Tema 4. El Procesador

FUNDAMENTOS DE LOS COMPUTADORES

Grupo de Arquitectura de Computadores (GAC)

Departamento de Electrónica e Sistemas
(Universidade da Coruña)

Índice I

Índice II

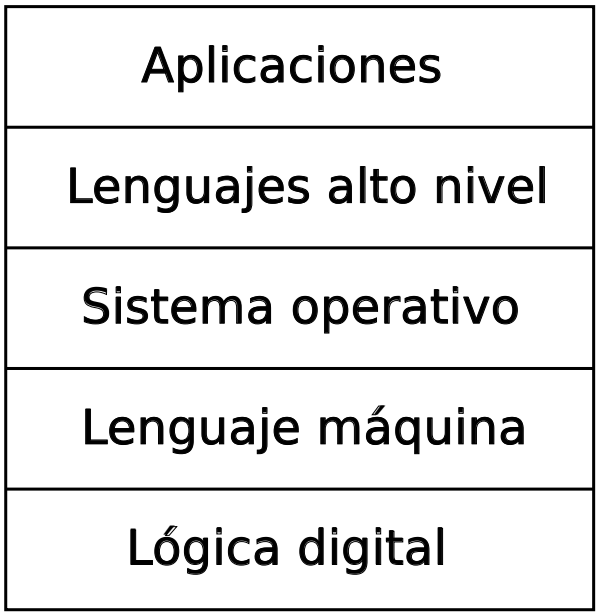
Jerarquía de niveles estructurales

Computador: Dispositivo electrónico formado por componentes y subsistemas digitales que permite el procesamiento de datos

Jerarquía de niveles estructurales

Computador: Dispositivo electrónico formado por componentes y subsistemas digitales que permite el procesamiento de datos

Jerarquía de niveles estructurales



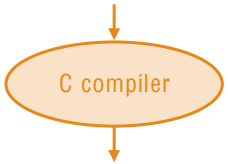
- Nivel digital: Se corresponde con la máquina física (hardware)
- Nivel instrucciones máquina: El lenguaje ensamblador es la notación simbólica del lenguaje máquina
- Nivel sistema operativo: Capa software que rodea al hardware para facilitar su utilización
- Nivel lenguajes de alto nivel: Los programas escritos en un lenguaje de alto nivel necesitan un compilador para traducirlos a lenguaje máquina

Jerarquía de niveles estructurales

Lenguaje de alto nivel

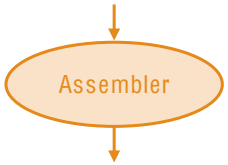
High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```



Assembly language program (for MIPS)

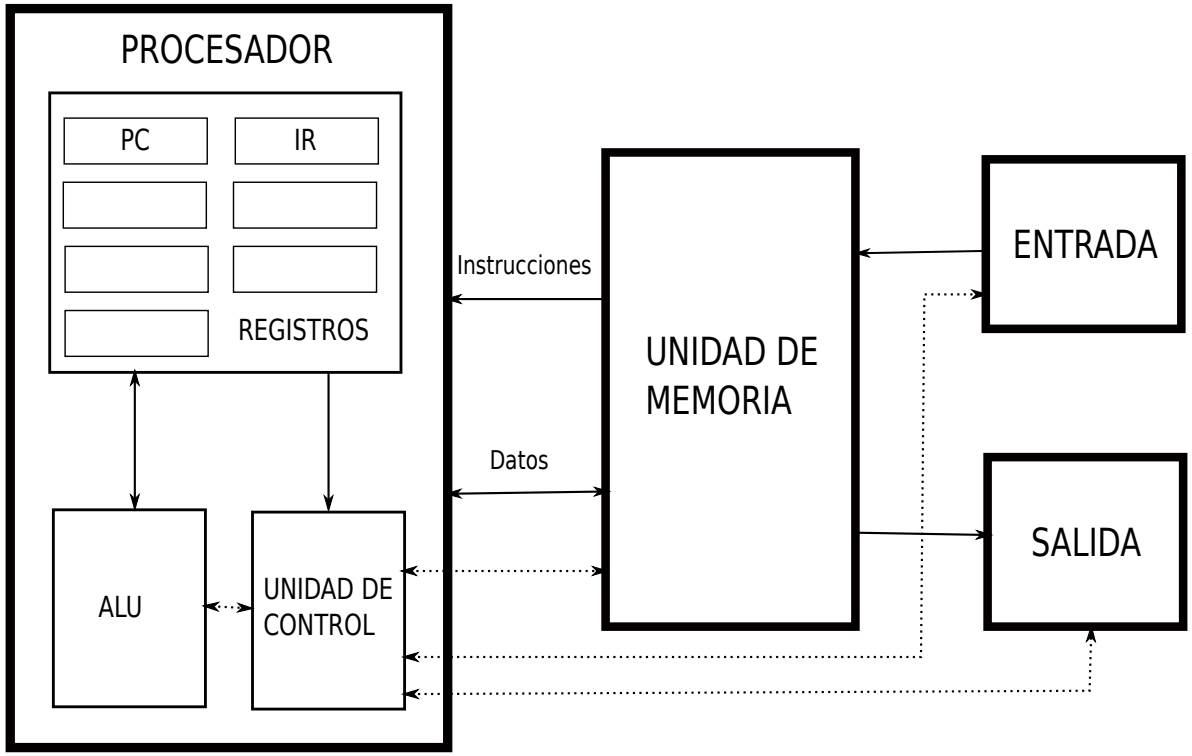
```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```



Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Jerarquía de niveles estructurales
Nivel digital



Ejecución de instrucciones

- Un computador funciona ejecutando continuamente instrucciones que operan sobre diferentes datos
- Instrucciones y datos se almacenan en memoria y se codifican empleando un número específico de bits
- El computador está diseñado para transferir bloques de información de un determinado tamaño
  - A cada uno de estos bloques se le denomina **palabra**
  - El tamaño, en número de bits, de cada palabra se denomina **ancho de palabra**
- Todos los elementos del computador (registros, buses, memorias, periféricos, ALUs) están diseñados para transferir, almacenar o procesar palabras.

Ejecución de instrucciones

- El proceso de ejecución de una instrucción sigue una secuencia de 4 pasos:
  1. Lectura o carga de la instrucción: memoria $\rightarrow$ registro de instrucción (IR, Instruction Register).
  2. Decodificación de la instrucción: La unidad de control lee el contenido del IR y decodifica la instrucción a ejecutar.
  3. Ejecución de la instrucción: Se activan las señales de control necesarias para la ejecución de la operación.
  4. Determinación de la siguiente instrucción: actualización del contador de programa (PC, Program Counter).

Ejecución de instrucciones

- El proceso de ejecución de una instrucción sigue una secuencia de 4 pasos:
  1. Lectura o carga de la instrucción: memoria $\rightarrow$ registro de instrucción (IR, Instruction Register).
  2. Decodificación de la instrucción: La unidad de control lee el contenido del IR y decodifica la instrucción a ejecutar.
  3. Ejecución de la instrucción: Se activan las señales de control necesarias para la ejecución de la operación.
  4. Determinación de la siguiente instrucción: actualización del contador de programa (PC, Program Counter).

Ejecución de instrucciones

- El proceso de ejecución de una instrucción sigue una secuencia de 4 pasos:
  1. Lectura o carga de la instrucción: memoria $\rightarrow$ registro de instrucción (IR, Instruction Register).
  2. Decodificación de la instrucción: La unidad de control lee el contenido del IR y decodifica la instrucción a ejecutar.
  3. Ejecución de la instrucción: Se activan las señales de control necesarias para la ejecución de la operación.
  4. Determinación de la siguiente instrucción: actualización del contador de programa (PC, Program Counter).

Ejecución de instrucciones

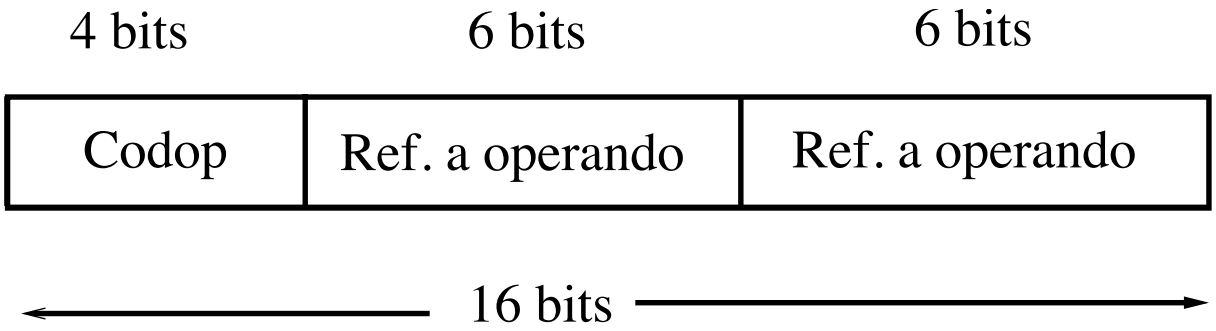
- El proceso de ejecución de una instrucción sigue una secuencia de 4 pasos:
  1. Lectura o carga de la instrucción: memoria $\rightarrow$ registro de instrucción (IR, Instruction Register).
  2. Decodificación de la instrucción: La unidad de control lee el contenido del IR y decodifica la instrucción a ejecutar.
  3. Ejecución de la instrucción: Se activan las señales de control necesarias para la ejecución de la operación.
  4. Determinación de la siguiente instrucción: actualización del contador de programa (PC, Program Counter).

- El funcionamiento de la CPU está determinado por las instrucciones que ejecuta
- El conjunto de instrucciones distintas que puede ejecutar se denomina **repertorio de instrucciones**
- El repertorio de instrucciones condiciona la implementación
- Cada instrucción debe contener toda la información necesaria para su ejecución

- Los elementos constitutivos de una instrucción máquina son:
  - Código de operación
  - Referencia a operandos fuente: uno o más
  - Referencia al operando resultado: si produce alguno
  - Referencia a la siguiente instrucción: en la mayoría de los casos es implícito y se refiere a la siguiente instrucción de la lista. En caso contrario, se suministrará la dirección de memoria

- Los elementos constitutivos de una instrucción máquina son:
  - Código de operación
  - Referencia a operandos fuente: uno o más
  - Referencia al operando resultado: si produce alguno
  - Referencia a la siguiente instrucción: en la mayoría de los casos es implícito y se refiere a la siguiente instrucción de la lista. En caso contrario, se suministrará la dirección de memoria
- Los operandos fuente y resultado estarán en alguna de las siguientes localizaciones:
  - Memoria: deberá indicarse la dirección
  - Registro de la CPU: identificados por un número único
  - Dispositivo de E/S: módulo y dispositivo E/S o dirección de memoria en caso de E/S asignada en memoria

- La instrucción se divide en campos correspondientes a los elementos constitutivos de la misma (código de operación, operandos, etc.)
- La división en campos y bits se denomina **formato de instrucción**.



- La mayoría de los repertorios de instrucciones emplean más de un formato

- Los aspectos más importantes en el diseño del repertorio de instrucciones son:
  - Repertorio de operaciones: Cuántas y qué operaciones considerar, y cuán complejas deben ser.
  - Tipos de datos: Los distintos tipos de datos con los que se efectúan operaciones.
  - Formatos de instrucciones: Longitud de la instrucción (en bits), tamaño de los distintos campos, etc.
  - Registros: Número de registros de la CPU que pueden ser referenciados por instrucciones y su uso.
  - Direccionamiento: El modo o modos de direccionamiento mediante los cuales puede especificarse la dirección de un operando.

Modelo de ejecución

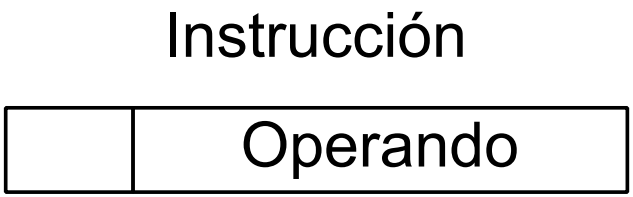
- El modelo de ejecución especifica el dispositivo (memoria, registro, etc.) que almacena los operandos de las instrucciones.
- Una misma arquitectura suele dar soporte a varios modelos de ejecución.
- Los modelos de ejecución posibles son:
  - Modelo de pila: POP y PUSH.
  - Modelo registro-registro: el modo más rápido. Se necesita transferir los datos.
  - Modelo registro-memoria: un operando en registro, otro en memoria. El resultado se almacena en un registro.
  - Modelo memoria-memoria: No se necesita transferir los datos. La ejecución requiere un alto número de ciclos.
- Los modelos soportados dependen fuertemente de la arquitectura.
  - P. ej. una ALU exclusivamente conectada con registros quedará limitada al modo registro-registro.

Modos de Direccionamiento

- Son los algoritmos empleados por el procesador para calcular las direcciones de las instrucciones y datos.
- La dirección real de memoria especificada por el modo de direccionamiento se denomina *dirección efectiva*.
- Modos de direccionamiento:
  - Direccionamiento inmediato.
  - Direccionamiento directo.
  - Direccionamiento indirecto.
  - Direccionamiento de registros.
  - Direccionamiento indirecto con registro.
  - Direccionamiento con desplazamiento:
    - Desplazamiento relativo.
    - Desplazamiento con registro-base.
    - Indexado.

Modos de Direccionamiento

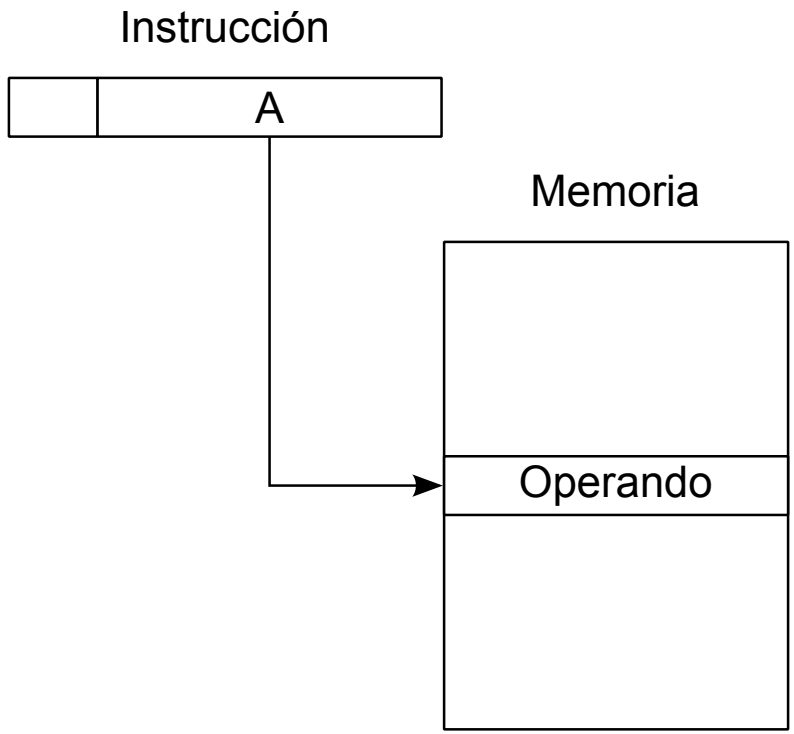
Direccionamiento inmediato



- El operando es una constante cuyo valor se almacena en el campo operando de la instrucción.
- Ventaja: no se requiere una referencia a memoria para obtener el operando.
- Desventaja: el tamaño del número está limitado por la longitud del campo de direcciones.

Modos de direccionamiento

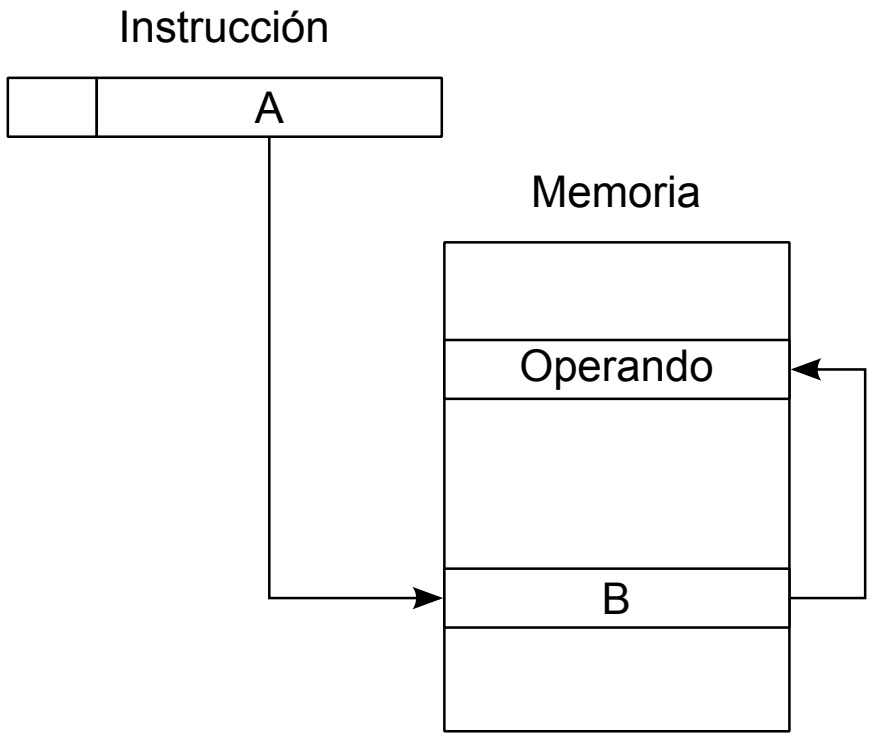
Direccionamiento directo



- El operando se encuentra almacenado en memoria en la dirección indicada por el campo operando.
- Limitación: proporciona un espacio de direcciones reducido.

Modos de direccionamiento

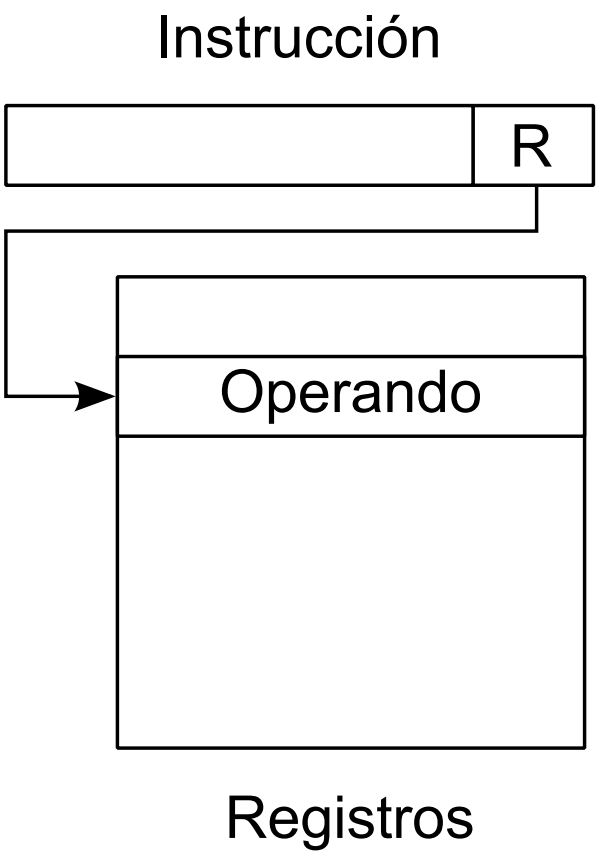
Direccionamiento indirecto



- El campo operando de la instrucción contiene la dirección a una posición de memoria que a su vez contiene la dirección del operando deseado.
- Desventaja: la ejecución de una instrucción requiere dos referencias a memoria principal.

Modos de direccionamiento

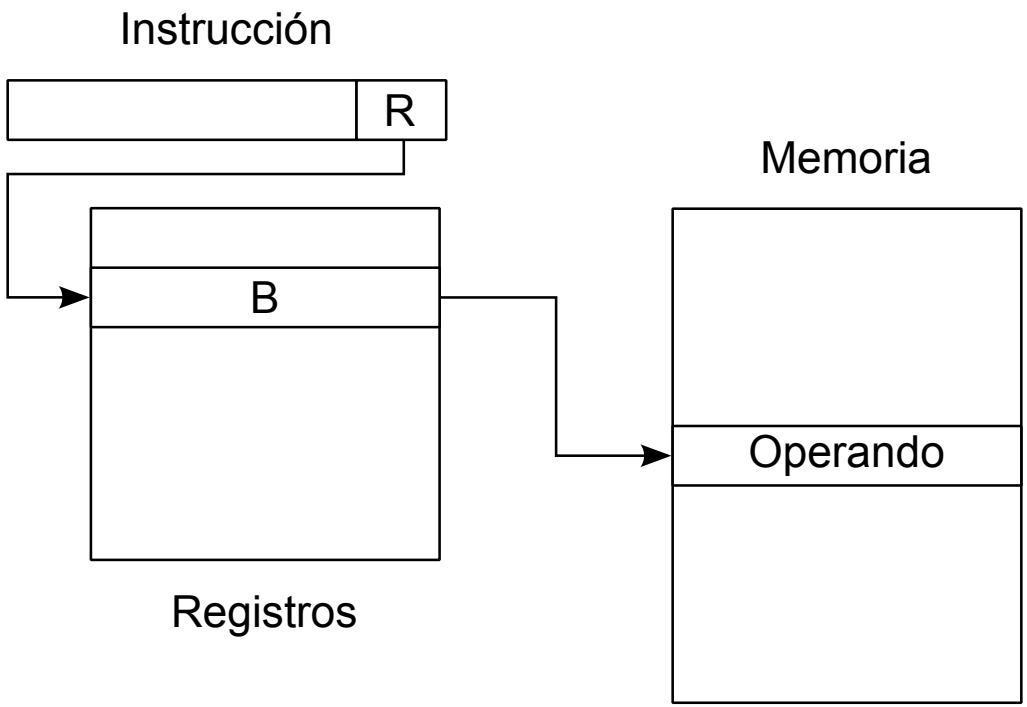
Direccionamiento de registros



- El operando referenciado se encuentra en un registro.
- El número de registro se especifica en el campo operando de la instrucción.
- Ventajas:
  - Sólo es necesario un campo pequeño de direcciones en la instrucción.
  - El tiempo de acceso a un registro es mucho menor que a memoria.
- Desventaja: espacio de direcciones muy limitado.

Modos de direccionamiento

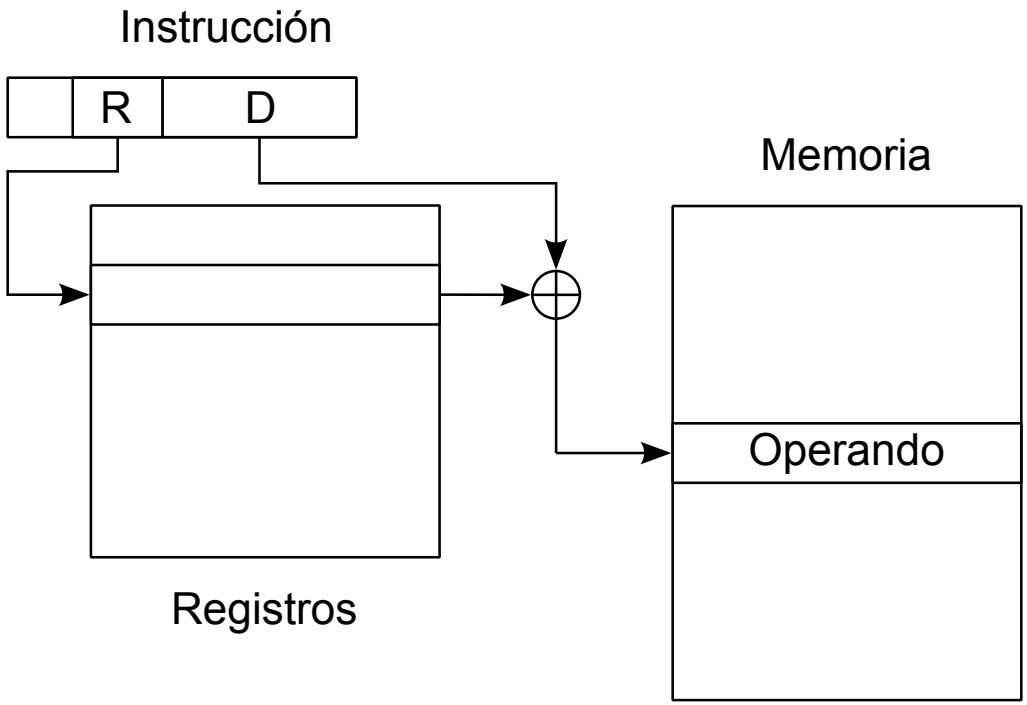
Direccionamiento indirecto con registro



- El operando de la instrucción contiene un número de registro que contiene la dirección de memoria del operando deseado.
- Ventaja: Emplea una referencia a memoria menos que el direccionamiento indirecto.

Modos de direccionamiento

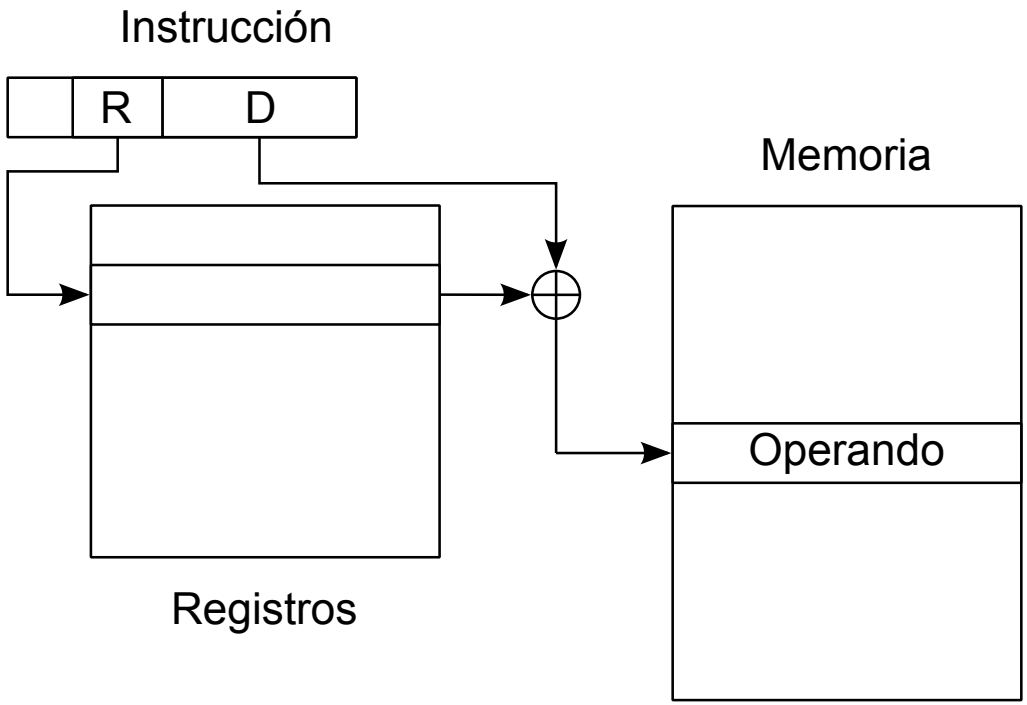
Direccionamiento con desplazamiento



- El campo operando contiene una dirección relativa o desplazamiento (D).
- La instrucción especifica otra dirección (usualmente un registro) donde se almacena información adicional de direccionamiento.
- La dirección efectiva se calcula como (D+R).

Modos de direccionamiento

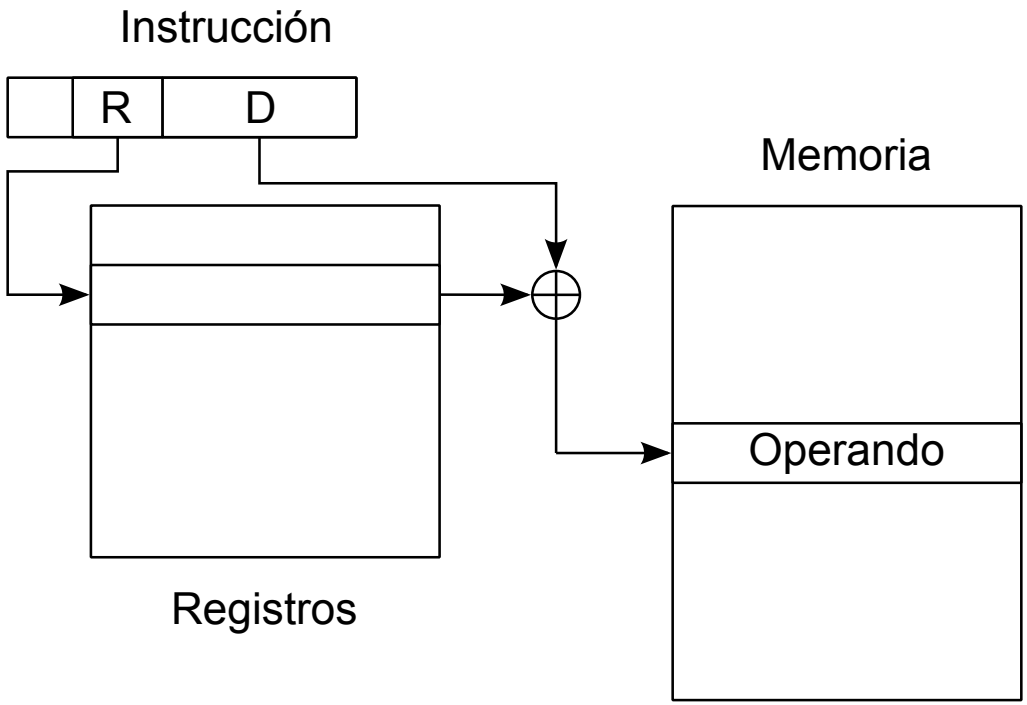
Direccionamiento con desplazamiento



- Desplazamiento relativo:
  - El registro referenciado (implícitamente) es el contador de programa.
  - La dirección efectiva es por tanto un desplazamiento relativo a la dirección de la instrucción.

Modos de direccionamiento

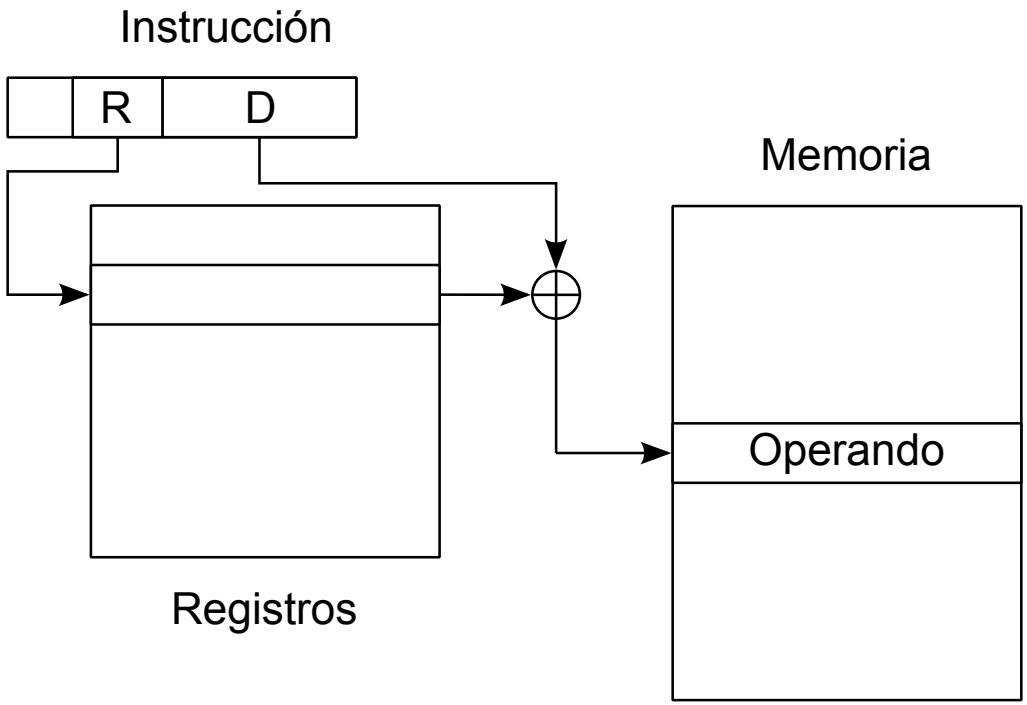
Direccionamiento con desplazamiento



- Direccionamiento con registro-base:
  - El registro referenciado (implícita o explícitamente) contiene una dirección de memoria.
  - El campo de dirección contiene un desplazamiento positivo o negativo desde esa dirección.

Modos de direccionamiento

Direccionamiento con desplazamiento



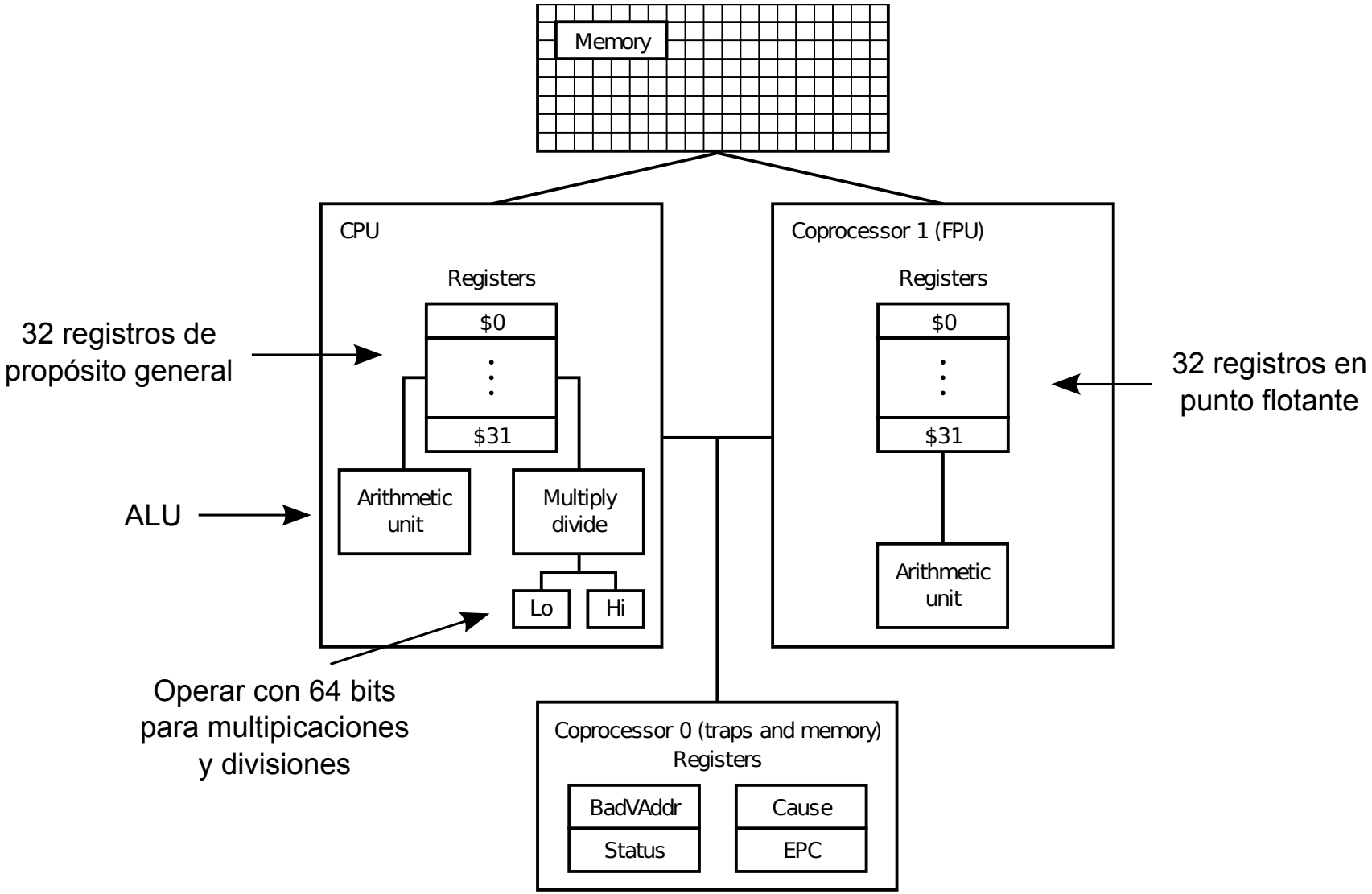
- Indexado:
  - El operando referencia una dirección de memoria.
  - El registro referenciado contiene un desplazamiento positivo desde esa posición.

Operaciones del repertorio de instrucciones

- Se pueden categorizar las operaciones de un repertorio de instrucciones en los siguientes tipos:
  - Transferencias de datos: *load* y *store*
  - Aritméticas
  - Lógicas
  - De E/S
  - De control de flujo:
    - Instrucciones de bifurcación o salto
    - Instrucciones de salto condicional
    - Instrucciones de llamada a subrutina
  - De control del sistema

- Dos filosofías a la hora de diseñar un repertorio de instrucciones:
  - *CISC (Complex Instruction Set Computer)*
    - Juego de instrucciones muy rico
    - Modos de direccionamiento potentes y complejos
    - Menos instrucciones de ejecución compleja
  - *RISC (Reduced Instruction Set Computer)*
    - Número relativamente pequeño de instrucciones disponibles
    - Sencillas y con pocos modos de direccionamiento
    - Más instrucciones de ejecución muy eficiente
- Estudiaremos el repertorio del MIPS (*Microprocessor without Interlocked Pipeline Stages*):
  - Ejemplo típico de diseño RISC.
  - Usado por NEC, Nintendo, SGI, Sony, ...

Arquitectura del MIPS R2000



Arquitectura del MIPS R2000

Registros

Nombre	Número	Uso	Preservado en llamada
`$zero`	0	Valor constante 0	n.a.
$at	1	Reservado para el ensamblador	n.a.
`$v0-$v1`	2-3	Valores para resultados y evaluación de expresiones	no
`$a0-$a3`	4-7	Paso de parámetros	sí
`$t0-$t7`	8-15	Registros temporales	no
`$s0-$s7`	16-23	Registros que deben preservarse	sí
`$t8-$t9`	24-25	Registros temporales	no
`$k0-$k1`	26-27	Reservado para núcleo de SO	n.a.
`$gp`	28	Puntero global	sí
`$sp`	29	Puntero de pila	sí
`$fp`	30	Puntero de bloque de activación	sí
`$ra`	31	Dirección de retorno	sí

Arquitectura del MIPS R2000

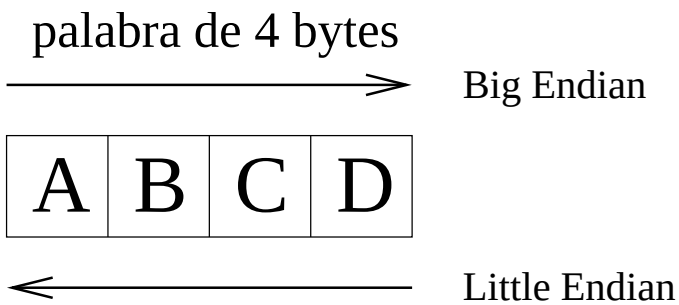
Características

- Longitud de palabra fija de 32 bits
- Direcciona bytes individuales, para indicar la dirección de una palabra en memoria hay que indicar la dirección del primer byte
  - 2 palabras consecutivas estarán separadas en 4 unidades
  - En las operaciones de transferencia entre memoria y registro las direcciones de memoria han de ser múltiplos de 4: **restricción de alineamiento**
  - Las direcciones múltiplos de 4 se llaman **direcciones alineadas**

Arquitectura del MIPS R2000

Características

- Representación de los bytes en una palabra
  - **Big-endian**: la dirección del dato es el byte más significativo del mismo
  - **Little-endian**: la dirección del dato es el byte menos significativo del mismo
  - **Bi-endian**: configurable (MIPS)

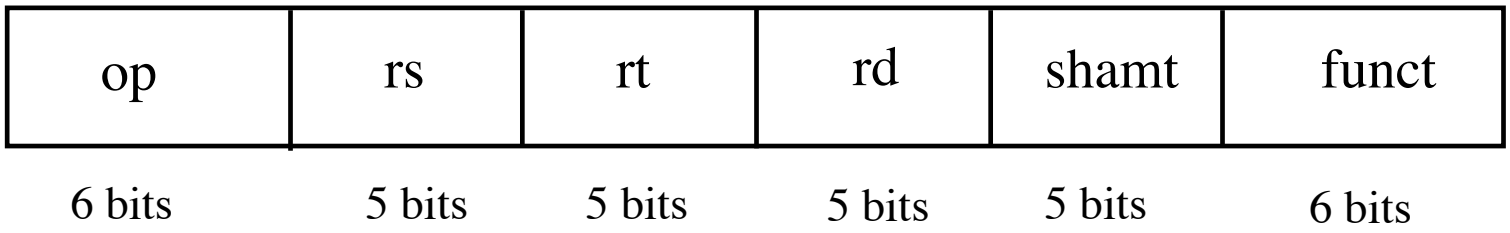


Tipos de instrucciones MIPS

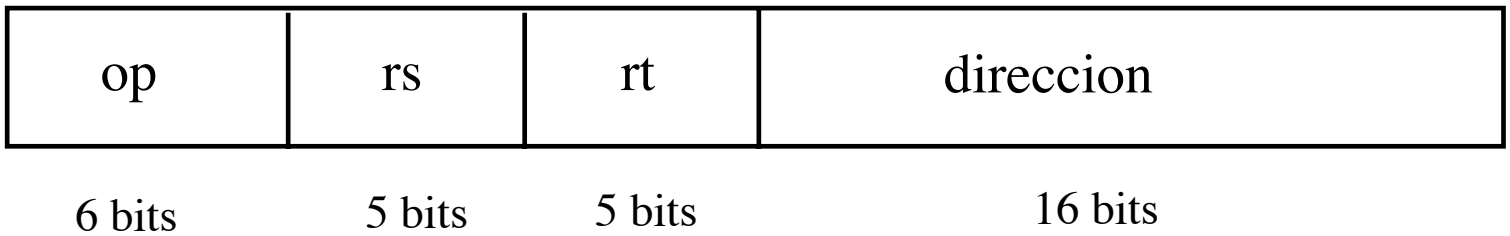
- Tipos de instrucciones soportados:
  - Transferencia de datos
  - Aritméticas y lógicas
  - De control de flujo:
    - Salto condicional
    - Bifurcación
- Características:
  - Longitud de todas las instrucciones fija de 32 bits.
  - Los operandos de las operaciones aritméticas son siempre registros
  - El acceso a memoria se hace a través de operaciones de carga/almacenamiento (transferencia de datos).

Formatos de las instrucciones MIPS

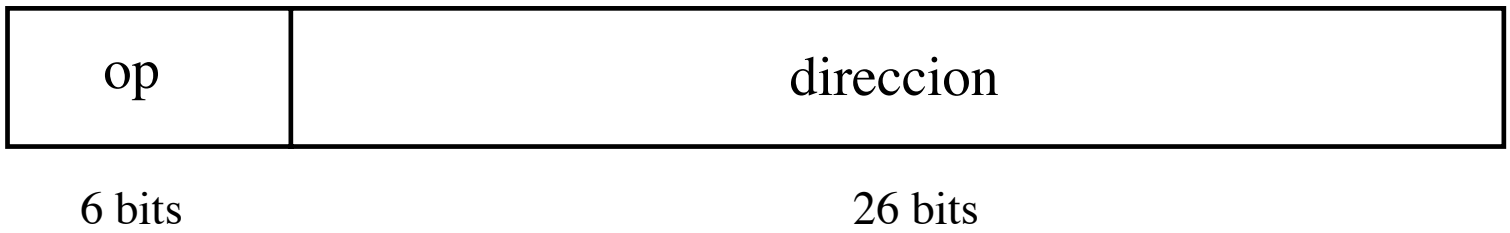
Tipo - R



Tipo - I



Tipo - J



Formatos de las instrucciones MIPS

Instrucciones aritmético-lógicas:

- add, sub, or, and, ...
- Formato tipo R
- Direccionamiento de registro

add $7,$3,$6

0	3	6	7	0	32

000000	00011	00110	00111	00000	100000
31	25	20	15	10	5 0

sub $7,$3,$6

0	3	6	7	0	34

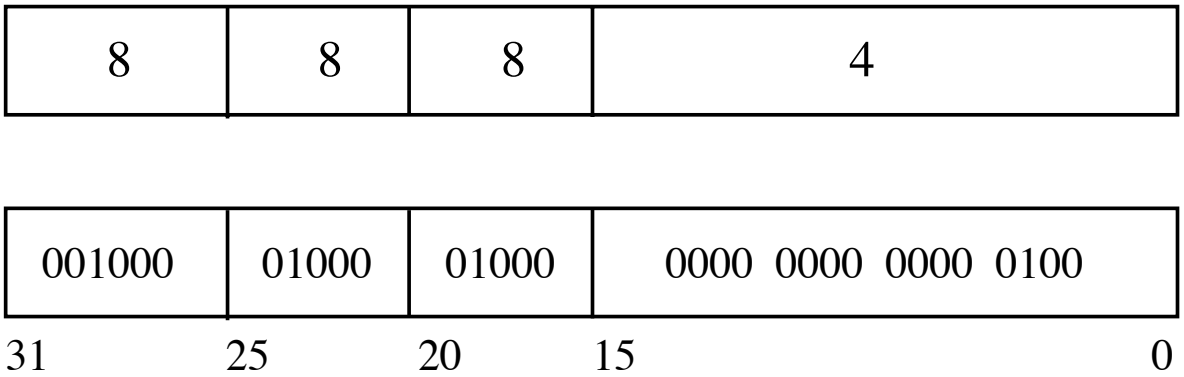
000000	00011	00110	00111	00000	100010
31	25	20	15	10	5 0

Formatos de las instrucciones MIPS

Operaciones aritméticas con operandos inmediatos:

- addi, andi, ori, ...
- Formato tipo I
- Direccionamiento de registro + direccionamiento inmediato

addi $8,$8,4

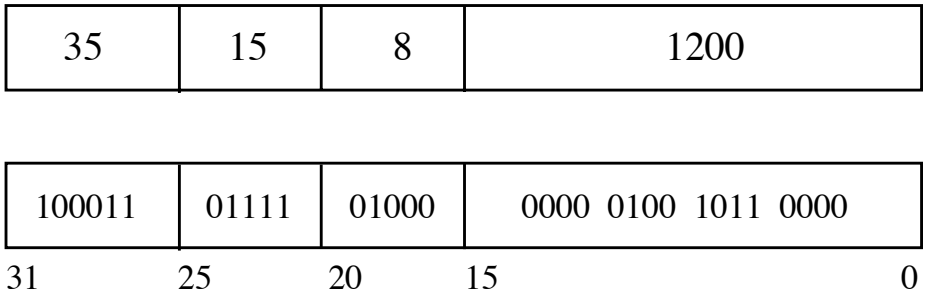


Formatos de las instrucciones MIPS

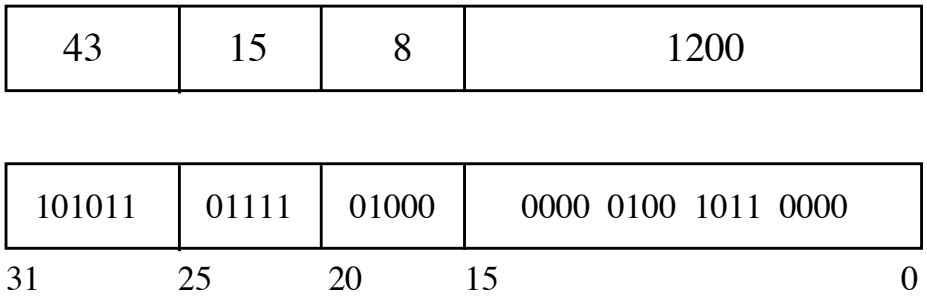
Operaciones de transferencia:

- lw (*load word*) y sw (*store word*)
- Tipo I
- Direccionamiento con desplazamiento

lw $8,1200($15)



sw $8,1200($15)

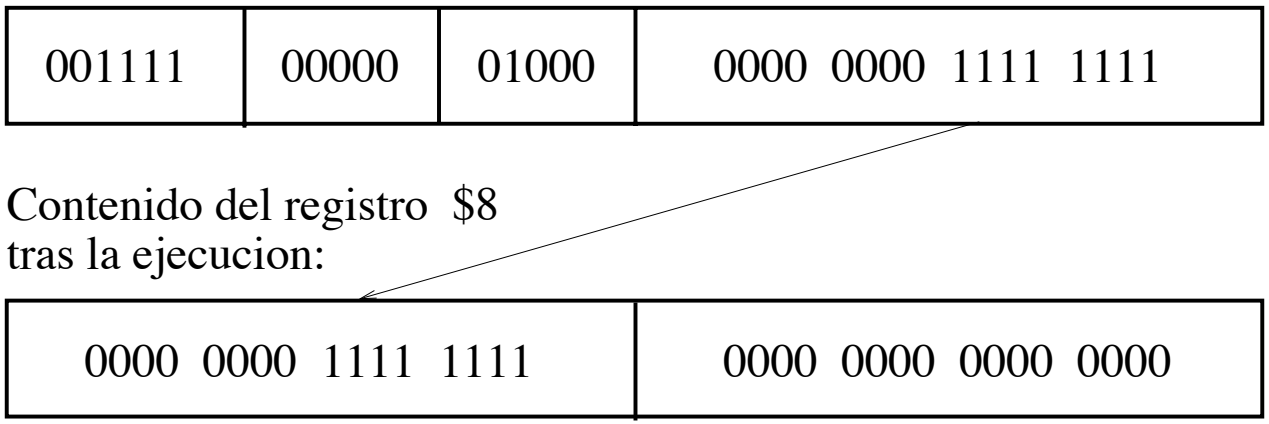


Formatos de las instrucciones MIPS

Operaciones de transferencia con operandos inmediatos:

- lui (*load upper inmediate*)
- Tipo I
- Direccionamiento inmediato

Version en lenguaje maquina: lui $8,255



($rt=inm«16)

Formatos de las instrucciones MIPS

Instrucciones de salto condicional:

- beq (*branch if equal*) y bne (*branch if not equal*)
- Tipo I
- Direccionamiento con desplazamiento relativo
- `beq $rs,$rt,despl` salta a la instrucción que se encuentra en la dirección de memoria `(PC+4)+4*despl` si `$rs` ==`$rt`.
- El ensamblador permite el uso de etiquetas para indicar la dirección de salto

Formatos de las instrucciones MIPS

beq $19,$20,L1

4	19	20	L1

000100	10011	10100	0001 1000 0011 1101

31 25 20 15 0

bne $19,$20,L1

5	19	20	L1

000101	10011	10100	0001 1000 0011 1101

31 25 20 15 0

Formatos de las instrucciones MIPS

- Para hacer comparaciones del tipo *menor que* tenemos la instrucción lógica **slt**
- `slt $rd,$rs,$rt`: `$rd=($rs<$rt)`
- `slti $rd,$rs,inm`: `$rd=($rs<inm)`

slt $8,$19,$20

0	19	20	8	0	42

slti $8,$19,10

10	19	8	10

- `slt`: Tipo R, direccionamiento de registro
- `slti`: Tipo I, direccionamiento de registro + inmediato

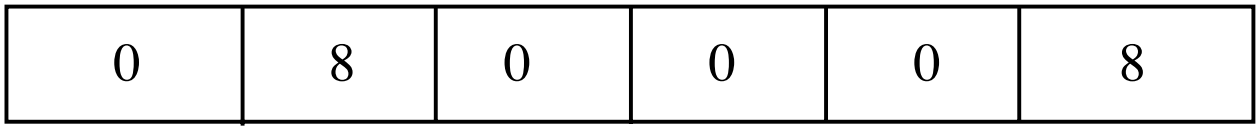
Formatos de las instrucciones MIPS

Instrucciones de bifurcación:

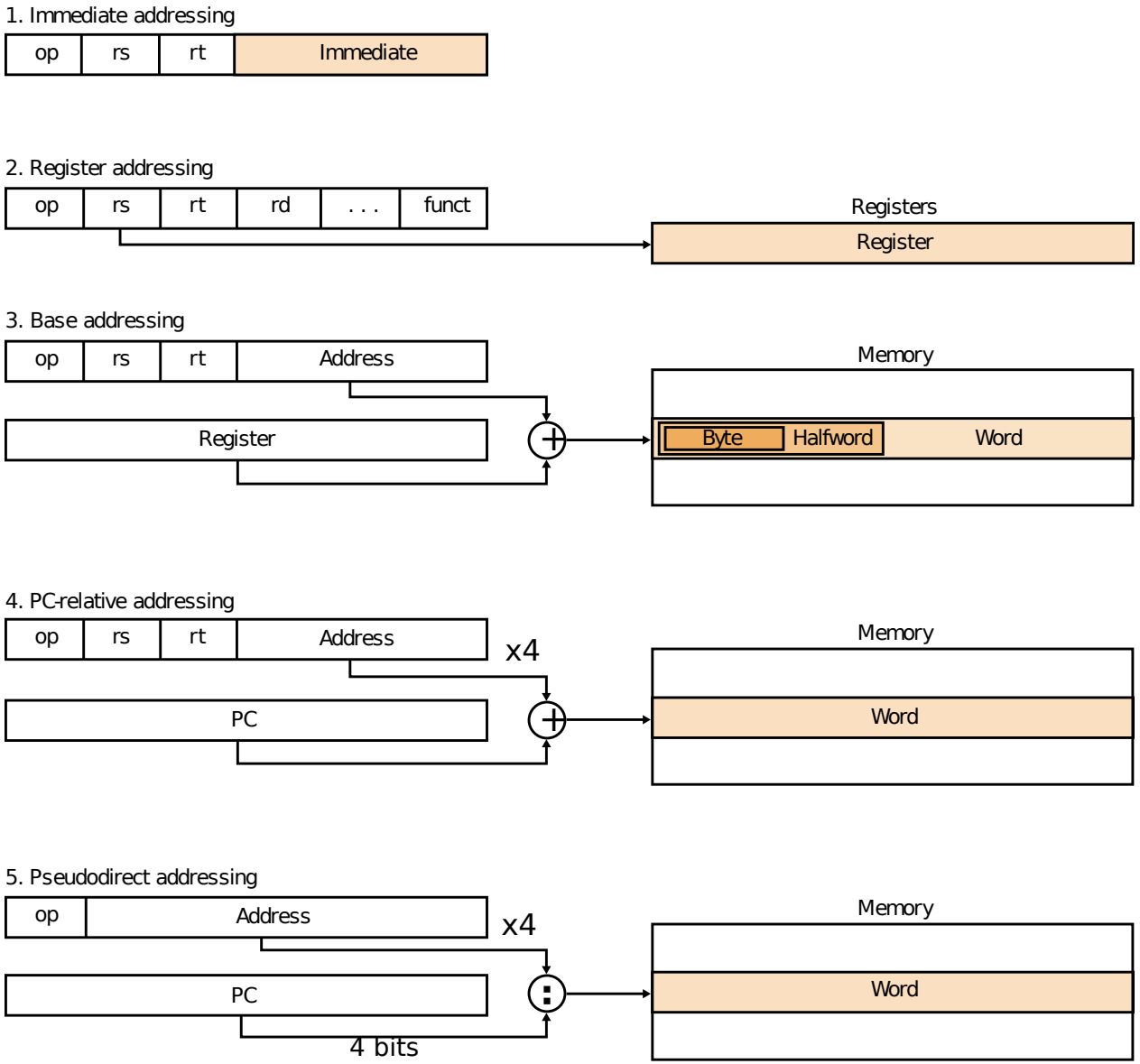
- `j` (*jump*): Tipo J, direccionamiento pseudodirecto
  - $PC \leftarrow (PC + 4)_{31-28} IR_{25-0} 00$
  - El ensamblador permite el uso de etiquetas
- `jr` (*jump register*): Tipo R, direccionamiento indirecto con registro.

j L1

jr $8



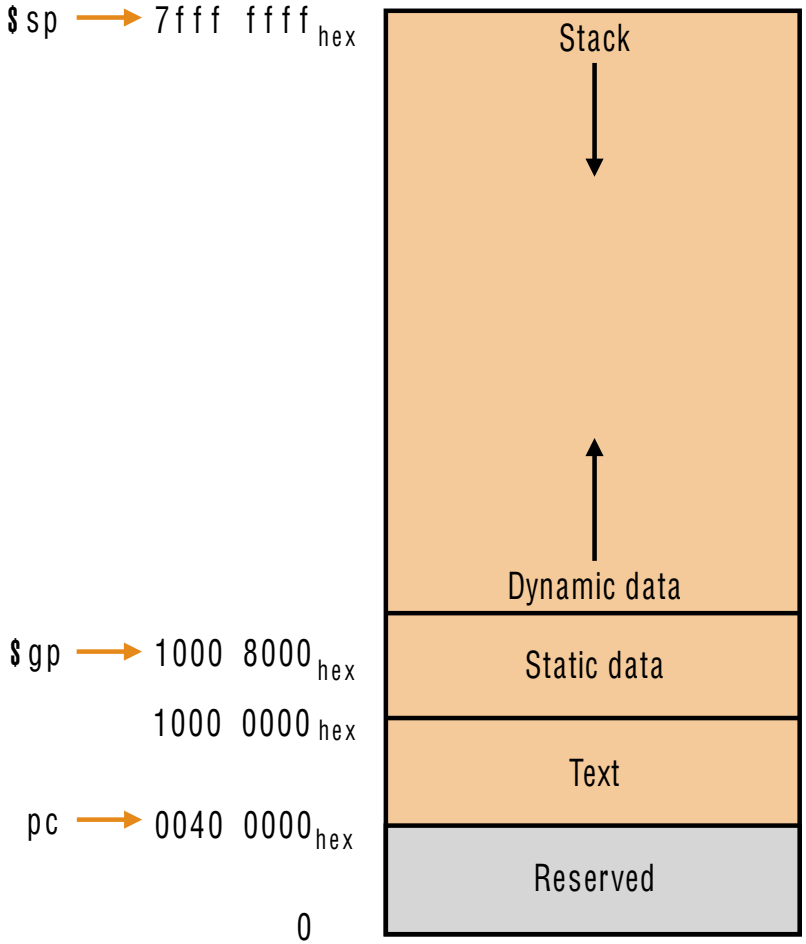
Modos de direccionamiento del MIPS



Llamadas a subrutinas

- Subrutina: herramienta para la estructuración de programas.
  - Aumenta la legibilidad del código.
  - Permite su reutilización.
- Pasos para la llamada a una subrutina:
  - Situar los parámetros en un lugar donde la subrutina pueda acceder a ellos.
  - Transferir el control a la subrutina.
  - Adquirir los recursos de almacenamiento necesarios para el procedimiento.
  - Realizar la tarea deseada.
  - Situar el valor del resultado en un lugar donde el programa que lo ha llamado pueda acceder a él.
  - Retornar el control al punto de origen.

Llamadas a subrutina
Distribución de la memoria en MIPS



- Generalmente, el invocador de la subrutina sitúa los parámetros en la pila.
- La subrutina colocará en la pila la información a guardar temporalmente.
- Al finalizar, liberará dicho espacio y dejará en la pila el resultado de la subrutina.
- Por último, el invocador recupera de la pila el resultado.

Llamadas a subrutina

Paso de parámetros a través de la pila

- Pila: espacio de memoria con estructura LIFO.
- El registro $sp apunta a la dirección más recientemente utilizada.
- Crece de las direcciones de memoria superiores hacia las inferiores.
- Ejemplo de utilización
  - *Push* salvando dos registros en la pila:

```
addi $sp, $sp, -8
sw $v0, 0($sp)
sw $v1, 4($sp)
```

  - *Pop* recuperando los datos de la pila a los registros:

```
lw $v0, 0($sp)
lw $v1, 4($sp)
addi $sp, $sp, 8
```

Llamadas a subrutina

Paso de parámetros a través de registros

- El acceso a los registros es más rápido que a memoria.
- El convenio de llamadas en MIPS se apoya en los registros para evitar accesos a pila:
  - `$a0-$a3`: paso de argumentos.
  - `$v0-$v1`: retorno de valores.
  - `$ra`: dirección de retorno.
- Si se necesita pasar o devolver más argumentos, será necesario utilizar la pila.

Llamadas a subrutina

Instrucción de salto

jal direccion_subrutina

- `jal` (*jump-and-link*): Salta a una dirección y simultáneamente salva la dirección de retorno (`$ra = PC+4`).
- Tipo J.
- Direccionamiento pseudodirecto. El ensamblador permite el uso de etiquetas.
- El salto de retorno se hará a través de `jr $ra`.

Llamadas a subrutina

Instrucción de salto

jal direccion_subrutina

- Uso:
  - El invocador pone los parámetros en `$a0-$a3` y llama a `jal`.
  - La subrutina realiza los cálculos, pone los resultados en `$v0-$v1` y devuelve el control.
  - Si se necesita pasar más de cuatro argumentos o devolver más de dos valores se utilizará la pila.

Llamadas a subrutina

Salvaguarda de registros

- Si la subrutina modifica los registros utilizados por la rutina invocadora, los valores deberán ser guardados y restaurados antes y después, respectivamente, de la ejecución de la subrutina.
- Dos convenios estándar:
  - Guardar invocador (*caller save*).
  - Guardar invocado (*callee save*).
- Para evitar el guardar y restaurar valores irrelevantes se usan las siguientes convenciones:
  - Registros temporales (`$t0`-`$t9`): no son preservados por el invocado. Es responsabilidad del invocador preservar su valor si fuera necesario.
  - Registros salvados (`$s0`-`$s7`): si el invocado los usa deberá salvar previamente su valor.

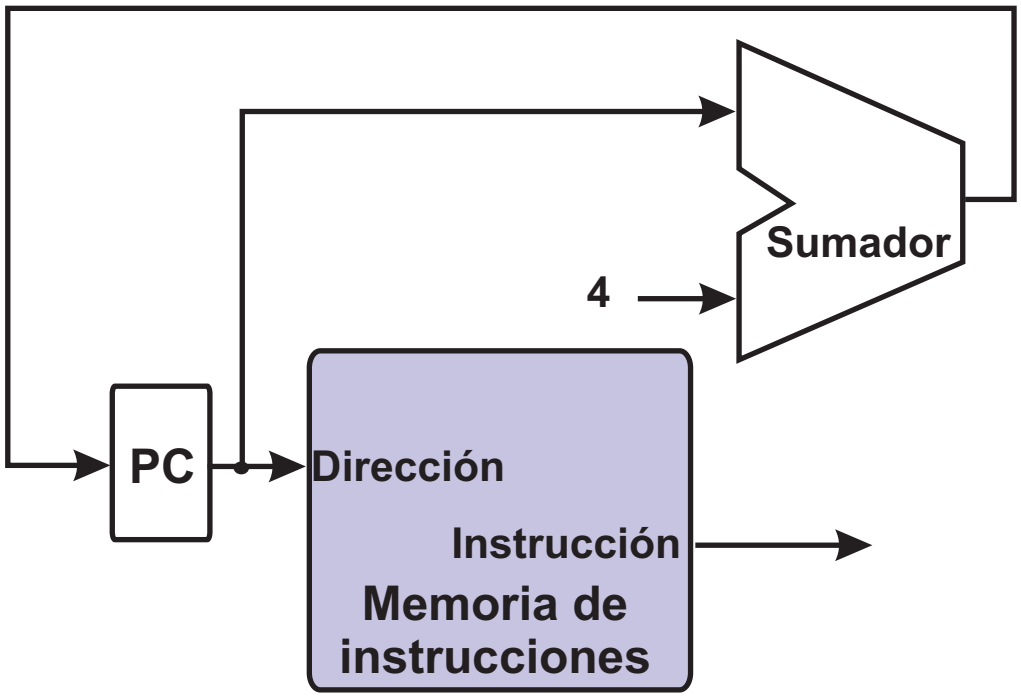
- La Unidad Central de Proceso (CPU) es la encargada de la ejecución de las instrucciones.
- La CPU se divide en dos bloques fundamentales:
  - Camino de datos: realiza las operaciones requeridas por las instrucciones máquina. Contiene:
    - Unidad aritmético-lógica.
    - Banco de registros.
    - Registros especiales: IR, PC, etc.
    - Buses internos: caminos de conexión entre los elementos de la CPU.
  - Unidad de control: gestiona el secuenciamiento de las operaciones ejecutadas en el camino de datos.

- La ejecución de una instrucción es un proceso dividido en cinco pasos:
  - **Captar la instrucción**: el procesador debe leer la instrucción de memoria.
  - **Interpretar la instrucción**: la instrucción debe decodificarse para determinar qué acción es necesaria.
  - **Captar datos**: la ejecución puede exigir leer datos de la memoria o de un módulo de E/S.
  - **Procesar datos**: la ejecución de una instrucción puede exigir llevar a cabo alguna operación aritmética o lógica.
  - **Escribir datos**: los resultados de la ejecución pueden tener que ser escritos en la memoria o en un módulo de E/S.

Introducción

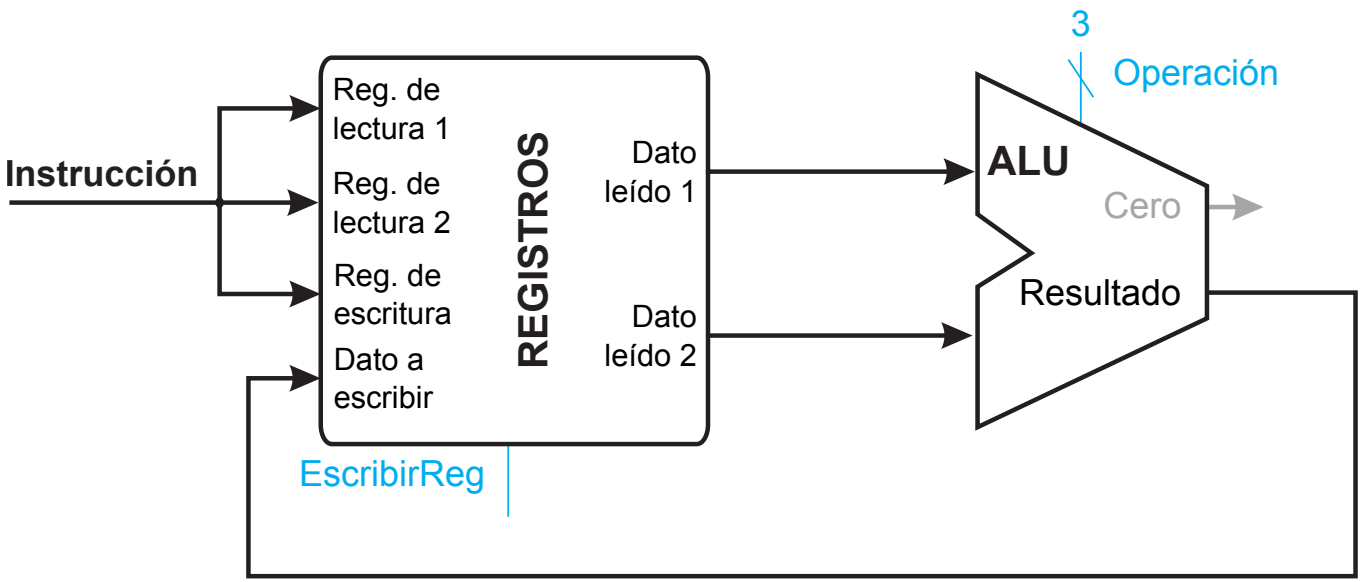
- Estudiaremos un diseño sencillo de una CPU (camino de datos y unidad de control) capaz de ejecutar el siguiente subconjunto del repertorio de instrucciones MIPS:
  - Instrucciones de transferencia de datos: `lw` y `sw`.
  - Instrucciones aritmético-lógicas: `add`, `sub`, `and`, `or` y `slt`.
  - Instrucciones de control de flujo: `beq` y `j`.
- Comenzaremos por la construcción del camino de datos.
  - Utilizaremos módulos digitales combinacionales y secuenciales
  - Para las módulos secuenciales asumiremos sincronización por flancos

Carga de la instrucción



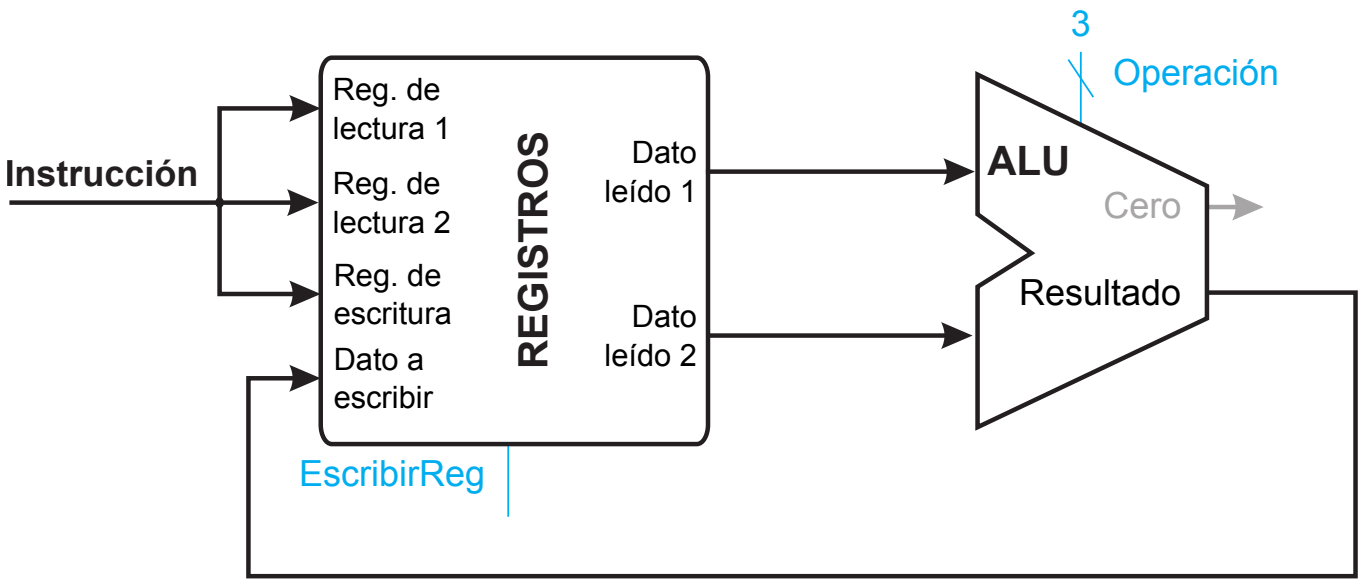
- La dirección de memoria en la que se localiza la instrucción a cargar la indica el PC.
- La siguiente instrucción se calcula incrementando el PC en 4 bytes.

Instrucciones aritmético-lógicas



- Instrucciones tipo R que involucran tres registros: dos de lectura y uno de escritura.
- La operación se realiza a través de la ALU.
- Ejemplo típico: `add $t1, $t2, $t3`

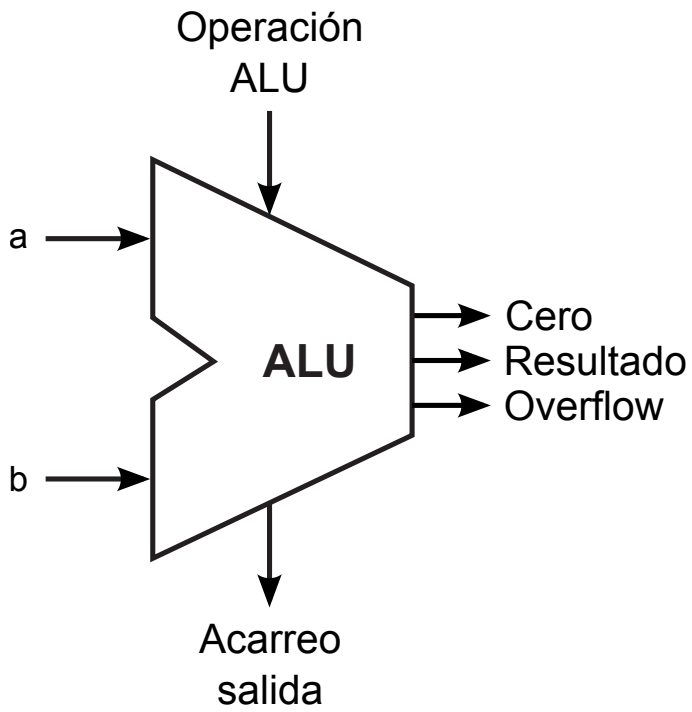
Instrucciones aritmético-lógicas



- Banco de registros:
  - Registros de 32 bits.
  - Dos puertos de lectura y uno de escritura.
  - Para escribir es necesario activar la señal `EscribirReg`.
  - Entradas (direcciones a leer/escribir): 5 bits.
  - Salidas: 32 bits (tamaño de palabra).

Instrucciones aritmético-lógicas

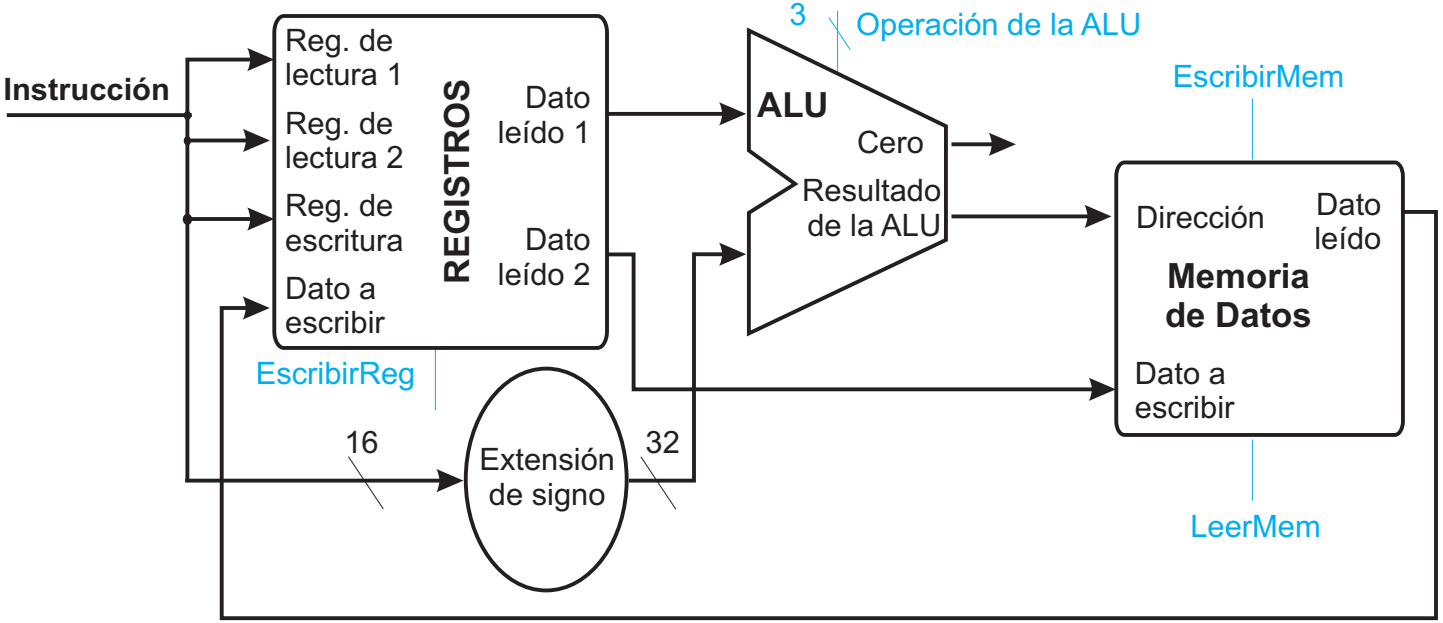
Unidad aritmético-lógica



Línea de control	Función
000	and
001	or
010	suma
110	resta
111	slt

La implementación de esta ALU para datos de 4 bits se estudió en el ejercicio 10 del boletín de combinacional

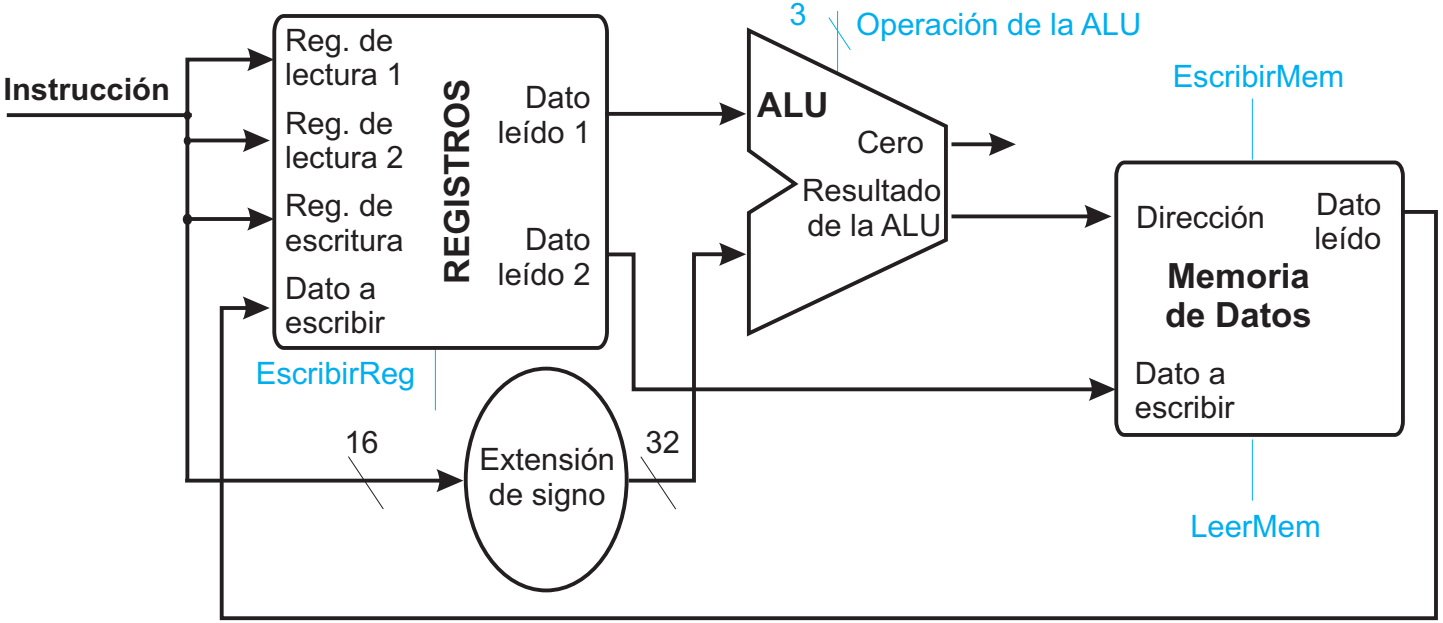
Instrucciones de transferencia de datos



- Instrucciones tipo I.
- Modo de direccionamiento: registro-base
- Ejemplos típicos:

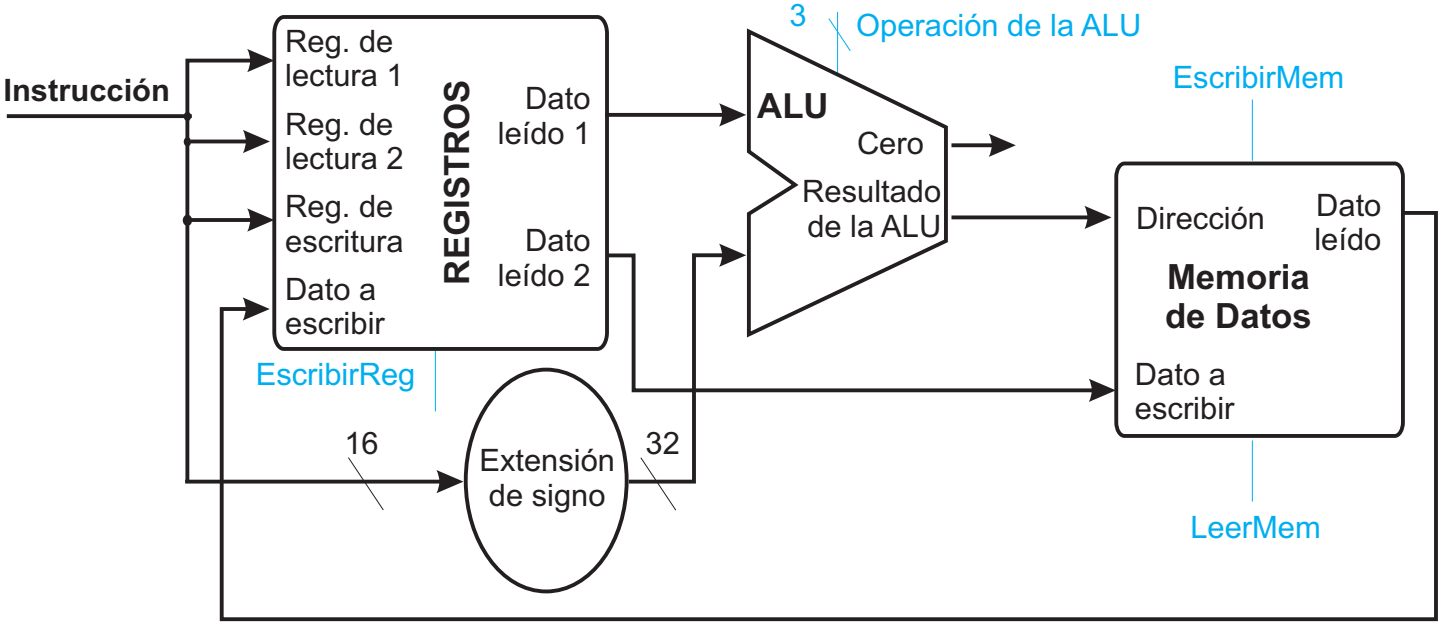
```
lw $t1, desp1($t2)
sw $t1, desp1($t2)
```

Instrucciones de transferencia de datos



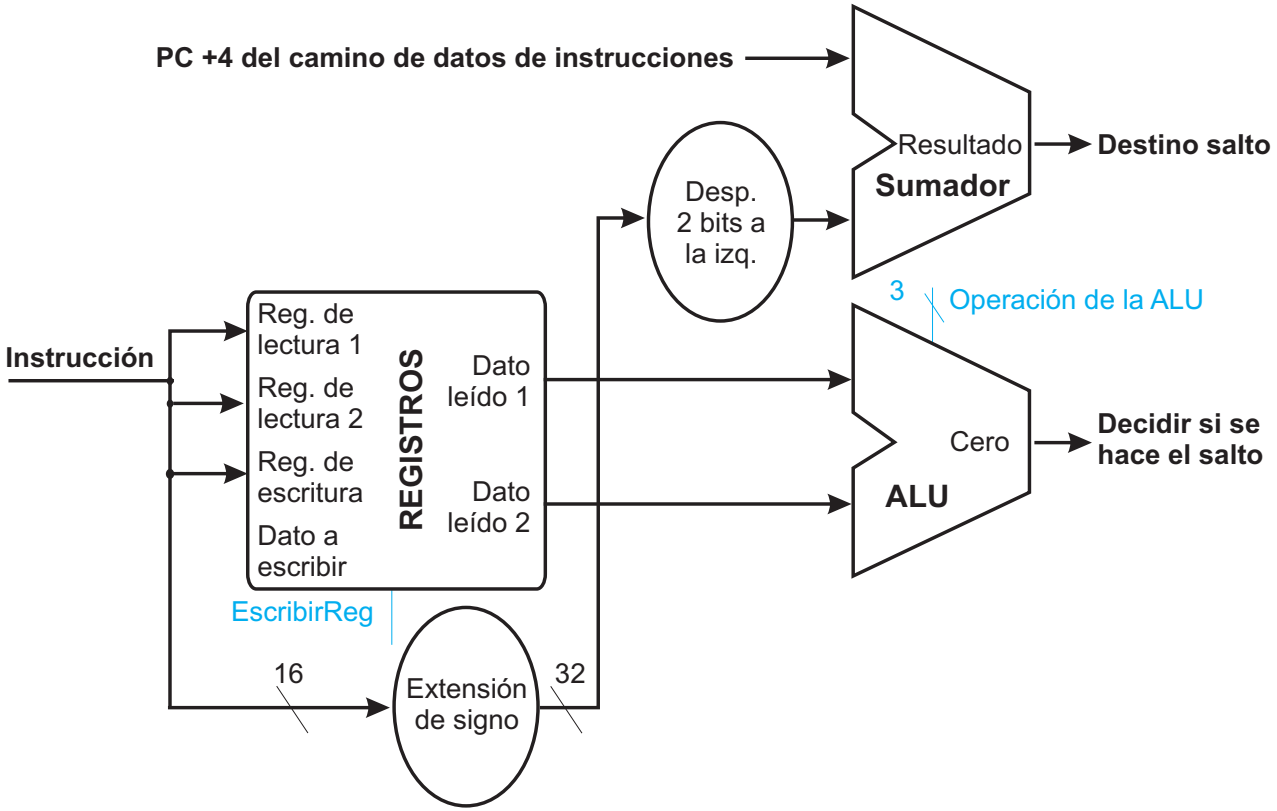
- Se utiliza la ALU para el cálculo de la dirección de memoria.
- El campo desplazamiento de la instrucción se extiende de 16 a 32 bits antes de realizar la suma.

Instrucciones de transferencia de datos



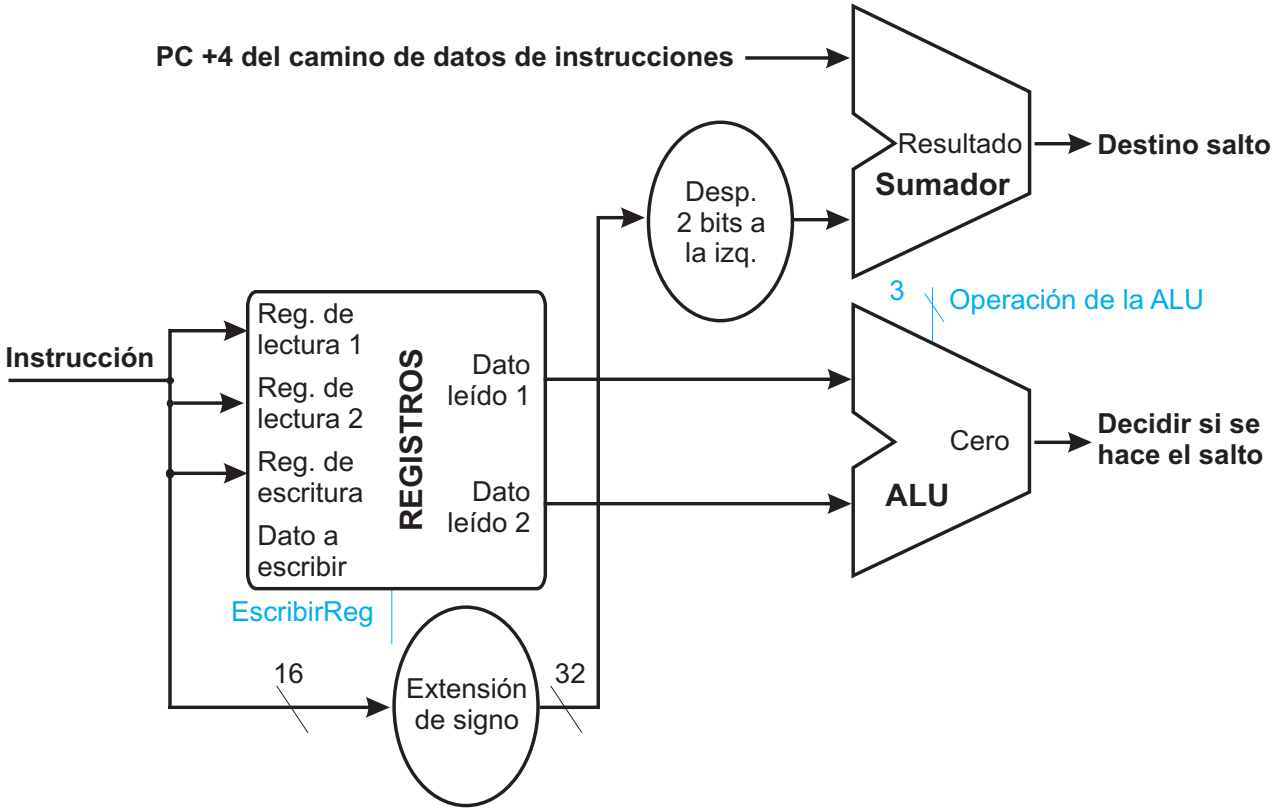
- Memoria de datos:
  - Entradas:
    - Señales de control de lectura y escritura.
    - Dirección a acceder (32 bits).
    - Palabra a escribir (32 bits).
  - Salidas:
    - Palabra leída (32 bits).

Instrucción de salto condicional



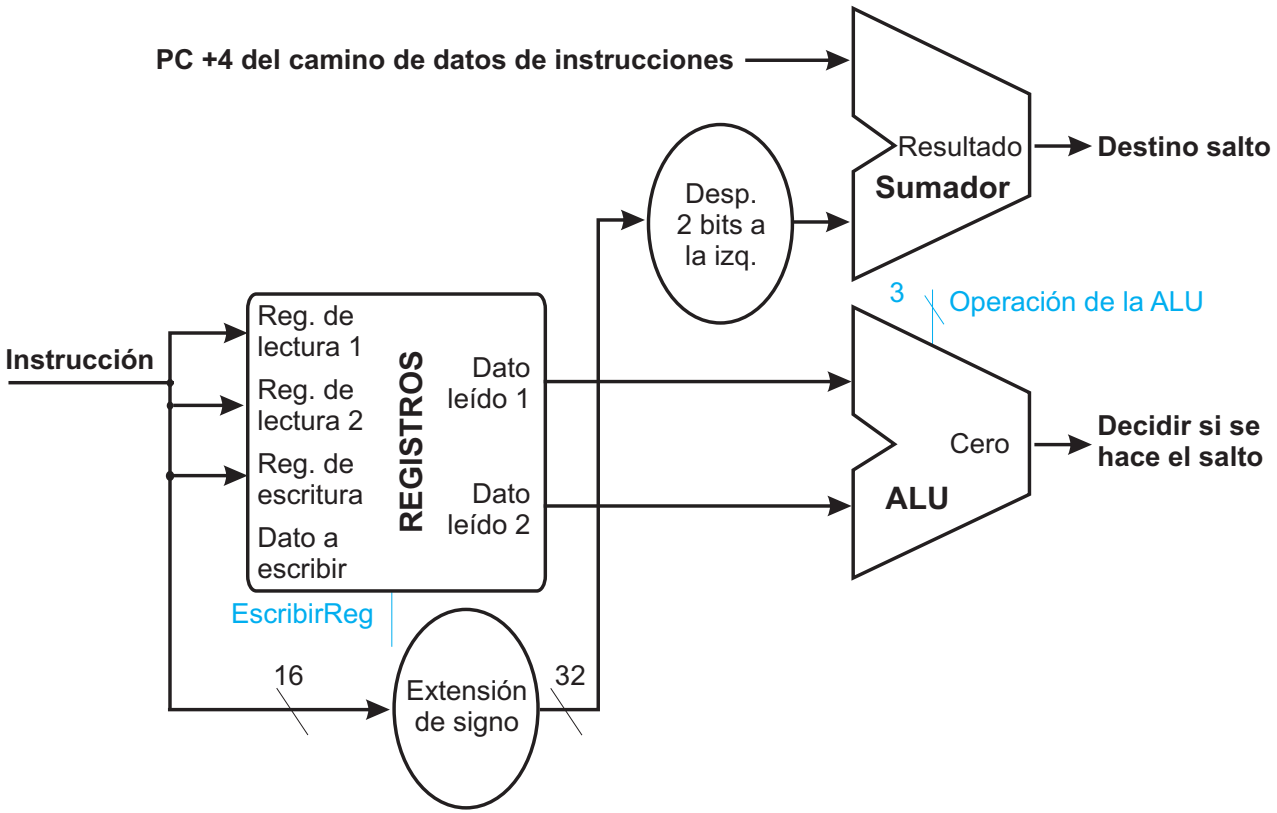
- Instrucción de tipo I: `beq $t1, $t2, despl`
- Evalúa si el contenido de $t1 y $t2 son iguales
- Si la condición se cumple aplica el salto

Instrucción de salto condicional



- Comparación:
  - Se realiza una resta, y se utiliza la salida Cero de la ALU como marcador de igualdad.

Instrucción de salto condicional



- Dirección de salto:
  - Salto relativo al PC+4 con signo
  - Dirección de salto: `(PC+4)+4*despl`
  - Los 16 bits del desplazamiento se extiende a 32 y se desplazan 2 posiciones a la izquierda.

Instrucción de salto incondicional

j L1

- Tipo J con direccionamiento pseudodirecto: `j L1`.
- Reemplaza los 28 bits de menor peso del PC con los 26 bits de menor peso de la instrucción desplazados 2 posiciones a la izquierda
- No es necesario la introducción de hardware adicional.

Camino de datos completo

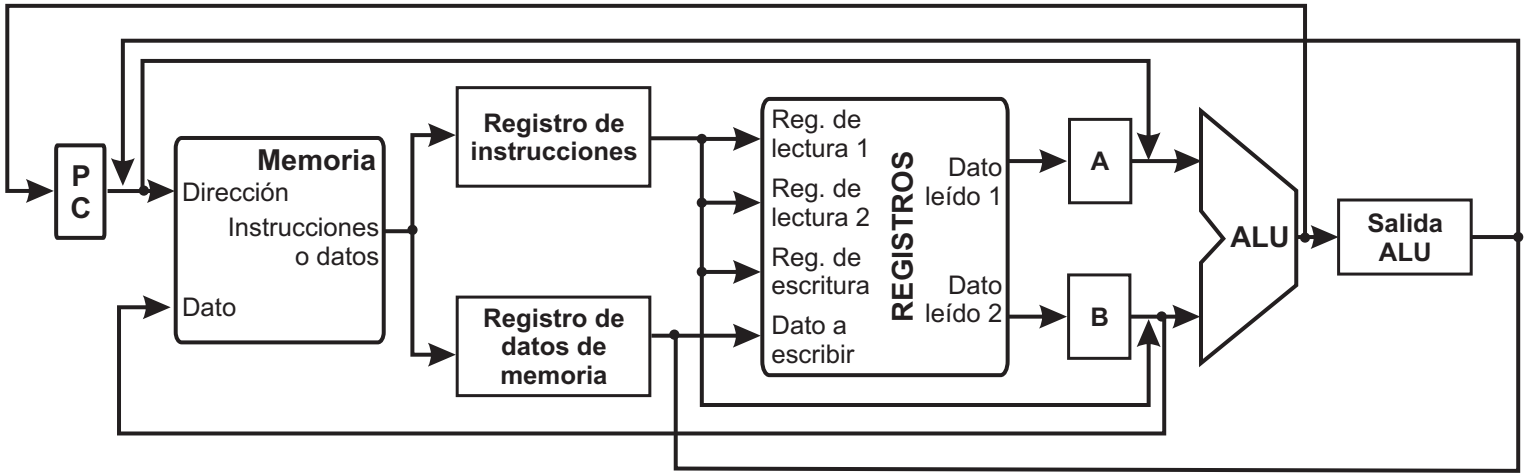
- Formado por la combinación de elementos necesitados por cada instrucción
- Cada una de las 5 etapas de ejecución de instrucciones se realiza en un ciclo de reloj diferente:
  - Se puede reutilizar una misma unidad funcional dentro de la misma instrucción si se utiliza en diferentes ciclos
    - No se necesita una memoria para instrucciones y otra para datos ya que el acceso a instrucciones y datos es en ciclos diferentes.
    - Para el calculo de la dirección de la siguiente instrucción (PC+4 o dirección de salto) podemos usar la ALU en lugar de los sumadores.

Camino de datos completo

- Los datos que vayan a utilizarse en los siguientes ciclos deben almacenarse en un elemento de almacenamiento (memoria o registro).
  - Los datos a utilizar por las siguientes instrucciones en ciclos posteriores se almacenarán en elementos de almacenamiento visibles al programador (banco de registros, PC o memoria).
  - Los datos a utilizar por la misma instrucción en ciclos posteriores se almacenarán en registros temporales.

Camino de datos completo

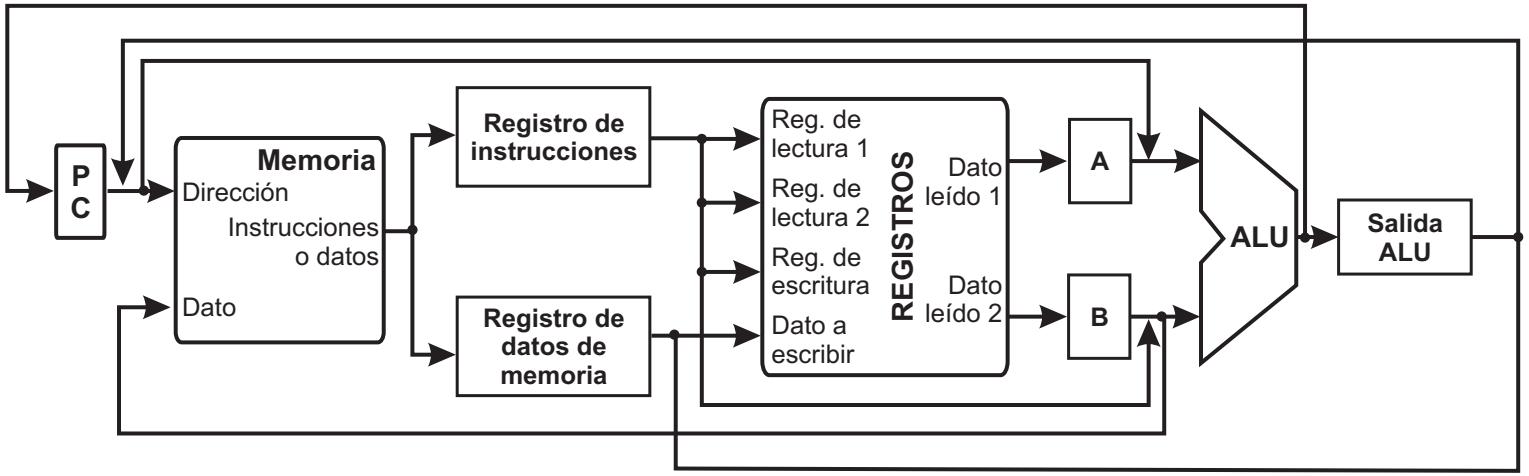
Visión de alto nivel del camino de datos completo



- Una sola ALU en vez de 1 ALU y dos sumadores
- Una memoria única para instrucciones y datos
- Registros temporales tras cada unidad funcional para almacenar su salida hasta que ese valor se utilice en el siguiente ciclo

Camino de datos completo

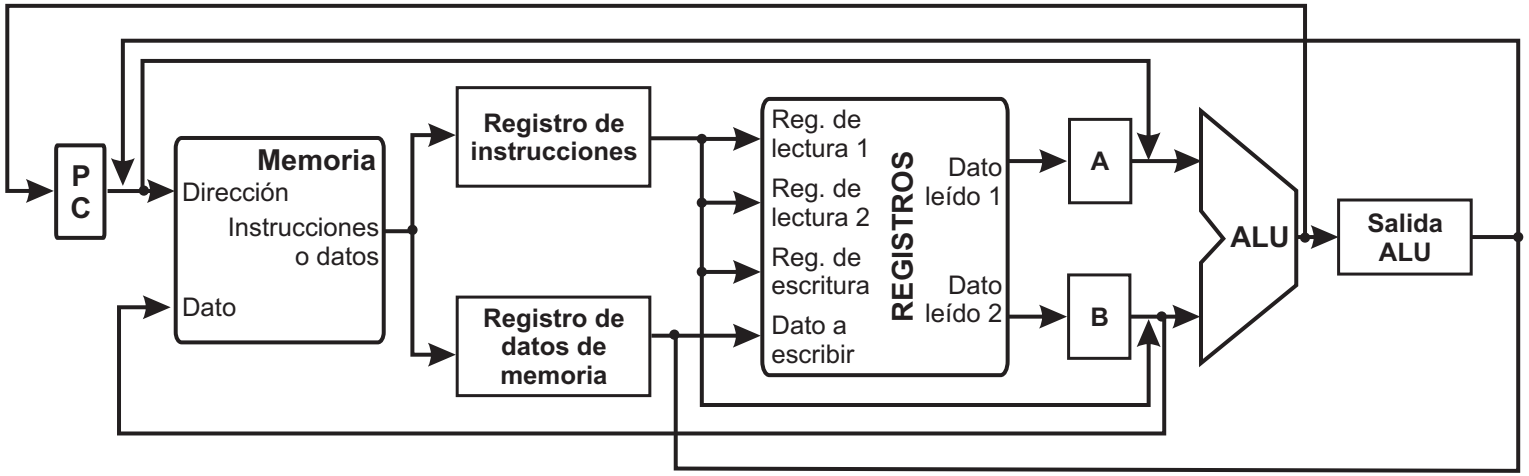
Visión de alto nivel del camino de datos completo



- Operaciones de la ALU:
  - Carga de la instrucción: Suma PC+4.
  - Instrucción de transferencia de datos: cálculo de la dirección a acceder.
  - Instrucción tipo R: Operación aritmética o lógica.
  - Salto condicional:
    - Evaluación de la condición de salto.
    - Cálculo de la dirección de destino.

Camino de datos completo

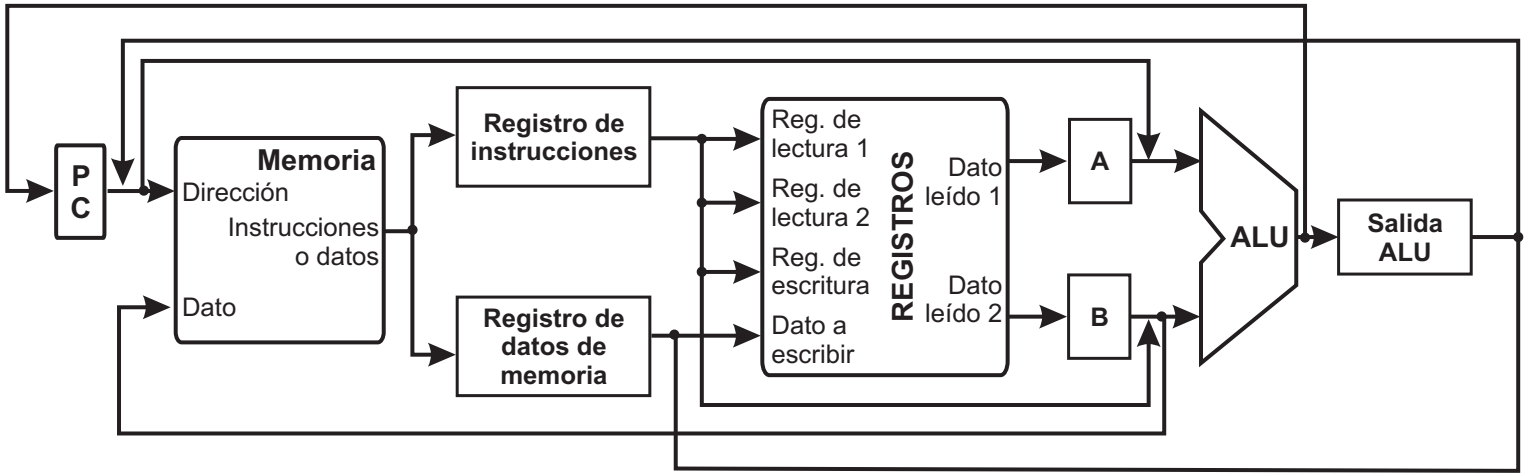
Visión de alto nivel del camino de datos completo



- Registros temporales:
  - Registro de instrucción (IR) y registro de datos de memoria (MDR):
    - Almacenan la instrucción o dato leído de memoria respectivamente.
  - Registros A y B: operandos leídos del banco de registros.
  - Salida ALU: salida de la ALU.
  - El registro IR requiere de una señal de control de escritura (necesita almacenar su valor más de un ciclo).

Camino de datos completo

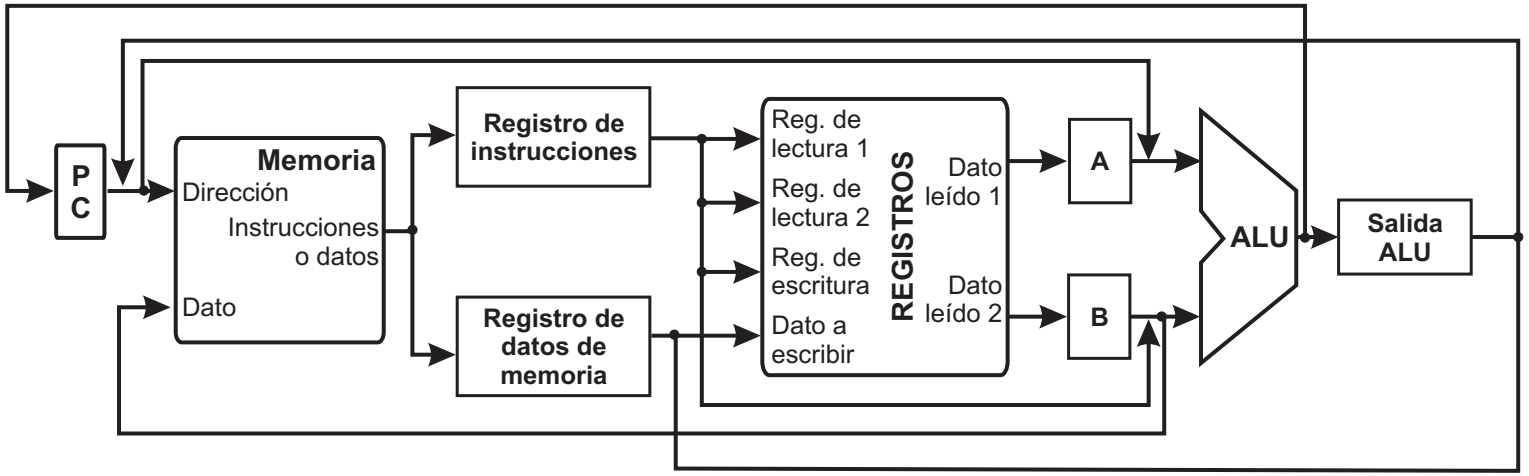
Visión de alto nivel del camino de datos completo



- Se introducen multiplexores para selección de entradas:
  - Origen de dirección de acceso a memoria:
    - Salida de la ALU (acceso a datos).
    - PC (acceso a instrucciones).
  - Registro de destino:
    - Instrucciones tipo R: rd (bits 11–15).
    - Instrucciones tipo I (cargas): rt (bits 16–20).

Camino de datos completo

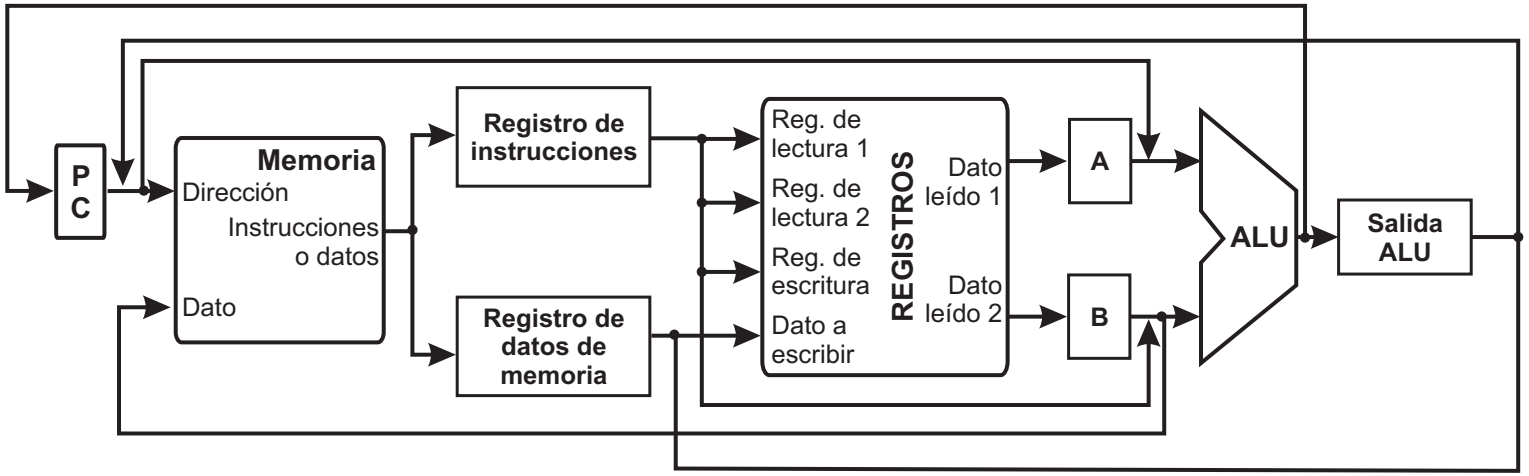
Visión de alto nivel del camino de datos completo



- Se introducen multiplexores para selección de entradas:
  - Dato a escribir:
    - De memoria (carga de datos).
    - ALU (instrucción aritmético-lógica).
  - Primera entrada de ALU:
    - registro A (acceso a datos e instrucción aritmético-lógica).
    - PC (carga de instrucción y saltos).

Camino de datos completo

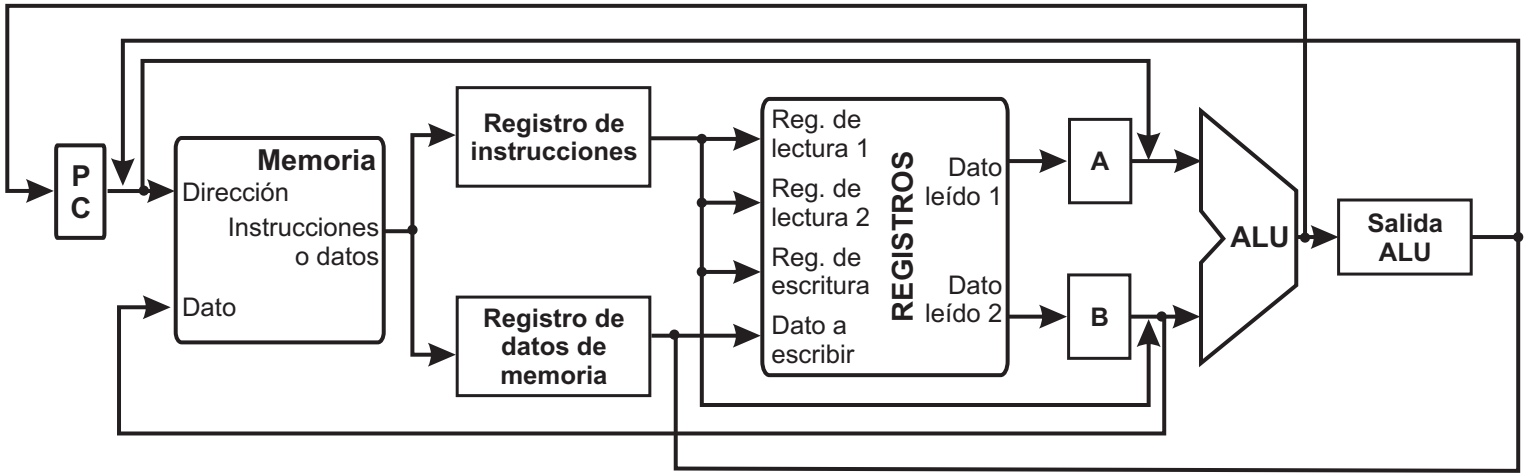
Visión de alto nivel del camino de datos completo



- Se introducen multiplexores para selección de entradas:
  - Segunda entrada de ALU:
    - Registro B (instrucción aritmético-lógica y salto condicional).
    - Constante 4 (incremento del PC).
    - Campo desplazamiento extendido (transferencia de datos).
    - Campo desplazamiento extendido y desplazado (saltos condicionales).

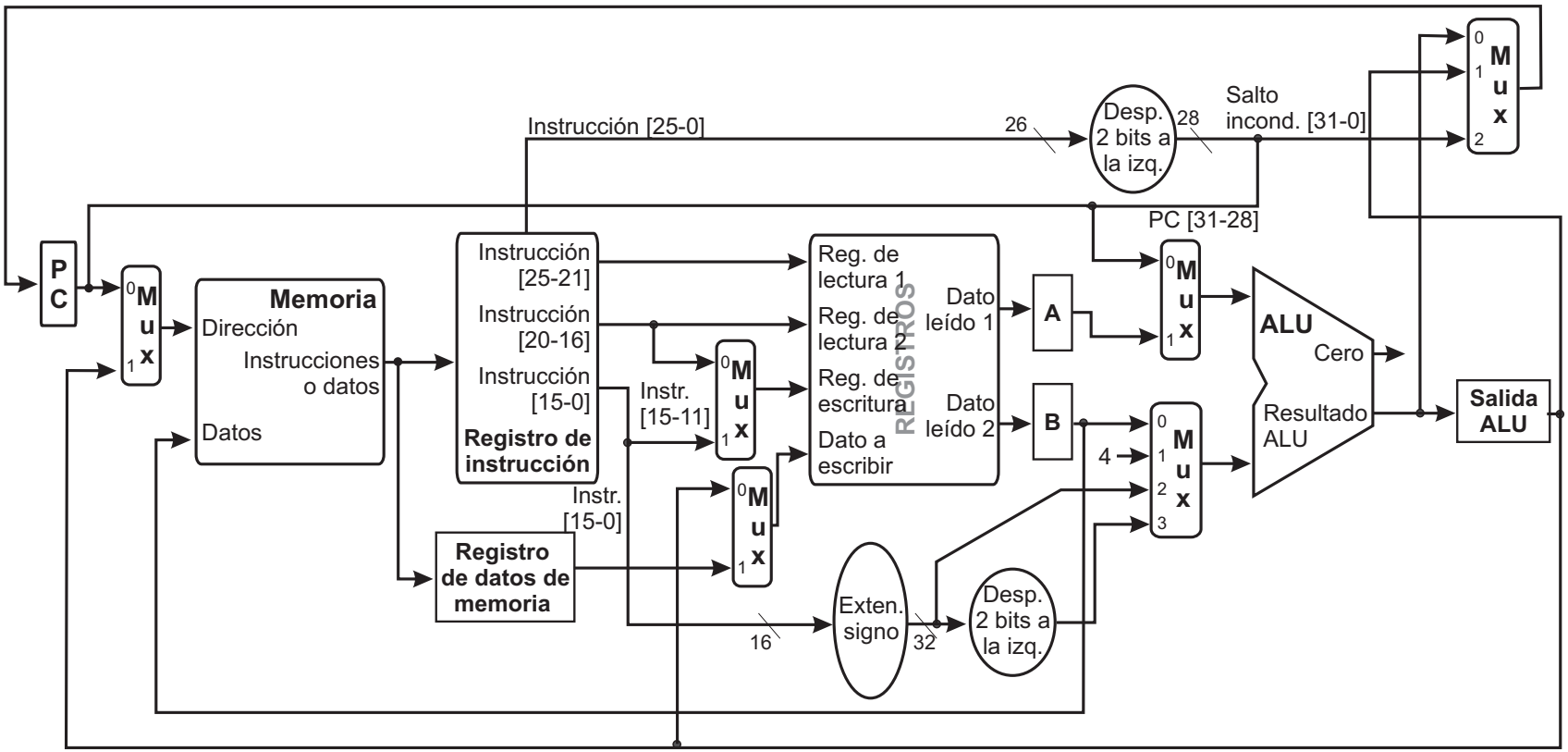
Camino de datos completo

Visión de alto nivel del camino de datos completo



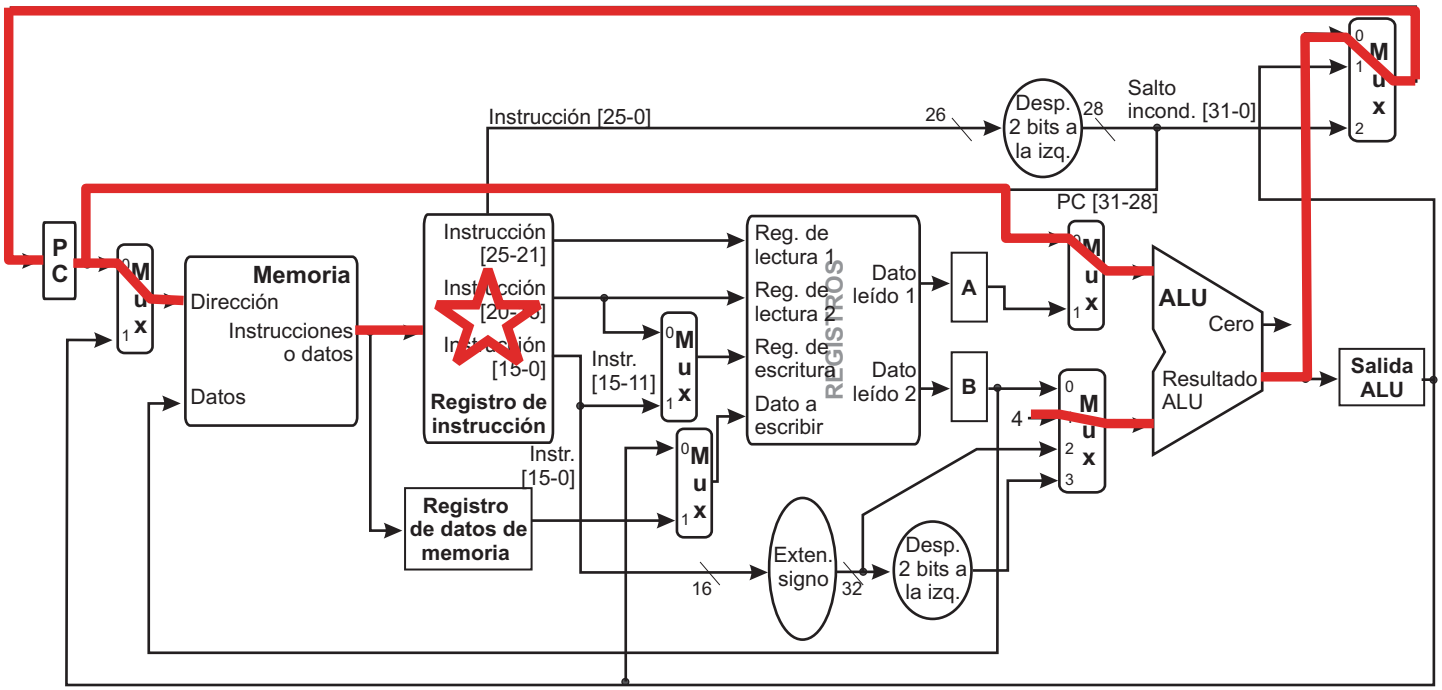
- Se introducen multiplexores para selección de entradas:
  - La dirección de la siguiente instrucción a ejecutar:
    - Resultado ALU: PC+4
    - Salida ALU: La dirección de salto calculada por la ALU en caso de salto condicional
    - La dirección de salto indicada en la propia instrucción en caso de salto incondicional

Camino de datos completo



Camino de datos completo

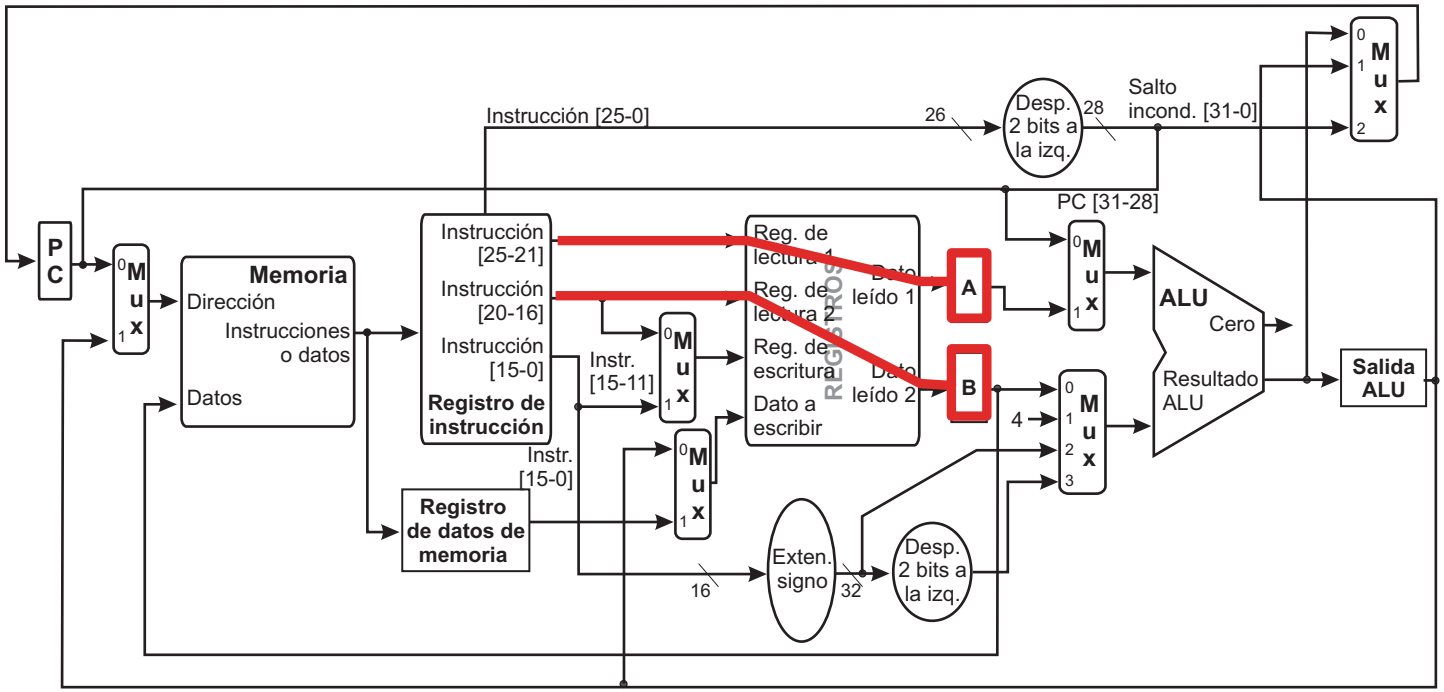
Carga de una instrucción



1. Se direcciona la memoria utilizando el PC.
2. Se efectúa la lectura y se escribe en el registro de instrucción.
3. Se incrementa el PC en 4 bytes.
4. Sincronización por flancos: el incremento del PC no será visible hasta el siguiente ciclo.

Camino de datos completo

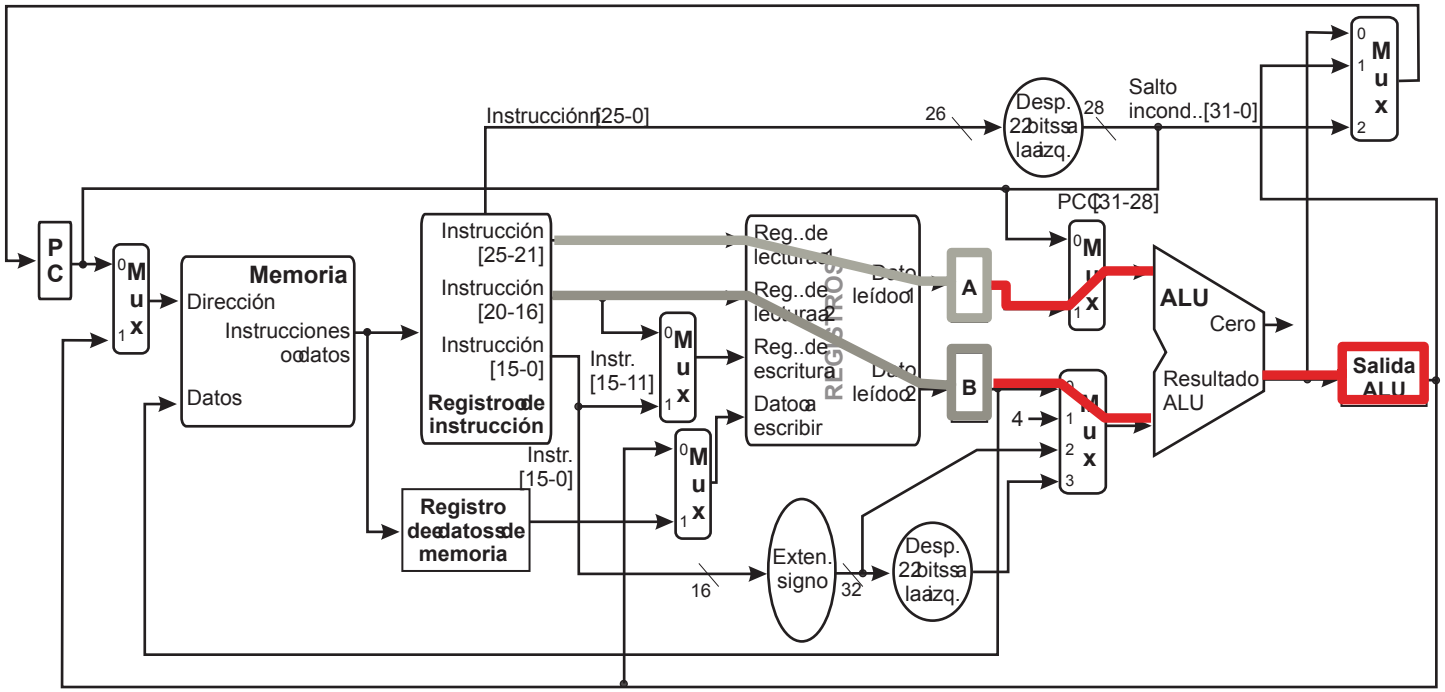
Instrucciones aritmético-lógicas



1. Lectura de los registros indicados por `rs` y `rt` y almacenamiento en los registros temporales A y B.
2. Ejecución por parte de la ALU de la operación especificada en el código de operación y almacenamiento del resultado en el registro SalidaALU.
3. Escritura del resultado en el registro espeficado por `rd`.

Camino de datos completo

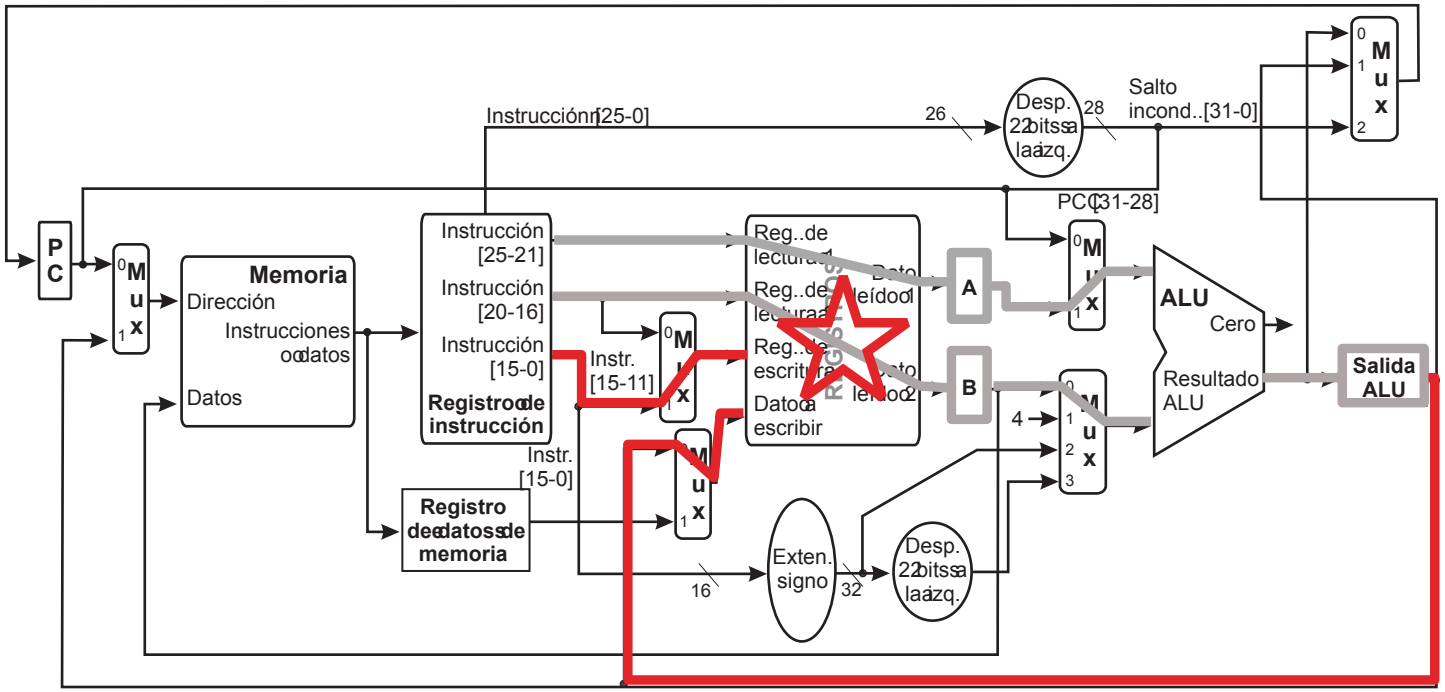
Instrucciones aritmético-lógicas



1. Lectura de los registros indicados por `rs` y `rt` y almacenamiento en los registros temporales A y B.
2. Ejecución por parte de la ALU de la operación especificada en el código de operación y almacenamiento del resultado en el registro SalidaALU.
3. Escritura del resultado en el registro espeficado por `rd`.

Camino de datos completo

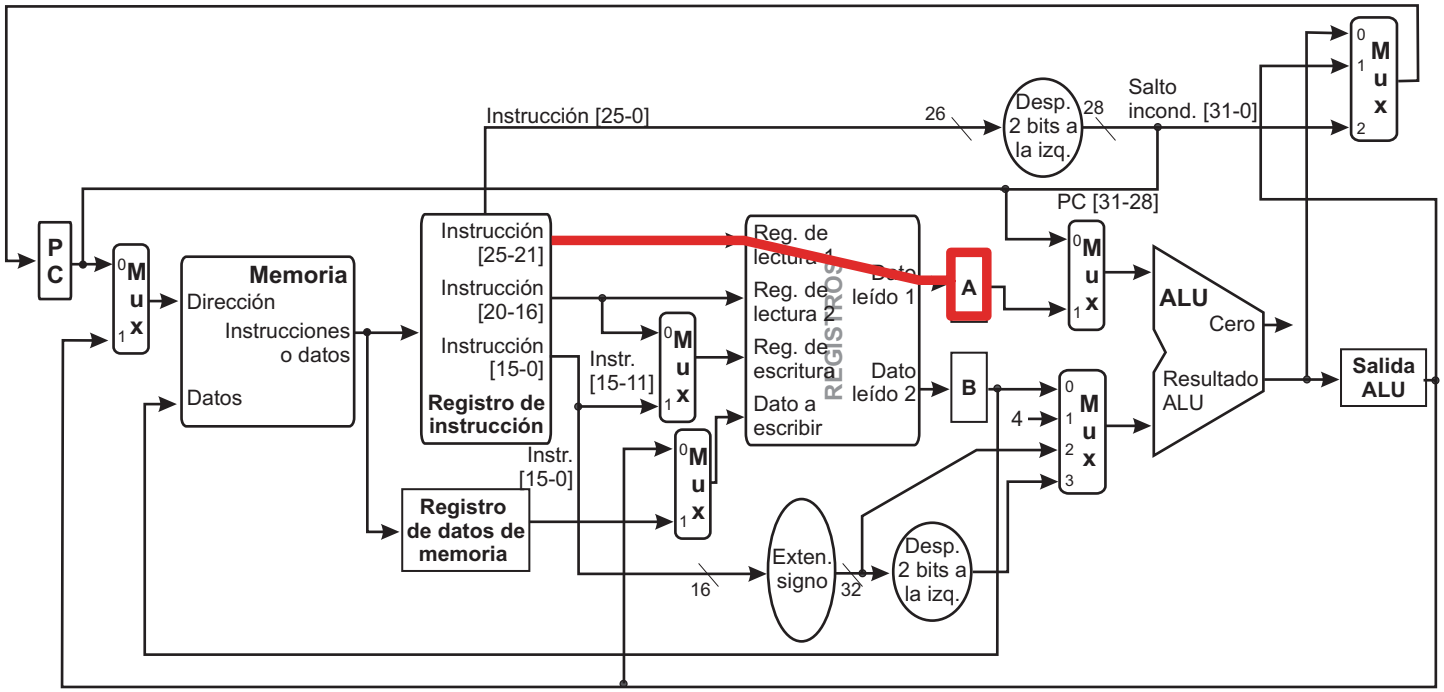
Instrucciones aritmético-lógicas



1. Lectura de los registros indicados por `rs` y `rt` y almacenamiento en los registros temporales A y B.
2. Ejecución por parte de la ALU de la operación especificada en el código de operación y almacenamiento del resultado en el registro SalidaALU.
3. Escritura del resultado en el registro espeficado por `rd`.

Camino de datos completo

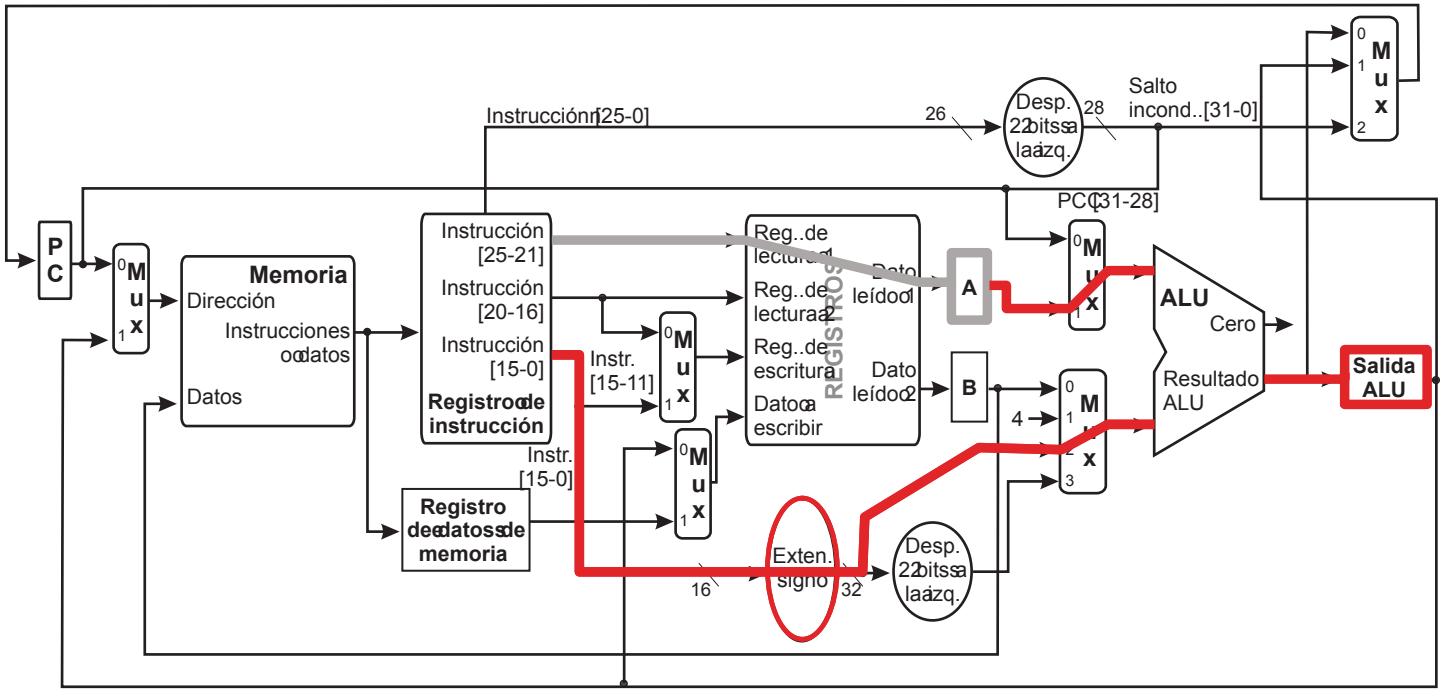
Instrucción `lw`



1. Lectura del registro base y almacenamiento en el registro temporal A.
2. Cálculo de la dirección de acceso como registro base más desplazamiento.
3. Acceso a memoria y carga del dato leído en el registro MDR.
4. Escritura del valor leído en el banco de registros.

Camino de datos completo

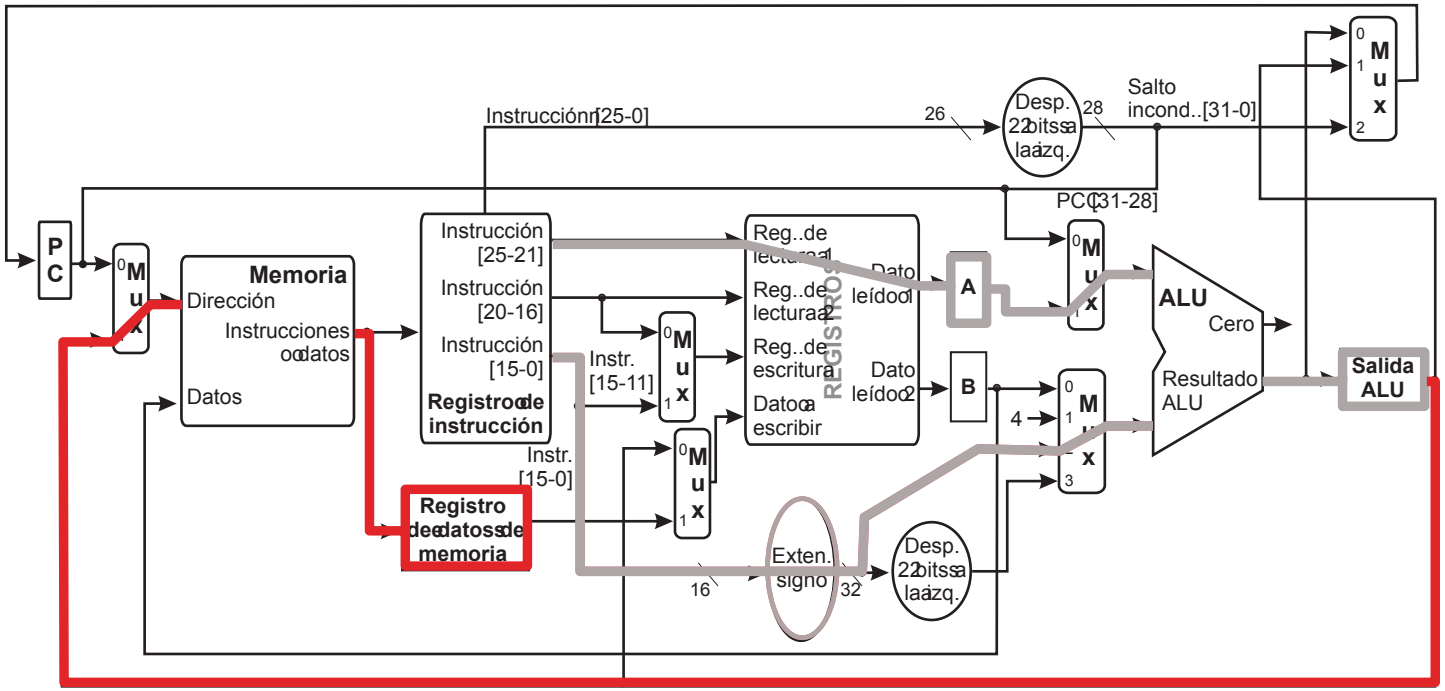
Instrucción `lw`



1. Lectura del registro base y almacenamiento en el registro temporal A.
2. Cálculo de la dirección de acceso como registro base más desplazamiento.
3. Acceso a memoria y carga del dato leído en el registro MDR.
4. Escritura del valor leído en el banco de registros.

Camino de datos completo

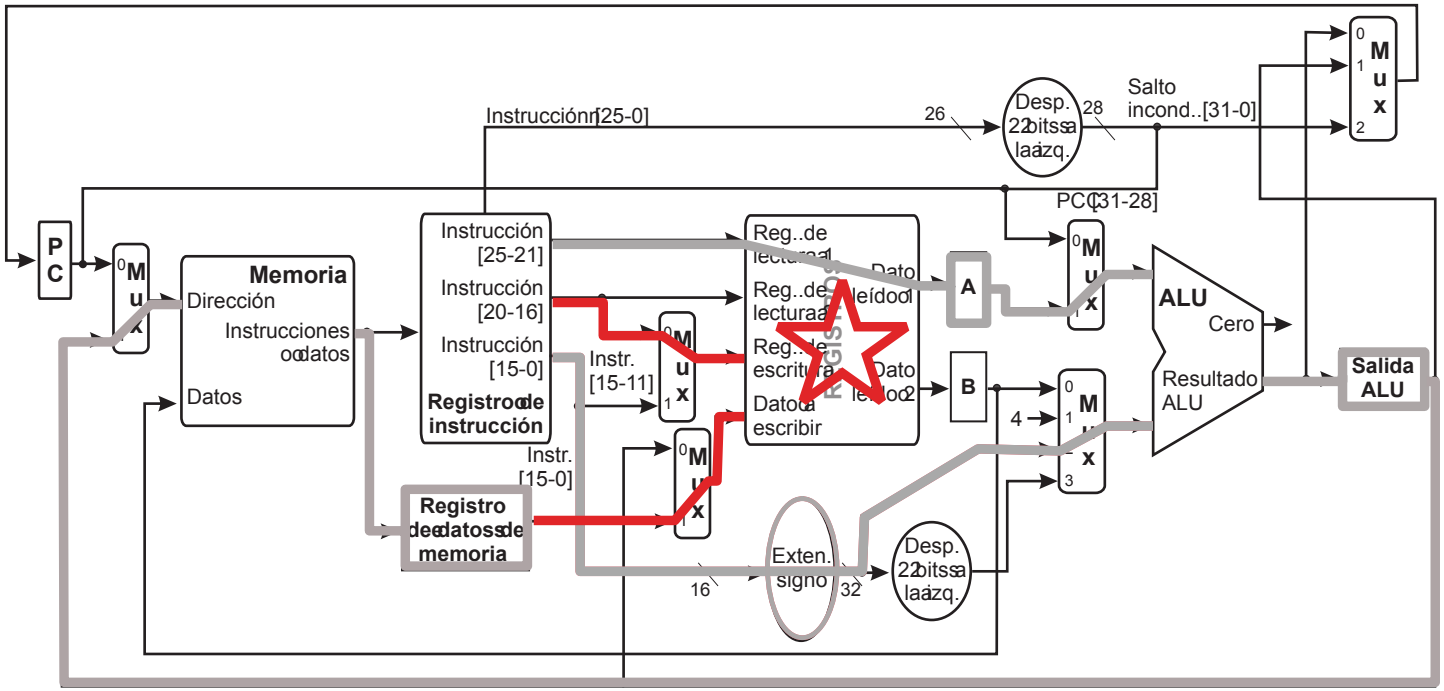
Instrucción `lw`



1. Lectura del registro base y almacenamiento en el registro temporal A.
2. Cálculo de la dirección de acceso como registro base más desplazamiento.
3. Acceso a memoria y carga del dato leído en el registro MDR.
4. Escritura del valor leído en el banco de registros.

Camino de datos completo

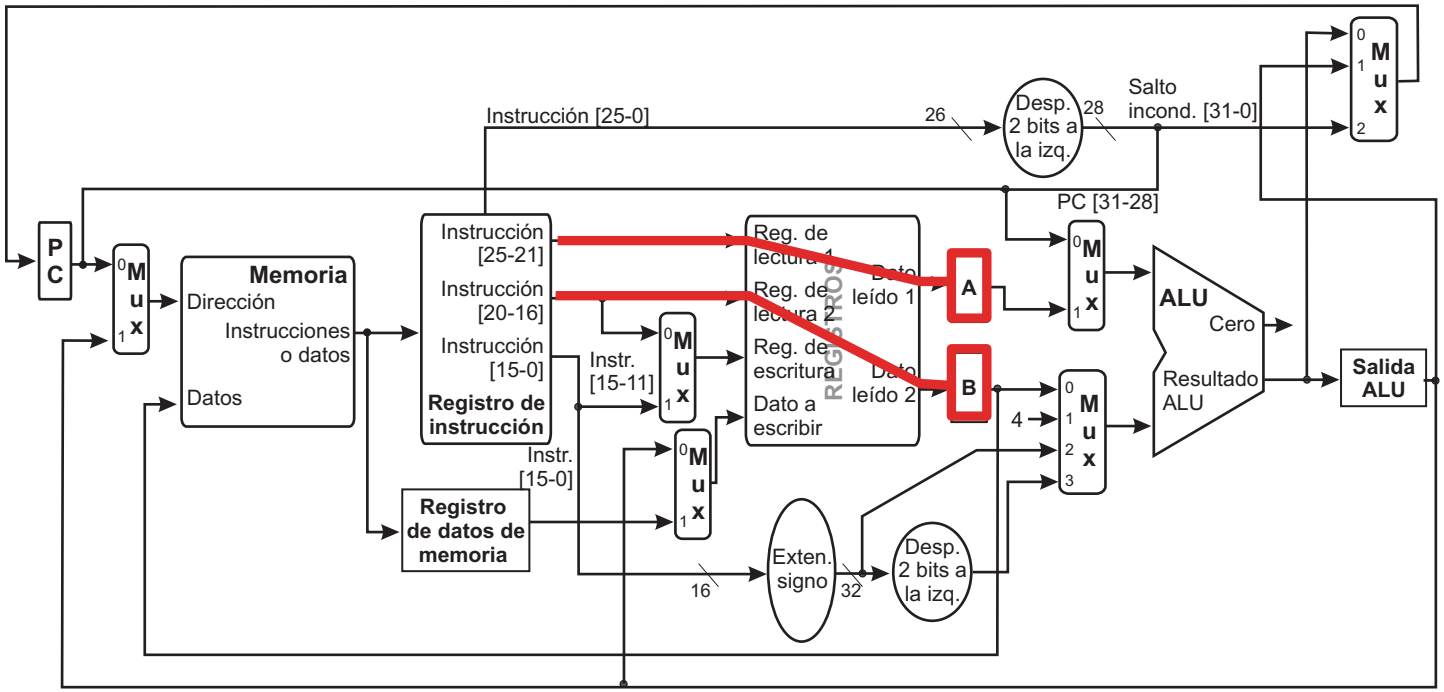
Instrucción `lw`



1. Lectura del registro base y almacenamiento en el registro temporal A.
2. Cálculo de la dirección de acceso como registro base más desplazamiento.
3. Acceso a memoria y carga del dato leído en el registro MDR.
4. Escritura del valor leído en el banco de registros.

Camino de datos completo

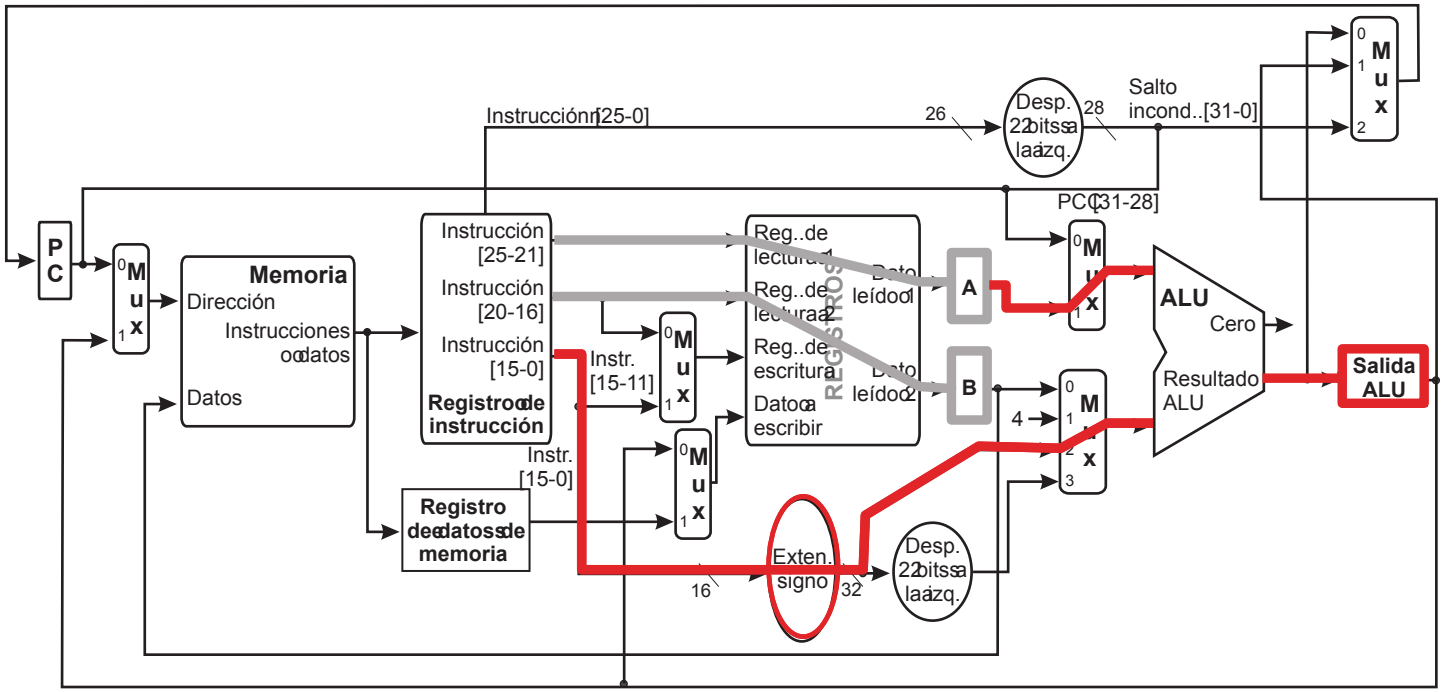
Instrucción sw



1. Carga en paralelo del registro base (registro temporal A) y dato a almacenar (registro temporal B).
2. Cálculo de la dirección de acceso como registro base más desplazamiento.
3. Acceso a memoria y almacenamiento del dato en el registro temporal B.

Camino de datos completo

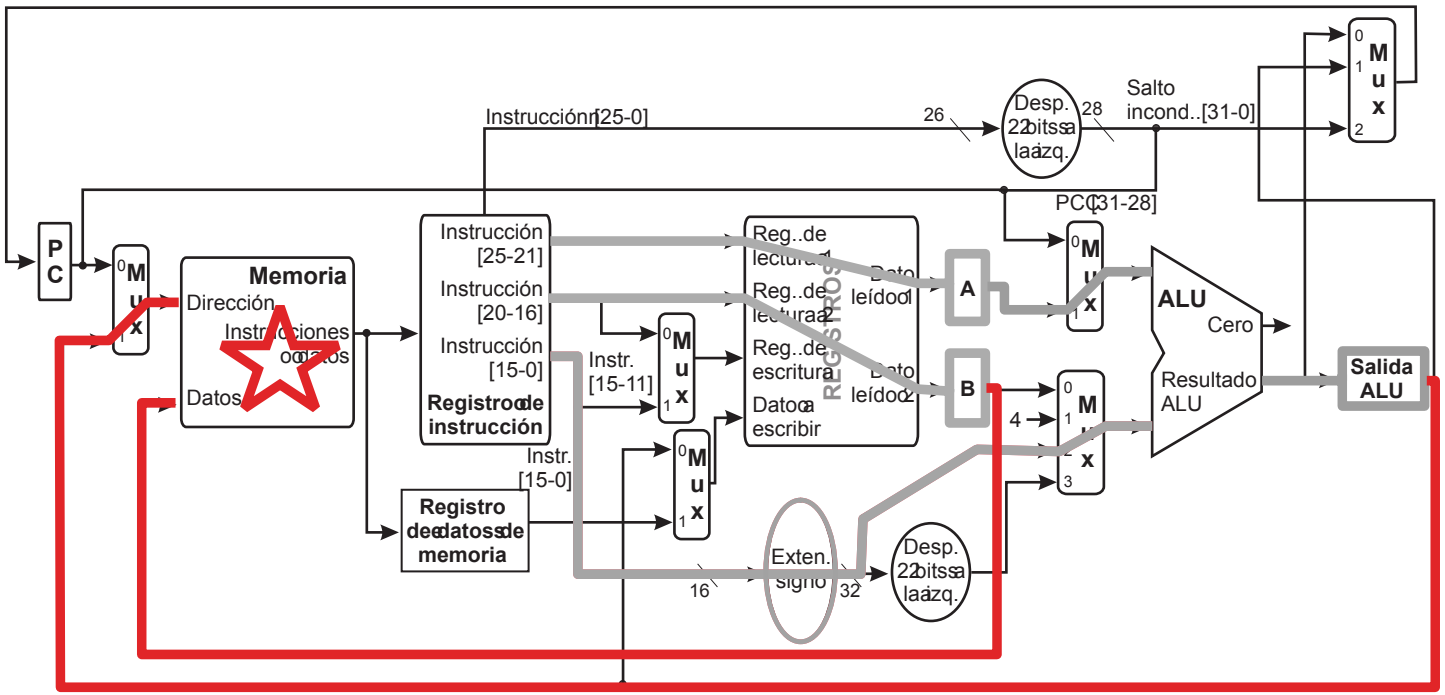
Instrucción sw



1. Carga en paralelo del registro base (registro temporal A) y dato a almacenar (registro temporal B).
2. Cálculo de la dirección de acceso como registro base más desplazamiento.
3. Acceso a memoria y almacenamiento del dato en el registro temporal B.

Camino de datos completo

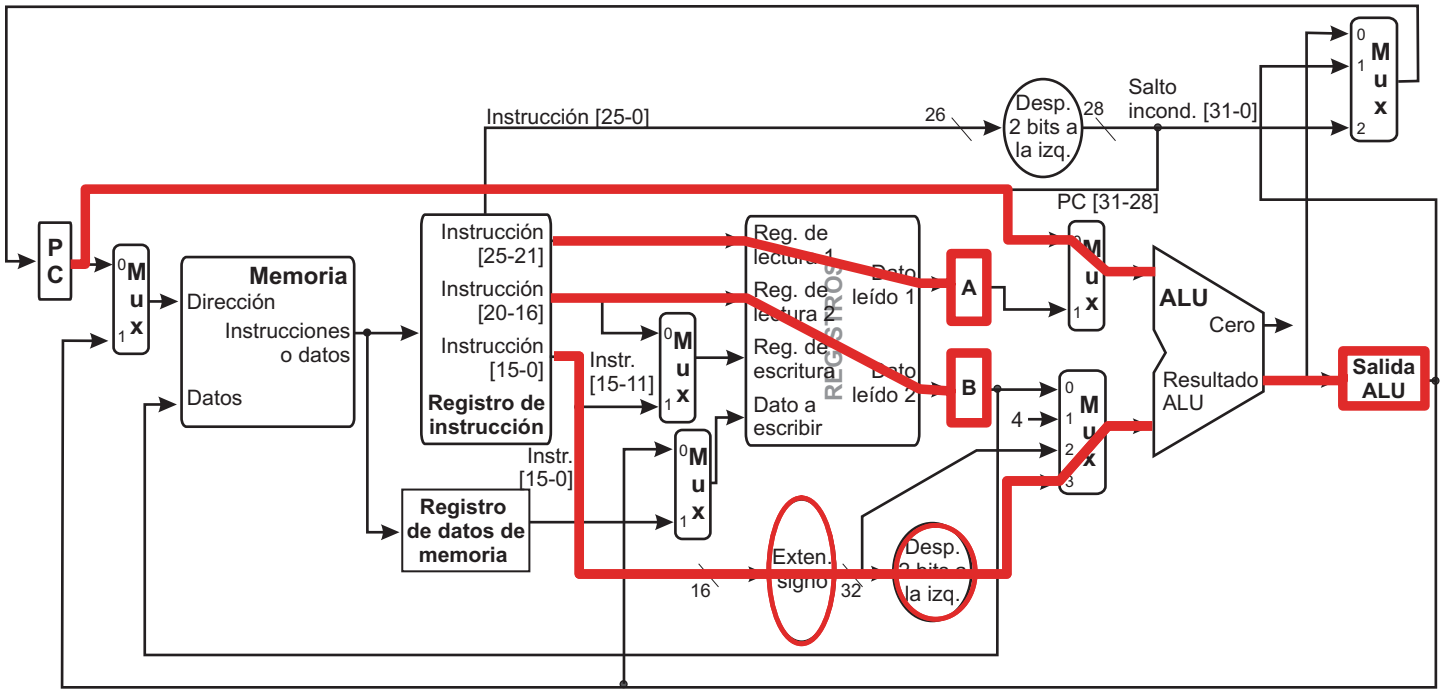
Instrucción sw



1. Carga en paralelo del registro base (registro temporal A) y dato a almacenar (registro temporal B).
2. Cálculo de la dirección de acceso como registro base más desplazamiento.
3. Acceso a memoria y almacenamiento del dato en el registro temporal B.

Camino de datos completo

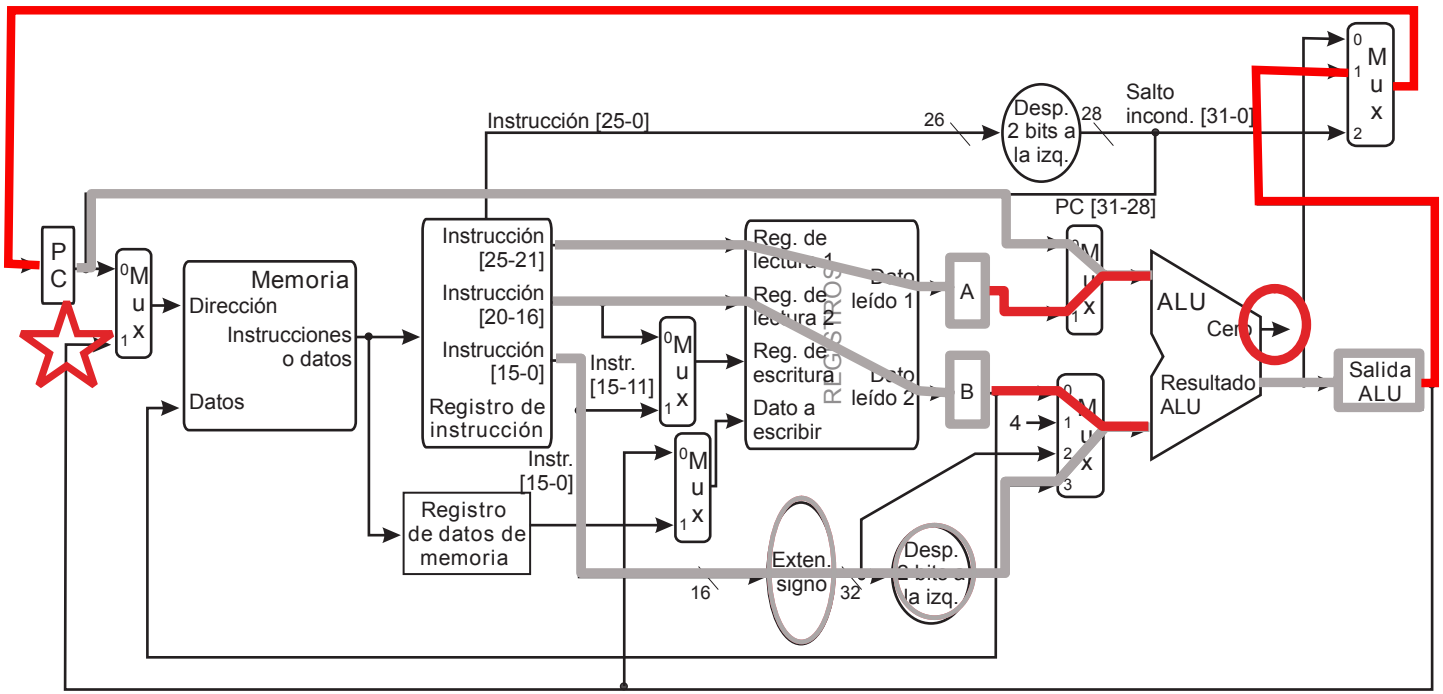
Instrucción beq



1. Carga en paralelo de los operandos a comparar (registros temporales A y B). Al mismo tiempo, cálculo de la dirección de destino del salto.
2. Resta de los operandos. La señal de Cero de la ALU se utilizará para activar la escritura en el PC. Si se cumple la condición se escribe el PC con el nuevo valor.

Camino de datos completo

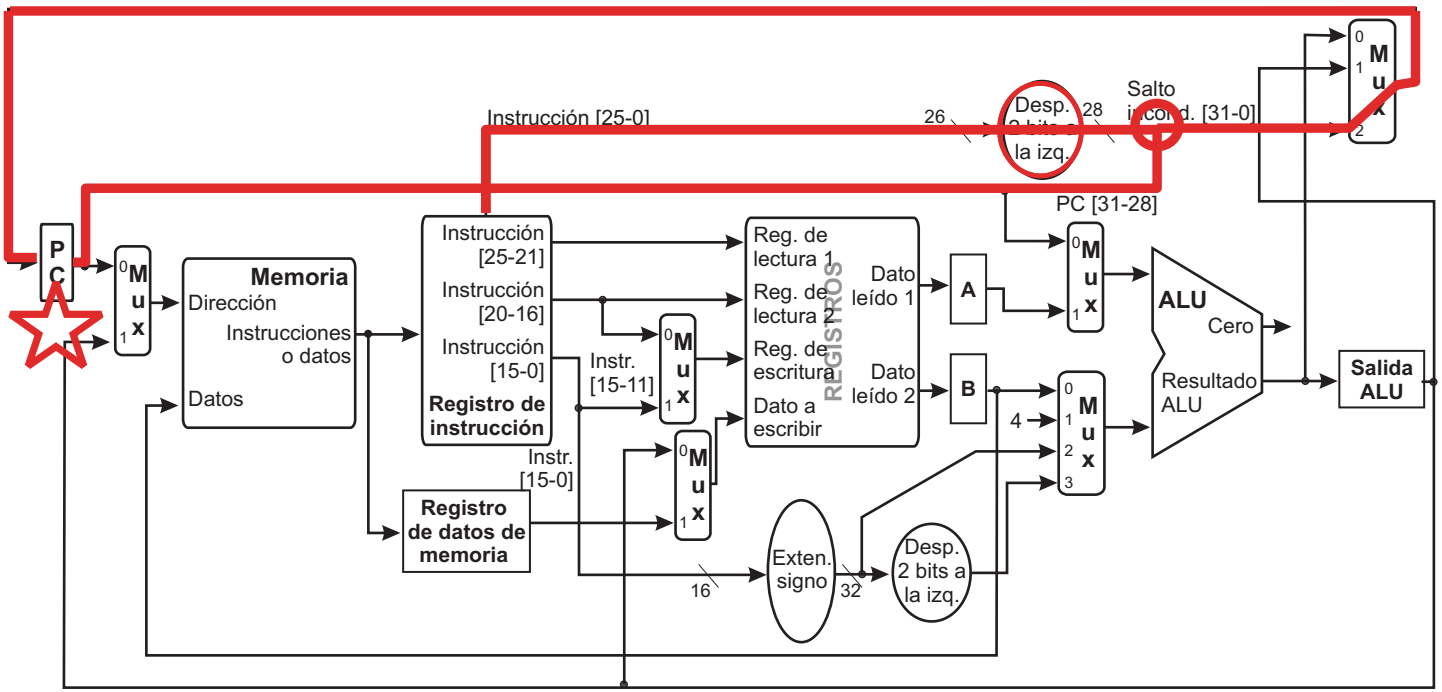
Instrucción beq



1. Carga en paralelo de los operandos a comparar (registros temporales A y B). Al mismo tiempo, cálculo de la dirección de destino del salto.
2. Resta de los operandos. La señal de Cero de la ALU se utilizará para activar la escritura en el PC. Si se cumple la condición se escribe el PC con el nuevo valor.

Camino de datos completo

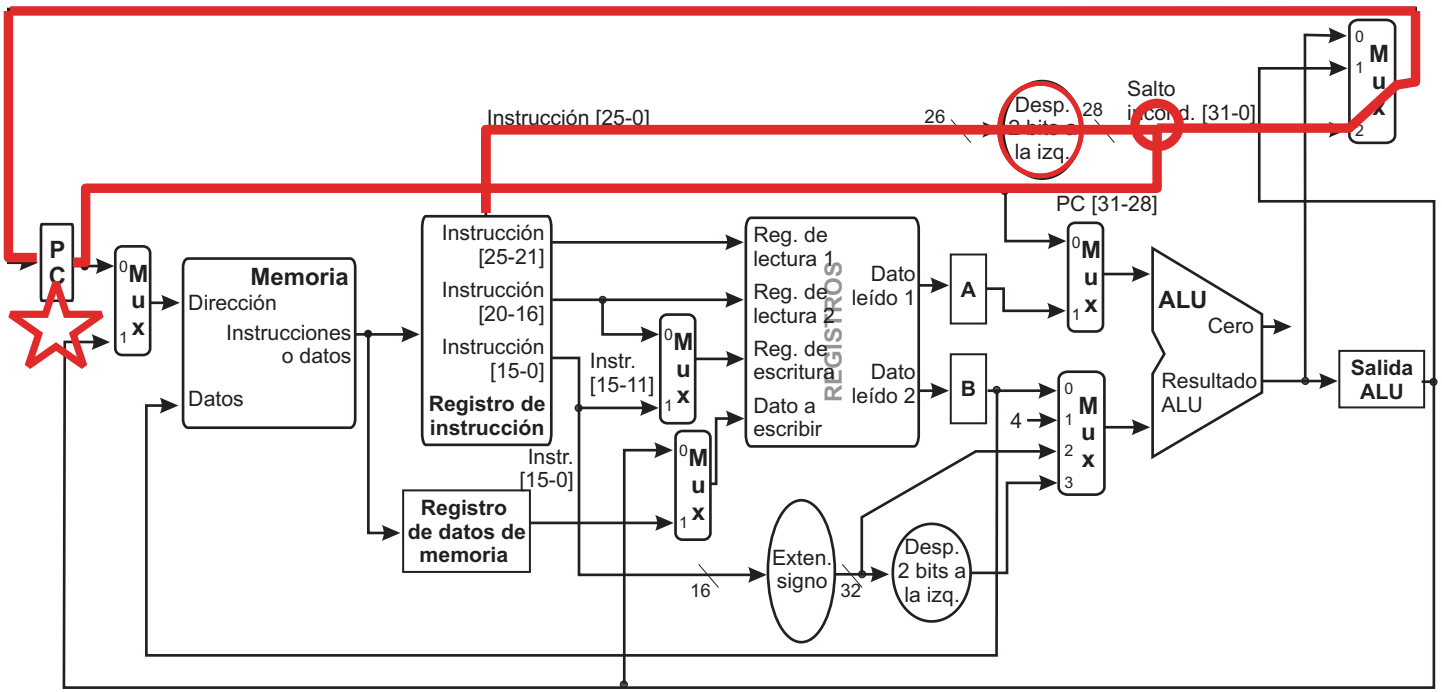
Instrucción j



- El único paso a realizar es la escritura del PC con la dirección de salto.
- Serían precisos tan solo dos ciclos para su ejecución, pero como veremos se ejecutará en 3 para simplificar la unidad de control.

Camino de datos completo

Instrucción j



- El único paso a realizar es la escritura del PC con la dirección de salto.
- Serían precisos tan solo dos ciclos para su ejecución, pero como veremos se ejecutará en 3 para simplificar la unidad de control.

Señales de control

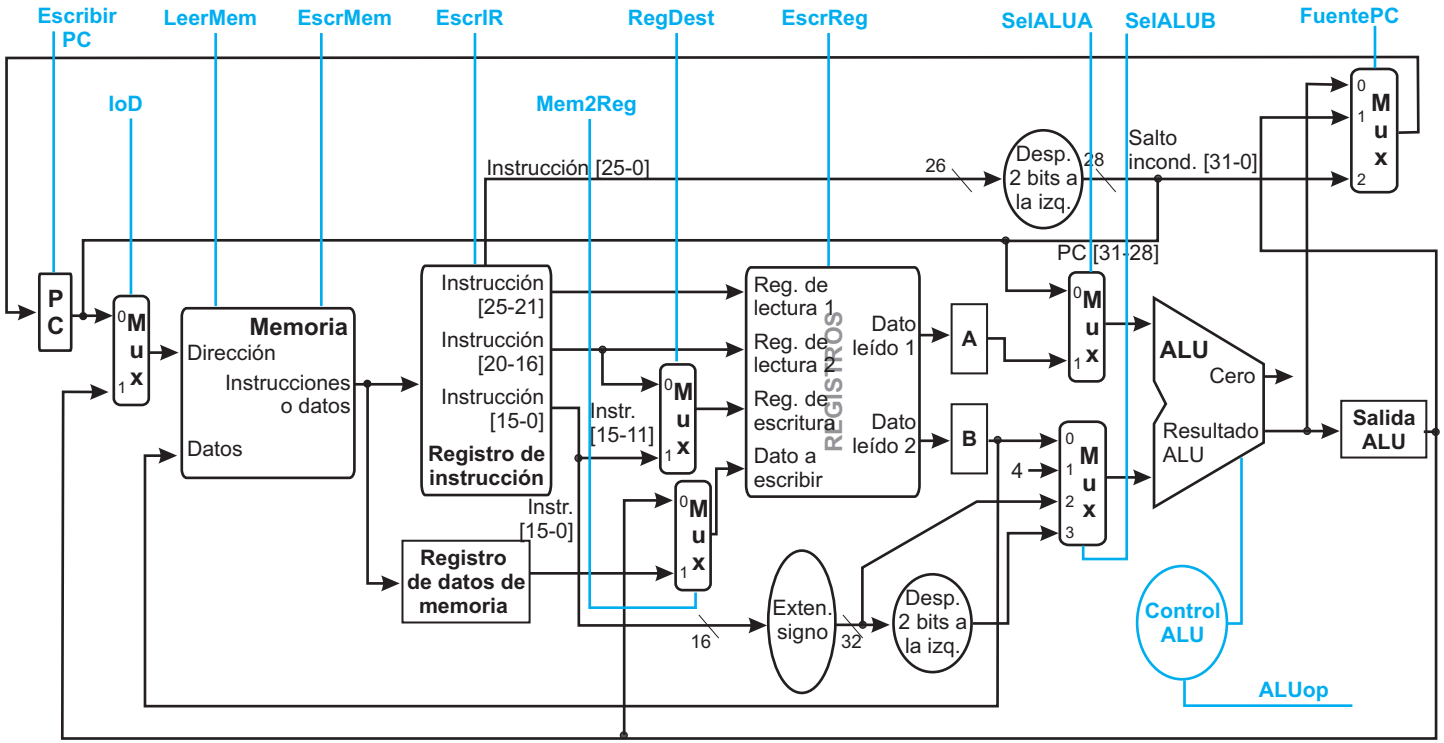
- Señales de control de escritura: PC, IR, memoria, banco de registros.
- Señal de lectura para la memoria.
- MUXes:
  - 2 entradas: 1 línea de control.
  - 4 entradas: 2 líneas de control.
- Unidad de control de la ALU:
  - Código de función + ALUop = 3 bits de control.

Señales de control
Unidad de control de la ALU

ALUOp	Campo de función	Acción	Líneas de control de la ALU
00	XXXXXX	suma	010
01	XXXXXX	resta	110
10	100000	suma	010
10	100010	resta	110
10	100100	AND	000
10	100101	OR	001
10	101010	`slt`	111

- 00: Usado en instrucciones `lw`, `sw` y `beq` (para calcular el destino del salto).
- 01: Usado en `beq` para determinar la igualdad de los operandos.
- 10: En instrucciones tipo R, la operación a realizar depende del campo de función de la instrucción.

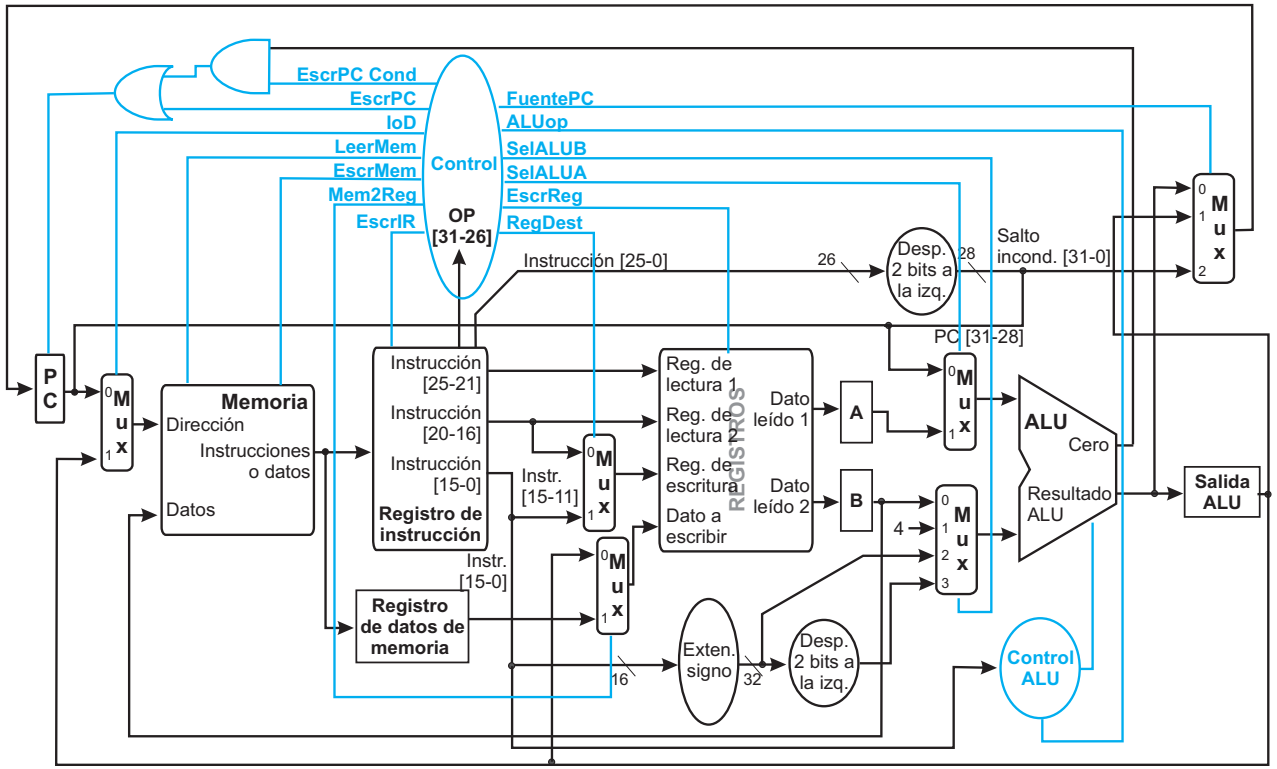
Señales de control



- ALUop, SelALUB y Fuente PC son de 2 bits.
- El resto son de 1 bit.
- Los únicos registros que requiere señal de escritura son IR y PC.

Señales de control

Camino de datos y unidad de control

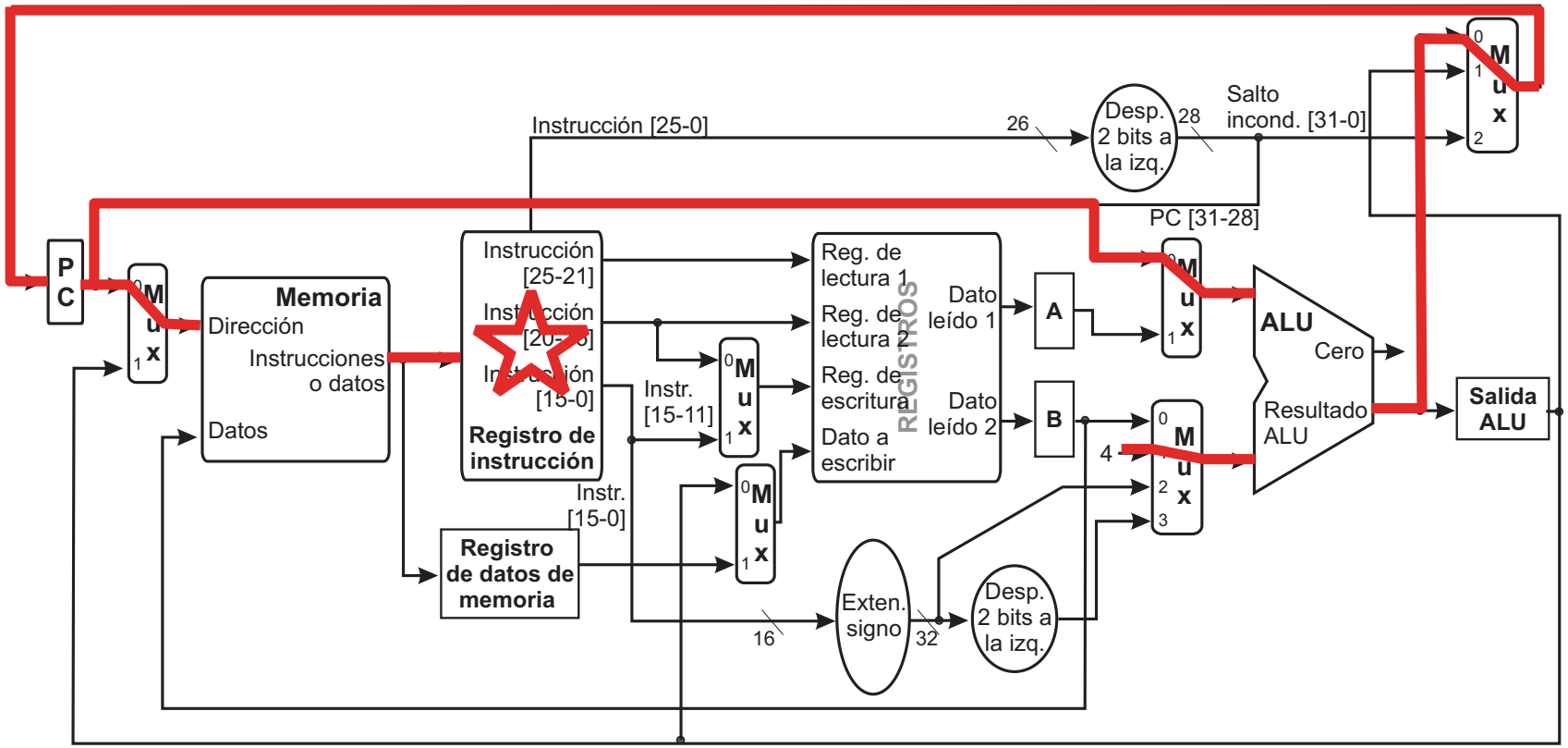


- Condiciones de escritura del PC:
  - Carga de la instrucción - Salto incondicional (`EscrPC=1`).
    o
  - Salto condicional con igualdad de operandos (`EscrPCCond=1` y `Cero ALU=1`).

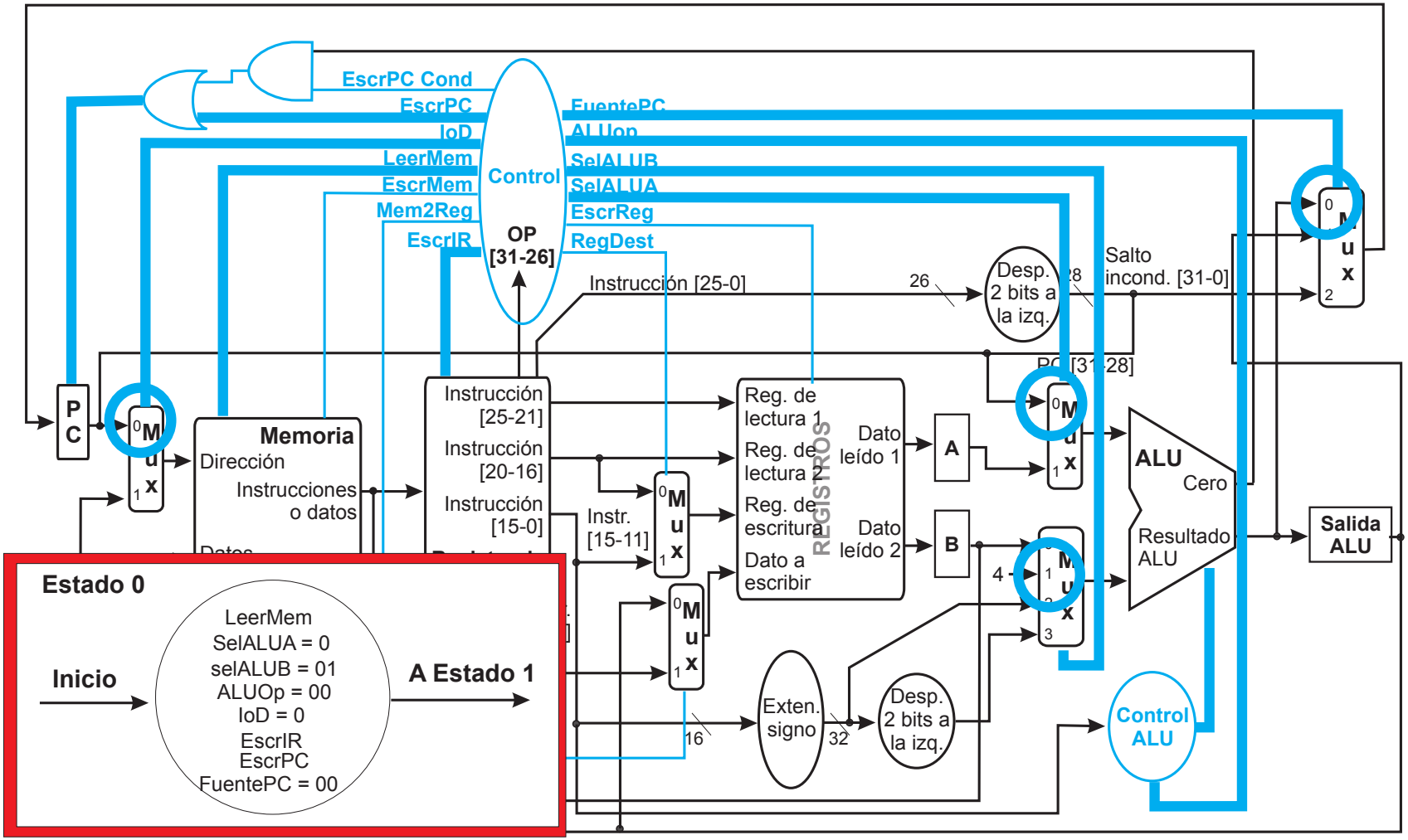
- La Unidad de Control debe especificar:
  - Las señales que se van a activar en cada paso
  - El paso siguiente de la secuencia
- Es un sistema secuencial síncrono:
  - Cada estado dura un ciclo de reloj y representa una etapa de la ejecución de una instrucción.
  - Entradas: los bits de la instrucción a ejecutar y los indicadores internos (p. ej. la salida Cero de la ALU).
  - Salidas: las señales de control a activar.

- Los dos primeros estados son comunes a todas las instrucciones:
  - Carga de la instrucción.
  - Decodificación.
- El resto dependen del tipo de instrucción y sus operandos.
- Tras terminar la ejecución de una instrucción el autómata vuelve al estado inicial para procesar la siguiente instrucción.

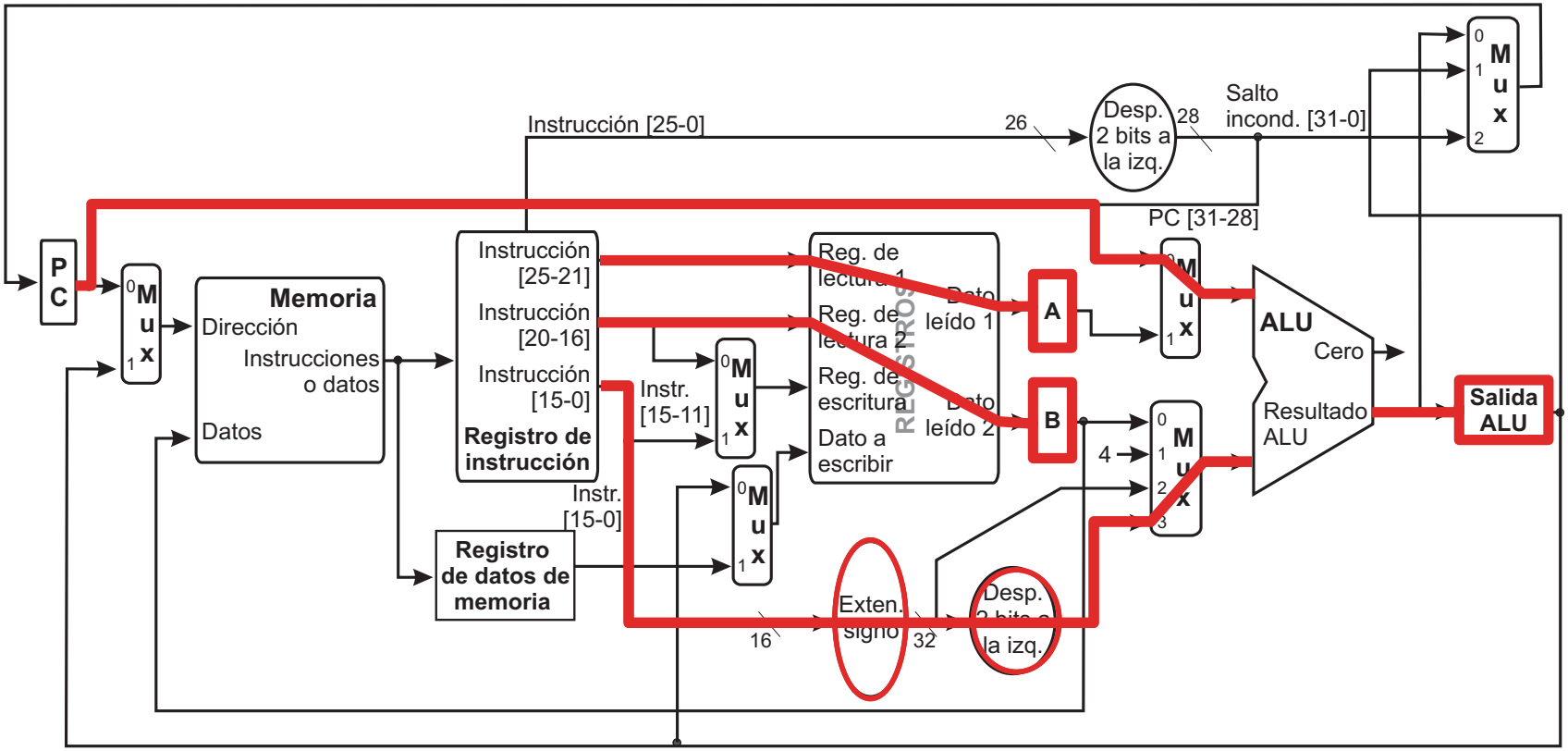
Etapa 1 – Carga de una instrucción



Etapa 1 – Carga de una instrucción

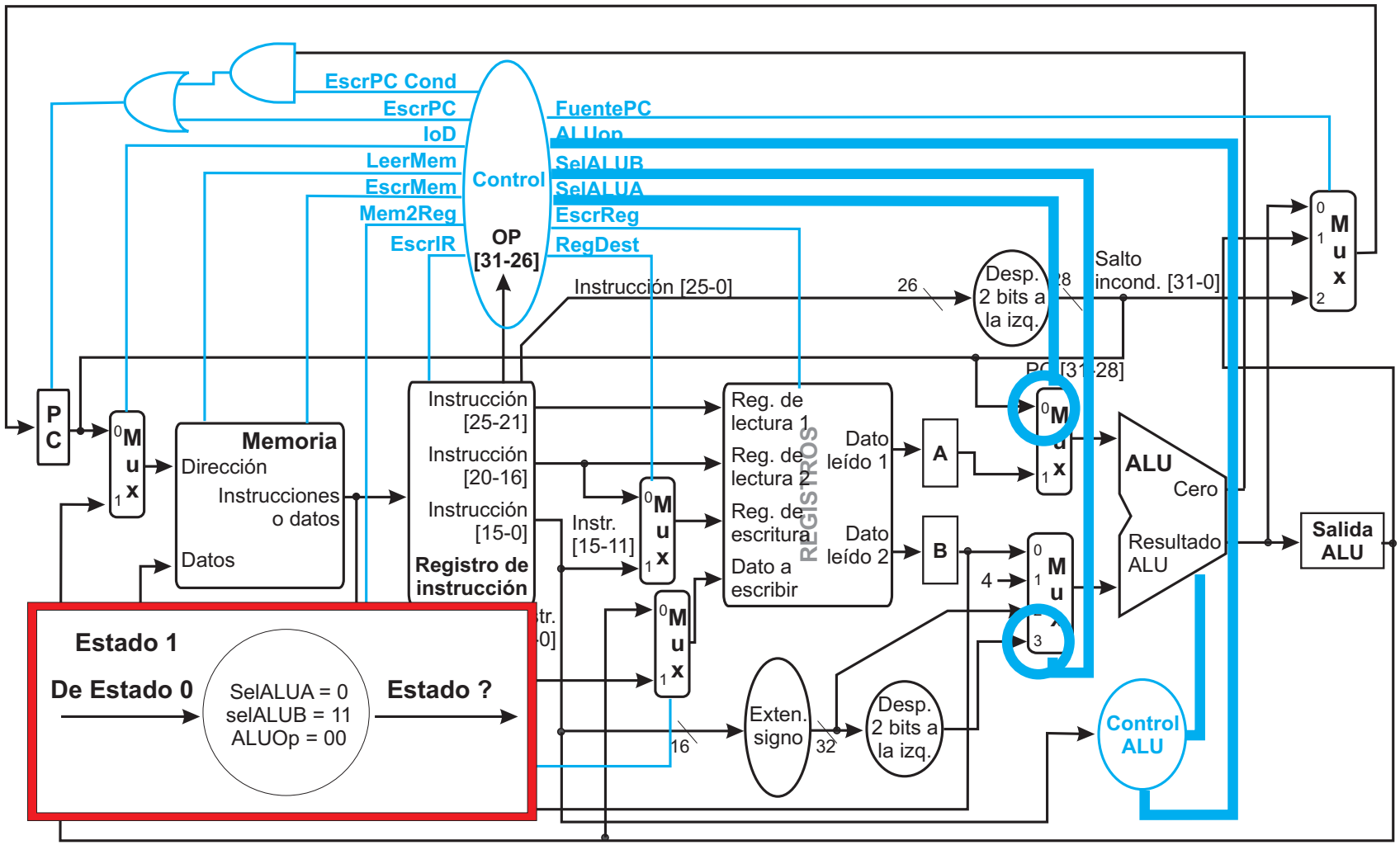


Etapa 2 – Decodificación de la instrucción y búsqueda de los registros



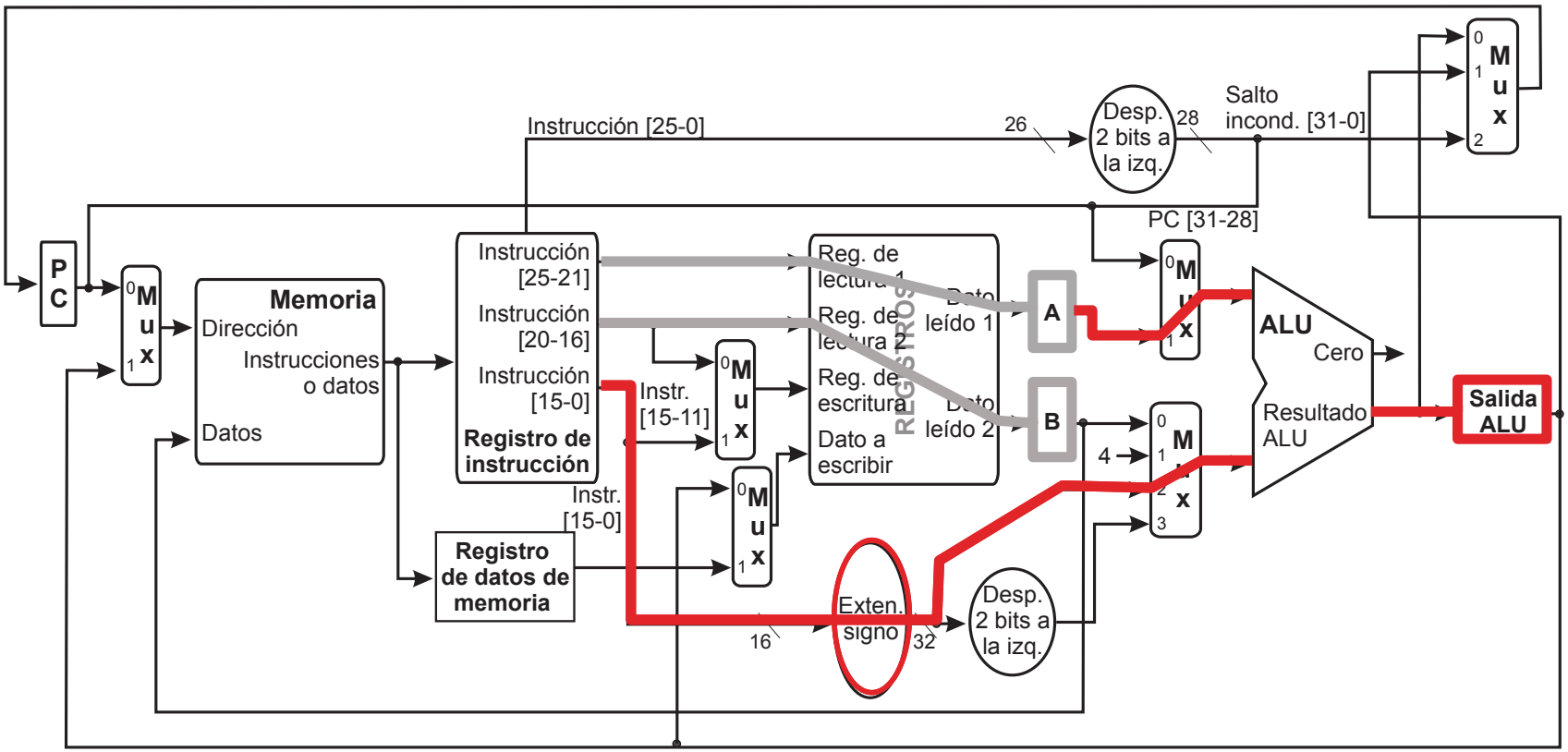
- Se decodifica la instrucción
- Se adelantan acciones que puedan ser útiles
  - Se leen los registros rt y rs y se almacenan en A y B.
  - Se calcula la dirección de salto potencial (beq).

Etapa 2 – Decodificación de la instrucción y búsqueda de los registros



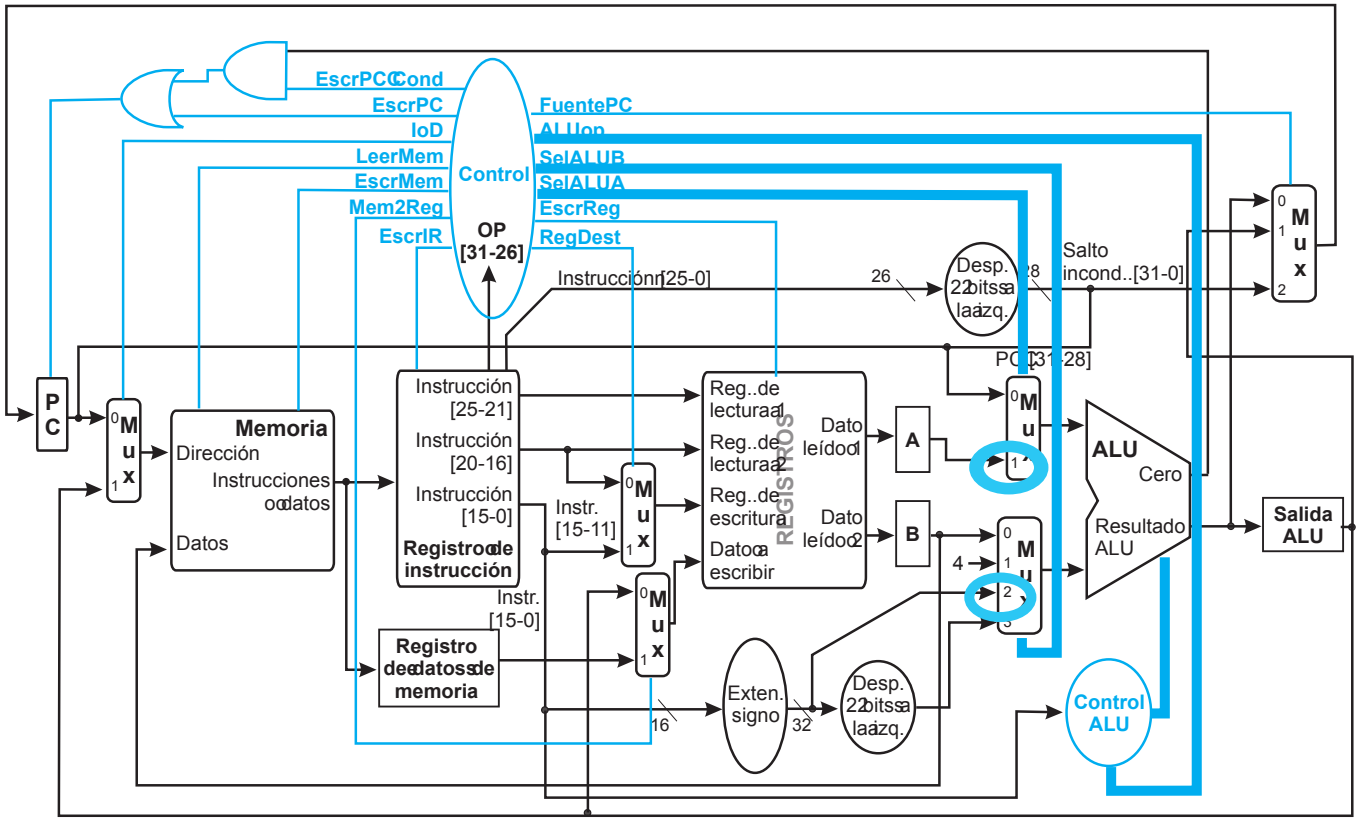
Instrucciones de acceso a memoria (`lw` o `sw`)

Etapa 3. Cálculo de la dirección de memoria



Instrucciones de acceso a memoria

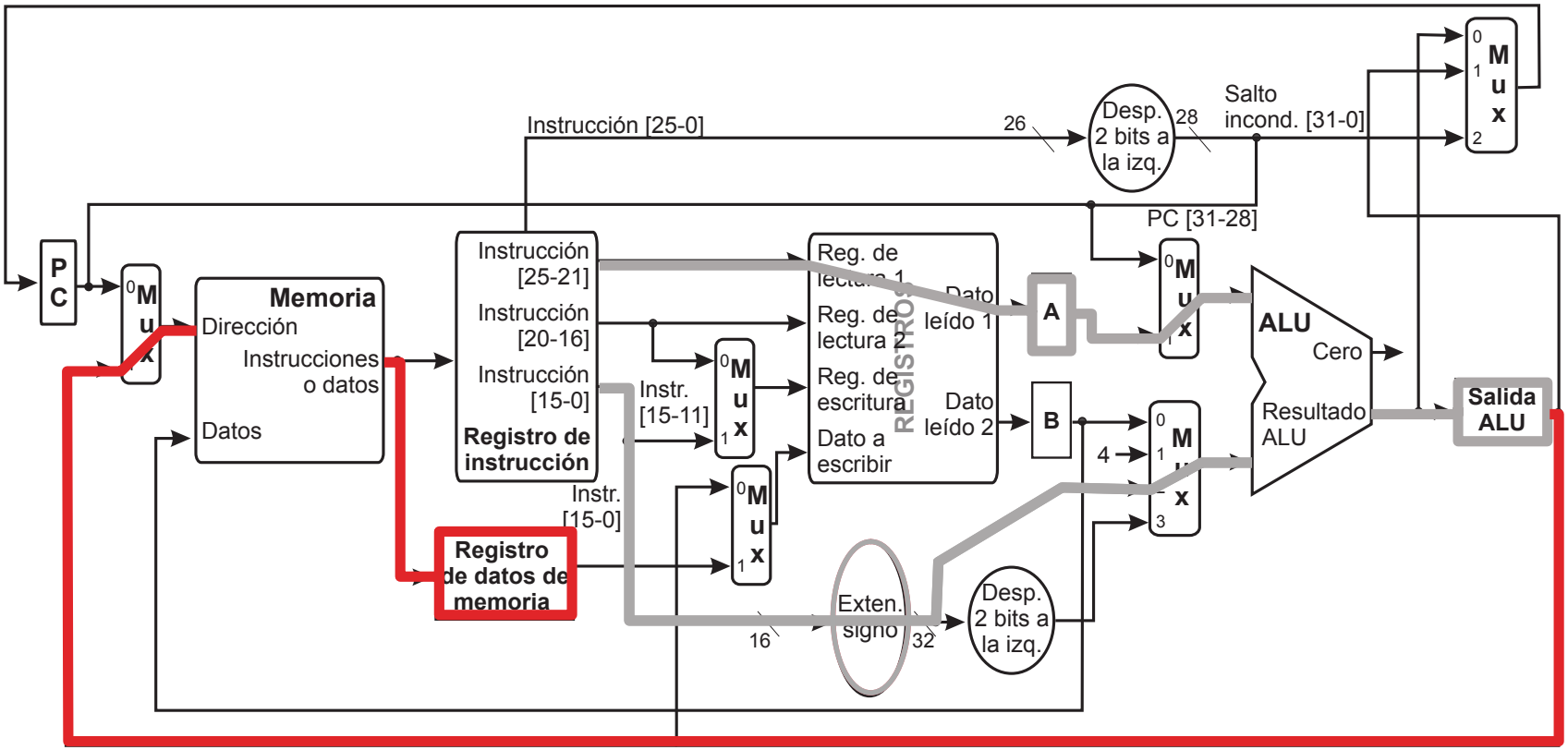
Etapa 3 (`lw` o `sw`). Cálculo de la dirección de memoria



- `SelALUA = 1`: Entrada A de la ALU es el registro A (registro base).
- `SelALUB = 10`: Entrada B es el campo desplazamiento con el signo extendido
- `ALUop = 00`: suma.

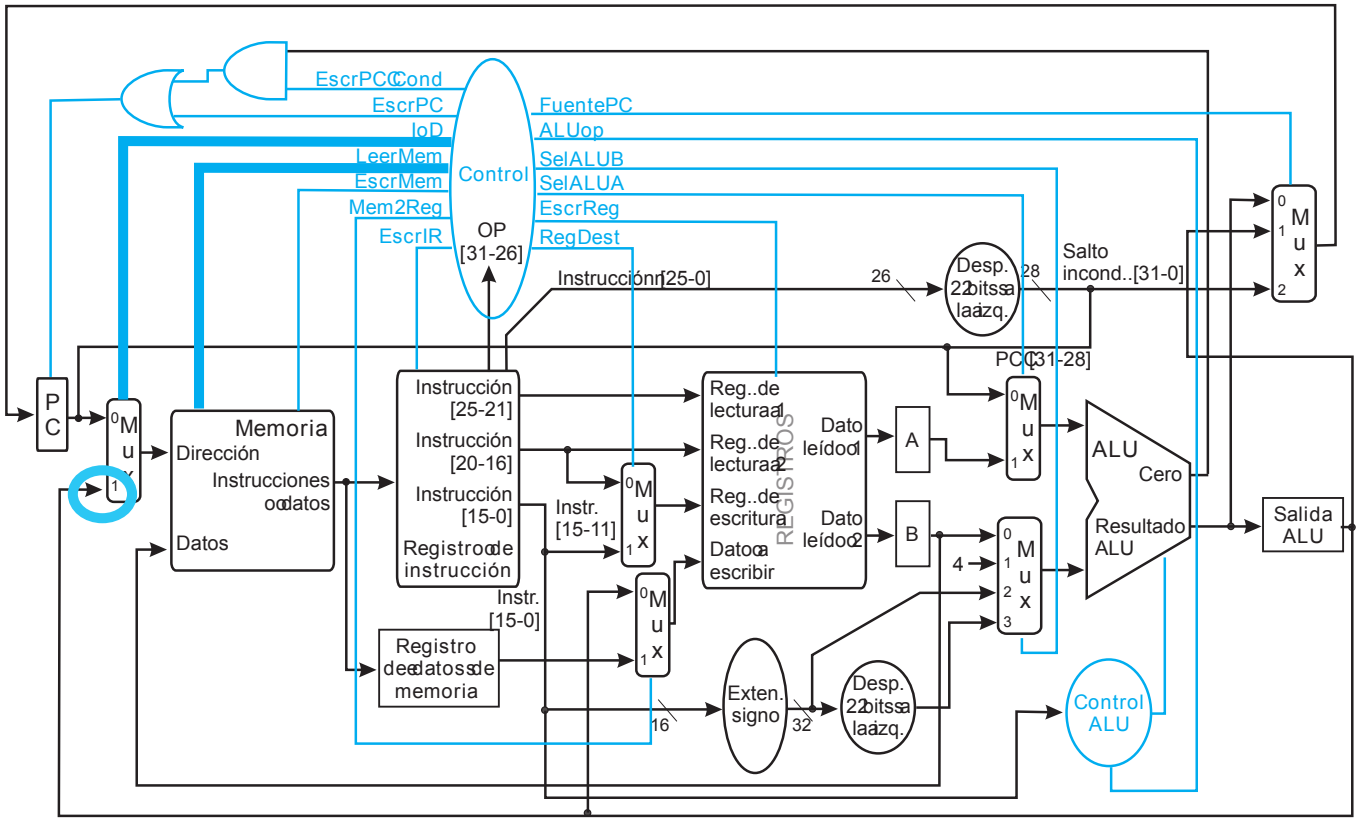
Instrucciones de acceso a memoria

Etapa 4 (lw). Acceso memoria



Instrucciones de acceso a memoria

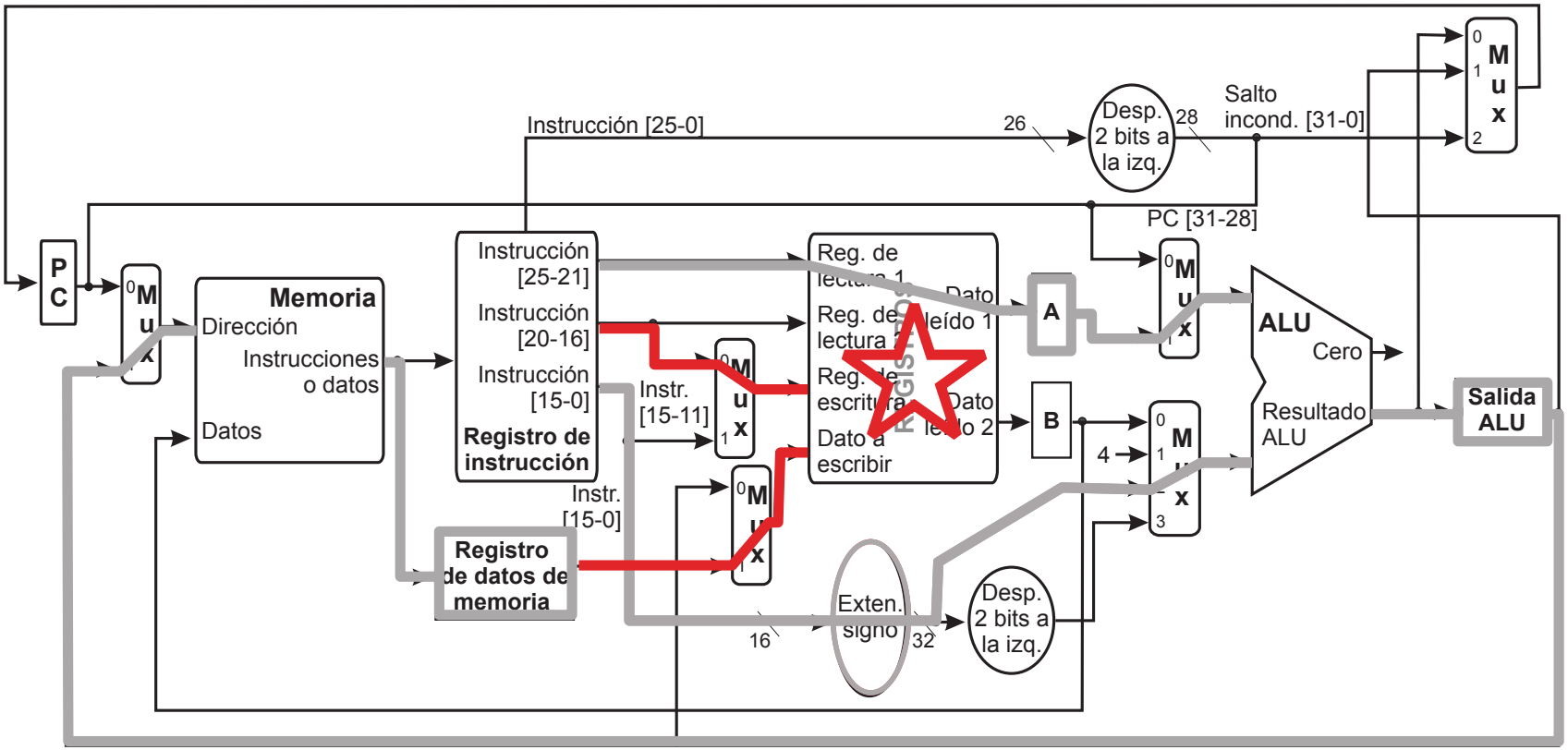
Etapa 4 (`lw`). Acceso memoria



- `LeerMem`: Activar la memoria para lectura.
- `IoD = 1`: Memoria direccionada por `SalidaALU`.
- El registro MDR no necesita señal de activación.

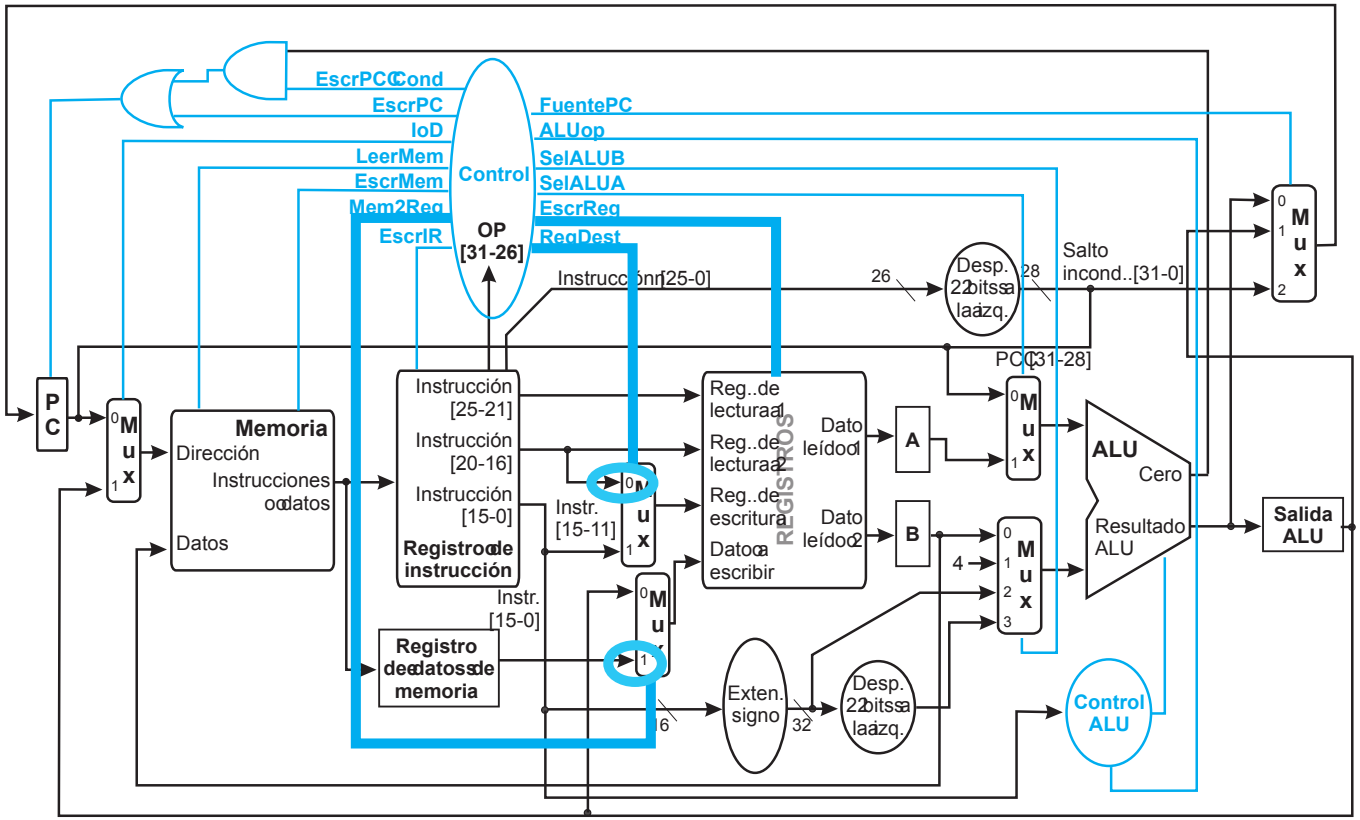
Instrucciones de acceso a memoria

Etapa 5 (`lw`): Escritura del dato de MDR al banco de registros



Instrucciones de acceso a memoria

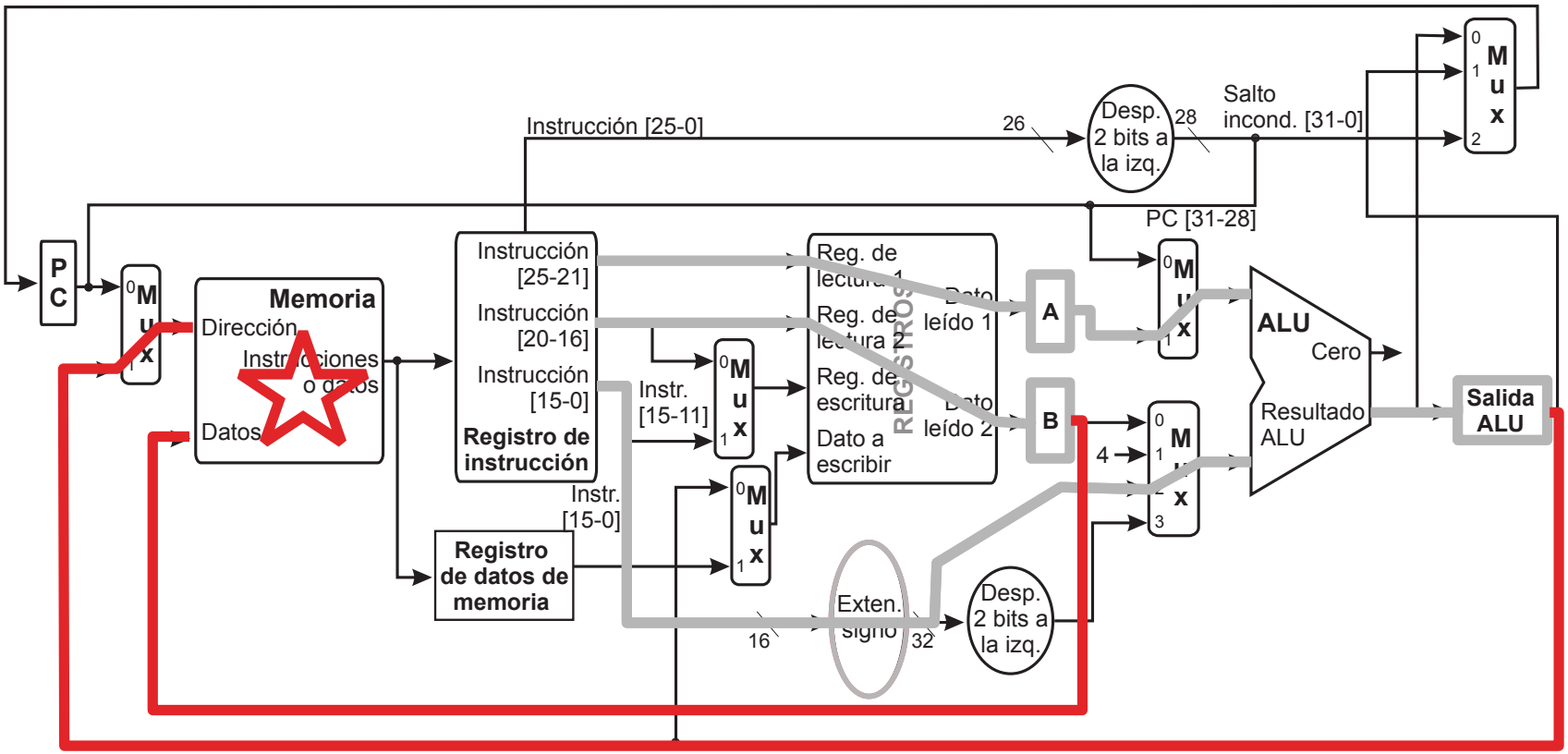
Etapa 5 (`lw`): Escritura del dato de MDR al banco de registros



- `EscrReg`: Activar la escritura en el banco de registros.
- `Mem2Reg = 1`: El dato a escribir proviene del MDR.
- `RegDest = 0`: El registro destino proviene de los bits 16–20 de la instrucción (campo `rt`).

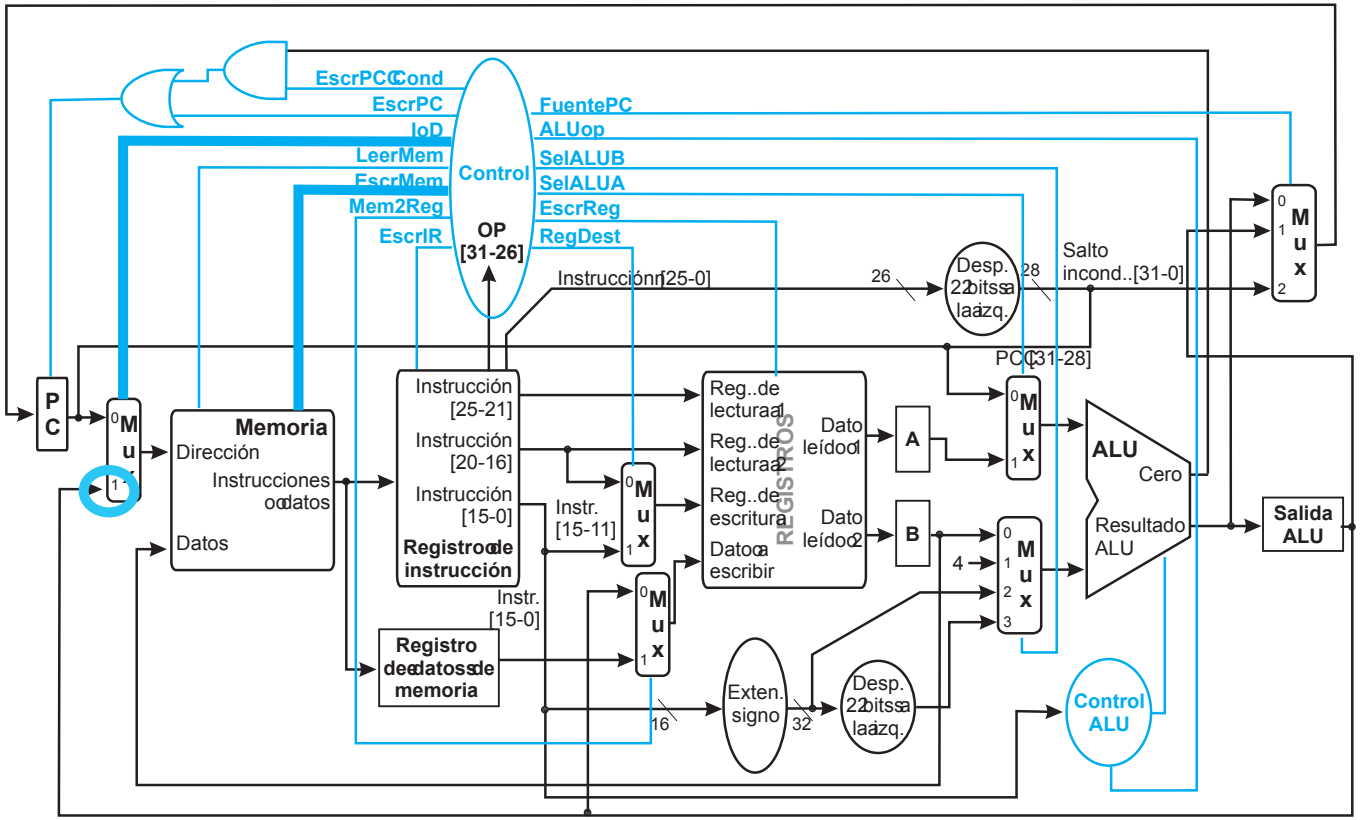
Instrucciones de acceso a memoria

Etapa 4 (sw): almacenamiento del dato en memoria



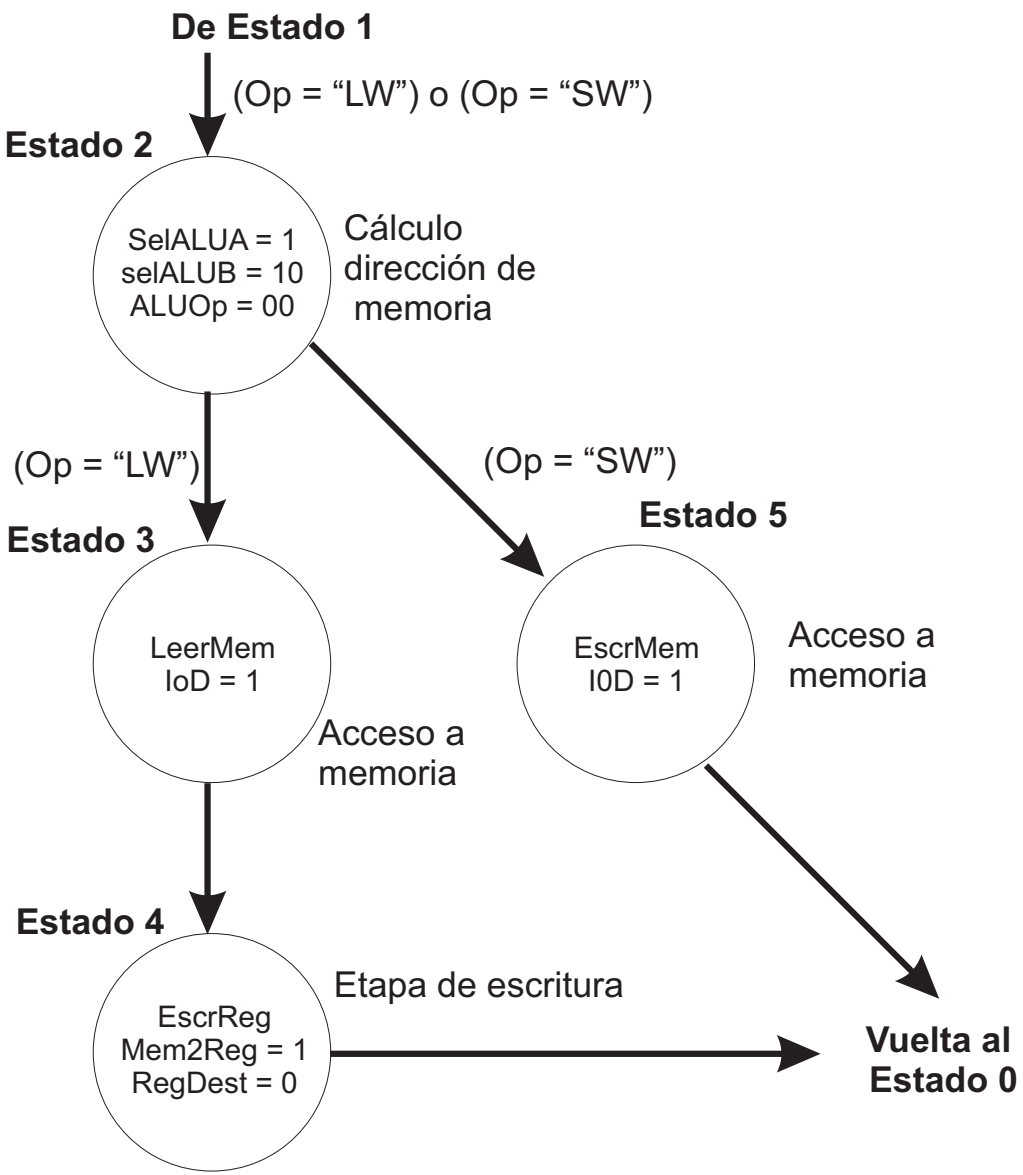
Instrucciones de acceso a memoria

Etapa 4 (`sw`): almacenamiento del dato en memoria



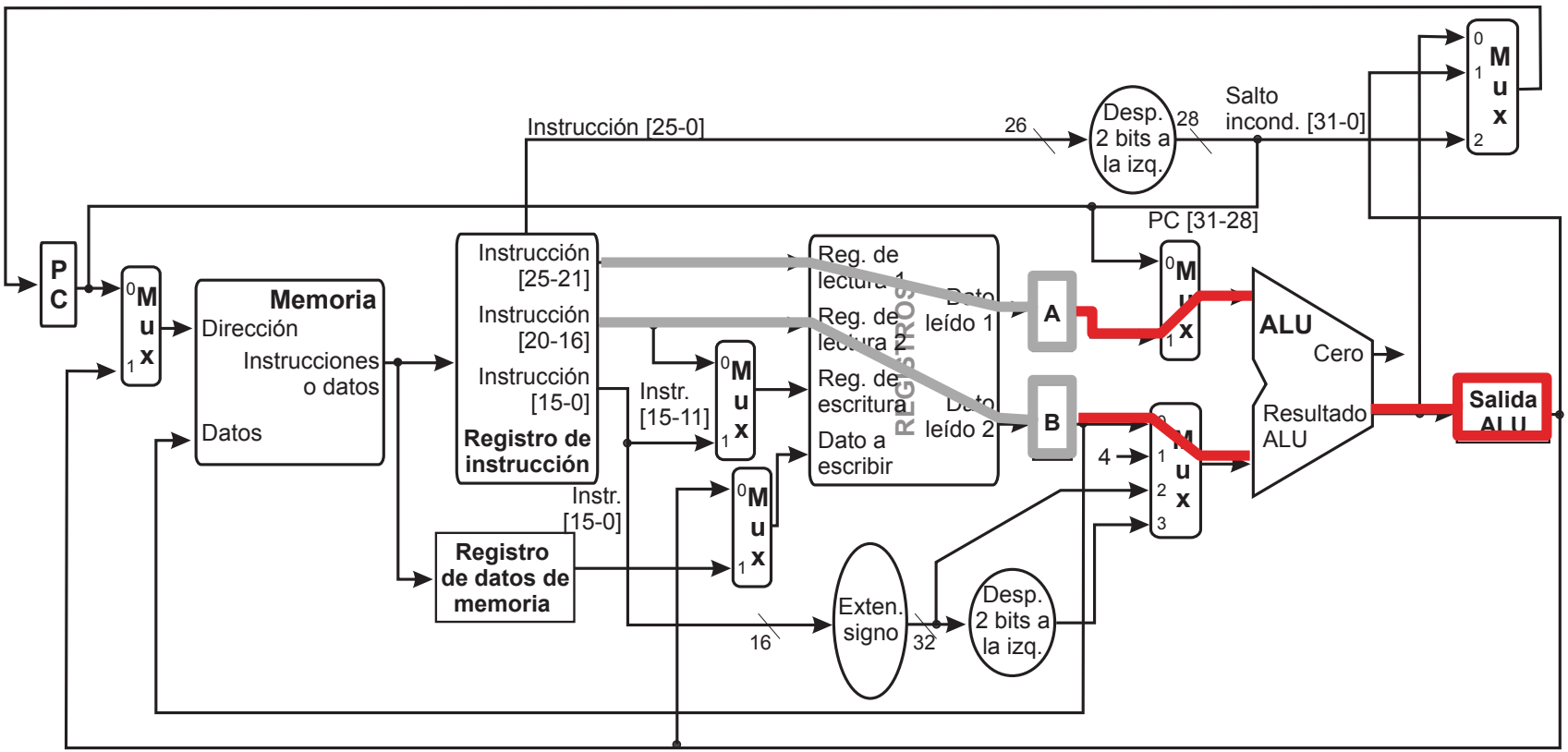
- `EscrMem`: Activar la escritura en memoria.
- `IoD = 1`: Memoria direccionada por `SalidaALU`.

Instrucciones de acceso a memoria (lw o sw)



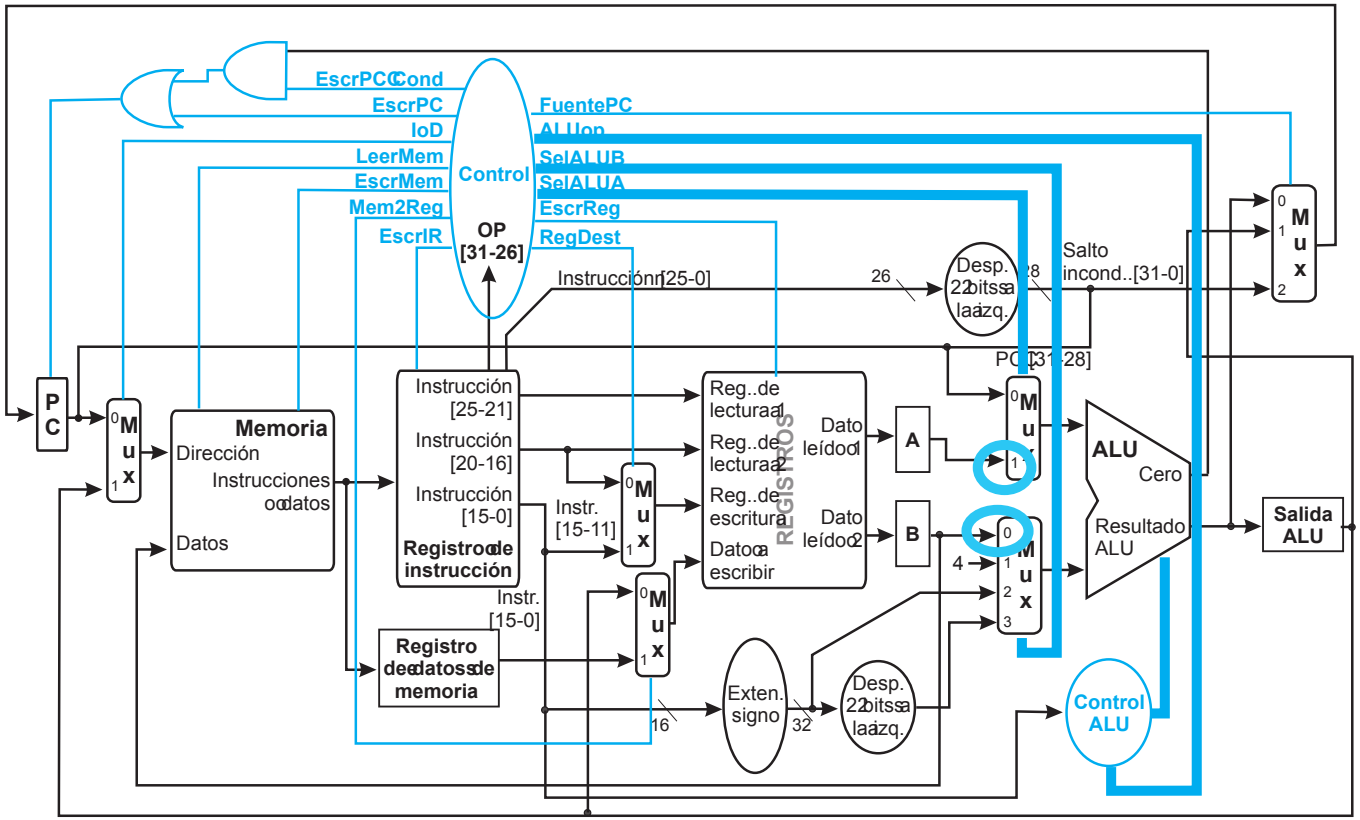
Instrucciones aritmético-lógicas

Etapa 3: Ejecución de la operación



Instrucciones aritmético-lógicas

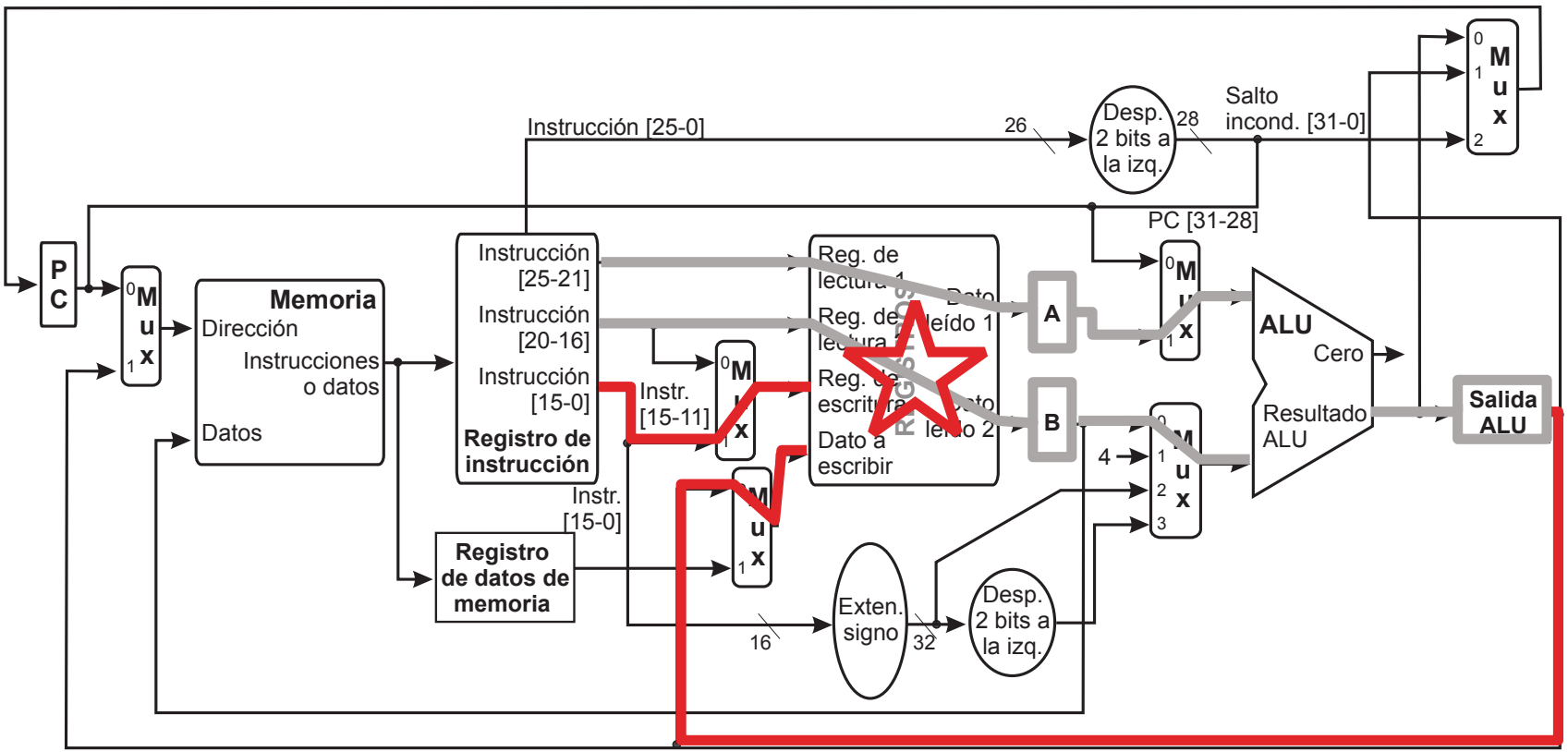
Etapa 3: Ejecución de la operación



- `SelALUA = 1`: Entrada A de la ALU es el registro A (operando `rs`).
- `SelALUB = 00`: Entrada B de la ALU es el registro B (operando `rt`).
- `ALUop = 10`: Operación determinada por el código de función.

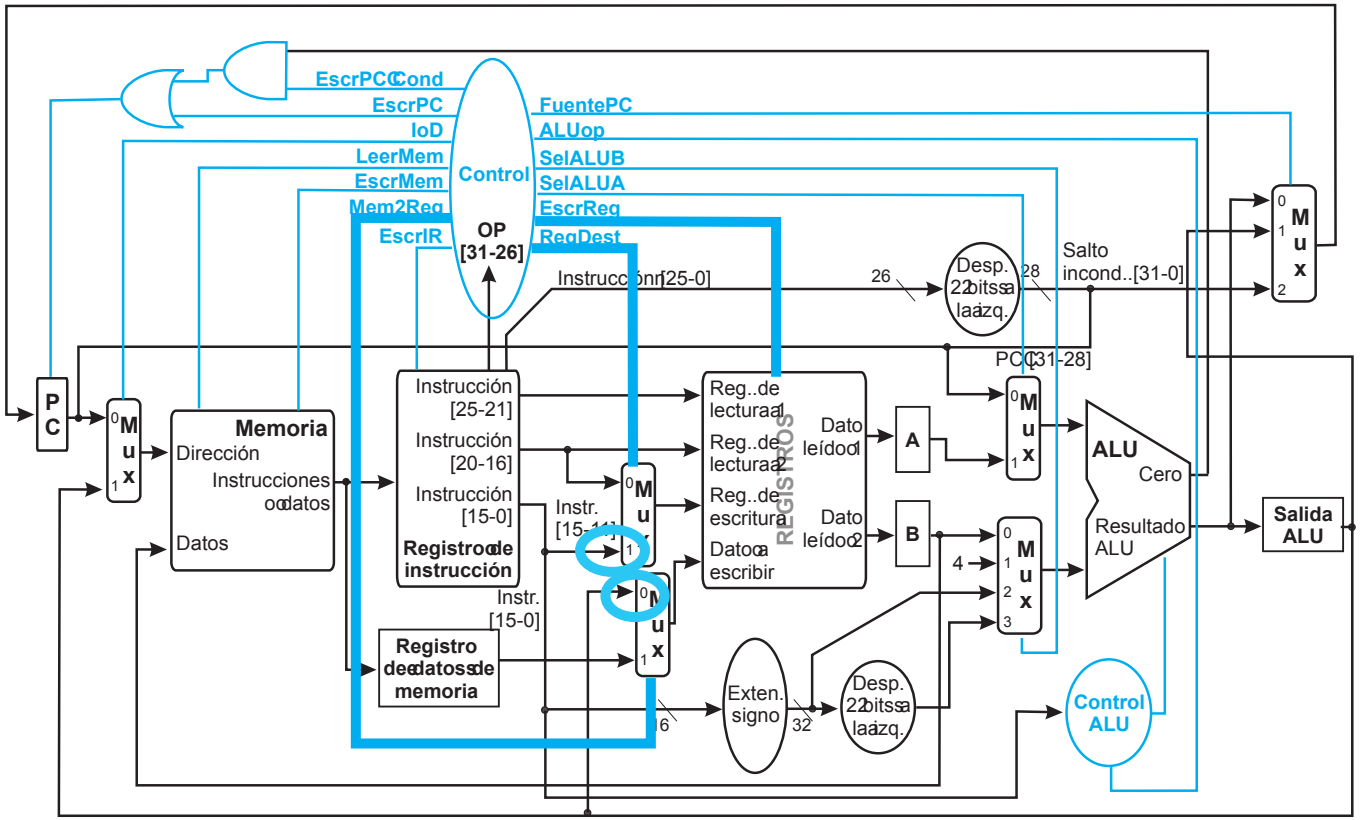
Instrucciones aritmético-lógicas

Etapa 4: Escritura del resultado



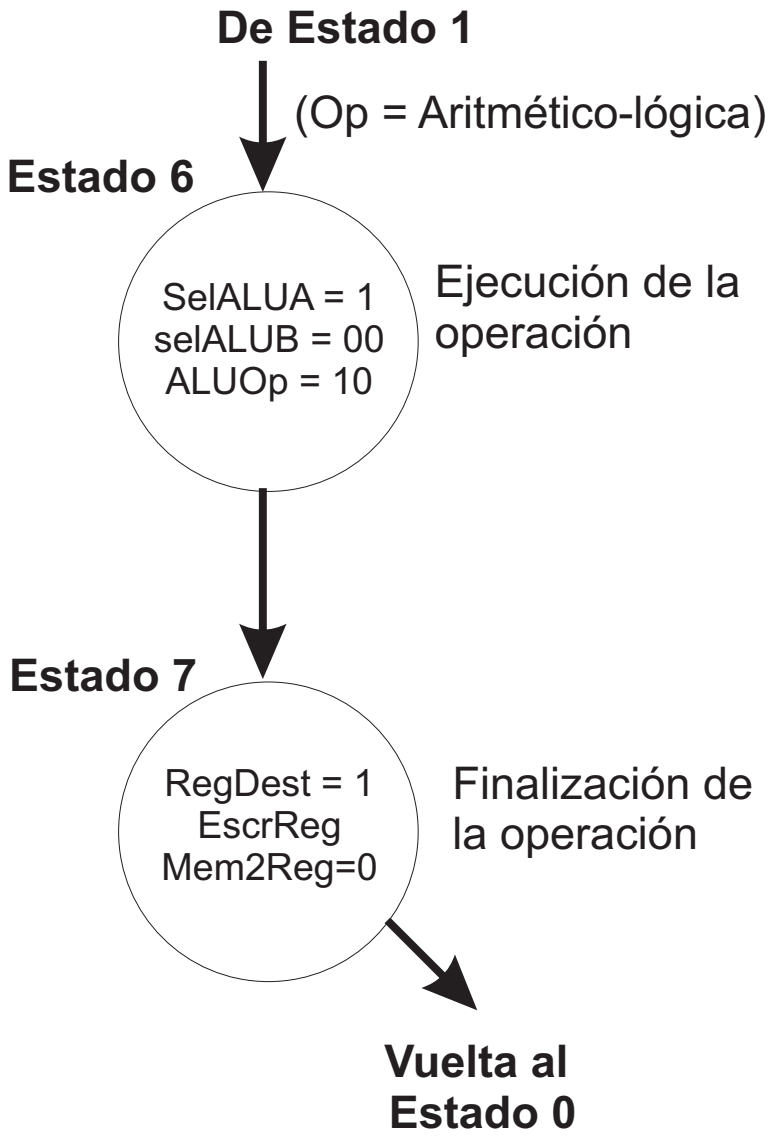
Instrucciones aritmético-lógicas

Etapa 4: Escritura del resultado



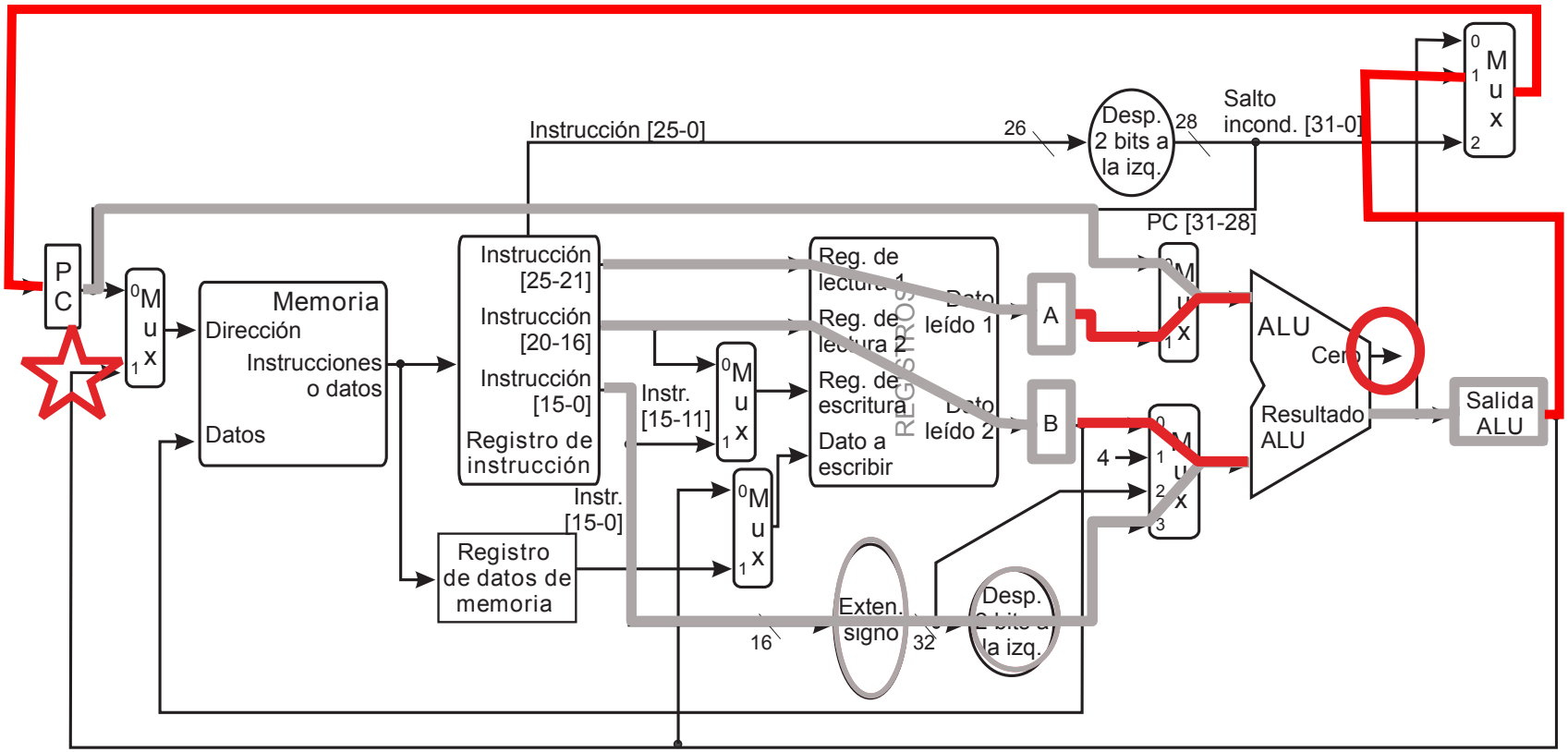
- `EscrReg`: Activa la escritura en el banco de registros.
- `Mem2Reg = 0`: El dato a escribir proviene de `SalidaALU`.
- `RegDest = 1`: El registro a escribir se encuentra en los bits 11–15 de la instrucción (campo `rd`).

Instrucciones aritmético-lógicas



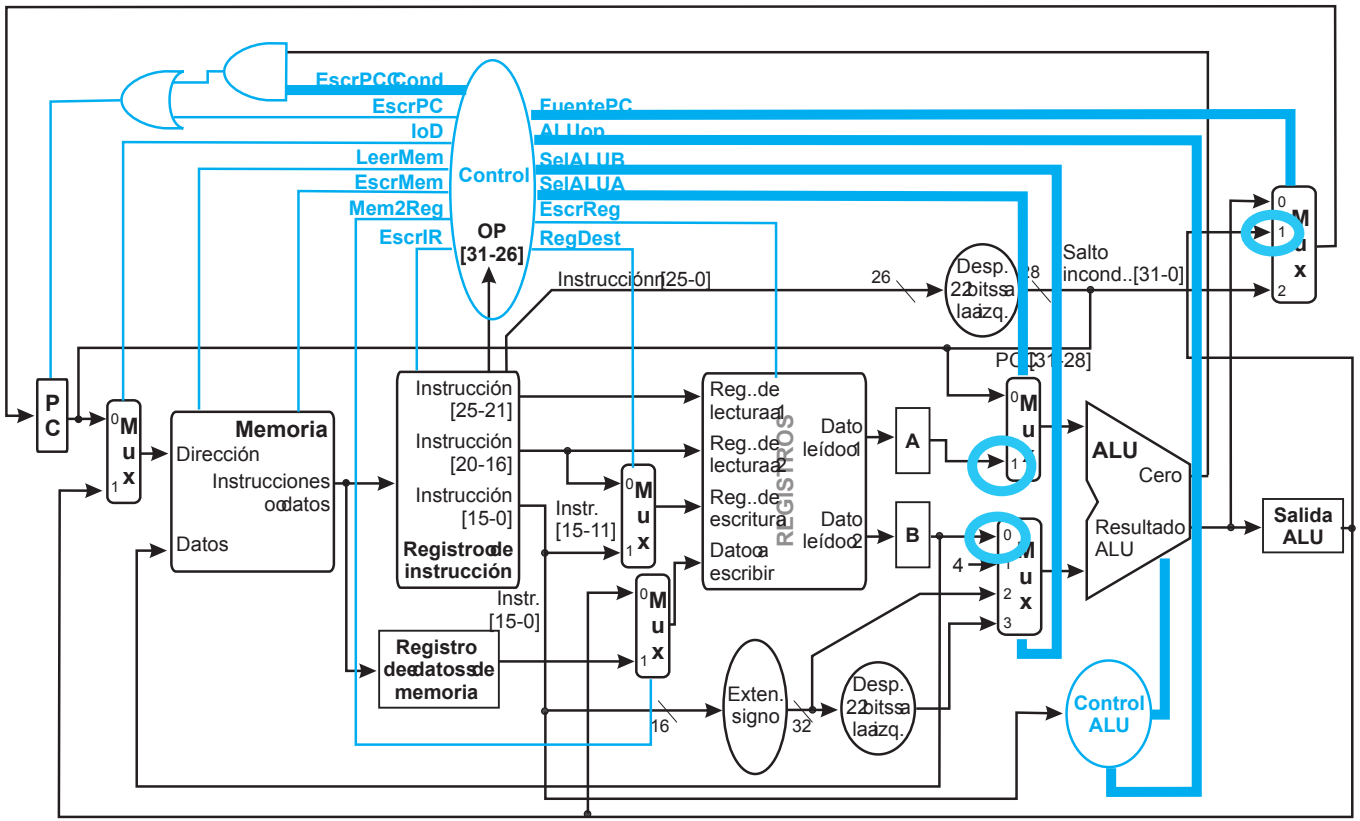
Instrucciones de salto condicional (beq)

Etapa 3: Comparación y determinación del nuevo PC



Instrucciones de salto condicional (beq)

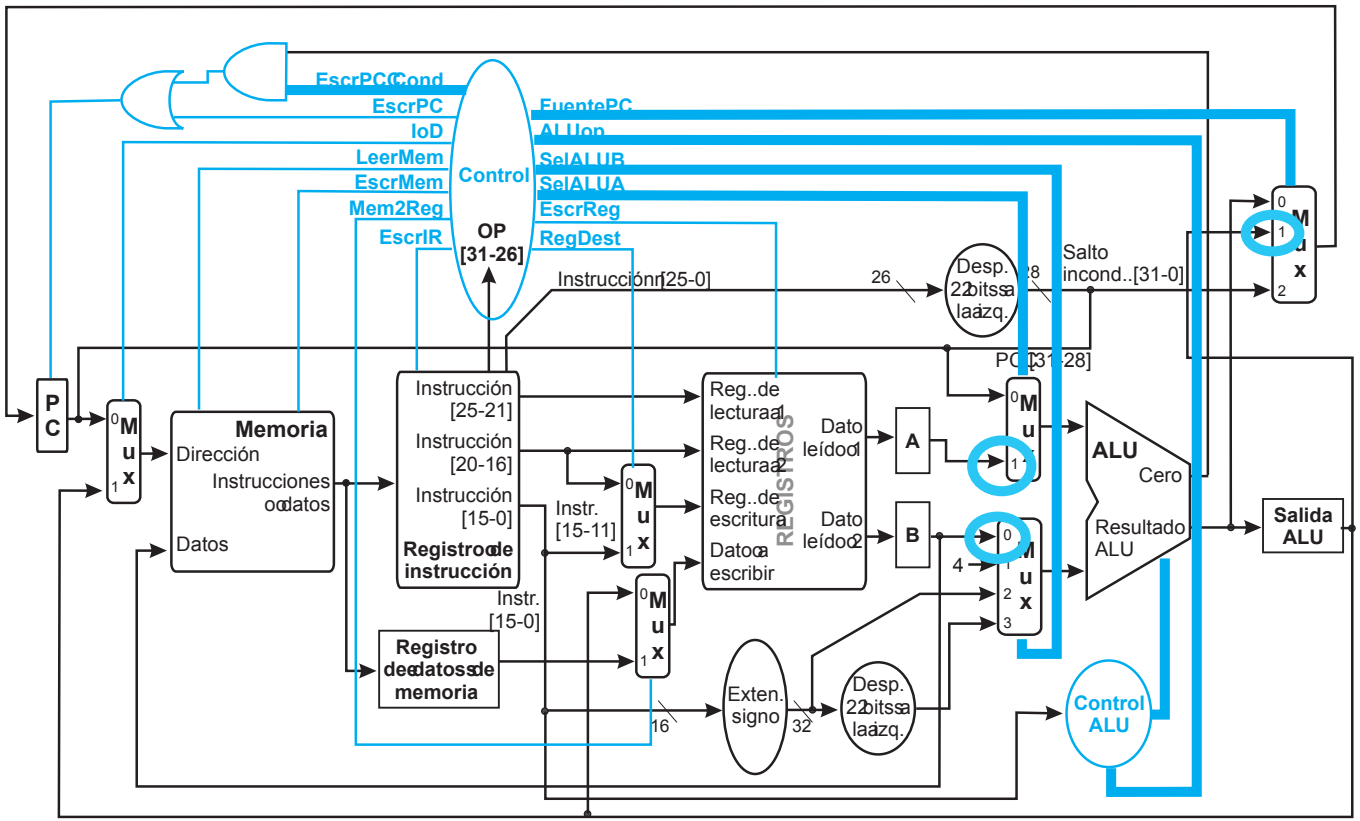
Etapa 3: Comparación y determinación del nuevo PC



- Comparación:
  - `SelALUA = 1`: Entrada A de la ALU es el registro A (`rs`).
  - `SelALUB = 00`: Entrada B de la ALU es el registro B (`rt`).
  - `ALUop = 01`: Resta.

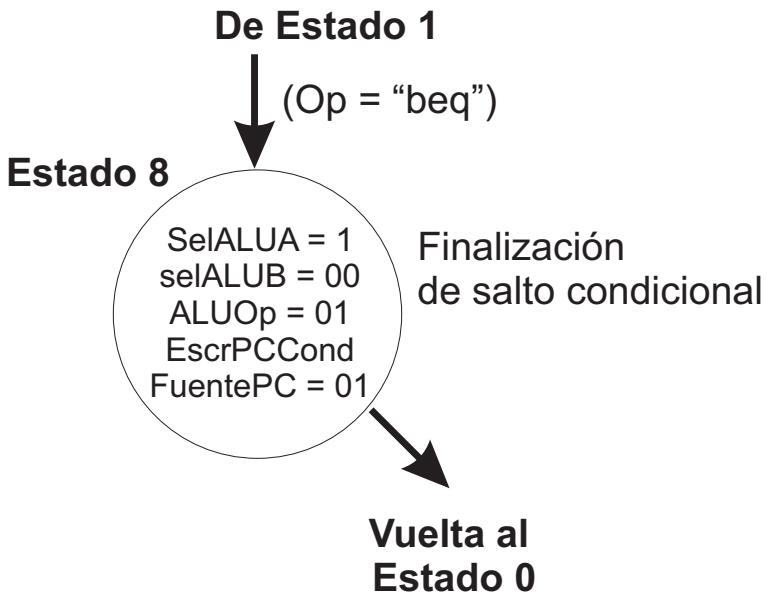
Instrucciones de salto condicional (beq)

Etapa 3: Comparación y determinación del nuevo PC



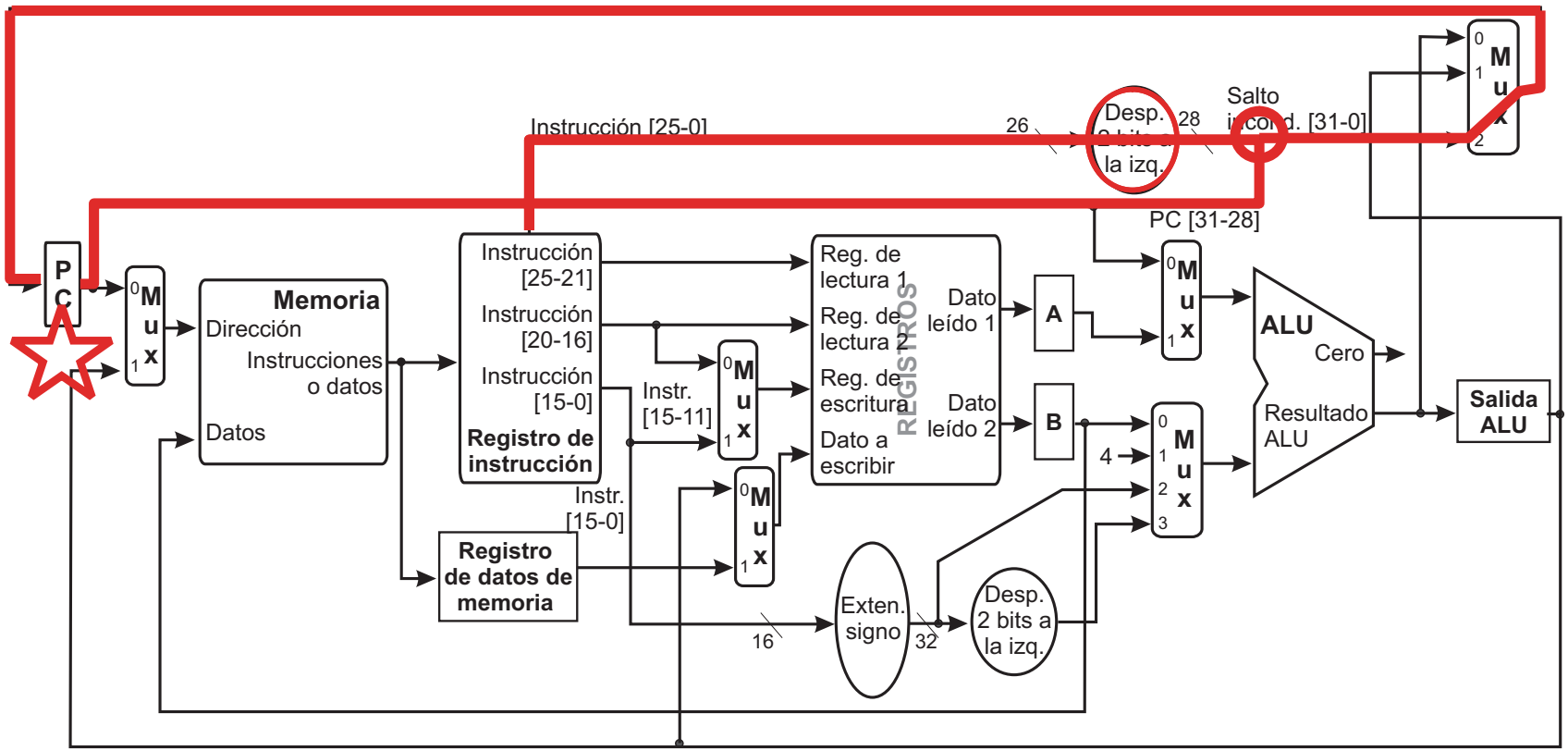
- Nuevo PC:
  - `EscrPCCond`: Activa la escritura del PC siempre y cuando, además, la salida Cero de la ALU sea 1.
  - `FuentePC = 01`: El nuevo PC se encuentra en `SalidaALU`.

Instrucciones de salto condicional (beq)



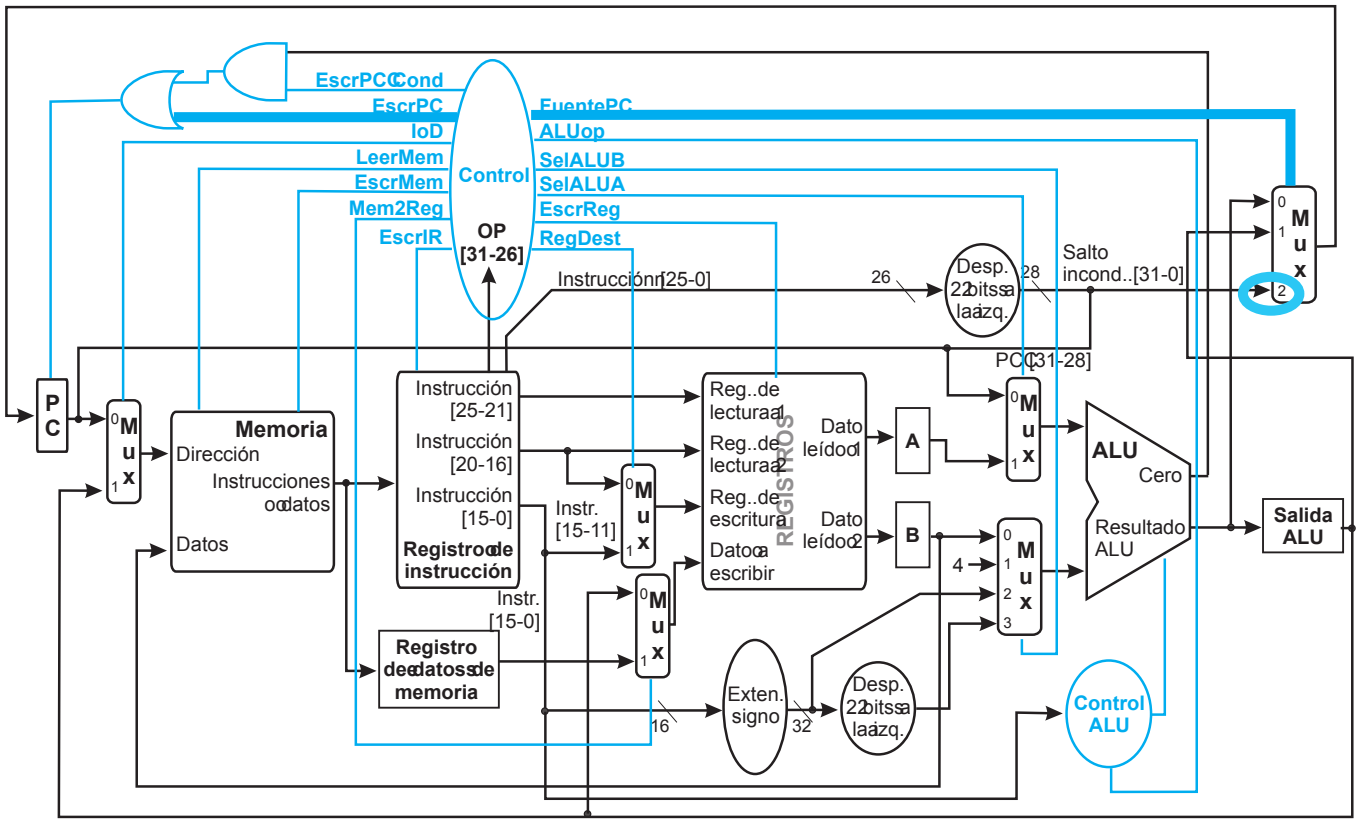
Instrucciones de salto incondicional (jump)

Etapa 3: Se escribe el nuevo PC



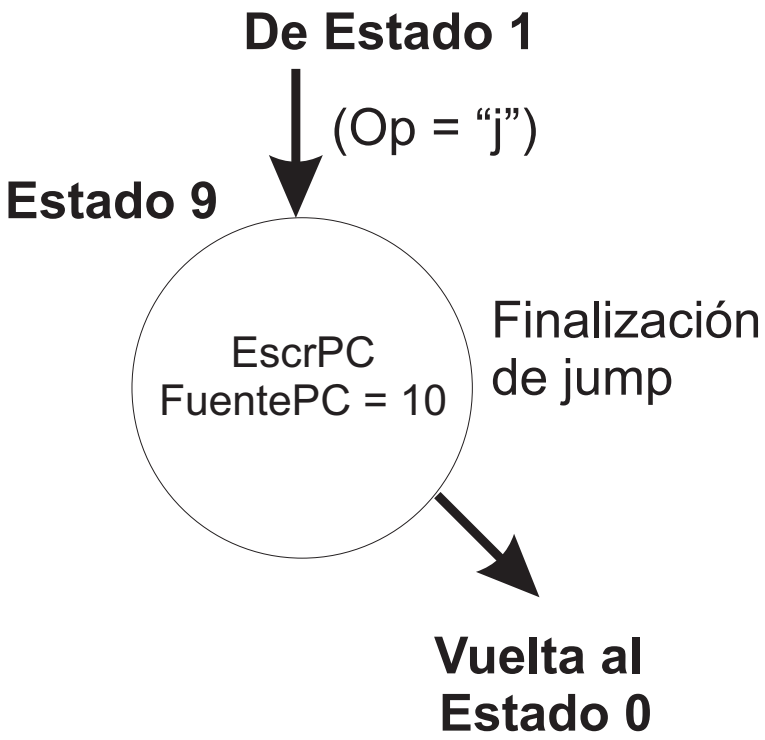
Instrucciones de salto incondicional (`jump`)

Etapa 3: Se escribe el nuevo PC

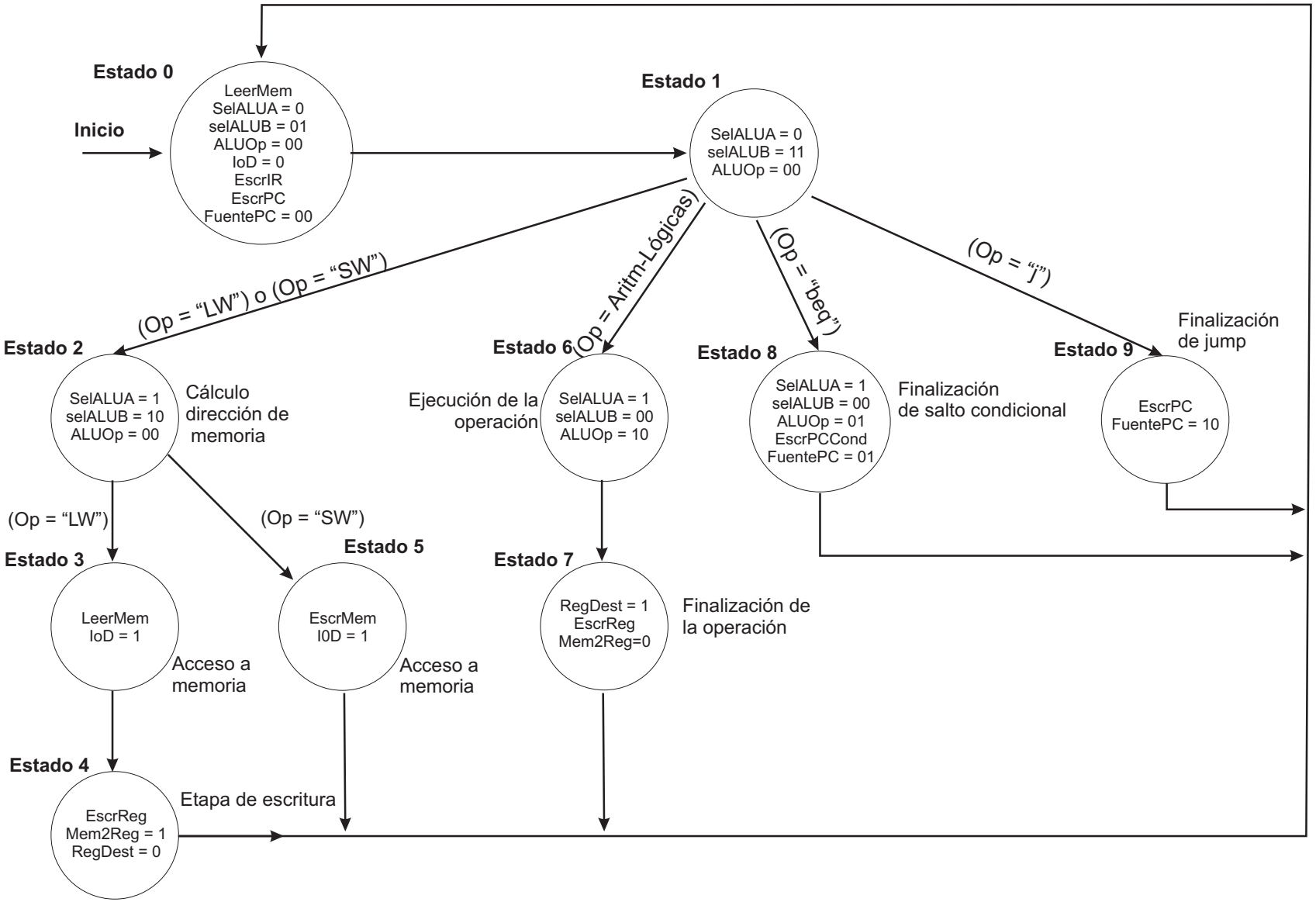


- `EscrPC`: Activar la escritura incondicional del PC.
- `FuentePC = 10`: El nuevo PC se obtiene tras desplazar dos bits a la izquierda los bits 0–25 de la instrucción, y concatenarlos con los 4 bits más significativos del PC.

Instrucciones de salto incondicional (jump)



Autómata de control completo



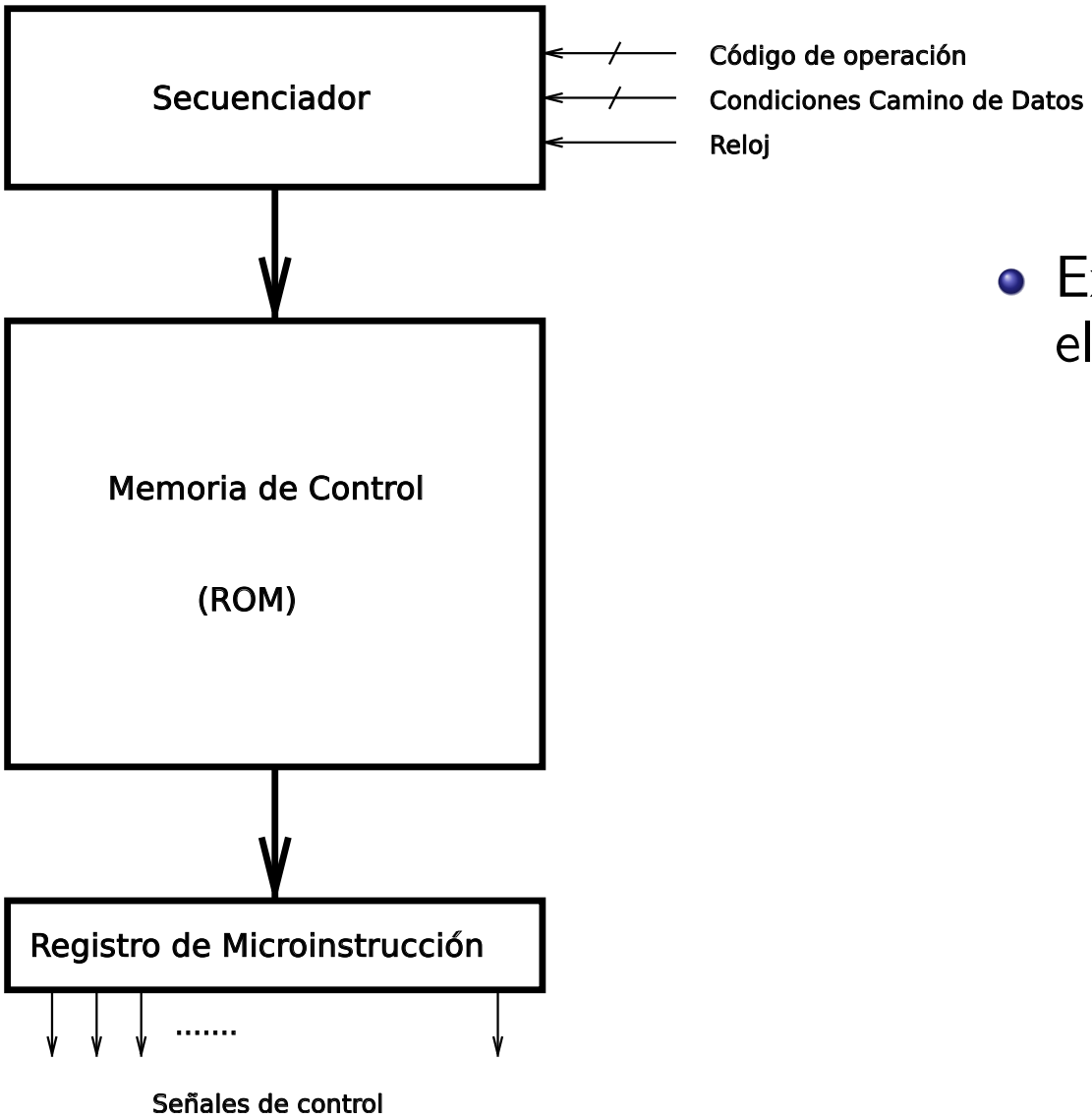
Autómata de control completo

- Dos técnicas diferentes de implementación:
  - **Unidad de control cableada**: se basa en alguno de los métodos clásicos de diseño
  - **Unidad de control microprogramada**: el autómata de control se representa en forma de programa de control
    - Más fácil de depurar, modificar y ampliar
    - Mejor opción para sistemas grandes y/o complejos

Definiciones

- El automáta se transforma en un programa de control
- Las instrucciones del programa de control se denominan **microinstrucciones**
- Cada microinstrucción define el conjunto de señales de control que se deben activar en un estado determinado
- El conjunto de las microinstrucciones se denomina **microprograma**
- El microprograma se almacenará en una memoria de solo lectura

Esquema



- Existen 2 formas de realizar el secuenciamiento:
  - Explícito: cada microinstrucción incluye la dirección de la microinstrucción siguiente
  - Implícito: existen dos tipos de microinstrucción, de control y de salto

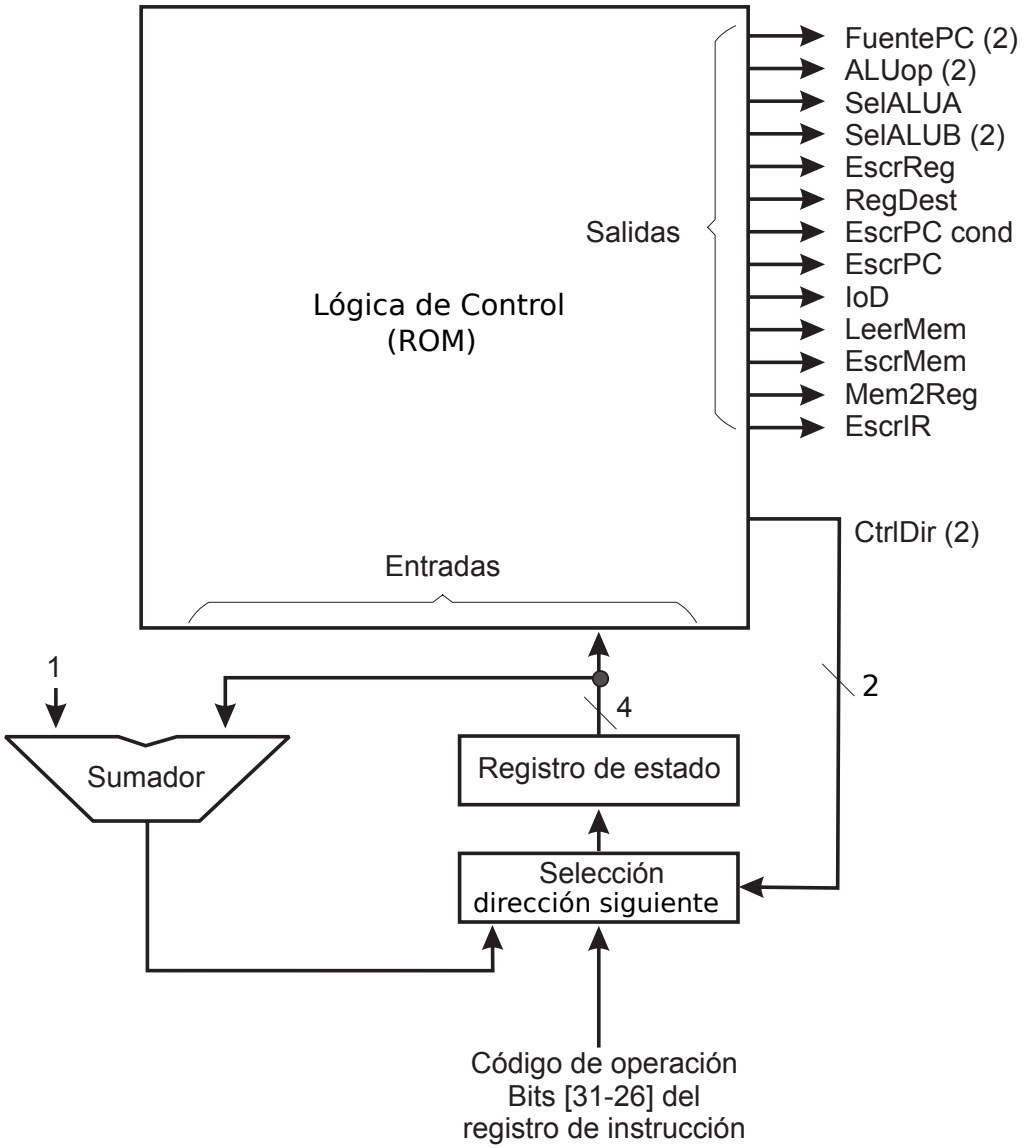
Secuenciamiento

- Vamos a implemetar la UC microprograma para el conjunto básico de instrucciones MIPS considerando secuenciamiento explícito
- Cada estado del autómata de control se transformará en una microinstrucción
- Elección de la siguiente microinstrucción, tres opciones:
  - Incrementar la dirección de la microinstrucción actual
  - Saltar a la primera microinstrucción para iniciar la ejecución de una nueva instrucción
  - Elegir la siguiente microinstrucción en función del estado en el que nos encontremos y la instrucción que estemos ejecutando
    - Tabla de envío: tabla con las direcciones destino
    - Una tabla para cada estado con múltiples estados destino

Secuenciamiento

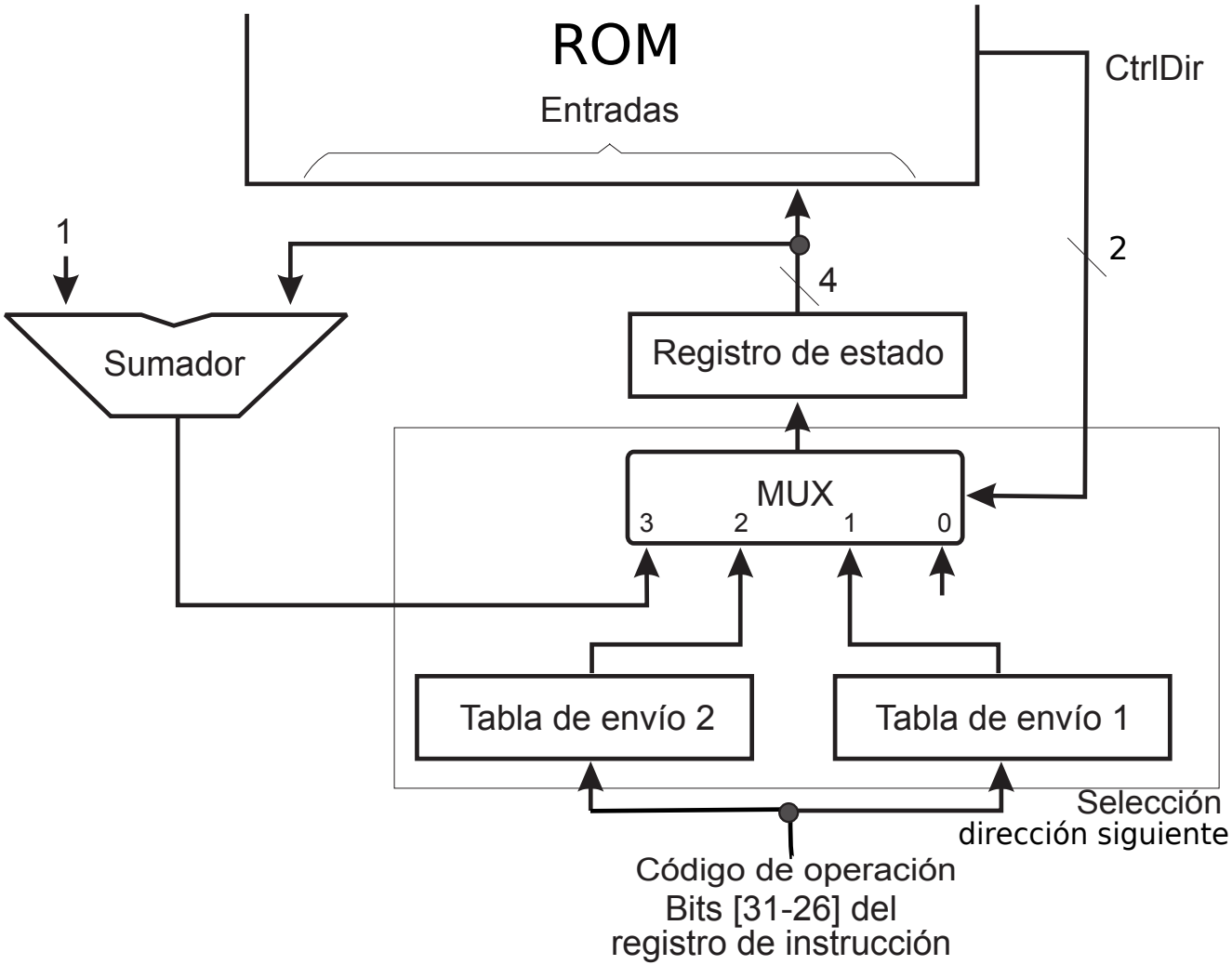
- Para la implementación de la UC del MIPS necesitamos dos tablas de envío
- Se necesitan 2 bits (CtrlDir) para especificar la siguiente microinstrucción
  - CtrlDir=00: Salta a la primera microinstrucción para comenzar una nueva instrucción
  - CtrlDir=01: Utiliza la tabla de envío 1
  - CtrlDir=10: Utiliza la tabla de envío 2
  - CtrlDir=11: Selecciona la siguiente microinstrucción

Implementación



- Cada punto de control del camino de datos se corresponde con un bit de las microinstrucciones
  - Cada microinstrucción consta de 18 bits
  - Los puntos de control podrían estar codificados para ahorrar bits

Implementación



Implementación

	EscrPC	EscrPCCond	IoD	LeerMem	EscrMem	EscrIR	Mem2Reg	FuentePC (2)	ALUOp (2)	SelALUB (2)	SelALUA	EscrReg	RegDest	CtrlDir (2)	
	1		0	1		1		0	0	1	0			3	estado 0
									0	3	0			1	estado 1
Mem1									0	2	1			2	estado 2
Lw2			1	1										3	estado 3
							1					1	0	0	estado 4
Sw2			1		1									0	estado 5
Rformat1									2	0	1			3	estado 6
							0					1	1	0	estado 7
Beq1		1						1	1	0	1			0	estado 8
Jump1	1							2						0	estado 9

Implementación

Tabla 1 de envío de microcódigo

Campo del código de operación	Nombre del código de operación	Valor
000000	FormatoR	Rformat1
000010	`jmp`	JUMP1
000100	`beq`	BEQ1
100011	`lw`	Mem1
101011	`sw`	Mem1

Tabla 2 de envío de microcódigo

Campo del código de operación	Nombre del código de operación	Valor
100011	`lw`	LW2
101011	`sw`	SW2

Temporización

Etapas en la ejecución de instrucciones

Nombre de la etapa	Acción para instrucciones del tipo R	Acción para instrucciones de acceso a memoria	Acción para saltos condicionales	Acción para instrucciones *Jump*
Carga de instrucción	IR = Memoria[PC] PC = PC + 4			
Decodificación de instrucciones/carga de los registros	A = Reg [IR[25-21]] B = Reg [IR[20-16]] SalidaALU = PC + (extensión-signo (IR[15-0]) << 2)			
Ejecución, cálculo de direcciones y finalización de saltos condicionales/*jump*	SalidaALU = A op B	SalidaALU = A + extensión-signo (IR[15-0])	si (A == B) entonces PC = SalidaALU	PC = PC [31-28]|| (IR[25-0]<<2)
Acceso a memoria y finalización de instrucciones de tipo R	Reg [IR[15-11]] = SalidaALU	*Load*: MDR = Memoria[SalidaALU] o *Store*: Memoria[SalidaALU] = B		
Finalización de la lectura de memoria		*Load*: Reg[IR[20-16]] = MDR		

Temporización
Ciclo de reloj

- El ciclo de reloj debería ser al menos tan largo como la más larga de las operaciones del camino de datos
- Ejemplo:
  - lectura de memoria: 2.5 ns
  - escritura en memoria: 3 ns
  - operación ALU: 2ns
  - acceso registros (lectura y escritura): 1ns
- Ciclo de reloj $\geqq$ 3ns
- Instrucción aritmética $\geqq$ 12 ns (4×3)

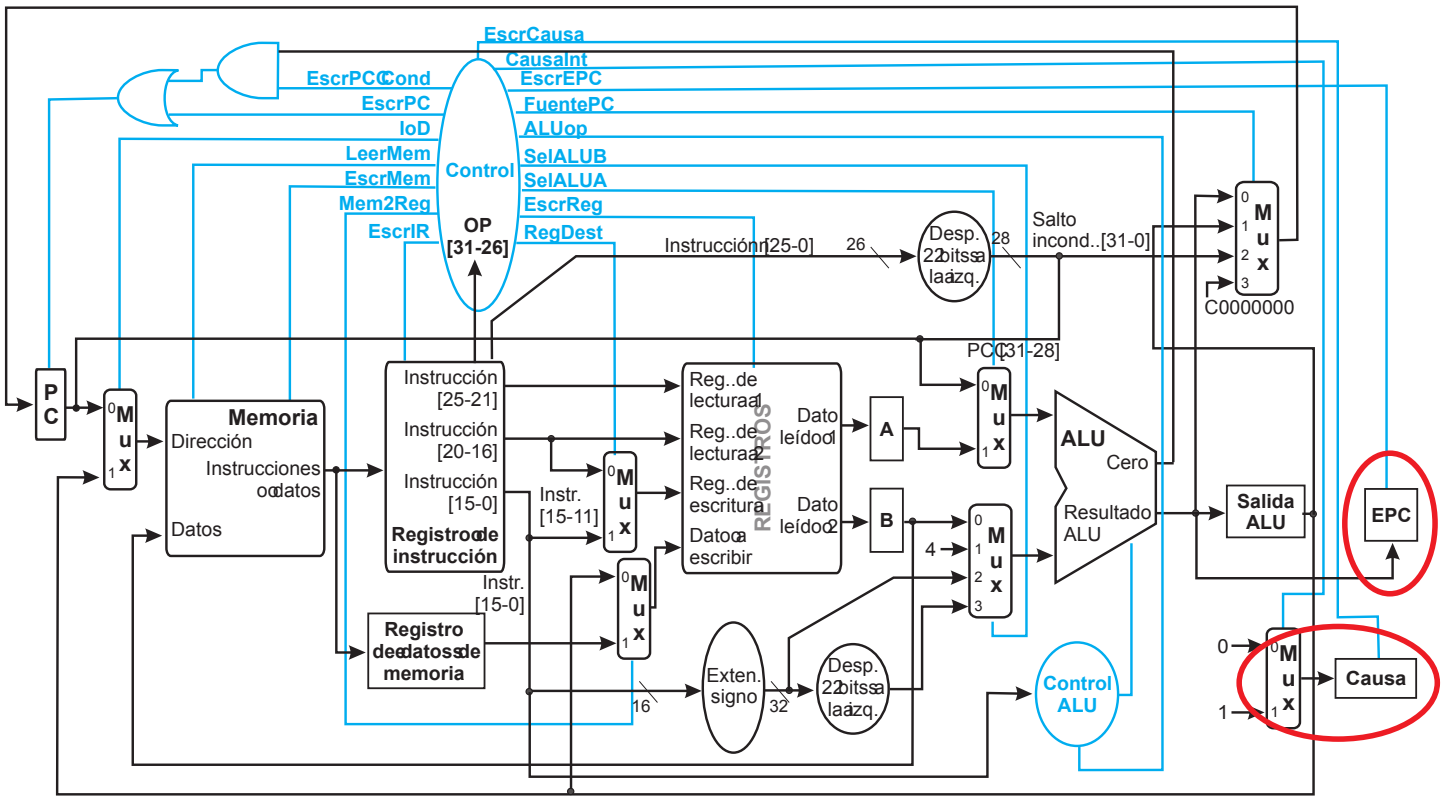
Excepciones

- Excepción: Suceso inesperado ocurrido en el procesador.
- Acciones a llevar a cabo ante una excepción:
  - Guardar la dirección de la instrucción causante en el registro contador de programa de excepción (EPC).
  - Transferir el control al SO, que ejecutará alguna rutina específica.
  - Finalizar el programa o continuar su ejecución (utilizando EPC).

Excepciones

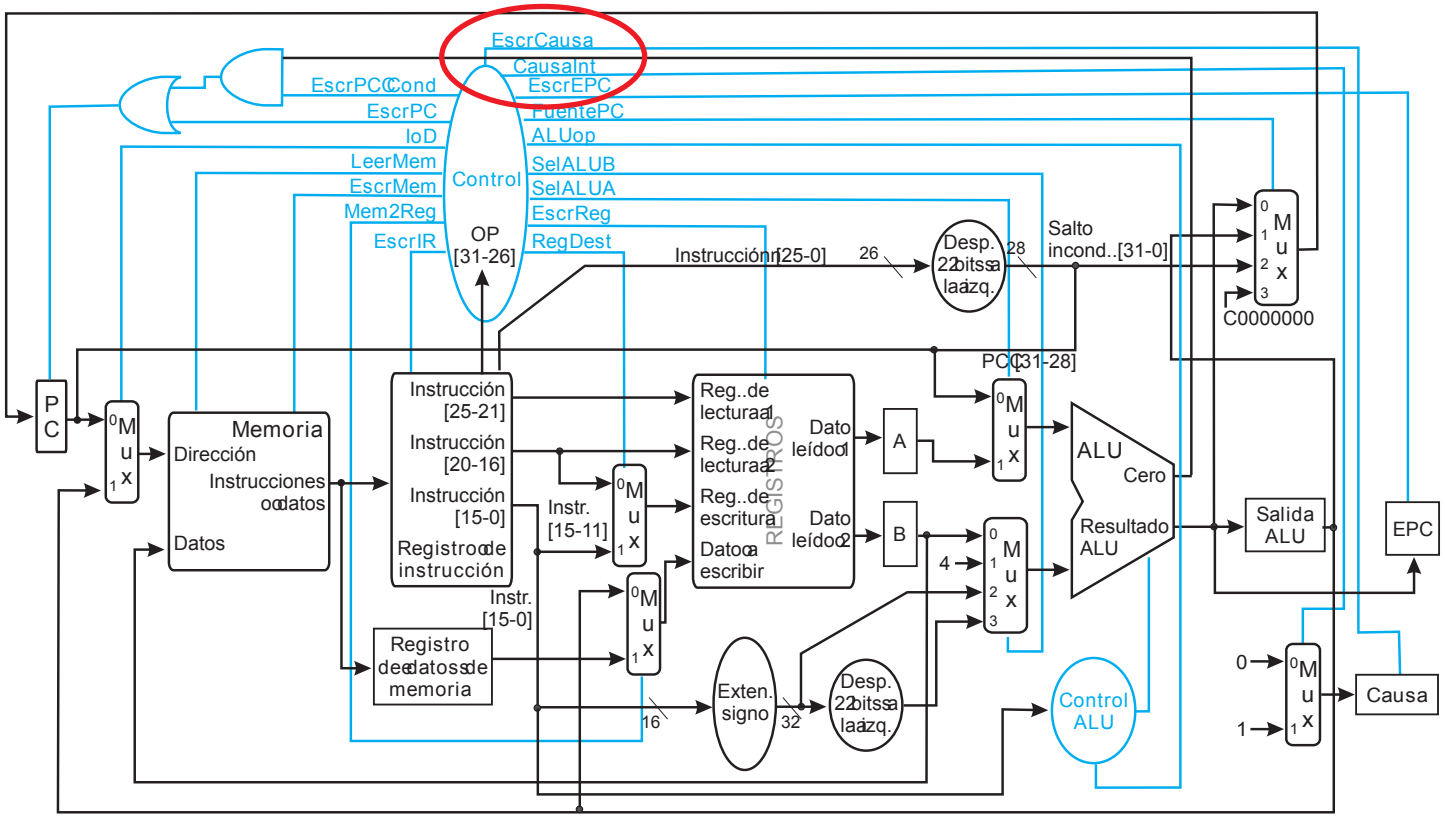
- Tipos de excepciones en nuestra implementación MIPS:
  - Ejecución de una instrucción no definida.
  - Desbordamiento aritmético.
- La rutina del SO a ejecutar dependerá del tipo de excepción.
- En el MIPS se incluye un registro denominado Registro de Causa (Causa) para almacenar la razón de la excepción
  - Causa=1: instrucción no definida
  - Causa=0: desbordamiento aritmético

Camino de datos modificado



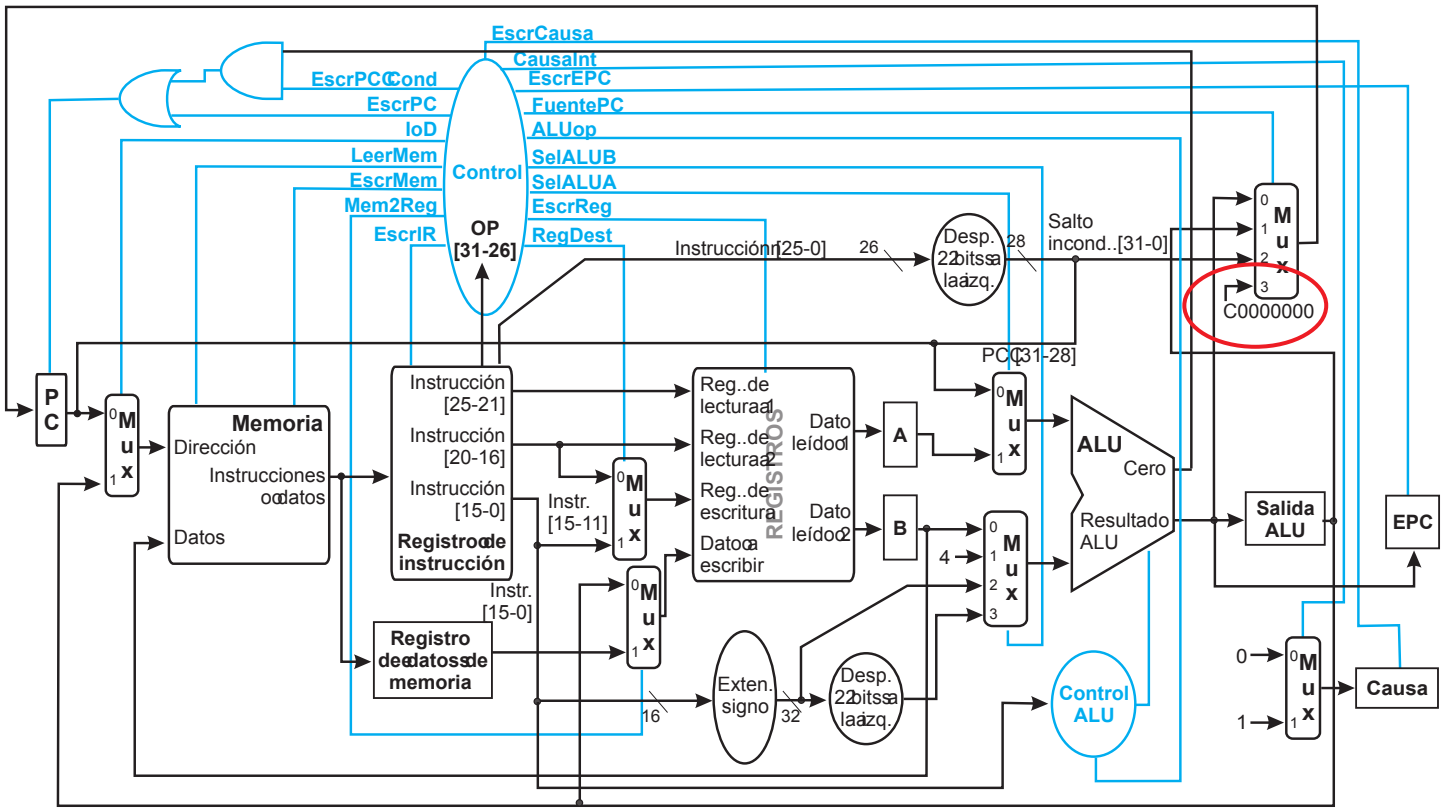
- EPC: registro de dirección de la instrucción causante (32 bits).
- Causa: Registro para almacenar la causa de la excepción (1 bit).
  - `Causa = 1`: Instrucción no definida.
  - `Causa = 0`: Desbordamiento aritmético.

Camino de datos modificado



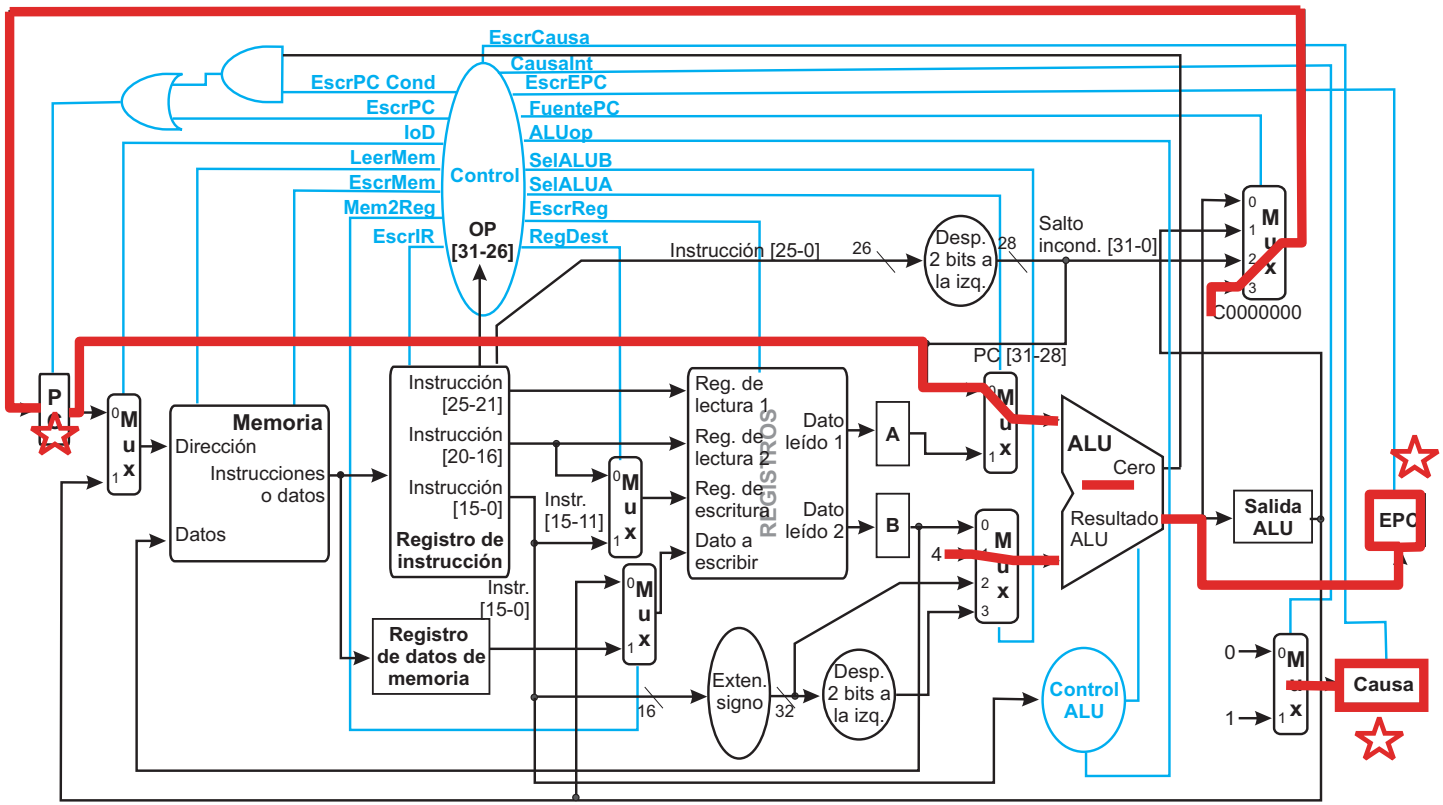
- Señales de control:
  - `EscrEPC`.
  - `EscrCausa`.
  - `CausaInt`: valor del registro Causa (1 bit).

Camino de datos modificado



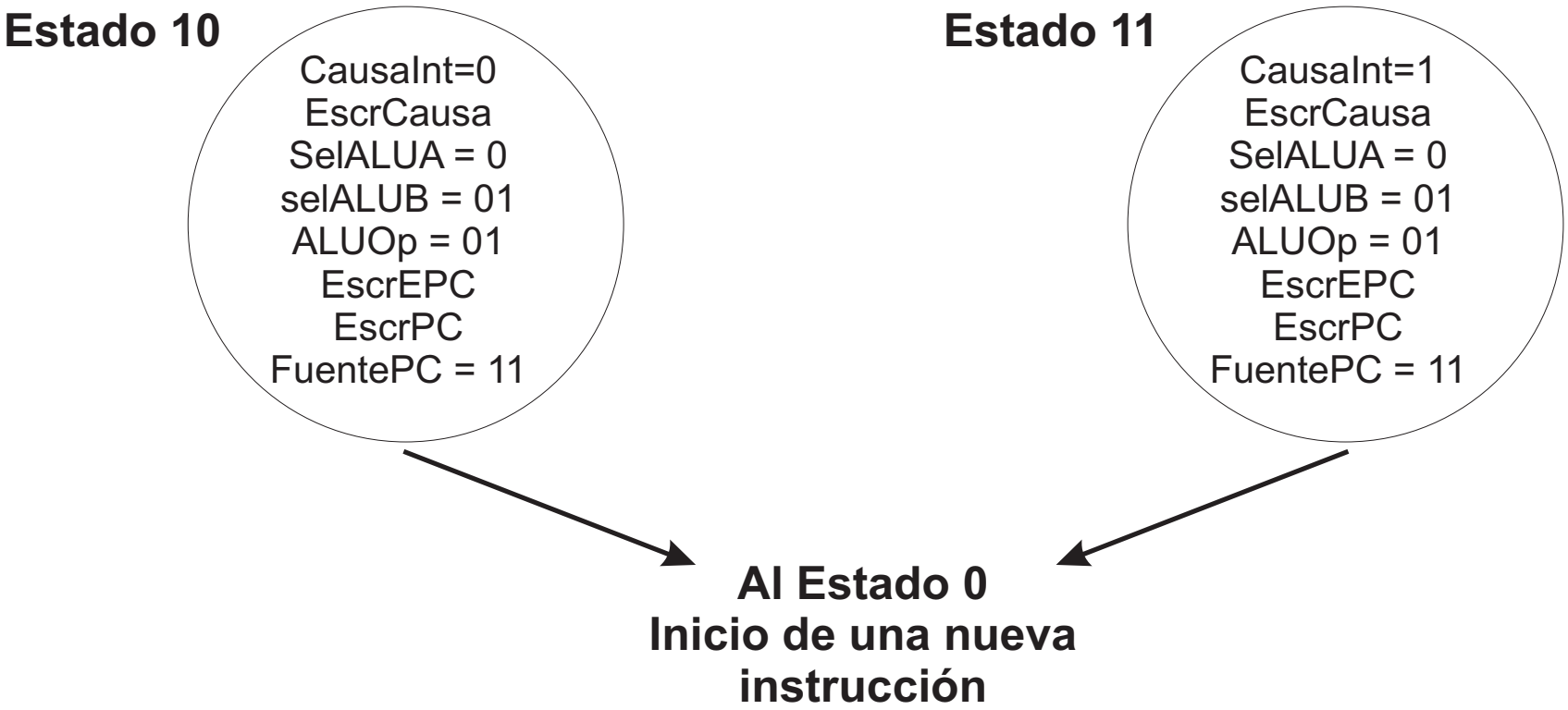
- Nueva entrada al MUX del PC: `0xC000 0000` (entrada al código del SO).

Procesamiento de una excepción

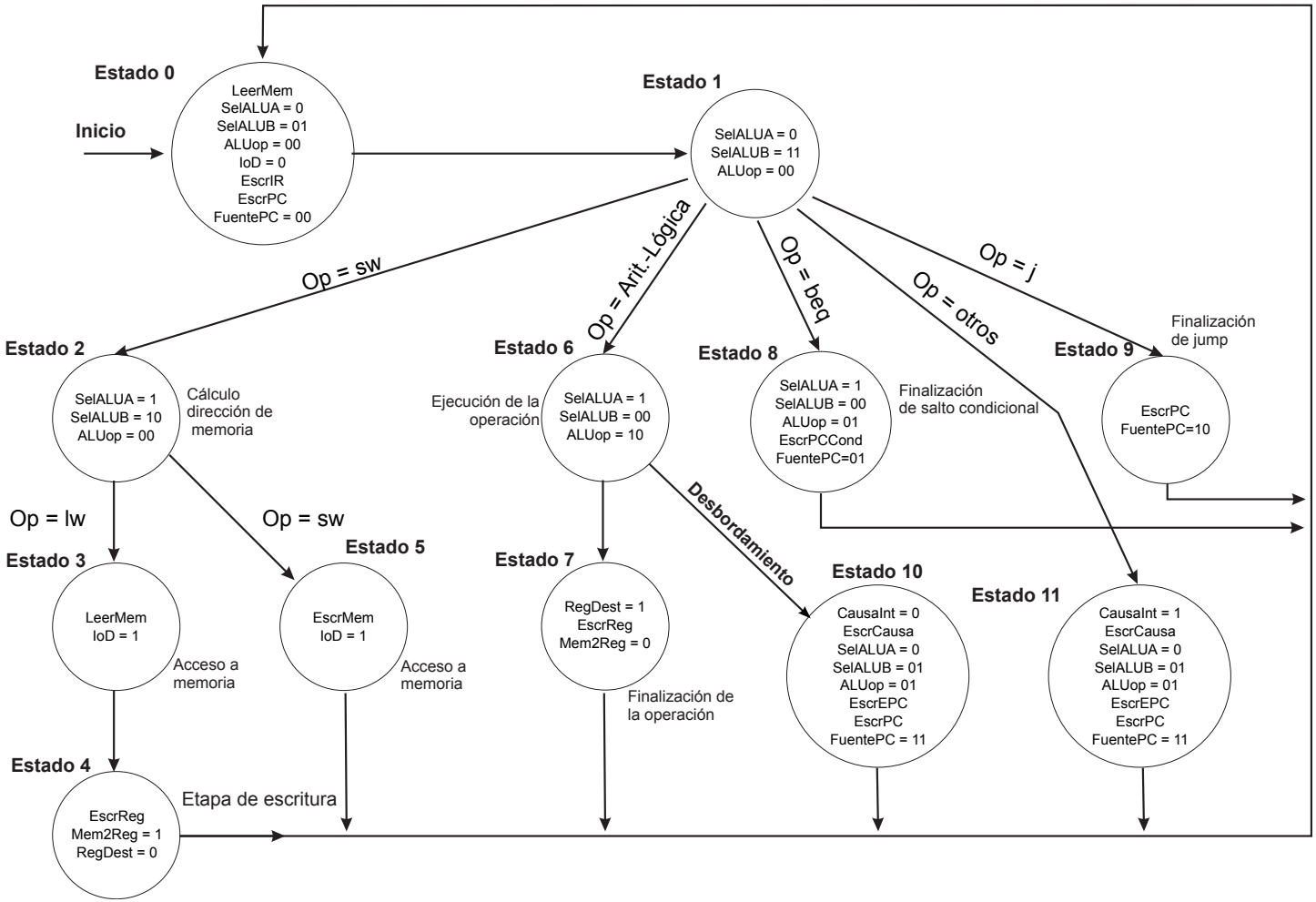


- Escribir en el registro Causa un 0 o un 1.
- Guardar la dirección de la instrucción causante (PC-4) en EPC.
- Poner en el PC la dirección de procesamiento de excepción del SO.

Nuevos estados de control

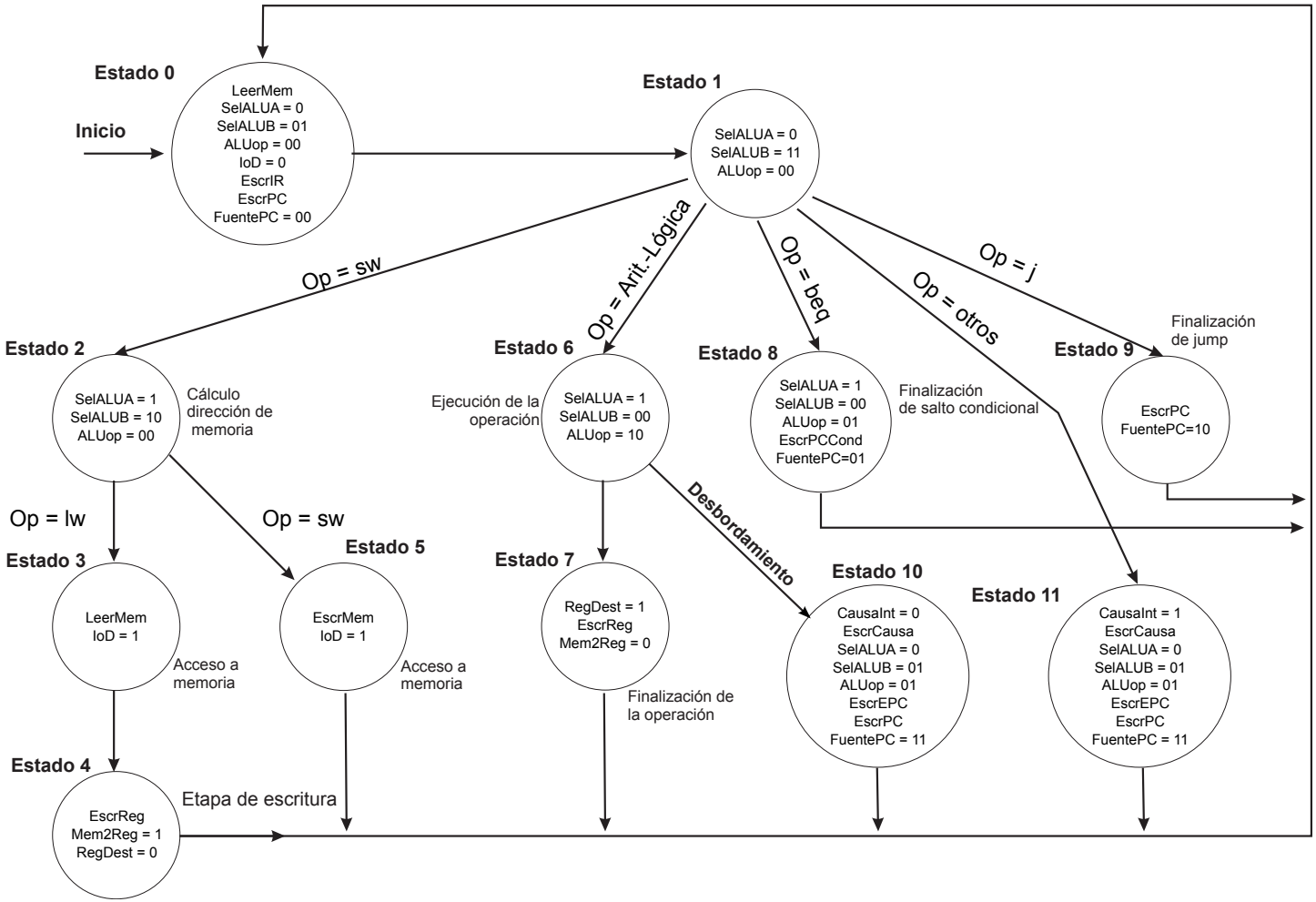


Unidad de control completa



- Instrucción no definida:
  - Se detecta al no estar definido el estado siguiente al estado 1 para el código de operación encontrado.
  - El estado siguiente será el estado 11.

Unidad de control completa



- Desbordamiento aritmético:
  - Se detecta cuando la ejecución de una instrucción aritmético-lógica (estado 6) arroja un desbordamiento.
  - El estado siguiente será el estado 10.

Unidad de control completa
Microprogramación

- Modificaciones a la UC microprogramada:
  - 2 nuevas microinstrucciones (estados 10 y 11)
  - Modificación tabla envío 1
  - Nueva tabla de envío 3 para bifurcación estado 6
  - Microinstrucciones de tamaño 22 bits en vez de 18:
    - CrtlDir podrá tomar 5 valores diferentes: ampliación a 3 bits
    - Nuevas señales de control: CausaInt, EscrCausa, EscrEPC

Unidad de control completa
Microprogramación

Tabla de envío 1

Código de operación	instrucción	Valor
000000	Tipo R	Rformat1
000010	jmp	JUMP1
000100	beq	BEQ1
100011	lw	Mem1
101011	sw	Mem 1
otro	otra	Nodef1

Tabla de envío 3

señal de desbordamiento	valor
0	Arit3
1	Ov3

Unidad de control completa

Microprogramación

	EscrPC	EscrPCCond	IoD	LeerMem	EscrMem	EscrIR	Mem2Reg	FuentePC (2)	ALUOp (2)	SelALUB (2)	SelALUA	EscrReg	RegDest	CtrlDir (3)	CausaInt	EScrCausa	EscrEPC	
	1		0	1		1		0	0	1	0			3				estado 0
									0	3	0			1				estado 1
Mem1									0	2	1			2				estado 2
Lw2			1	1										3				estado 3
							1					1	0	0				estado 4
Sw2			1		1									0				estado 5
Rformat1									2	0	1			4				estado 6
Arit3							0					1	1	0				estado 7
Beq1		1						1	1	0	1			0				estado 8
Jump1	1							2						0				estado 9
Ov3	1							3	1	1	0			0	0	1	1	estado 10
Nodef1	1							3	1	1	0			0	1	1	1	estado 11

Mejoras del rendimiento en los procesadores actuales

- Hay dos posibilidades para incrementar el rendimiento:
  - Mejorar la tecnología de los circuitos integrados, lo cual reduciría el tiempo necesario para ejecutar cada una de las etapas y permitiría reducir el ciclo de reloj.
  - Mejorar el diseño del procesador:
    - **Procesador Segmentado:** Trabaja con varias instrucciones a la vez, cada una en una etapa diferente. Se pretende conseguir la ejecución de una instrucción por ciclo (límite máximo).
    - **Procesador Superscalar:** Implementa paralelismo a nivel de instrucción. Ejecuta más de 1 instrucción por ciclo de reloj gracias a la replicación de unidades funcionales.
- Todas las CPUs de propósito general actuales son segmentadas y superscalares.