión de etapas** se conoce como **ciclo de instrucción**.
  - Indefinidamente, un procesador realiza un ciclo de instrucción tras otro.



# Modelo Von Neumann (1945)

## Estructura

- Sus principales componentes estructurales son:
  - **Procesador (CPU)**: controla el funcionamiento del computador y procesa los datos según las instrucciones de un programa almacenado
  - **Subsistema de memoria**: almacena datos/instrucciones (programa)
  - **Subsistema de entrada/salida**: transfiere datos entre el computador y el entorno externo
  - **Subsistema de interconexión**: proporciona un medio de comunicación entre el procesador, la memoria y la E/S.





# Conceptos básicos

## Arquitectura del procesador

- La **arquitectura del procesador** o **arquitectura del repertorio de instrucciones** es el conjunto de atributos del procesador visibles por:
  - El programador en lenguaje ensamblador.
  - El compilador de un lenguaje de alto nivel.
- Supone un **contrato entre el SW y el HW** que engloba los siguientes elementos:
  - **Tipos elementales de datos** soportados por las instrucciones.
  - Modelo de memoria y **organización de información en memoria**.
  - **Registros del procesador** accesibles por el programador.
  - **Mecanismos para indicar la localización de los operandos** de una instrucción.
  - **Conjunto de instrucciones** que puede ejecutar el procesador.
  - **Formato** y codificación de la instrucción máquina.
- La **arquitectura del procesador** **abstrae la complejidad** de su diseño HW indicando **qué hace** el procesador pero **sin concretar cómo** lo hace.



# Conceptos básicos

## Estructura del procesador

- La **estructura de un procesador** o **microarquitectura** es la **organización concreta de bloques HW** con los que está diseñada una arquitectura.
  - Una misma arquitectura puede implementarse con microarquitecturas diferentes diseñadas, incluso, por distintos fabricantes.
- Se denomina **familia** al conjunto de procesadores con la **misma arquitectura** pero **distinta implementación**:
  - Distinta tecnología, distintas prestaciones, distinto precio...
- Existen una gran número de **familias arquitectónicas** diferentes:
  - x86, ARM, MIPS, SPARC, PowerPC, RISC-V...
  - Todos los **procesadores de una misma familia** son compatibles entre sí y **pueden ejecutar exactamente los mismos programas**.
  - La **compatibilidad hacia atrás** permite a los miembros más modernos de una familia poder ejecutar programas desarrollados para los antiguos.

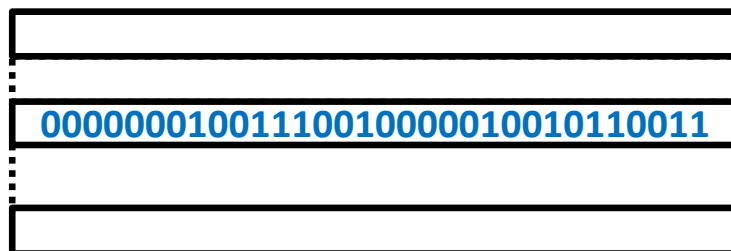


# Conceptos básicos

## Código máquina vs. código ensamblador

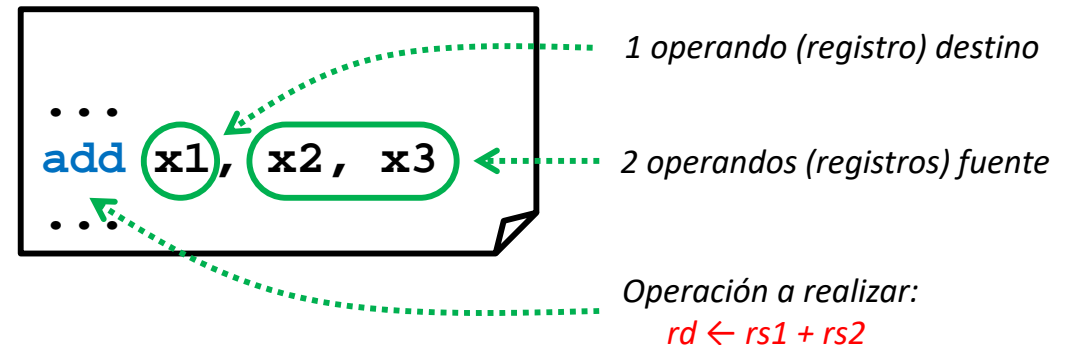
- La **orden más elemental** que un procesador puede ejecutar se denomina **instrucción**.
  - Define la **operación** a realizar y los **operandos** con los que hacerla.
  - El hardware de un computador solo interpreta y ejecuta **instrucciones codificadas en binario**, es lo que se denomina **lenguaje máquina**.
  - El **lenguaje ensamblador** (*assembly language*) es una **representación legible** por humanos del **lenguaje máquina**.
    - Las instrucciones en ensamblador usan **nemotécnicos** indicar la operación y los operandos.
    - Normalmente existe una **relación 1:1** entre instrucciones máquina y ensamblador.

### Código máquina RISC-V



Memoria

### Ensamblador RISC-V





# Conceptos básicos

## Compilación vs. ensamblado

- **Ensamblador** (*assembler*): software que traduce instrucciones en ensamblador a instrucciones en código máquina.
- **Compilador** (*compiler*): software que traduce un programa escrito en lenguaje de alto nivel (i.e. C/C++) en un programa en ensamblador.
  - En general, la relación entre sentencias de alto nivel y ensamblador es 1:n.

### Lenguaje C/C++

```
int a, b, c, d;  
...  
d = (a + b) - c;  
...
```

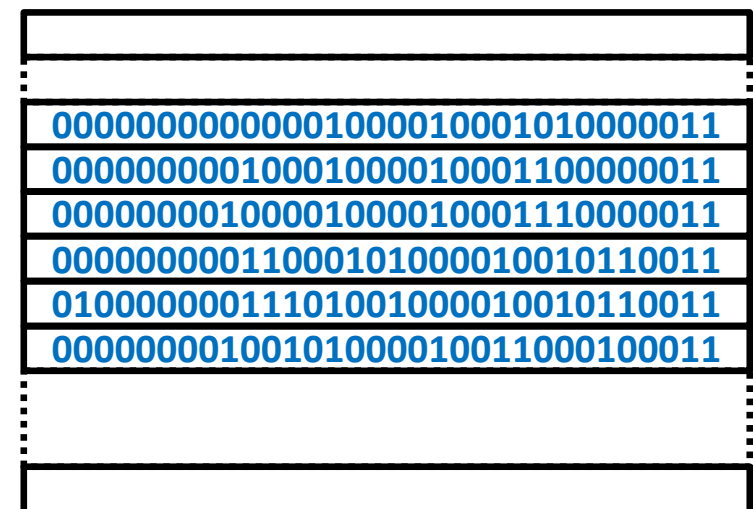
$a \rightarrow x5$      $c \rightarrow x7$   
 $b \rightarrow x6$      $d \rightarrow x9$

Vinculación de variables C  
con registros del RISC-V

### Ensamblador RISC-V

```
...  
lw x5, 0(x8)  
lw x6, 4(x8)  
lw x7, 8(x8)  
add x9, x5, x6  
sub x9, x9, x7  
sw x9, 12(x8)  
...
```

### Código máquina RISC-V



Memoria

# Acerca de *Creative Commons*



## ■ Licencia CC (*Creative Commons*)

- Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:



**Reconocimiento** (*Attribution*):

En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.



**No comercial** (*Non commercial*):

La explotación de la obra queda limitada a usos no comerciales.



**Compartir igual** (*Share alike*):

La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: <https://creativecommons.org/licenses/by-nc-sa/4.0/>