

Tema 4: Diseño de un microprocesador

Febrero de 2011

- 1 Introducción
- 2 Visión general de la implementación
- 3 El camino de datos
- 4 Control del camino de datos

Índice

- 1 **Introducción**
- 2 Visión general de la implementación
- 3 El camino de datos
- 4 Control del camino de datos

Objetivos

El objetivo de este tema es ver cómo se construye un procesador capaz de ejecutar el ISA que se ve en el tema 3 a partir de las puertas lógicas que vimos en los primeros dos temas.

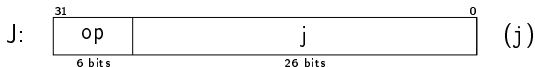
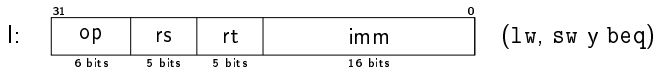
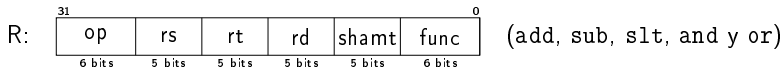
- Por simplicidad implementaremos sólo un subconjunto de las instrucciones:
 - Aritmético-lógicas: `add`, `sub`, `and`, `or` y `slt`.
 - Carga y almacenamiento: `lw` y `sw`.
 - Salto condicional: `beq`.
 - Salto incondicional: `j`.
- Al finalizar el tema, debéis ser capaces de modificar el procesador que veremos aquí para añadirle cualquier tipo de instrucción (incluso instrucciones no presentes en el ISA real de MIPS).

Codificación de las instrucciones (1/4)

- Es necesario saber cómo se codifican las instrucciones para entender cómo funciona el procesador que las ejecuta.
 - Uno de los primeros pasos que todo procesador tendrá que realizar antes de ejecutar una instrucción es **decodificarla**.

Codificación de las instrucciones (1/4)

- Es necesario saber cómo se codifican las instrucciones para entender cómo funciona el procesador que las ejecuta.
 - Uno de los primeros pasos que todo procesador tendrá que realizar antes de ejecutar una instrucción es **decodificarla**.
- En MIPS, las instrucciones enteras se codifican utilizando uno de los siguientes 3 formatos:



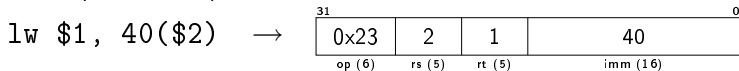
Codificación de las instrucciones (2/4)

- Aritmético-lógicas (formato R):

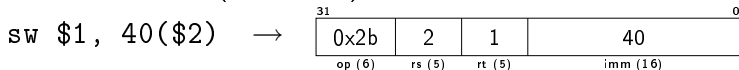
add \$1, \$2, \$3	→	<table border="1" style="border-collapse: collapse; width: 100%; text-align: center;"> <tr> <td style="width: 16.6%;">³¹</td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;">⁰</td> </tr> <tr> <td style="height: 30px; vertical-align: middle;">0</td> <td style="height: 30px; vertical-align: middle;">2</td> <td style="height: 30px; vertical-align: middle;">3</td> <td style="height: 30px; vertical-align: middle;">1</td> <td style="height: 30px; vertical-align: middle;">0</td> <td style="height: 30px; vertical-align: middle;">0x20</td> </tr> <tr> <td style="font-size: small;">op (6)</td> <td style="font-size: small;">rs (5)</td> <td style="font-size: small;">rt (5)</td> <td style="font-size: small;">rd (5)</td> <td style="font-size: small;">shamt (5)</td> <td style="font-size: small;">func (6)</td> </tr> </table>	³¹					⁰	0	2	3	1	0	0x20	op (6)	rs (5)	rt (5)	rd (5)	shamt (5)	func (6)
³¹					⁰															
0	2	3	1	0	0x20															
op (6)	rs (5)	rt (5)	rd (5)	shamt (5)	func (6)															
sub \$1, \$2, \$3	→	<table border="1" style="border-collapse: collapse; width: 100%; text-align: center;"> <tr> <td style="width: 16.6%;">³¹</td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;">⁰</td> </tr> <tr> <td style="height: 30px; vertical-align: middle;">0</td> <td style="height: 30px; vertical-align: middle;">2</td> <td style="height: 30px; vertical-align: middle;">3</td> <td style="height: 30px; vertical-align: middle;">1</td> <td style="height: 30px; vertical-align: middle;">0</td> <td style="height: 30px; vertical-align: middle;">0x22</td> </tr> <tr> <td style="font-size: small;">op (6)</td> <td style="font-size: small;">rs (5)</td> <td style="font-size: small;">rt (5)</td> <td style="font-size: small;">rd (5)</td> <td style="font-size: small;">shamt (5)</td> <td style="font-size: small;">func (6)</td> </tr> </table>	³¹					⁰	0	2	3	1	0	0x22	op (6)	rs (5)	rt (5)	rd (5)	shamt (5)	func (6)
³¹					⁰															
0	2	3	1	0	0x22															
op (6)	rs (5)	rt (5)	rd (5)	shamt (5)	func (6)															
and \$1, \$2, \$3	→	<table border="1" style="border-collapse: collapse; width: 100%; text-align: center;"> <tr> <td style="width: 16.6%;">³¹</td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;">⁰</td> </tr> <tr> <td style="height: 30px; vertical-align: middle;">0</td> <td style="height: 30px; vertical-align: middle;">2</td> <td style="height: 30px; vertical-align: middle;">3</td> <td style="height: 30px; vertical-align: middle;">1</td> <td style="height: 30px; vertical-align: middle;">0</td> <td style="height: 30px; vertical-align: middle;">0x24</td> </tr> <tr> <td style="font-size: small;">op (6)</td> <td style="font-size: small;">rs (5)</td> <td style="font-size: small;">rt (5)</td> <td style="font-size: small;">rd (5)</td> <td style="font-size: small;">shamt (5)</td> <td style="font-size: small;">func (6)</td> </tr> </table>	³¹					⁰	0	2	3	1	0	0x24	op (6)	rs (5)	rt (5)	rd (5)	shamt (5)	func (6)
³¹					⁰															
0	2	3	1	0	0x24															
op (6)	rs (5)	rt (5)	rd (5)	shamt (5)	func (6)															
or \$1, \$2, \$3	→	<table border="1" style="border-collapse: collapse; width: 100%; text-align: center;"> <tr> <td style="width: 16.6%;">³¹</td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;">⁰</td> </tr> <tr> <td style="height: 30px; vertical-align: middle;">0</td> <td style="height: 30px; vertical-align: middle;">2</td> <td style="height: 30px; vertical-align: middle;">3</td> <td style="height: 30px; vertical-align: middle;">1</td> <td style="height: 30px; vertical-align: middle;">0</td> <td style="height: 30px; vertical-align: middle;">0x25</td> </tr> <tr> <td style="font-size: small;">op (6)</td> <td style="font-size: small;">rs (5)</td> <td style="font-size: small;">rt (5)</td> <td style="font-size: small;">rd (5)</td> <td style="font-size: small;">shamt (5)</td> <td style="font-size: small;">func (6)</td> </tr> </table>	³¹					⁰	0	2	3	1	0	0x25	op (6)	rs (5)	rt (5)	rd (5)	shamt (5)	func (6)
³¹					⁰															
0	2	3	1	0	0x25															
op (6)	rs (5)	rt (5)	rd (5)	shamt (5)	func (6)															
slt \$1, \$2, \$3	→	<table border="1" style="border-collapse: collapse; width: 100%; text-align: center;"> <tr> <td style="width: 16.6%;">³¹</td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;"></td> <td style="width: 16.6%;">⁰</td> </tr> <tr> <td style="height: 30px; vertical-align: middle;">0</td> <td style="height: 30px; vertical-align: middle;">2</td> <td style="height: 30px; vertical-align: middle;">3</td> <td style="height: 30px; vertical-align: middle;">1</td> <td style="height: 30px; vertical-align: middle;">0</td> <td style="height: 30px; vertical-align: middle;">0x2a</td> </tr> <tr> <td style="font-size: small;">op (6)</td> <td style="font-size: small;">rs (5)</td> <td style="font-size: small;">rt (5)</td> <td style="font-size: small;">rd (5)</td> <td style="font-size: small;">shamt (5)</td> <td style="font-size: small;">func (6)</td> </tr> </table>	³¹					⁰	0	2	3	1	0	0x2a	op (6)	rs (5)	rt (5)	rd (5)	shamt (5)	func (6)
³¹					⁰															
0	2	3	1	0	0x2a															
op (6)	rs (5)	rt (5)	rd (5)	shamt (5)	func (6)															

Codificación de las instrucciones (3/4)

- Carga (formato I):

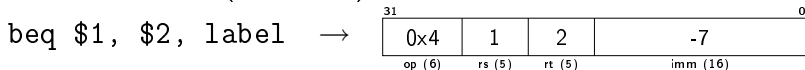


- Almacenamiento (formato I):

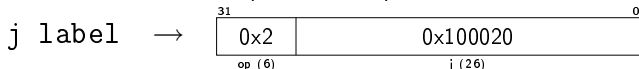


Codificación de las instrucciones (4/4)

- Salto condicional (formato I):



- Salto incondicional (formato J):



En ambos casos suponemos que las instrucciones de salto se encuentran en la dirección 0x00400098 y que la instrucción de destino se encuentra en la dirección 0x00400080.

Modelo del tiempo de ejecución

- Usaremos la siguiente expresión para modelar el tiempo de ejecución de un programa concreto, usando un ISA concreto y una implementación concreta de dicho ISA:

$$T_{ejec} = N_{inst} \times CPI \times T_{ciclo}$$

Modelo del tiempo de ejecución

- Usaremos la siguiente expresión para modelar el tiempo de ejecución de un programa concreto, usando un ISA concreto y una implementación concreta de dicho ISA:

$$T_{ejec} = N_{inst} \times CPI \times T_{ciclo}$$

- Nos permite comparar entre sí la velocidad de distintos programas, ISAs, o implementaciones del ISA.

Modelo del tiempo de ejecución

- Usaremos la siguiente expresión para modelar el tiempo de ejecución de un programa concreto, usando un ISA concreto y una implementación concreta de dicho ISA:

$$T_{ejec} = N_{inst} \times CPI \times T_{ciclo}$$

- Nos permite comparar entre sí la velocidad de distintos programas, ISAs, o implementaciones del ISA.
- El **número de instrucciones** depende del ISA y del compilador (y del programa, por supuesto).

Modelo del tiempo de ejecución

- Usaremos la siguiente expresión para modelar el tiempo de ejecución de un programa concreto, usando un ISA concreto y una implementación concreta de dicho ISA:

$$T_{ejec} = N_{inst} \times CPI \times T_{ciclo}$$

- Nos permite comparar entre sí la velocidad de distintos programas, ISAs, o implementaciones del ISA.
- Los **ciclos por instrucción** y el **tiempo de ciclo** dependen de la implementación del procesador.

Distintas opciones de implementación

El tiempo de ciclo y los ciclos por instrucción están relacionados entre sí y dependen de la implementación.

Distintas opciones de implementación

El tiempo de ciclo y los ciclos por instrucción están relacionados entre sí y dependen de la implementación.

- **Implementación monociclo:** cada instrucción tarda un ciclo
 $\Rightarrow CPI = 1$, pero $\uparrow T_{ciclo}$.

Distintas opciones de implementación

El tiempo de ciclo y los ciclos por instrucción están relacionados entre sí y dependen de la implementación.

- **Implementación monociclo:** cada instrucción tarda un ciclo $\Rightarrow CPI = 1$, pero $\uparrow T_{ciclo}$.
- **Implementación multiciclo:** cada instrucción se divide en varios pasos, cada paso tarda un ciclo $\Rightarrow CPI > 1$ y $\downarrow T_{ciclo}$.

Distintas opciones de implementación

El tiempo de ciclo y los ciclos por instrucción están relacionados entre sí y dependen de la implementación.

- **Implementación monociclo:** cada instrucción tarda un ciclo $\Rightarrow CPI = 1$, pero $\uparrow T_{ciclo}$.
- **Implementación multiciclo:** cada instrucción se divide en varios pasos, cada paso tarda un ciclo $\Rightarrow CPI > 1$ y $\downarrow T_{ciclo}$.
- **Implementación segmentada:** cada instrucción se divide en varios pasos, cada paso tarda un ciclo, se inicia la ejecución de una instrucción cada ciclo $\Rightarrow CPI_{segmentado} < CPI_{multiciclo}$ ($CPI_{segmentado} \approx 1$) y $\downarrow T_{ciclo}$.

Distintas opciones de implementación

El tiempo de ciclo y los ciclos por instrucción están relacionados entre sí y dependen de la implementación.

- **Implementación monociclo:** cada instrucción tarda un ciclo $\Rightarrow CPI = 1$, pero $\uparrow T_{ciclo}$.
- **Implementación multiciclo:** cada instrucción se divide en varios pasos, cada paso tarda un ciclo $\Rightarrow CPI > 1$ y $\downarrow T_{ciclo}$.
- **Implementación segmentada:** cada instrucción se divide en varios pasos, cada paso tarda un ciclo, se inicia la ejecución de una instrucción cada ciclo $\Rightarrow CPI_{segmentado} < CPI_{multiciclo}$ ($CPI_{segmentado} \approx 1$) y $\downarrow T_{ciclo}$.
- **Implementación superescalar:** cada instrucción se divide en varios pasos, cada paso tarda un ciclo, se inicia la ejecución de varias instrucciones cada ciclo $\Rightarrow CPI < 1$ y $\downarrow T_{ciclo}$.

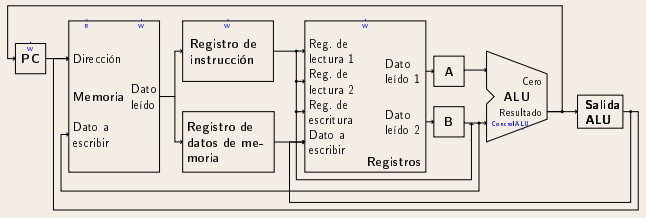
Índice

- 1 Introducción
- 2 Visión general de la implementación**
- 3 El camino de datos
- 4 Control del camino de datos

Visión general de la implementación

- Diseñaremos un procesador **multiciclo**: cada instrucción se descompone en varios pasos y cada paso se ejecuta en un ciclo de reloj.

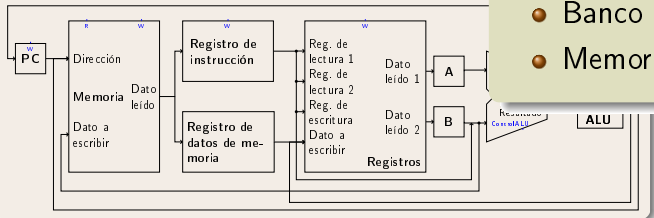
Esquema de la implementación



Visión general de la implementación

- Diseñaremos un procesador **multiciclo** que se descompone en varios pasos y cada uno tiene su propio reloj de reloj.

Esquema de la implementación



Usaremos componentes que vimos en temas anteriores:

- Puertas lógicas.
- Registros.
- ALU.
- Banco de registros
- Memoria.

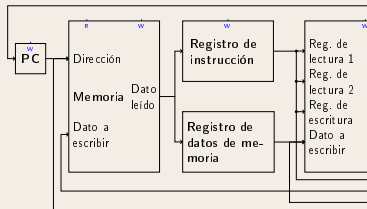
Visión general de la implementación

- Diseñaremos un procesador que se descomponga en varios bloques que se sincronizan con una señal de reloj.

Utilizaremos la metodología de **sincronización por pulsos de reloj**.

- Todos los componentes se conectan a la misma señal de reloj.
- Las entradas a los bloques lógicos combinatoriales serán valores que se calcularon en el ciclo anterior.
- El valor de salida de un bloque combinatorial se escribirá al final del ciclo actual (en el flanco activo) en algún elemento de estado.

Esquema de la implementación



Visión general de la implementación

Importante:

Se deben guardar en elementos de estado todos los datos calculados en un ciclo que se necesiten en ciclos posteriores.

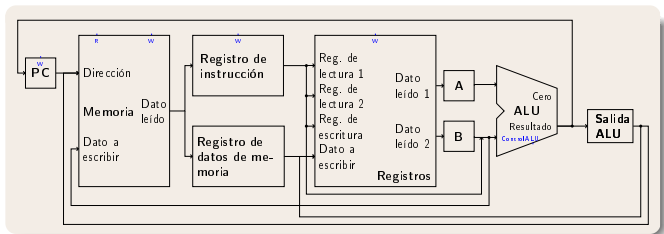
- Banco de registros.
- Registro PC.
- Memoria.
- Registros auxiliares.

Utilizaremos la metodología de **sincronización por pulsos de reloj**.

- Todos los componentes se conectan a la misma señal de reloj.
- Las entradas a los bloques lógicos combinatoriales serán valores que se calcularon en el ciclo anterior.
- El valor de salida de un bloque combinatorial se escribirá al final del ciclo actual (en el flanco activo) en algún elemento de estado.

Qué da tiempo a hacer en un ciclo

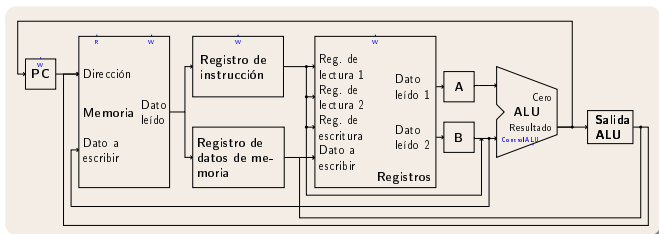
Supondremos que la **ALU**, la **memoria** y el **banco de registros** son las unidades funcionales principales:



Qué da tiempo a hacer en un ciclo

Supondremos que la **ALU**, la **memoria** y el **banco de registros** son las unidades funcionales principales:

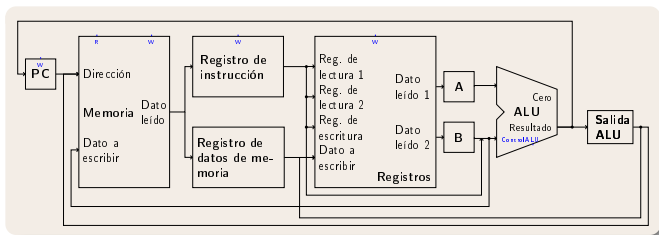
- La más lenta de estas unidades determinará el tiempo de ciclo.



Qué da tiempo a hacer en un ciclo

Supondremos que la **ALU**, la **memoria** y el **banco de registros** son las unidades funcionales principales:

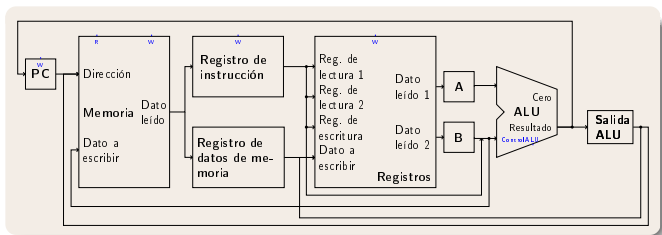
- La más lenta de estas unidades determinará el tiempo de ciclo.
- ⇒ La entrada de una unidad funcional principal no puede depender de valores calculados por otra unidad funcional **en el mismo ciclo** (no se pueden conectar en serie).



Qué da tiempo a hacer en un ciclo

Supondremos que la **ALU**, la **memoria** y el **banco de registros** son las unidades funcionales principales:

- La más lenta de estas unidades determinará el tiempo de ciclo.
- ⇒ La entrada de una unidad funcional principal no puede depender de valores calculados por otra unidad funcional **en el mismo ciclo** (no se pueden conectar en serie).
- ⇒ Se usarán registros auxiliares para comunicarlas.

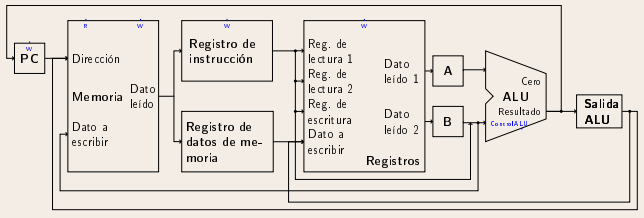


Índice

- 1 Introducción
- 2 Visión general de la implementación
- 3 El camino de datos**
- 4 Control del camino de datos

Instrucciones paso a paso

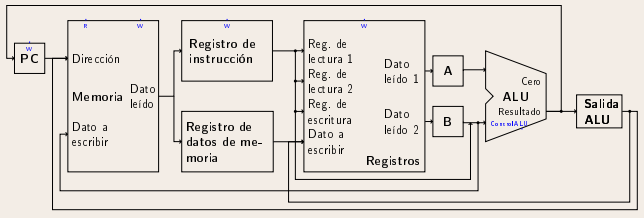
Esquema de la implementación



- Veremos cómo dividimos cada una de las instrucciones que nuestro procesador será capaz de ejecutar en pequeños pasos.

Instrucciones paso a paso

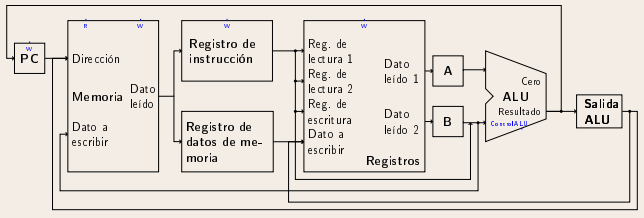
Esquema de la implementación



- Veremos cómo dividimos cada una de las instrucciones que nuestro procesador será capaz de ejecutar en pequeños pasos.
- Nos interesa balancear lo mejor posible el trabajo entre los pasos para aprovechar al máximo el tiempo de cada ciclo.

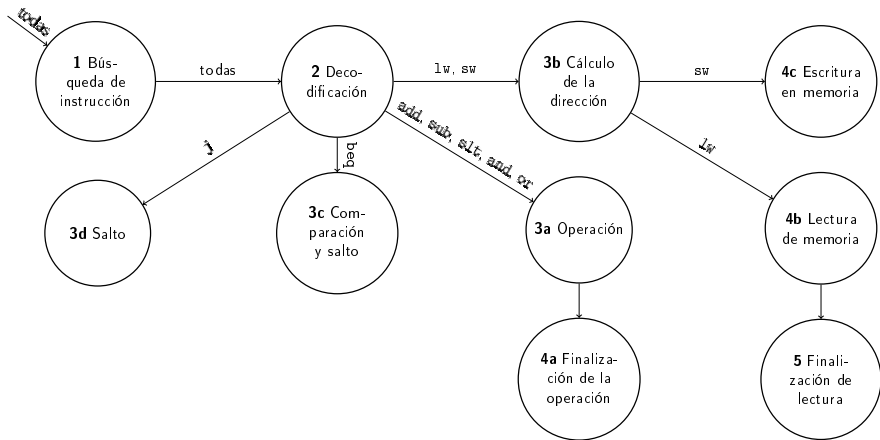
Instrucciones paso a paso

Esquema de la implementación

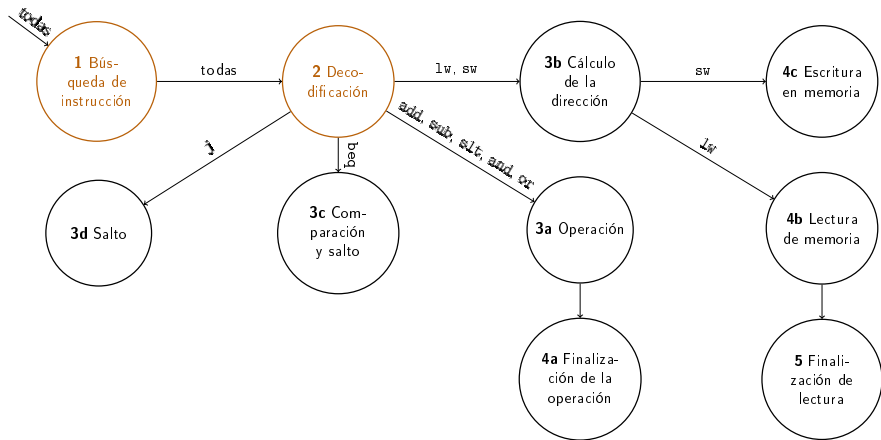


- Veremos cómo dividimos cada una de las instrucciones que nuestro procesador será capaz de ejecutar en pequeños pasos.
- Nos interesa balancear lo mejor posible el trabajo entre los pasos para aprovechar al máximo el tiempo de cada ciclo.
- En cada paso necesitaremos que se activen distintos componentes, y que se conecten entre sí de formas diferentes.

Descomposición de la ejecución de las instrucciones

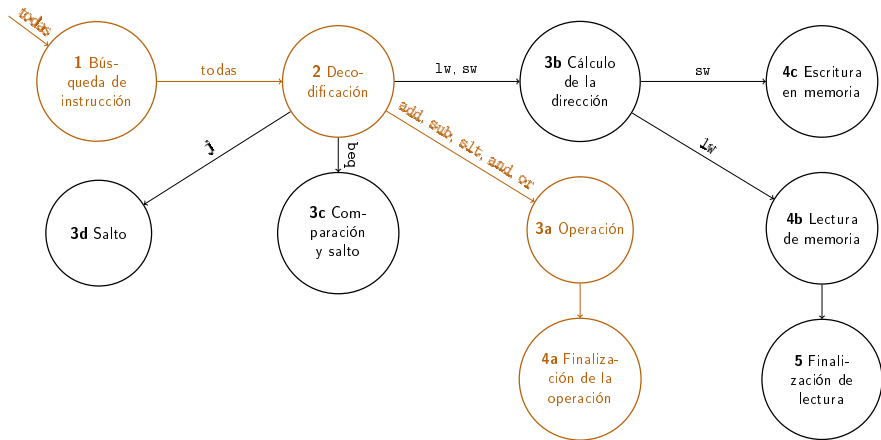


Descomposición de la ejecución de las instrucciones



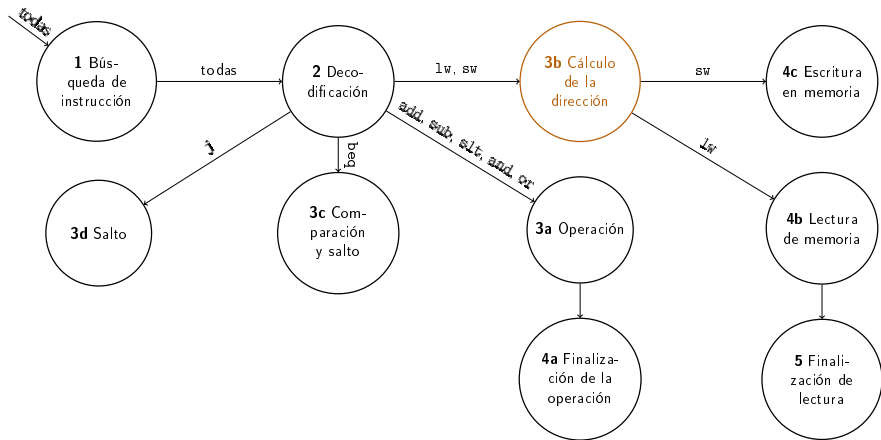
→ Los dos primeros pasos son comunes a todas las instrucciones.

Descomposición de la ejecución de las instrucciones



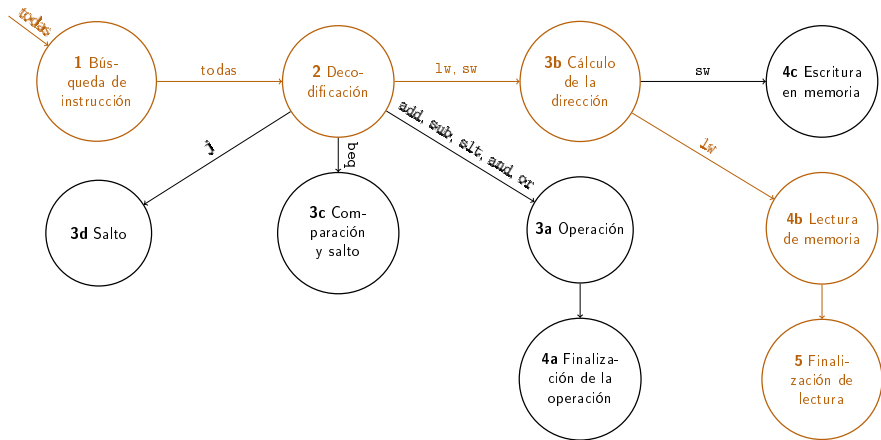
→ Las instrucciones aritmético-lógicas tendrán 4 pasos ⇒ tardarán 4 ciclos.

Descomposición de la ejecución de las instrucciones



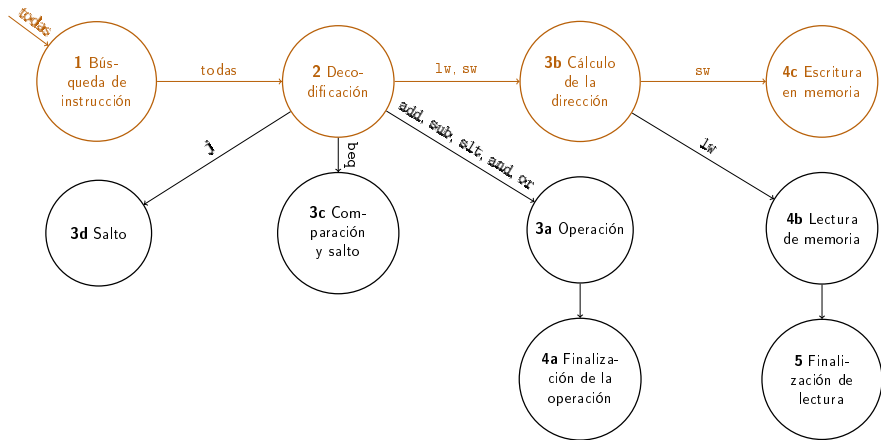
→ Las instrucciones de acceso a memoria tienen un paso común de cálculo de la dirección.

Descomposición de la ejecución de las instrucciones



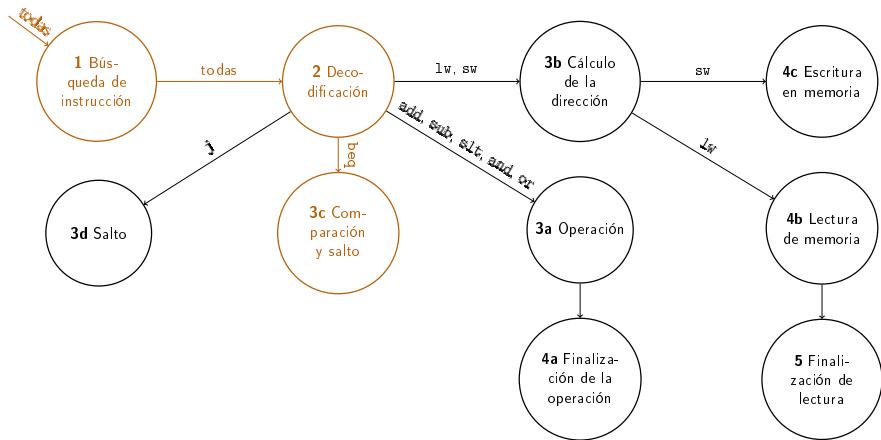
→ La instrucción `lw` es la más larga, con 5 pasos.

Descomposición de la ejecución de las instrucciones



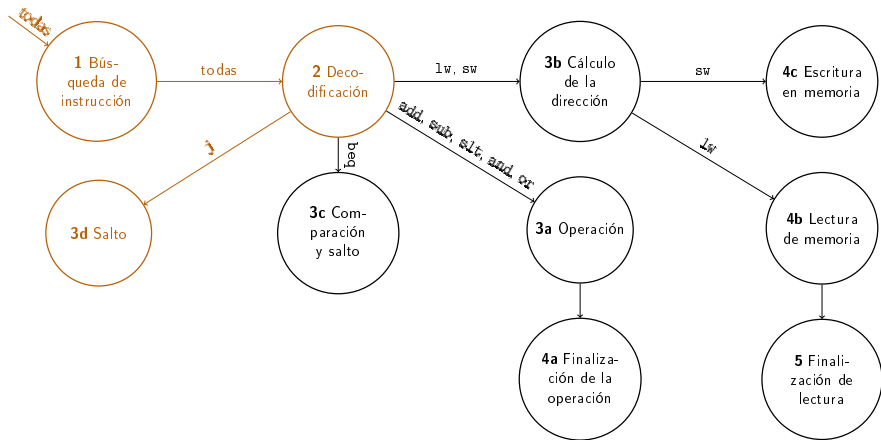
→ La instrucción *sw* necesita 4 pasos.

Descomposición de la ejecución de las instrucciones



→ Los saltos condicionales necesitan sólo 3 pasos.

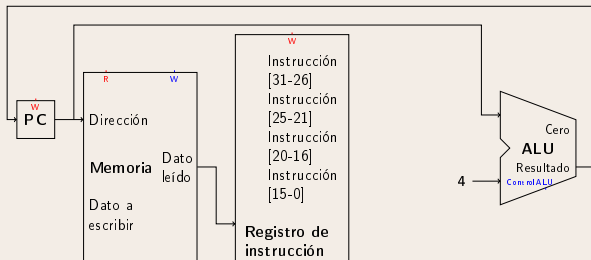
Descomposición de la ejecución de las instrucciones



→ Y los saltos incondicionales, también 3 sólo.

Paso 1: búsqueda de la instrucción

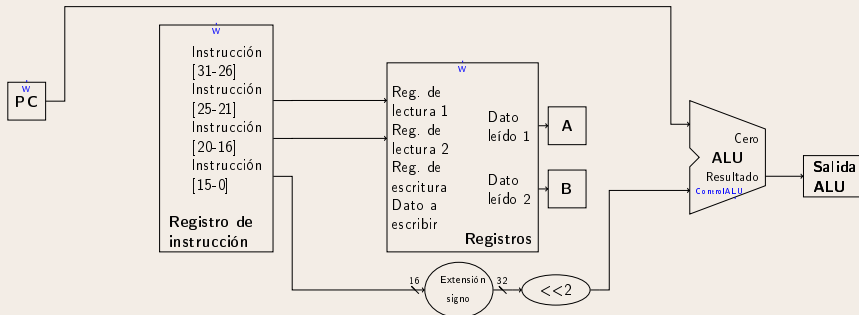
$$IR = \text{Memoria}[\text{PC}]$$

$$\text{PC} = \text{PC} + 4$$


Paso 2: decodificación y lectura de operandos

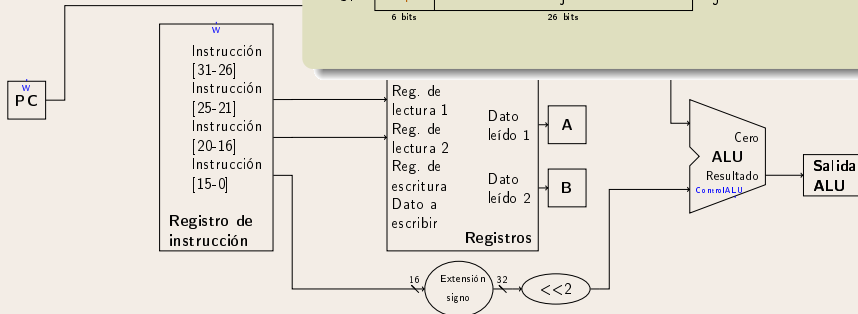
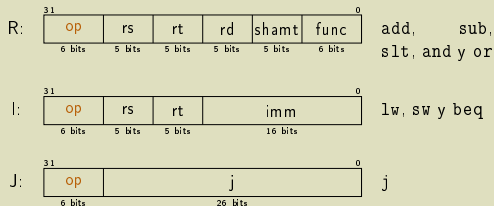
$$A = \text{Reg}[\text{IR}[25-21]]$$

$$B = \text{Reg}[\text{IR}[20-16]]$$

$$\text{ALUOut} = \text{PC} + (\text{extender_signo}(\text{IR}[15-0]) \ll 2)$$


Paso 2: decodificación y lectura de operandos

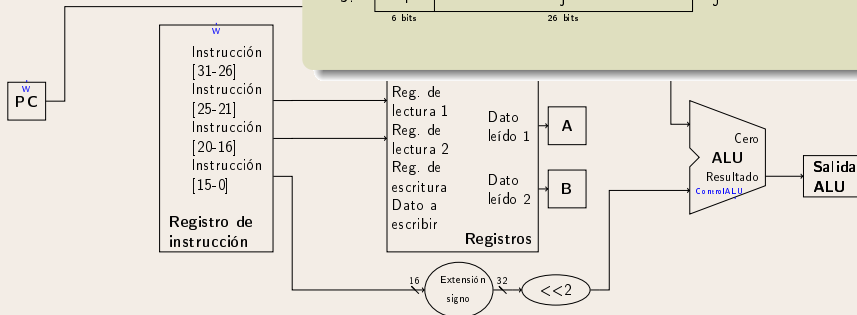
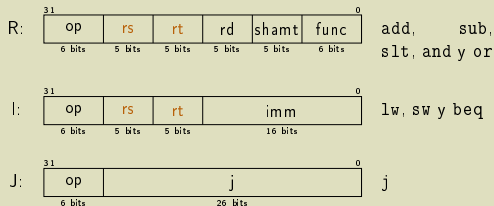
$A = \text{Reg}[\text{IR}[25-21]]$
 $B = \text{Reg}[\text{IR}[20-16]]$
 $\text{ALUOut} = \text{PC} + (\text{exten})$



- Se envía el código de operación a la unidad de control.

Paso 2: decodificación y lectura de operandos

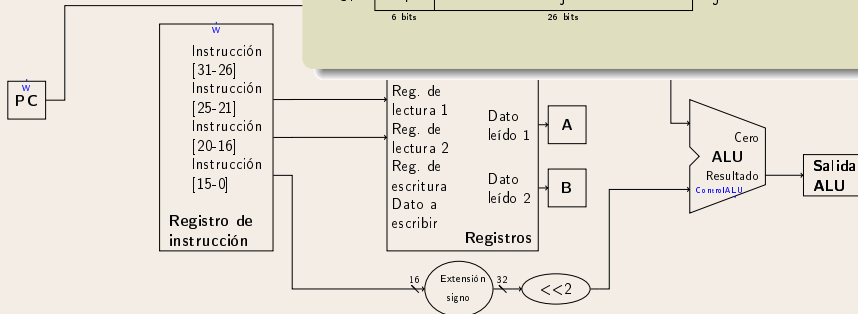
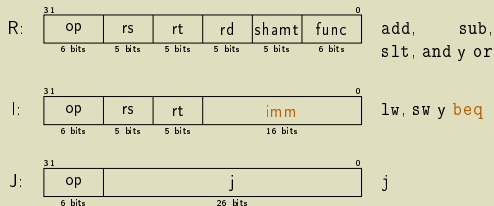
$A = \text{Reg}[\text{IR}[25-21]]$
 $B = \text{Reg}[\text{IR}[20-16]]$
 $\text{ALUOut} = \text{PC} + (\text{exten})$



- Leemos **especulativamente** los registros apuntados por **rs** y **rt**.

Paso 2: decodificación y lectura de operandos

$A = \text{Reg}[\text{IR}[25-21]]$
 $B = \text{Reg}[\text{IR}[20-16]]$
 $\text{ALUOut} = \text{PC} + (\text{exters})$

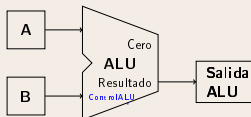


- Se calcula **especulativamente** el destino del salto condicional.

Paso 3a: operación

- Instrucciones add, sub, and, or y slt.

ALUOut = A func B

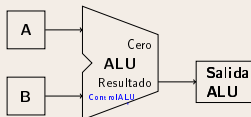


Paso 3a: operación



- Instrucciones add, sub, and, or y slt.

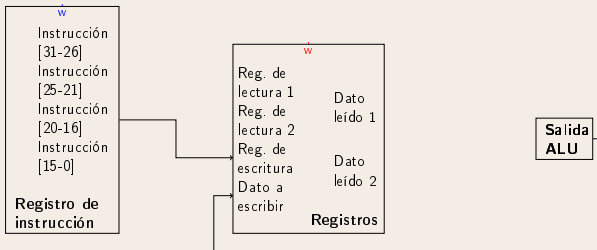
$$\text{ALUOut} = A \text{ func } B$$



Paso 4a: finalización de la operación

- Instrucciones add, sub, and, or y slt.

$\text{Reg}[\text{Ir}[15-11]] = \text{ALUOut}$

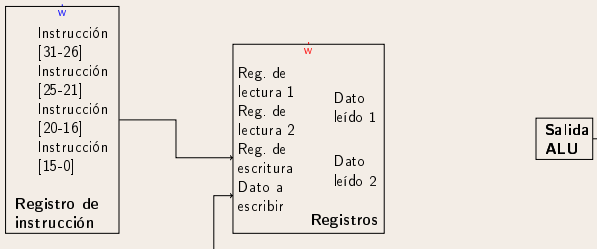


Paso 4a: finalización de la operación



- Instrucciones add, sub, and,

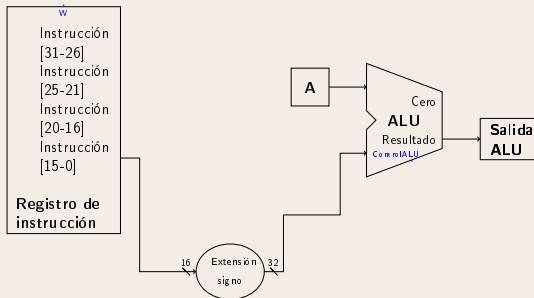
$\text{Reg}[\text{Ir}[15-11]] = \text{ALUOut}$



Paso 3b: cálculo de la dirección

- Instrucciones lw y sw.

$$\text{ALUOut} = A + \text{extender_signo}(\text{IR}[15-0])$$

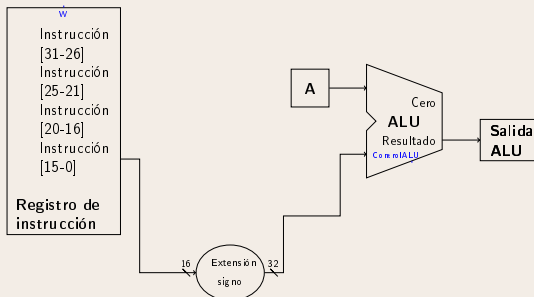


Paso 3b: cálculo de la dirección

- Instrucciones lw y sw.



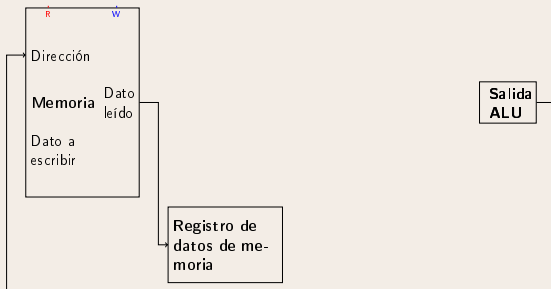
$$\text{ALUOut} = A + \text{extender_signo}(\text{IR}[15-0])$$



Paso 4b: lectura de memoria

- Instrucción lw.

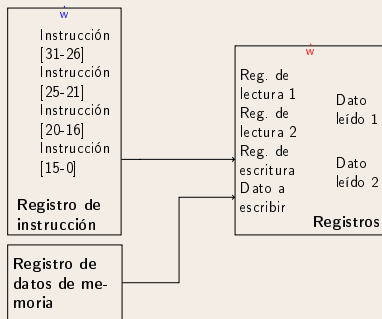
MDR = Memoria[ALUOut]



Paso 5: finalización de la lectura

- Instrucción lw.

$\text{Reg}[\text{IR}[20-16]] = \text{MDR}$

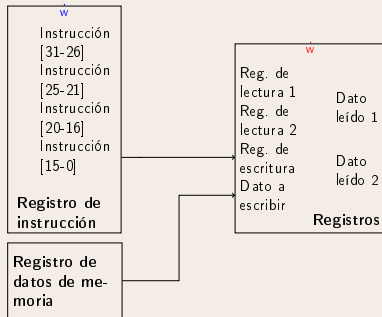


Paso 5: finalización de la lectura

- Instrucción lw.



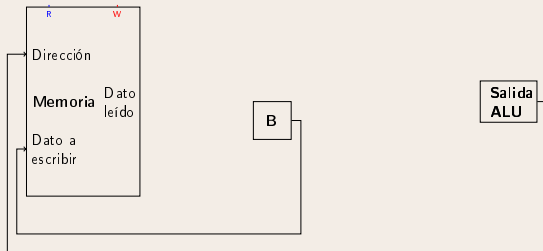
$\text{Reg}[\text{IR}[20-16]] = \text{MDR}$



Paso 4b: escritura en memoria

- Instrucción sw.

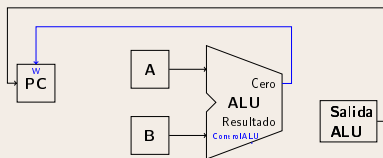
Memoria[ALUOut] = B



Paso 3c: comparación y salto

- Instrucción beq.

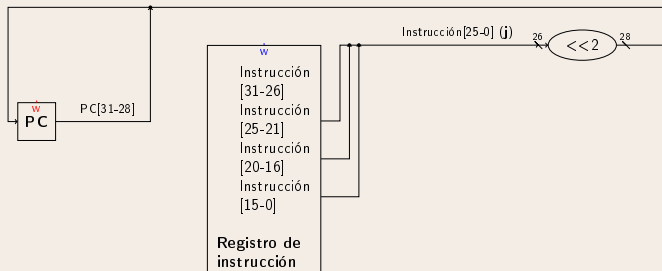
si (A == B) entonces PC = ALUOut



Paso 3d: salto

- Instrucción j.

$$PC = (PC[31-28] \ll 28) \parallel (IR[25-0] \ll 2)$$

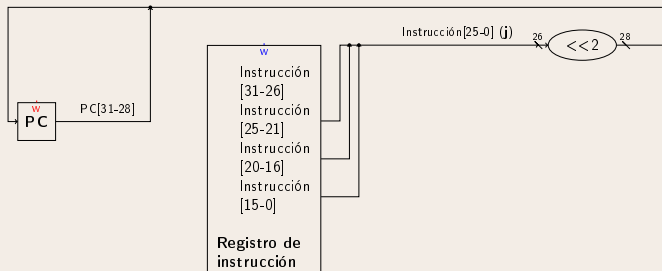


Paso 3d: salto

- Instrucción j.



$$PC = (PC[31-28] \ll 28) \mid (IR[25-0] \ll 2)$$



Combinando todos los pasos

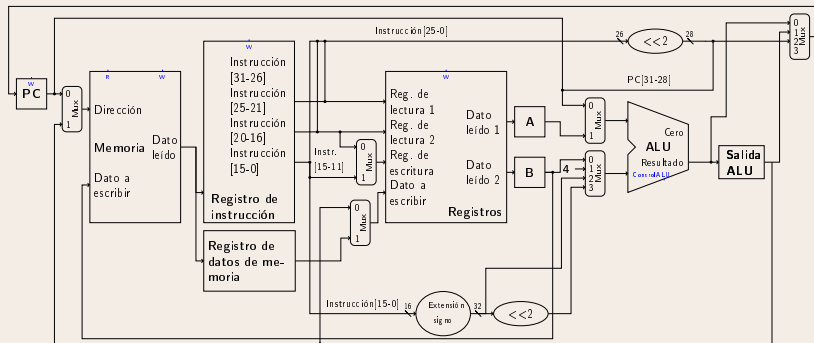
- Necesitamos un camino de datos capaz de realizar todos los pasos que hemos visto en las transparencias anteriores.

Combinando todos los pasos

- Necesitamos un camino de datos capaz de realizar todos los pasos que hemos visto en las transparencias anteriores.
 - Pero cada paso necesita que las unidades funcionales estén conectadas de forma diferente entre sí.

Combinando todos los pasos

- Necesitamos un camino de datos capaz de realizar todos los pasos que hemos visto en las transparencias anteriores.
 - Pero cada paso necesita que las unidades funcionales estén conectadas de forma diferente entre sí.
- ⇒ **Solución:** añadir multiplexores.



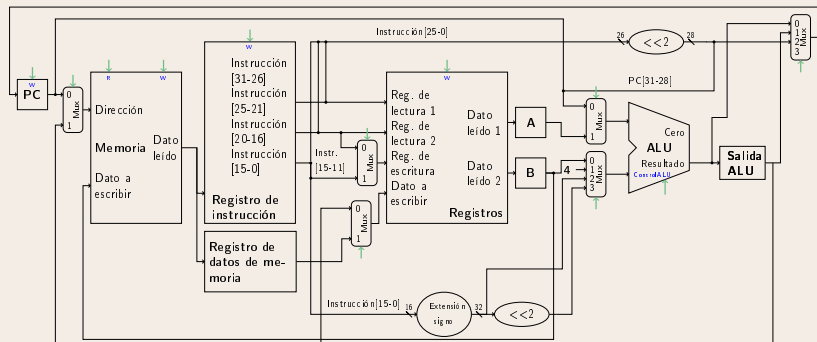
Índice

- 1 Introducción
- 2 Visión general de la implementación
- 3 El camino de datos
- 4 Control del camino de datos**

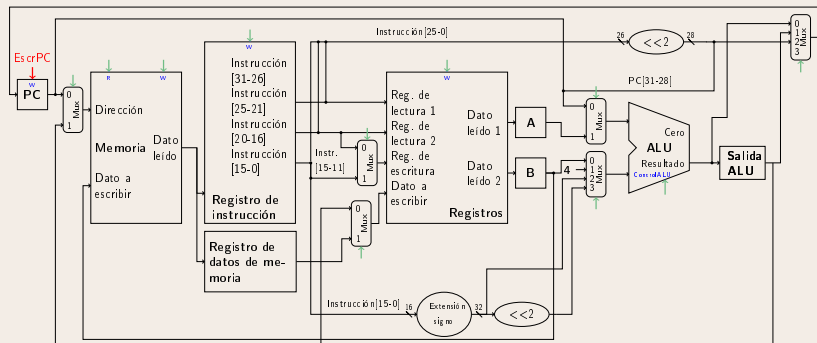
Objetivo de la unidad de control

- Necesitamos un componente que se encargue de guiar la ejecución de instrucciones en nuestro camino de datos \Rightarrow **unidad de control**. Deberá:
 - Seleccionar las entradas adecuadas de los multiplexores en cada paso.
 - Activar las señales de habilitación de escritura de los registros y la señal de lectura de memoria en los pasos que sea necesario.
 - Indicar a la ALU qué operación realizar.
- Definiremos un conjunto de **señales de control** que serán generadas por la unidad de control y que conectaremos a los puertos adecuados de cada componente del camino de datos.
 - Añadiremos un poco de lógica combinacional para simplificar el control de la ALU y para implementar los saltos condicionales.

Señales de control



Señales de control

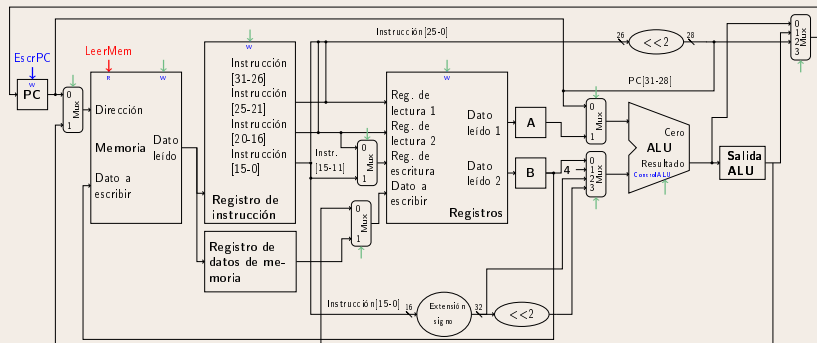


- **EscrPC:** controla la escritura en el registro PC.

0 No se escribe.

1 Se actualiza con el valor que haya a la entrada.

Señales de control

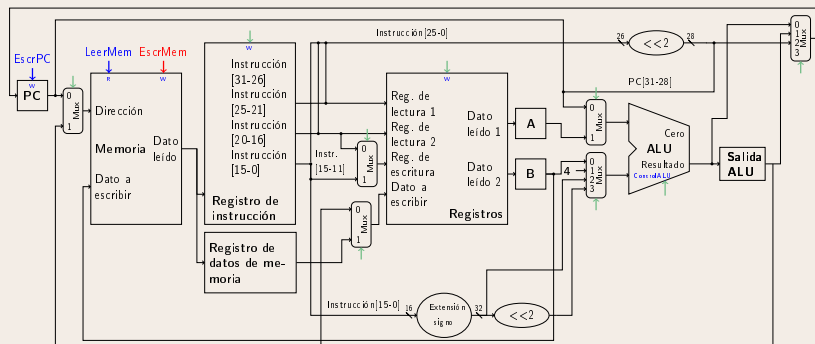


- **LeerMem:** activa la lectura de memoria.

0 No se lee.

1 Se lee la dirección de memoria indicada.

Señales de control

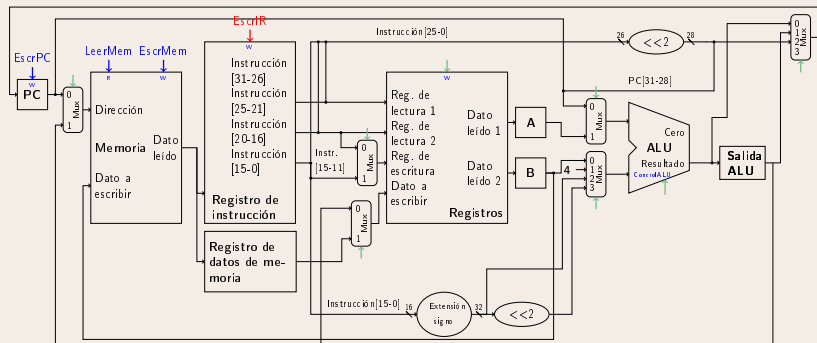


- **EscrMem**: activa la escritura en memoria.

0 No se escribe.

1 Se escribe el dato en la dirección de memoria indicada.

Señales de control

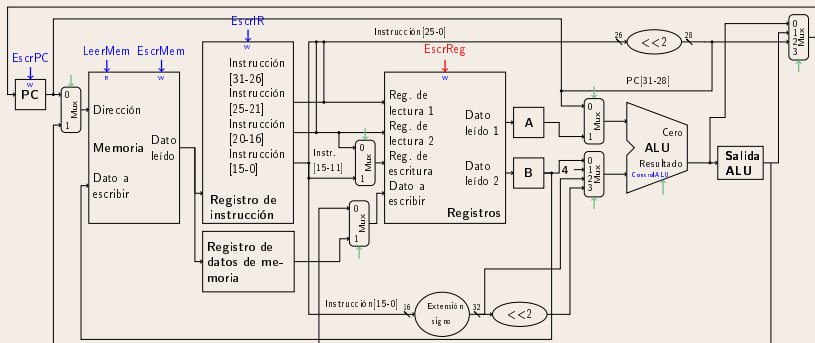


- **Es crIR**: controla la escritura en el registro IR.

0 Se conserva el contenido actual del registro IR.

1 Se escribe la instrucción leída de memoria.

Señales de control

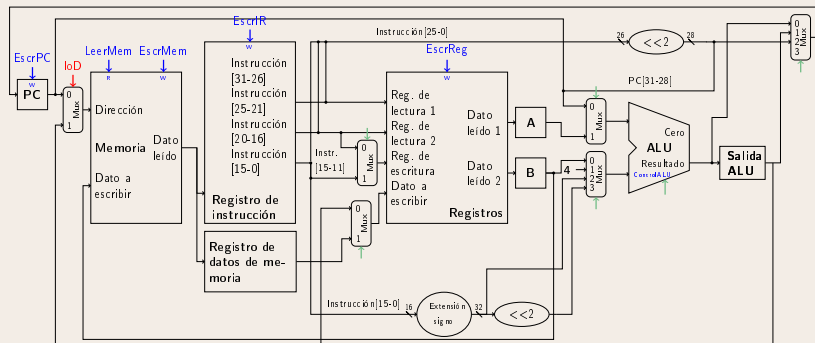


- **EscrReg:** controla la escritura en el banco de registros.

0 No se escribe.

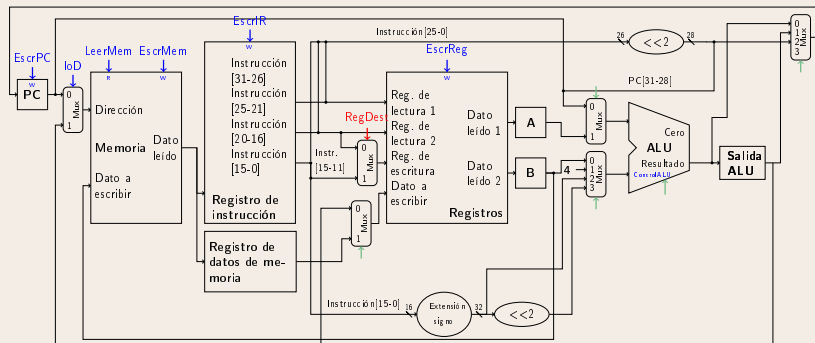
1 Se escribe el dato en el registro indicado.

Señales de control



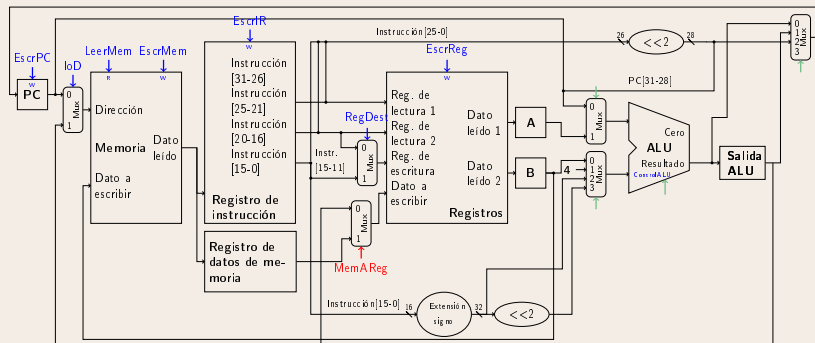
- **loD:** selecciona qué dirección se usa para acceder a la memoria.
 - 0 El PC suministra la dirección para acceder a memoria.
 - 1 ALUOut proporciona la dirección para acceder a memoria.

Señales de control



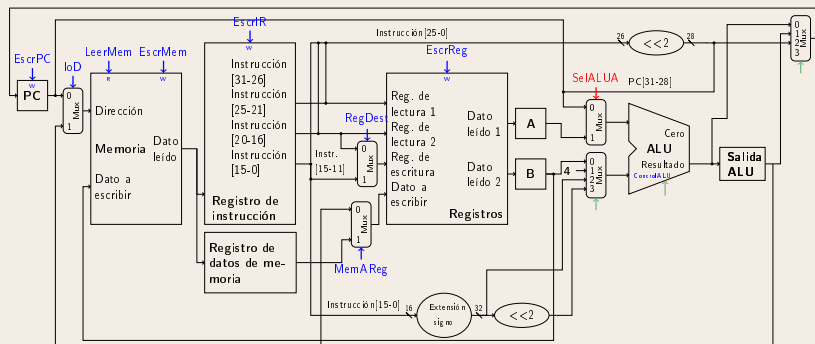
- **RegDest:** selecciona qué campo elige el registro a escribir.
 - 0 El registro destino viene determinado por el campo **rt**.
 - 1 El registro destino viene determinado por el campo **rd**.

Señales de control



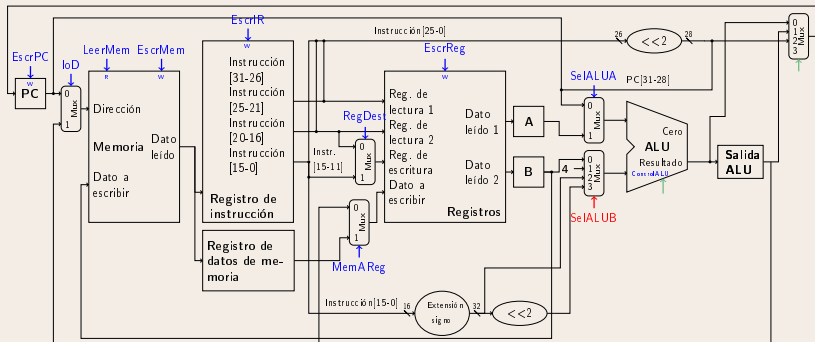
- **MemAReg:** selecciona qué se escribe en el banco de registros.
 - 0 El valor que proviene del registro ALUOut.
 - 1 El valor que proviene del registro MDR.

Señales de control



- **SelALUA:** selecciona el primer operando de la ALU.
 - 0 El primer operando de la ALU es el PC.
 - 1 El primer operando de la ALU es el registro A.

Señales de control



- **SelALUB:** selecciona el segundo operando de la ALU.

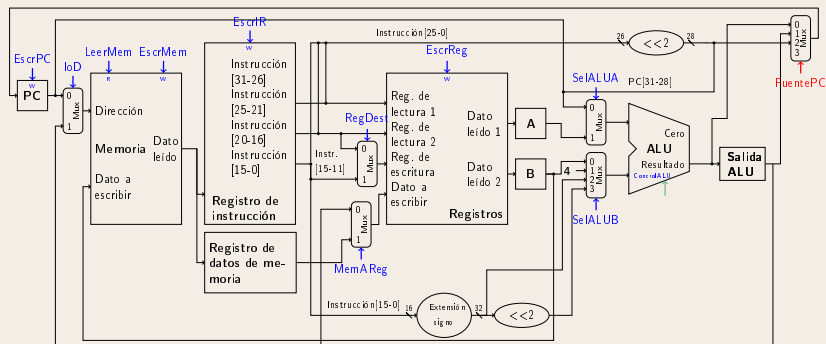
00 El registro B.

01 La constante 4.

10 Los 16 bits menos significativos de IR, extendidos de signo.

11 Igual que 10, pero desplazado 2 bits a la izquierda.

Señales de control



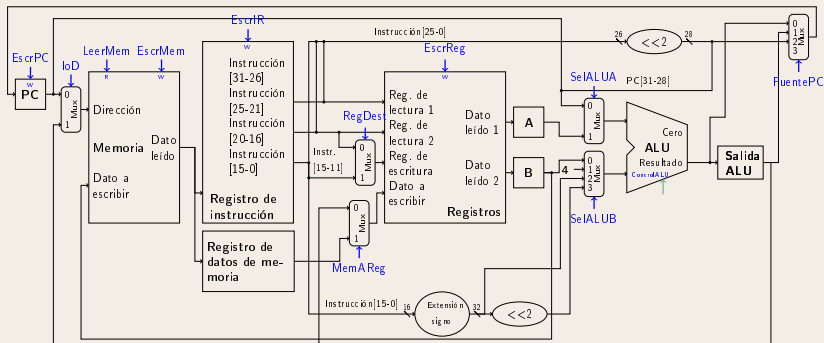
- **FuentePC:** selecciona qué se escribe en el registro PC.

00 La salida de la ALU.

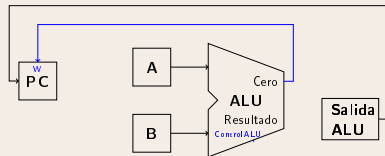
01 El contenido de ALUOut.

10 El campo **j** desplazado dos bits a la izquierda y concatenado con los 4 bits más significativos del PC actual.

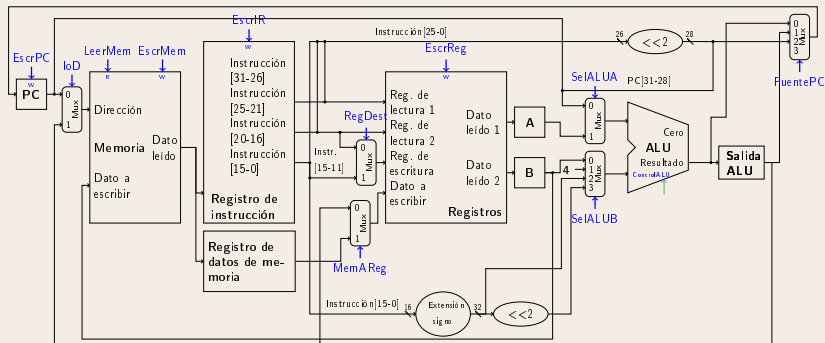
Señales de control



¿Cómo hacemos que funcione la instrucción beq (paso 3c)?

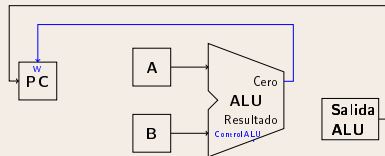


Señales de control

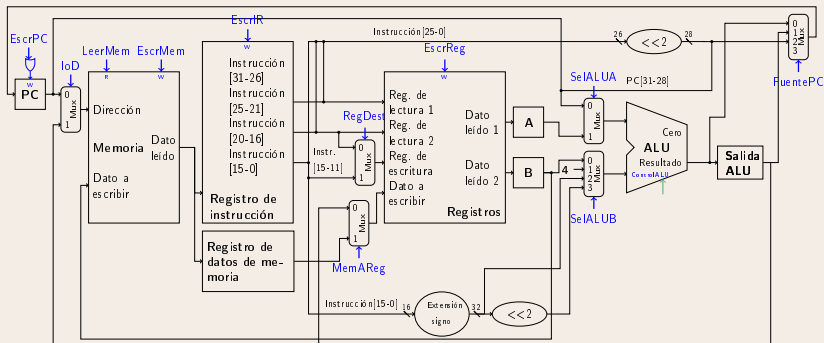


¿Cómo hacemos que funcione la instrucción beq (paso 3c)?

→ Tenemos que añadir un par de puertas lógicas.

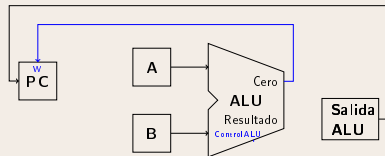


Señales de control

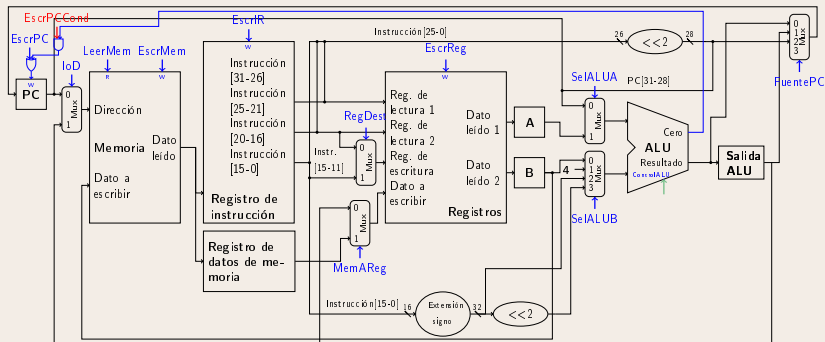


¿Cómo hacemos que funcione la instrucción *beq* (paso 3c)?

→ Tenemos que añadir un par de puertas lógicas.

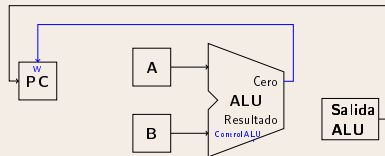


Señales de control

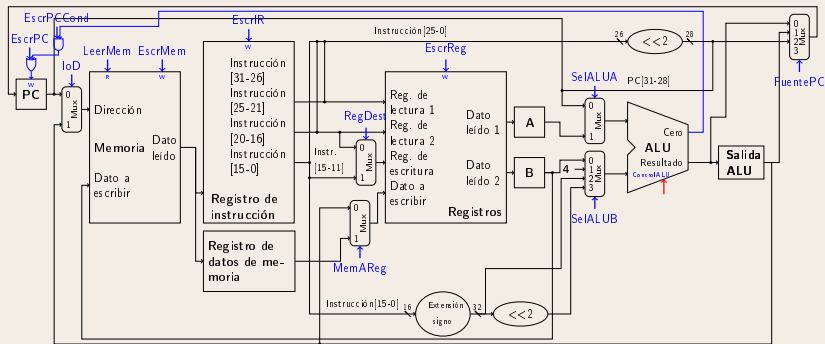


¿Cómo hacemos que funcione la instrucción beq (paso 3c)?

→ Tenemos que añadir un par de puertas lógicas.



Señales de control



Sólo falta controlar la ALU.

ControlALU	Operación
000	AND
001	OR
010	Suma
110	Resta
111	slt

Control de la ALU

- Para simplificar el control, vamos a introducir una señal de control auxiliar que nos ayude a generar la señal **ControlALU**.

Necesitamos que la ALU:

Pasos	Acción
1, 2 y 3b	Sumar
3c	Restar
3a	Depende

Control de la ALU

- Para simplificar el control, vamos a introducir una señal de control auxiliar que nos ayude a generar la señal **ControlALU**.

Necesitamos que la ALU:

Pasos	Acción
1, 2 y 3b	Sumar
3c	Restar
3a	Depende



Control de la ALU

- Para simplificar el control, vamos a introducir una señal de control auxiliar que nos ayude a generar la señal **ControlALU**.

Necesitamos que la ALU:

Pasos	Acción
1, 2 y 3b	Sumar
3c	Restar
3a	Depende



- ALUOp**: controla la acción de la ALU.
 - 00 La ALU realiza una suma.
 - 01 La ALU realiza una resta.
 - 10 Realiza un AND, OR, suma, resta o comparación según **func**.

Control de la ALU

- La **unidad de control de la ALU** será un circuito combinacional que generará la señal **ControlALU** a partir de **ALUOp** y **func**.

ALUOp	Acción	ControlALU
00	Suma	010
01	Resta	110
10	Según func →	

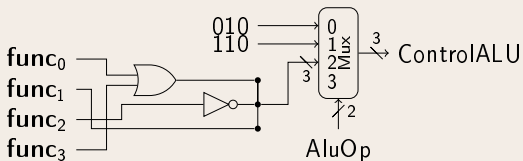
Instr.	func	Operación	ControlALU
and	100100	AND bit a bit	000
or	100101	OR bit a bit	001
add	100000	Suma	010
sub	100010	Resta	110
slt	101010	Comparación	111

Control de la ALU

- La **unidad de control de la ALU** será un circuito combinacional que generará la señal **ControlALU** a partir de **ALUOp** y **func**.

ALUOp	Acción	ControlALU
00	Suma	010
01	Resta	110
10	Según func →	

Instr.	func	Operación	ControlALU
and	100100	AND bit a bit	000
or	100101	OR bit a bit	001
add	100000	Suma	010
sub	100010	Resta	110
slt	101010	Comparación	111

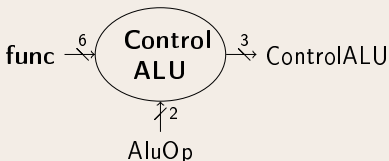


Control de la ALU

- La **unidad de control de la ALU** será un circuito combinacional que generará la señal **ControlALU** a partir de **ALUOp** y **func**.

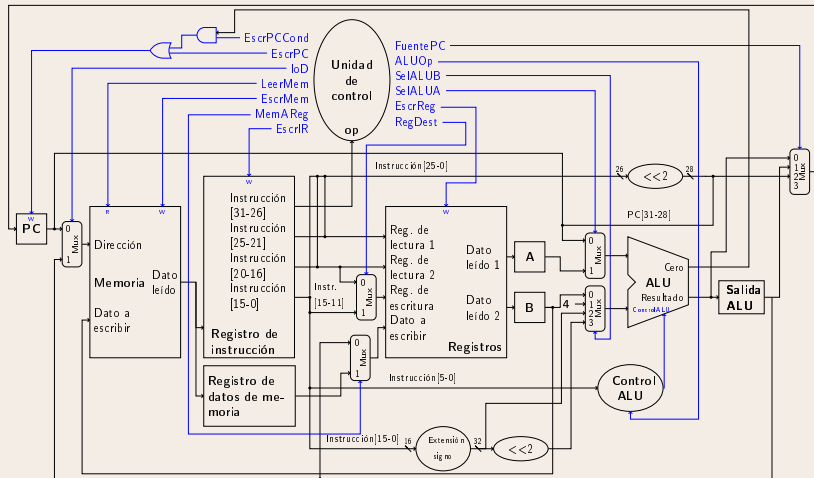
ALUOp	Acción	ControlALU
00	Suma	010
01	Resta	110
10	Según func →	

Instr.	func	Operación	ControlALU
and	100100	AND bit a bit	000
or	100101	OR bit a bit	001
add	100000	Suma	010
sub	100010	Resta	110
slt	101010	Comparación	111



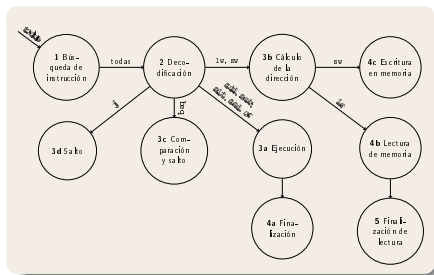
Visión completa del procesador

La **unidad de control** generará las señales adecuadas a cada paso:



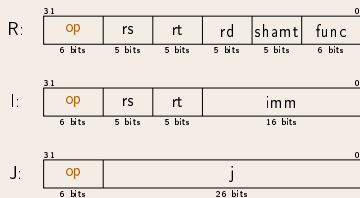
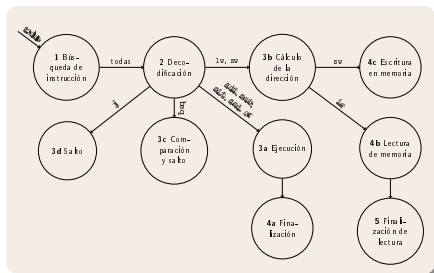
Implementación de la unidad de control

- El valor de las señales de control que hemos definido depende **únicamente** del paso actual en el que nos encontremos.
- La transición de un paso a otro depende de la instrucción actual.



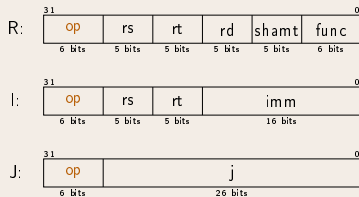
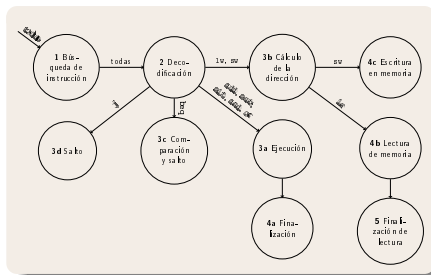
Implementación de la unidad de control

- El valor de las señales de control que hemos definido depende **únicamente** del paso actual en el que nos encontremos.
- La transición de un paso a otro depende de la instrucción actual.
→ Campo **op**.



Implementación de la unidad de control

- El valor de las señales de control que hemos definido depende **únicamente** del paso actual en el que nos encontremos.
- La transición de un paso a otro depende de la instrucción actual.
→ Campo **op**.

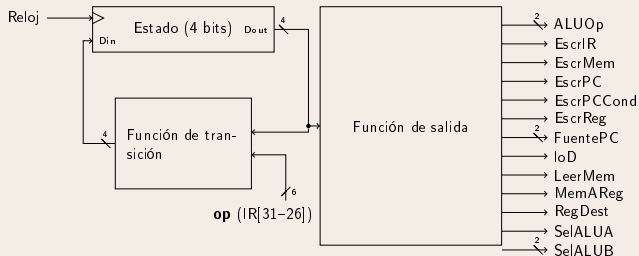


⇒ Podemos implementar la unidad de control mediante un **sistema secuencial síncrono** modelado como una máquina de Moore.

Implementación de la unidad de control

- **Estado:** paso actual.
- **Entradas:** bits del campo **op** de la última instrucción leída.
- **Salidas:** valores de las señales de control.

Esquema de implementación

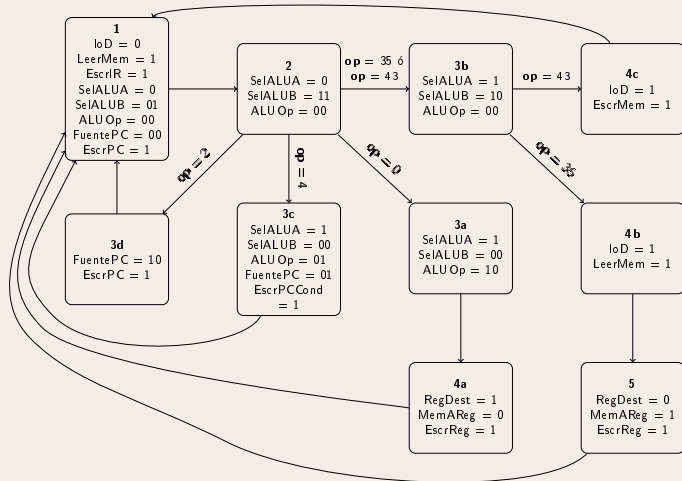


Implementación de la unidad de control

Autómata de control

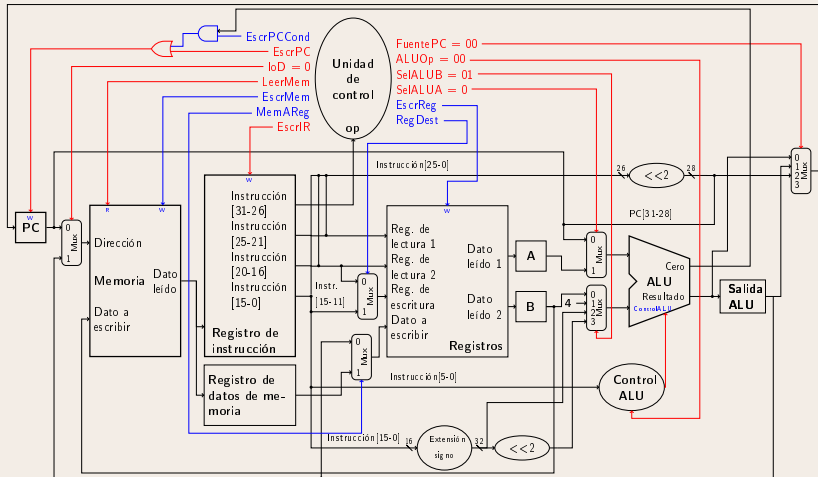
- E
- E
- S

Esque



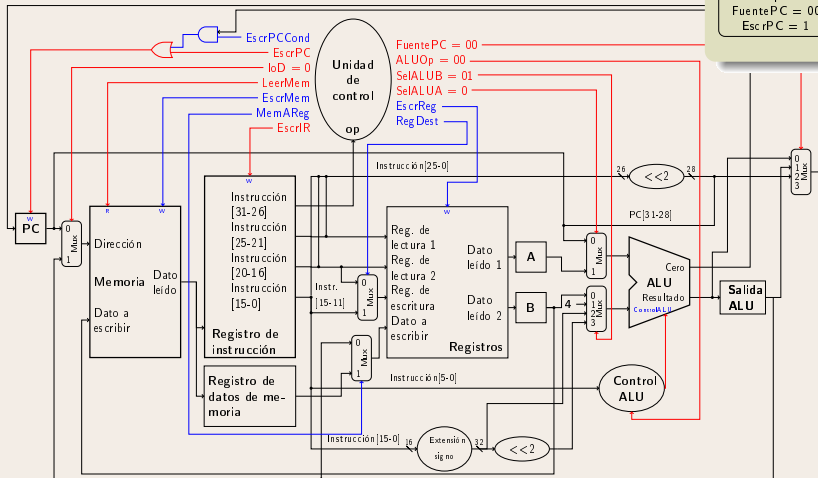
Ejemplo de ejecución de add \$1, \$2, \$3

Paso 1



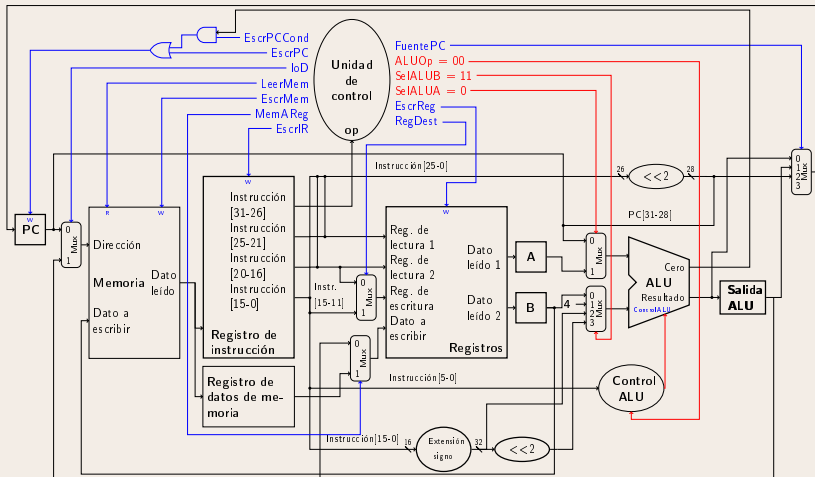
Ejemplo de ejecución de add \$1, \$2, \$3

Paso 1



Ejemplo de ejecución de add \$1, \$2, \$3

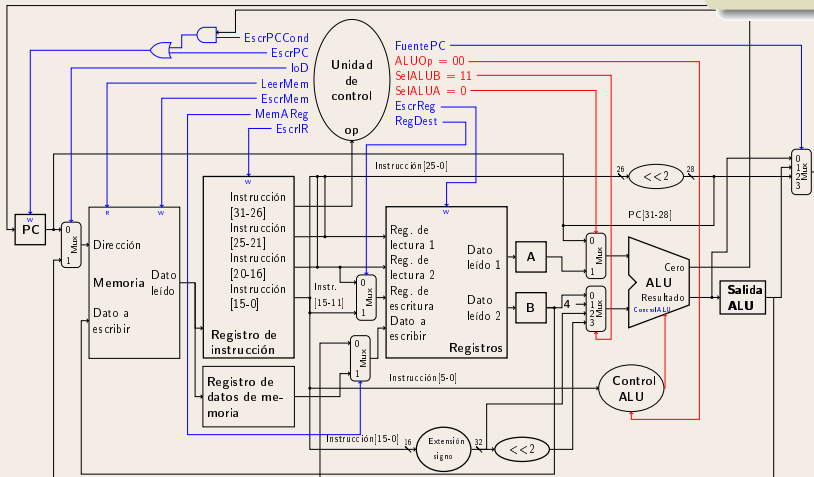
Paso 2



Ejemplo de ejecución de add \$1, \$2, \$3

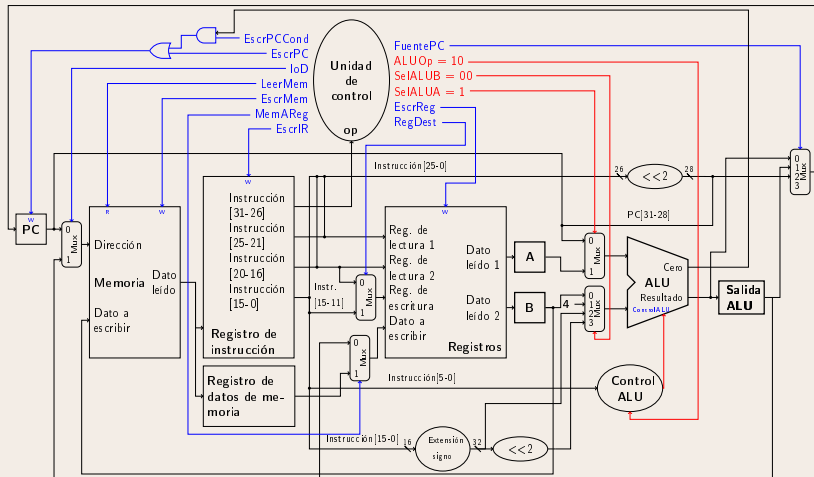
Paso 2

2
 SelALUA = 0
 SelALUB = 11
 ALUOp = 00



Ejemplo de ejecución de add \$1, \$2, \$3

Paso 3a

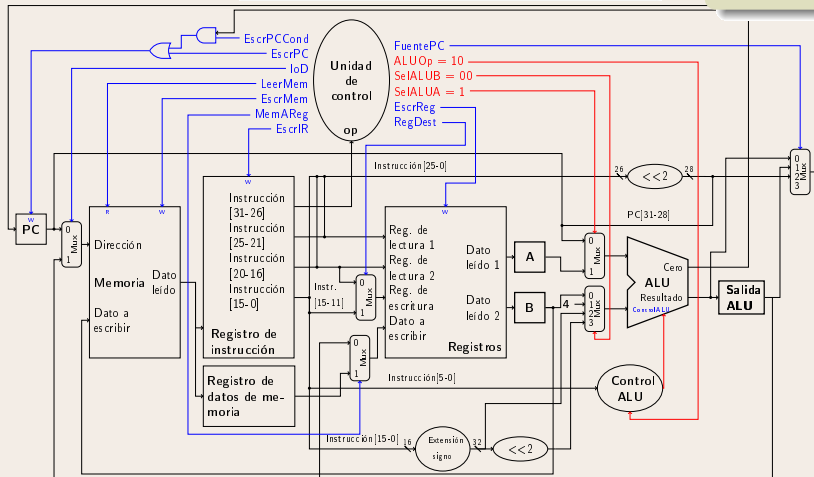


Ejemplo de ejecución de una instrucción R



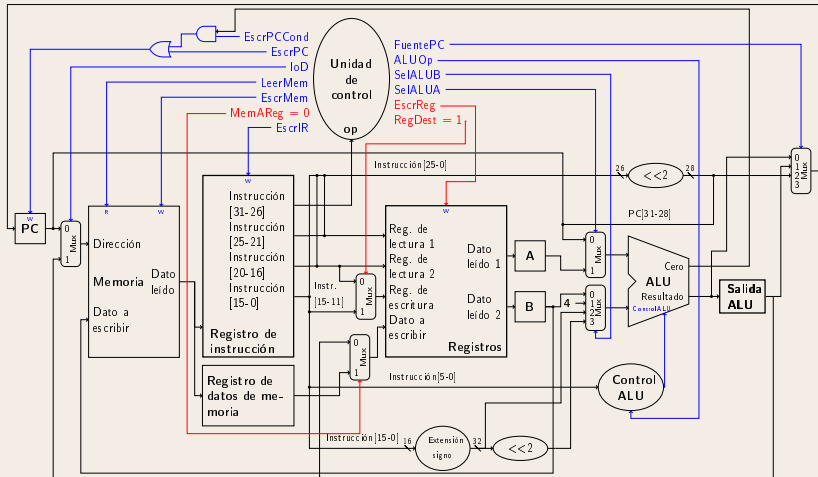
3a
 SelALUA = 1
 SelALUB = 00
 ALUOp = 10

Paso 3a



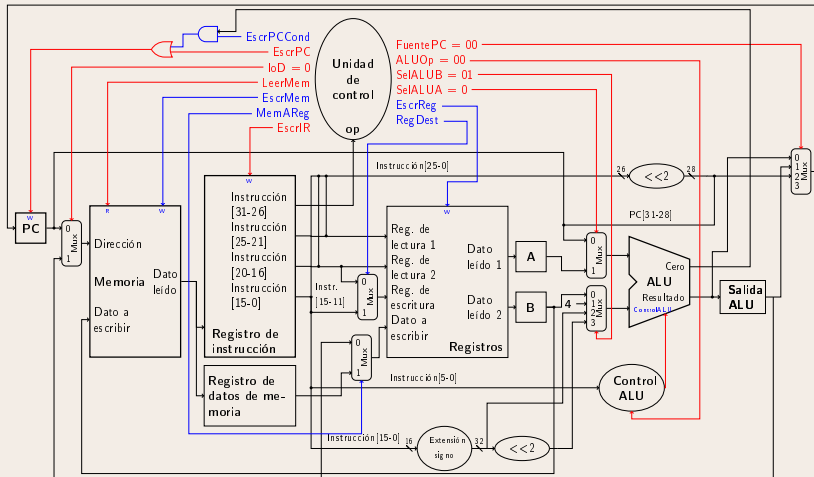
Ejemplo de ejecución de add \$1, \$2, \$3

Paso 4a



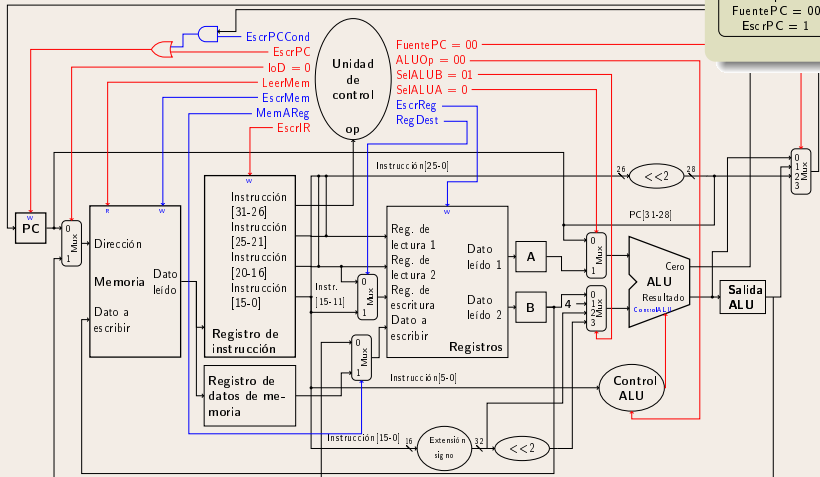
Ejemplo de ejecución de `lw $1, label($2)`

Paso 1



Ejemplo de ejecución de `lw $1, label($0)`

Paso 1



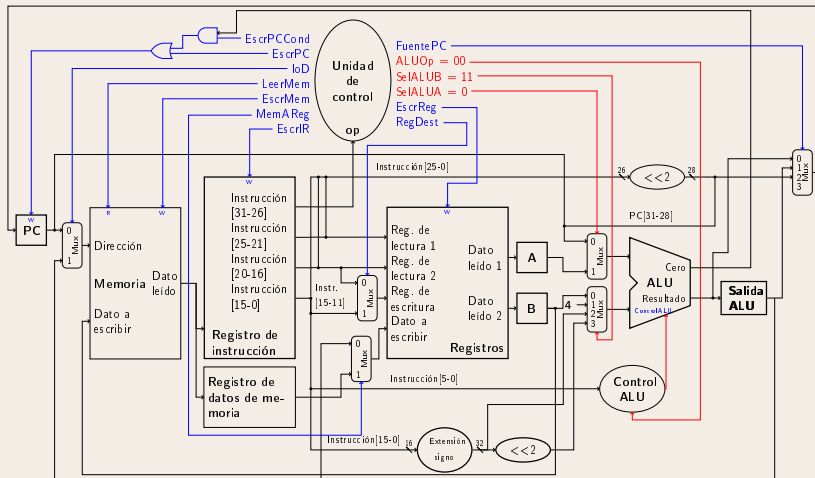
```

1
loD = 0
LeerMem = 1
EscrR = 1
SelALUA = 0
SelALUB = 01
ALUOp = 00
FuentePC = 00
EscrPC = 1

```

Ejemplo de ejecución de `lw $1, label($2)`

Paso 2

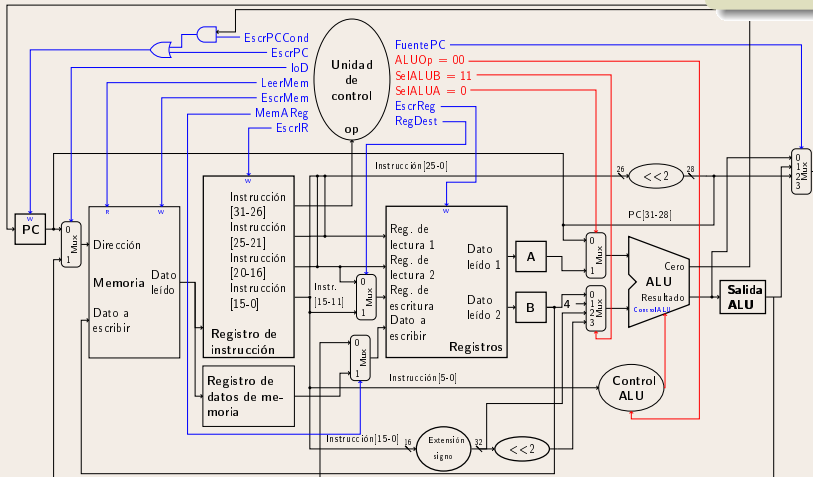


Ejemplo de ejecución de `lw $1, label($2)`

2

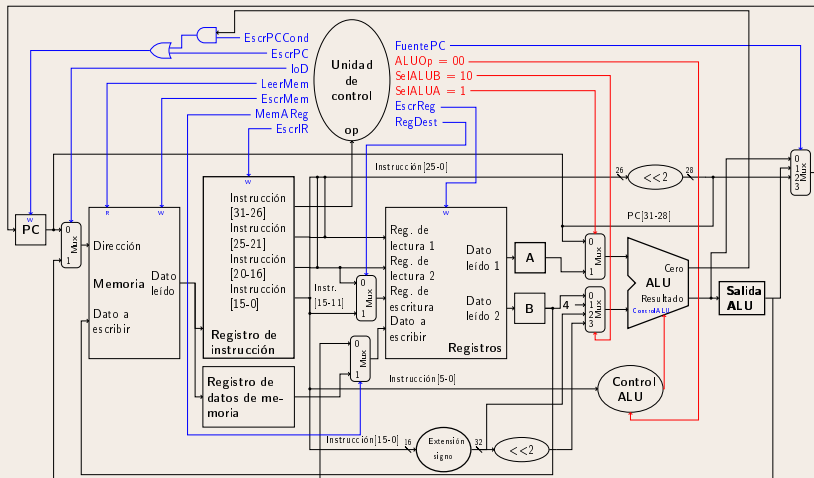
SelALUA = 0
 SelALUB = 11
 ALUOp = 00

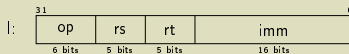
Paso 2



Ejemplo de ejecución de `lw $1, label($2)`

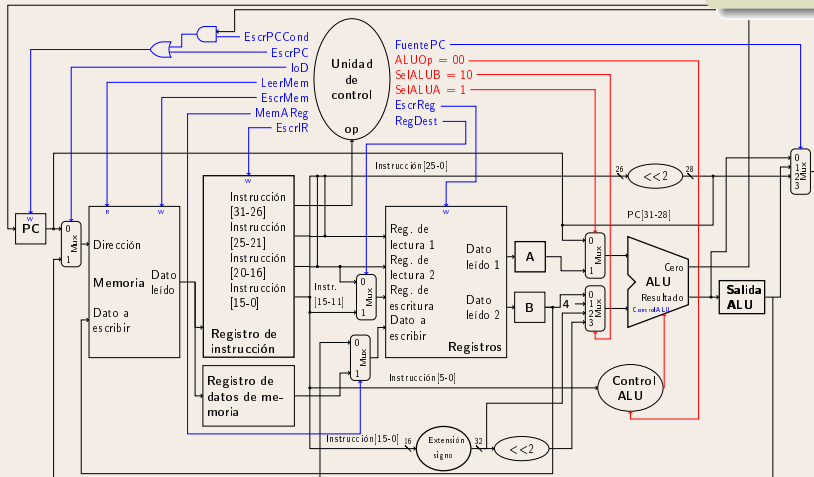
Paso 3b



Ejemplo de ejecución de $lwr \$1, 10001(\$2)$ 

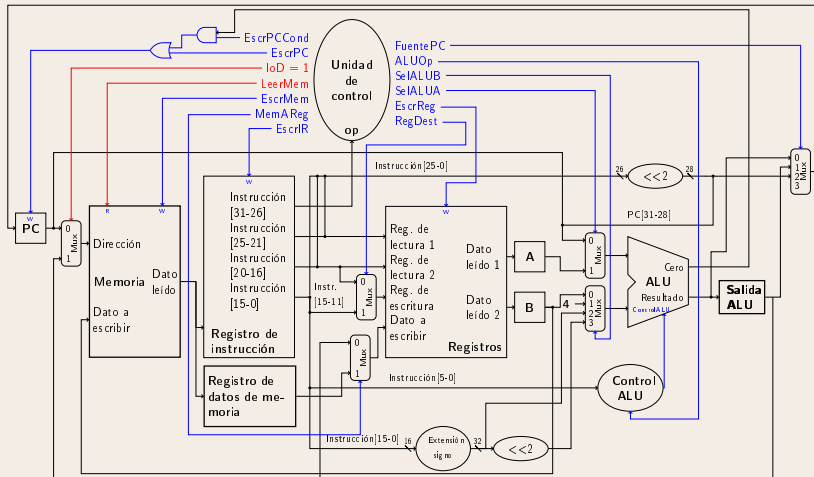
3b
 SelALUA = 1
 SelALUB = 10
 ALUOp = 00

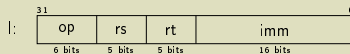
Paso 3b



Ejemplo de ejecución de `lw $1, label($2)`

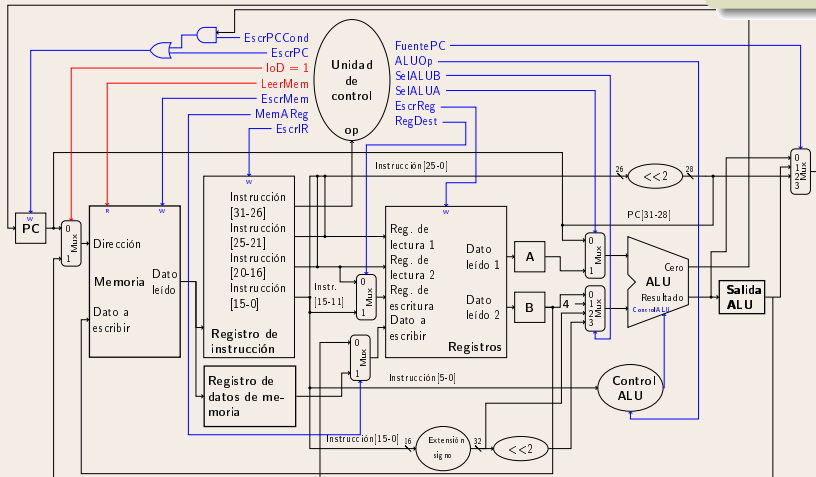
Paso 4b



Ejemplo de ejecución de $lwr \$1, 10001(\$2)$ 

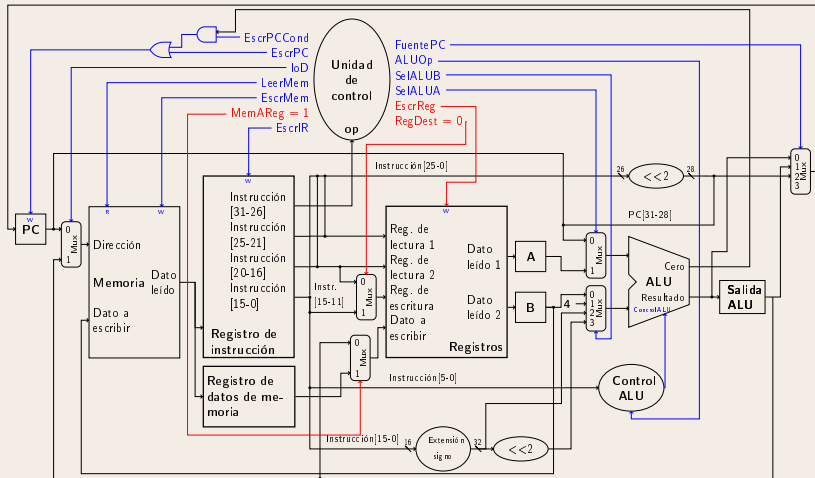
4b
 loD = 1
 LeerMem = 1

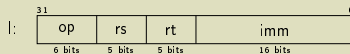
Paso 4b



Ejemplo de ejecución de `lw $1, label($2)`

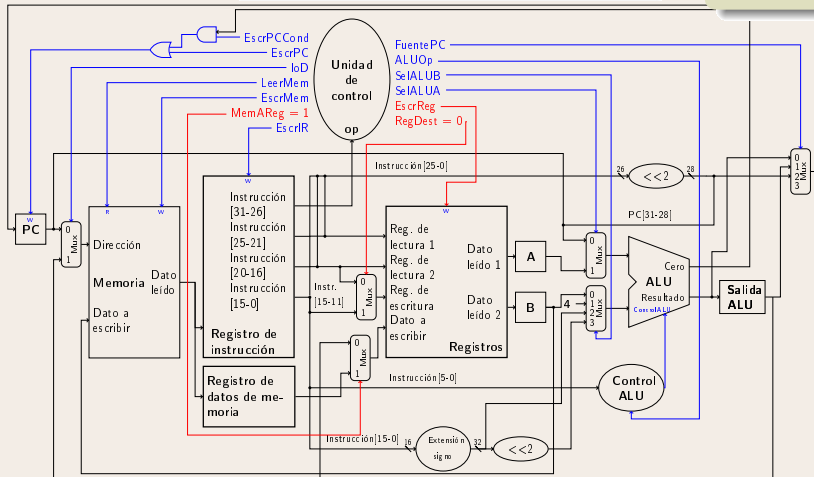
Paso 5



Ejemplo de ejecución de $lwr \$1, 10001(\$2)$ 

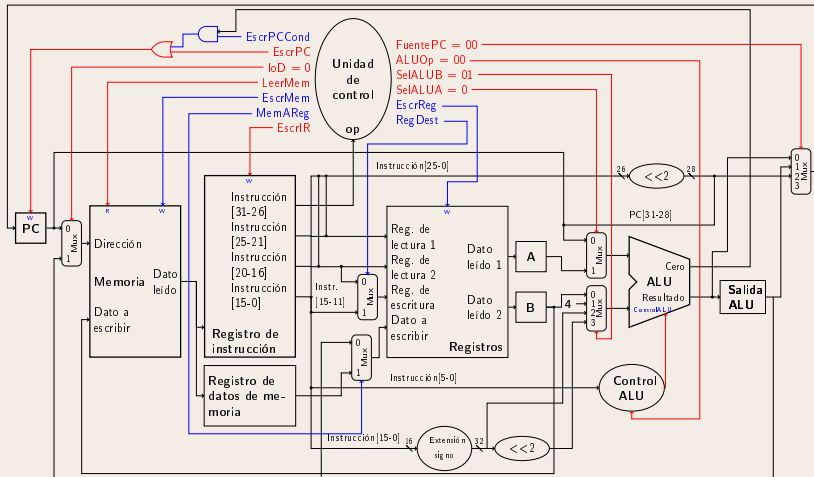
5
 RegDest = 0
 MemAReg = 1
 EscrReg = 1

Paso 5



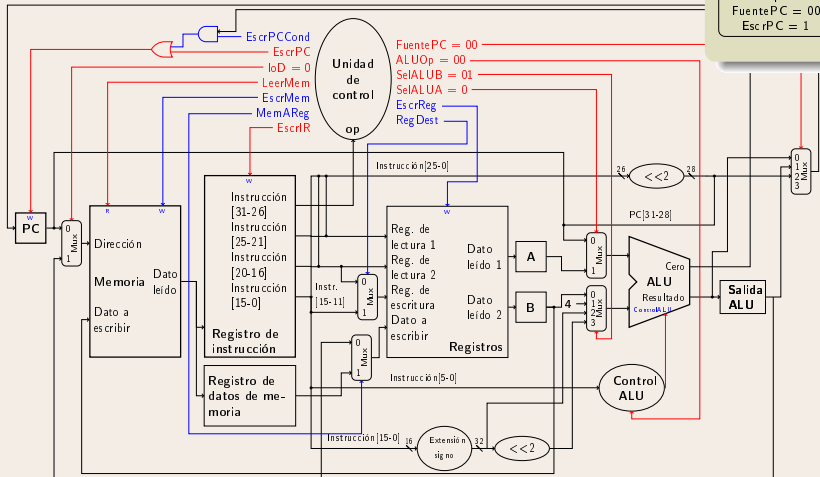
Ejemplo de ejecución de `sw $1, label($2)`

Paso 1



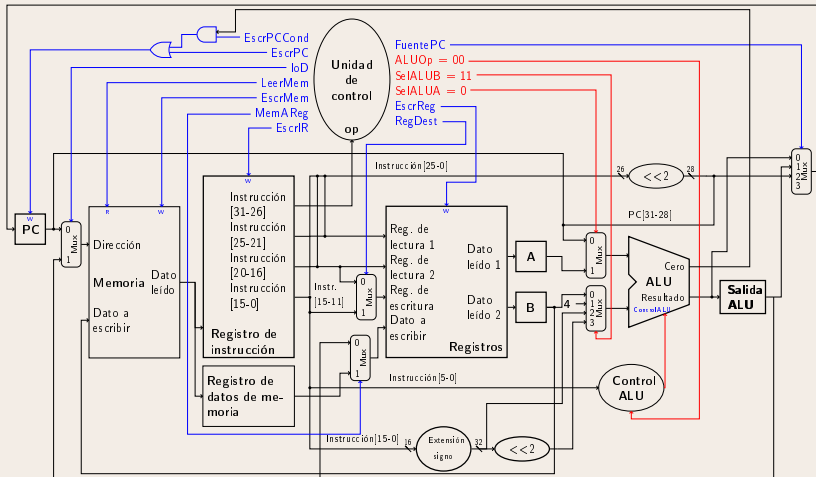
Ejemplo de ejecución de sw \$1, label(\$?)

Paso 1



Ejemplo de ejecución de `sw $1, label($2)`

Paso 2

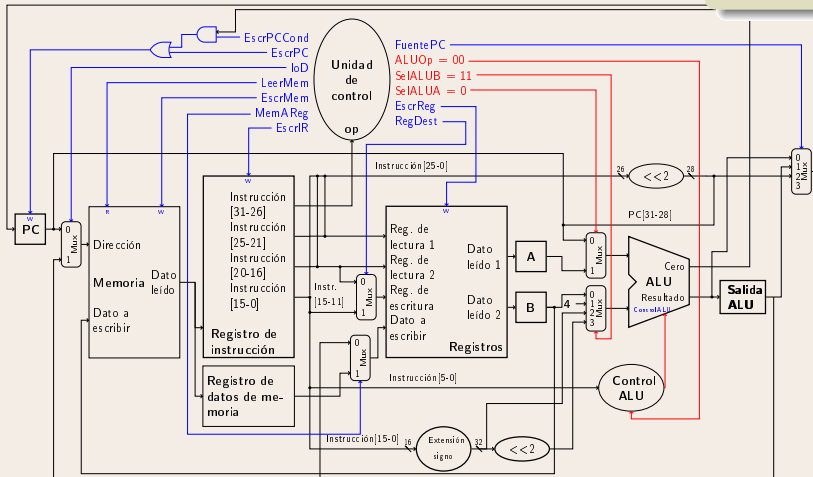


Ejemplo de ejecución de sw \$1, label(\$?)

2

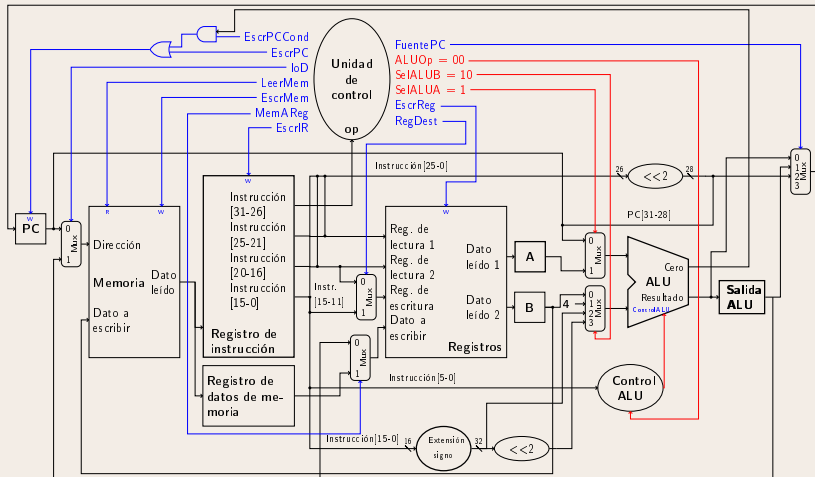
SelALUA = 0
 SelIALUB = 11
 ALUOp = 00

Paso 2



Ejemplo de ejecución de `sw $1, label($2)`

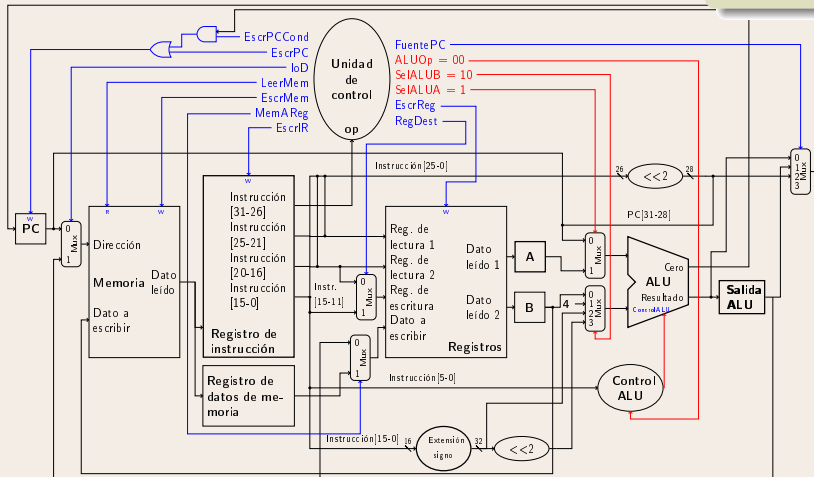
Paso 3b



Ejemplo de ejecución de $sw \$1, 16(\$2)$ 

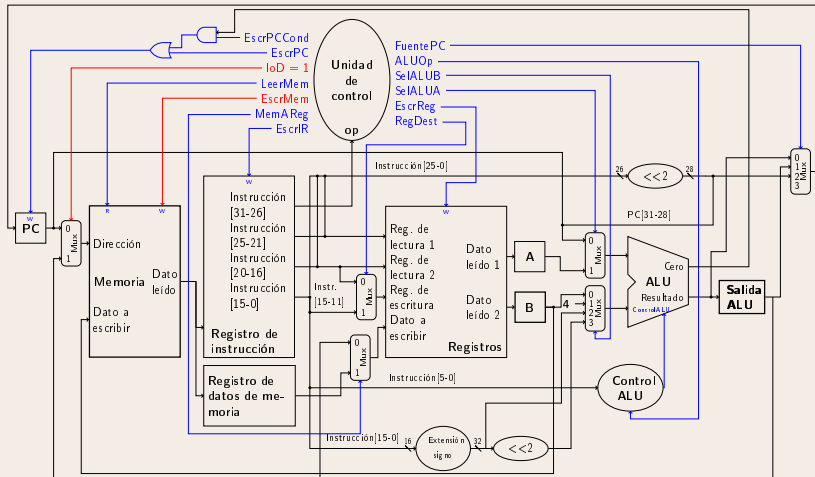
3b
 SelALUA = 1
 SelALUB = 10
 ALUOp = 00

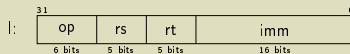
Paso 3b



Ejemplo de ejecución de sw \$1, label(\$2)

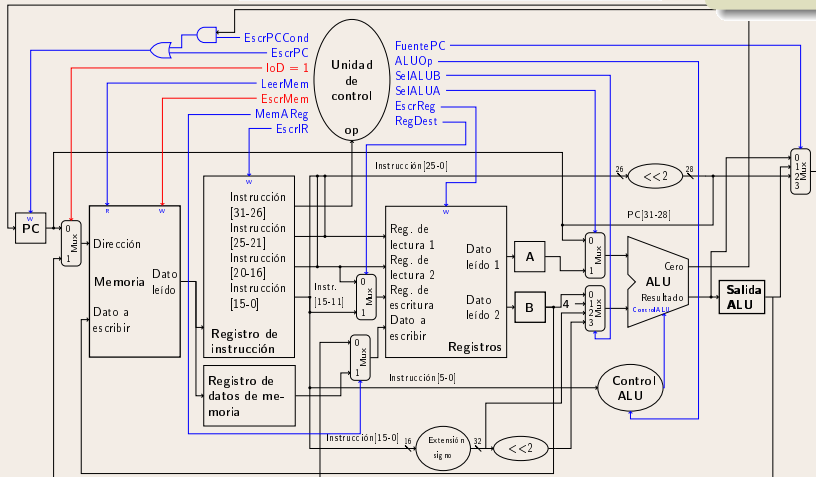
Paso 4c



Ejemplo de ejecución de $sw \$1, 16(\$2)$ 

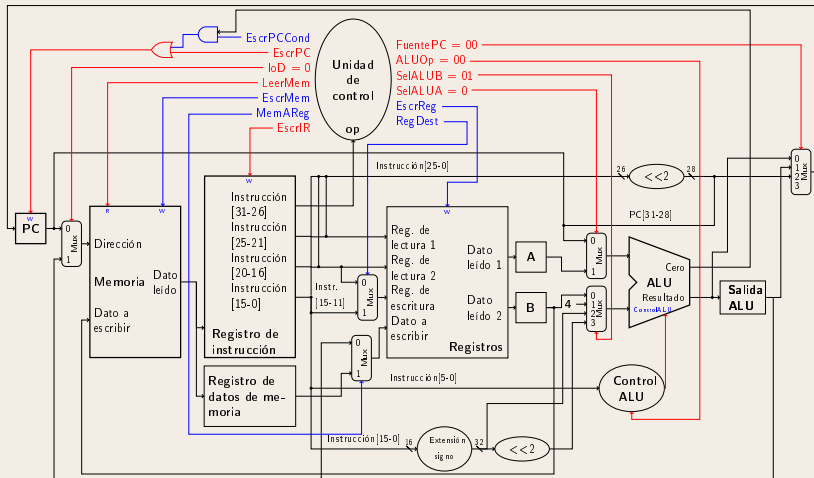
4c
 loD = 1
 EscrMem = 1

Paso 4c



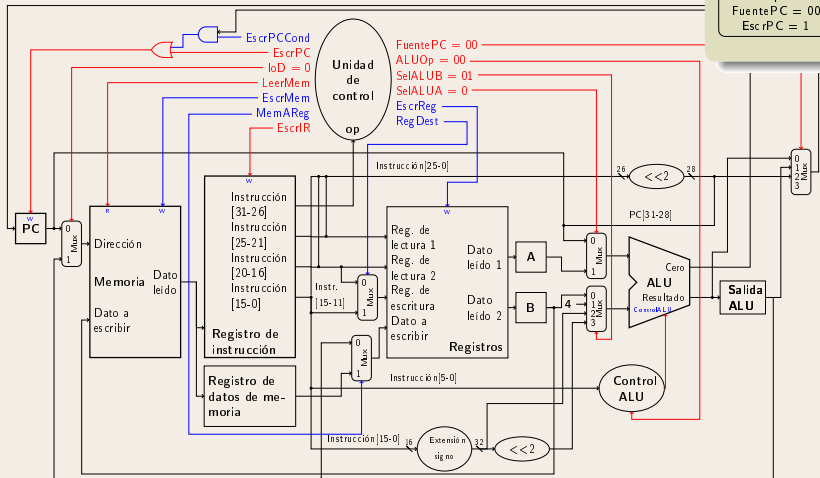
Ejemplo de ejecución de beq \$1, \$2, label

Paso 1



Ejemplo de ejecución de beq \$1, \$2, label

Paso 1



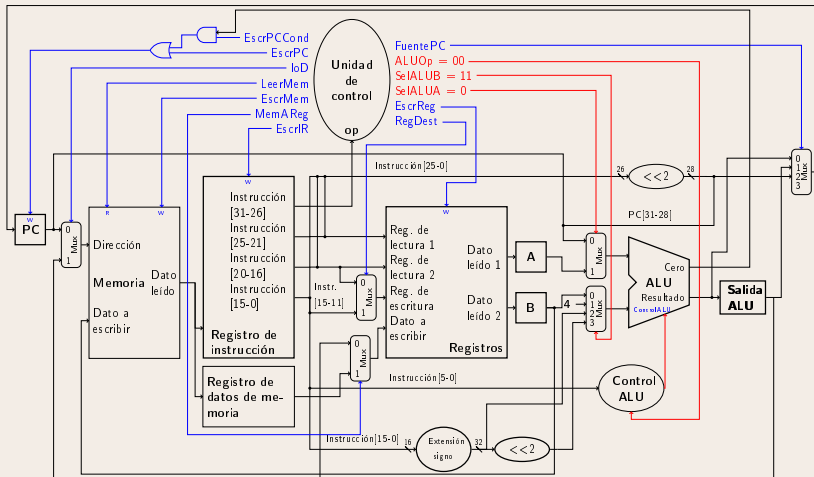
```

1
loD = 0
LeerMem = 1
Es crR = 1
SelALUA = 0
SelALUB = 01
ALUOp = 00
FuentePC = 00
Es crPC = 1

```

Ejemplo de ejecución de beq \$1, \$2, label1

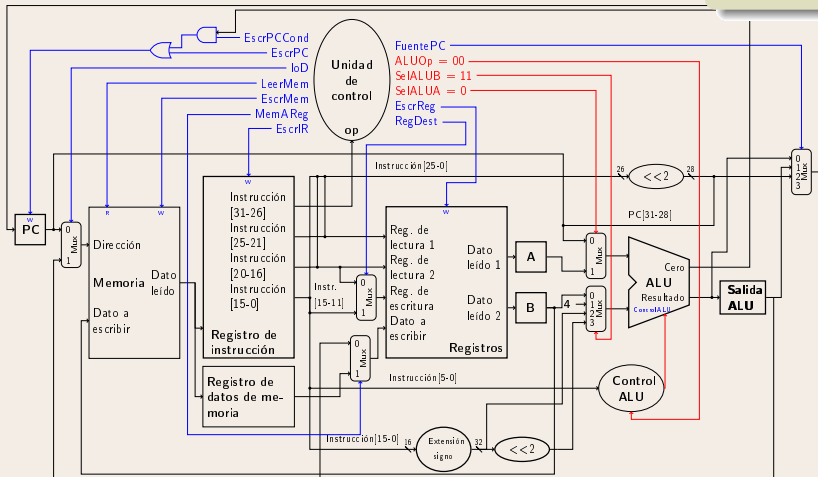
Paso 2



Ejemplo de ejecución de beq \$1, \$2, label

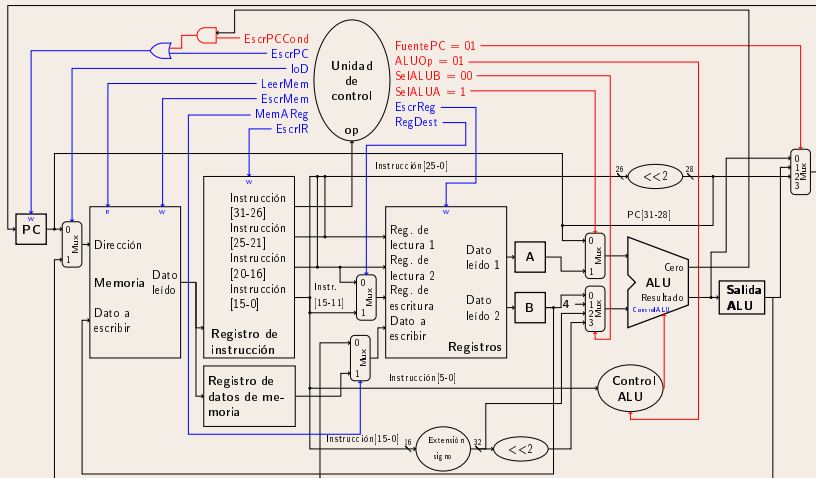
2
 SelALUA = 0
 SelALUB = 11
 ALUOp = 00

Paso 2



Ejemplo de ejecución de beq \$1, \$2, label

Paso 3c

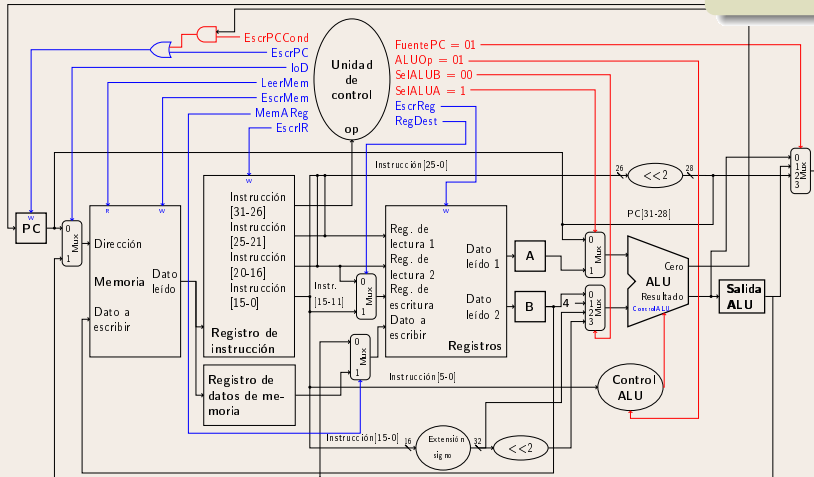


Ejemplo de ejecución de `beq $1, $2, label`

Paso 3c

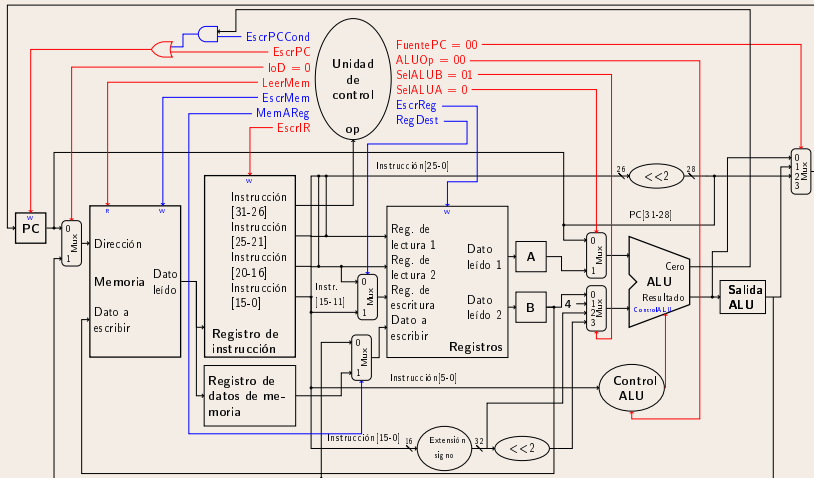


3c
 SelALUA = 1
 SelALUB = 00
 ALUOp = 01
 FuentePC = 01
 EscrPCCond = 1



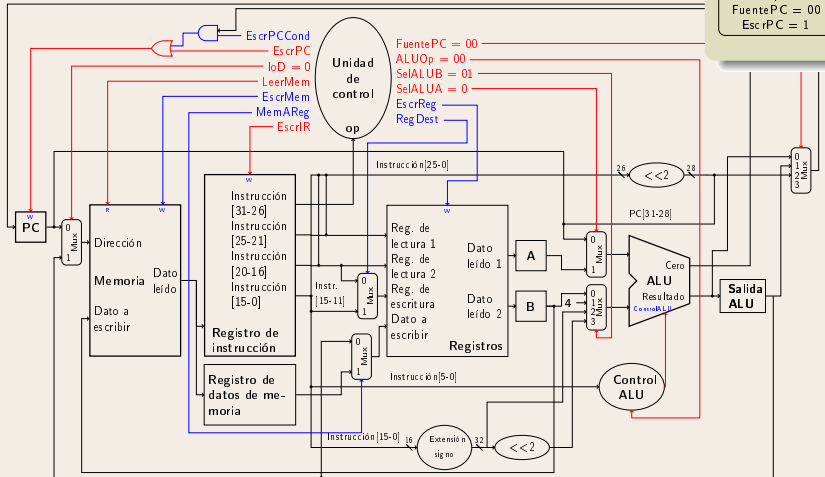
Ejemplo de ejecución de `j label`

Paso 1



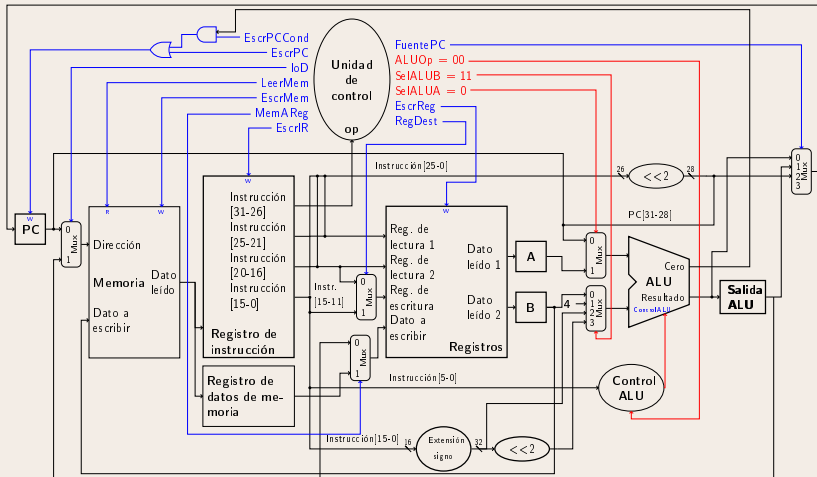
Ejemplo de ejecución de j label

Paso 1



Ejemplo de ejecución de `j label`

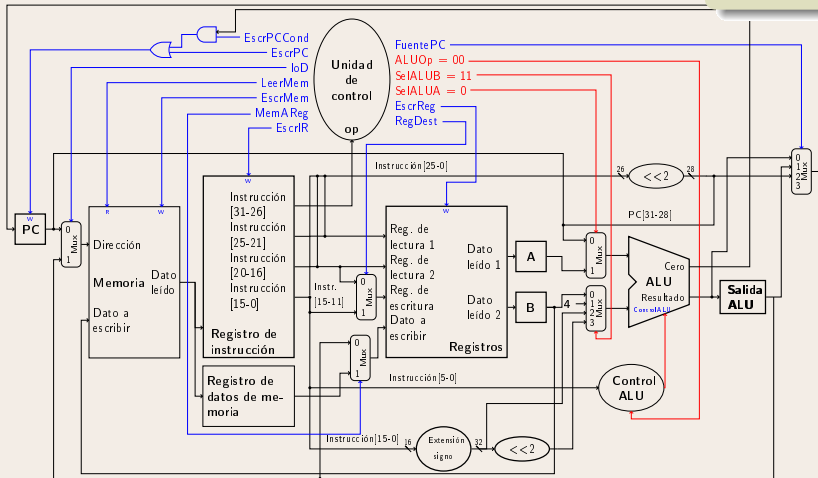
Paso 2



Ejemplo de ejecución de `j label`

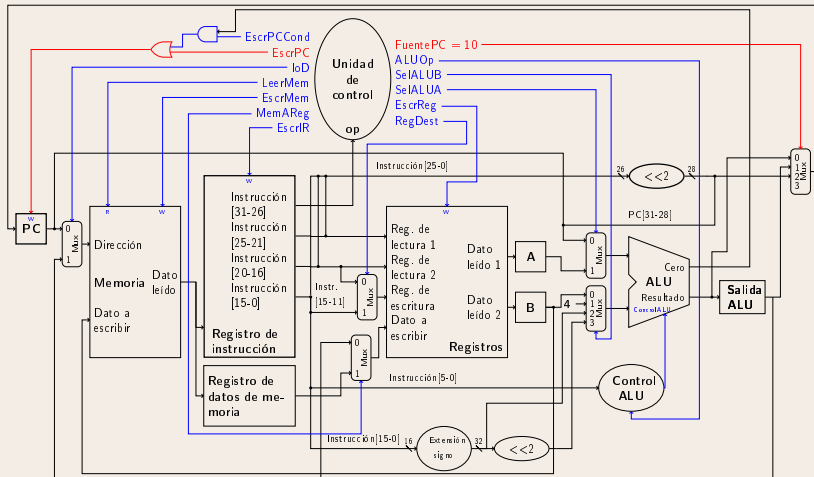
Paso 2

2
 SelALUA = 0
 SelALUB = 11
 ALUOp = 00



Ejemplo de ejecución de `j label`

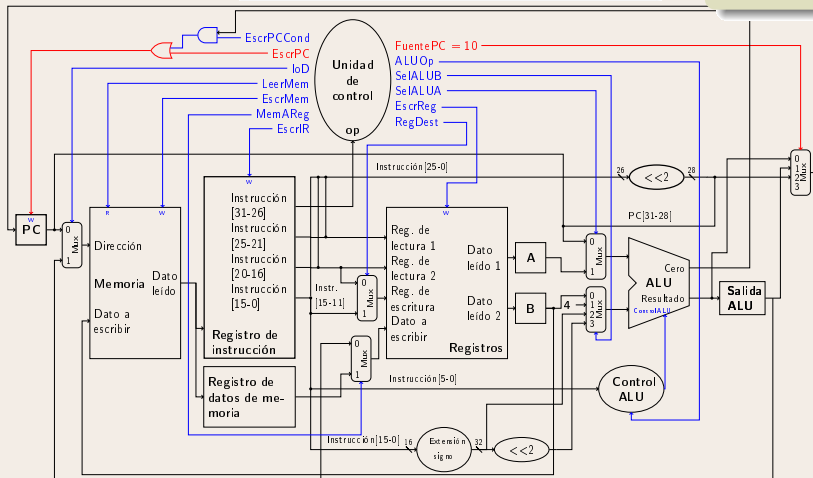
Paso 3d



Ejemplo de ejecución de `jal`

3d
FuentePC = 10
EscrPC = 1

Paso 3d



Visión completa del procesador

