Universidad Rey Juan Carlos

Arquitectura e Ingeniería de Computadores

Grado de Ingeniería del Software

TEMA 3. CAMINO DE DATOS Y DE CONTROL

Luis Rincón Córcoles
Ángel Serrano Sánchez de León

Universidad Rey Juan Carlos

ÍNDICE

Universidad Rey Juan Carlos

INTRODUCCIÓN

- Tamaño de palabra (W): 32 bits
- Banco de 32 registros de propósito general, 2 de lectura y 1 de escritura: 32 bits
- Registros de propósito específico (PC=contador de programa y otros): 32 bits
- Memoria con unidad direccionable el byte, en la que se accede a byte (B), media palabra (H), palabra (W)
- Direcciones de memoria: 32 bits
- Instrucciones: 32 bits (formatos: R, I y J)
- ALU para datos de 32 bits

3

- Repaso de las características principales del procesador MIPS.

VÍDEO

https://youtu.be/ipzHvbMmweI

4

- La instrucción en código máquina (binario) se organiza según diferentes formatos.
- Este vídeo es un recordatorio de los formatos de instrucción del procesador MIPS (contenido estudiado en Introducción a la Informática).

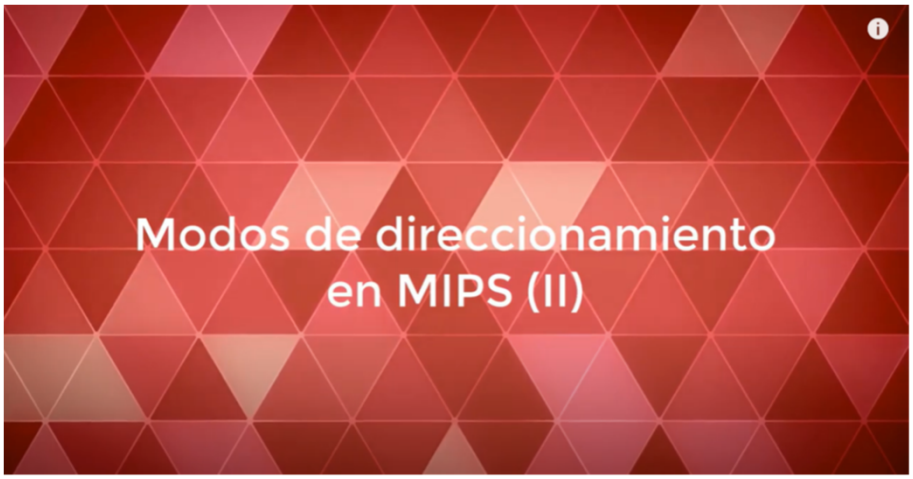
VÍDEO

https://youtu.be/PoXmikP-zfw

5

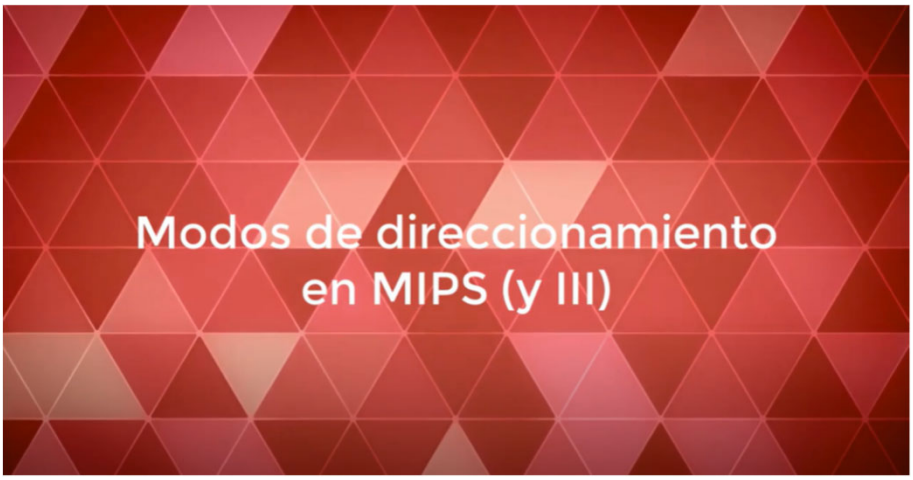
- Los modos de direccionamiento son las maneras de indicar la ubicación de los operandos de una instrucción.
- En resumen, los operandos pueden ubicarse en:
  - Registros del banco.
  - Memoria, para lo cual surge la pregunta de cómo se codifica la dirección de memoria del operando.
  - En la propia instrucción (inmediatos).
- Este vídeo es un recordatorio de los modos de direccionamiento del procesador MIPS (contenido estudiado en Introducción a la Informática).

VÍDEO



https://youtu.be/-6g_ThvoyqY

VÍDEO



https://youtu.be/O4cBavmFAGo

INTRODUCCIÓN

- Principios básicos de diseño:
  - Hacer rápido el caso común y correcto el caso infrecuente.
  - La simplicidad favorece la regularidad.

- El rendimiento de un computador está afectado por el **recuento dinámico de instrucciones**, la **frecuencia del reloj** del sistema y el **CPI (ciclos/instrucción)**.
  - Recuento dinámico de instrucciones: depende del repertorio de instrucciones de la máquina y del compilador.
  - El repertorio de instrucciones de máquina influye fuertemente en la realización del procesador:
    - Elección del camino de datos.
    - Elección de la estrategia de control del procesador.
  - La realización del procesador determina el tiempo de ciclo de reloj y el CPI.

- **Camino de datos (ruta de datos) de un procesador**: conjunto de elementos de almacenamiento y cálculo junto con los enlaces que los interconectan y las señales de control que regulan el funcionamiento de todo el conjunto.

- **Estrategia de control de un procesador**: técnica empleada para la generación de señales de control que gobiernan el funcionamiento del procesador.

8

- A la hora de diseñar un computador, será preciso que todas las instrucciones, tanto las más utilizadas como las menos frecuentes, funcionen correctamente.
- Sin embargo, hablando en términos de rendimiento, es mucho más rentable hacer rápido el caso común que hacer rápido el caso infrecuente.
- Por otra parte, si las instrucciones son sencillas, el diseño será más regular y, por ende, más simple. Y si el diseño es sencillo, normalmente será más rápido.
- Es muy importante conseguir que el computador alcance un rendimiento elevado.
  - Si consideramos que el rendimiento del computador se mide por el tiempo de ejecución de los programas (más bien por el inverso del tiempo de ejecución), es evidente que el rendimiento está afectado por:
    - El **recuento dinámico de instrucciones** (número de instrucciones ejecutadas y su tipo).
    - La **duración del ciclo de reloj** del procesador (o por su inverso, la frecuencia del reloj).
    - El número de ciclos que cada instrucción tarda en ejecutarse (más comúnmente conocido como **CPI**).
- El **camino de datos** o **ruta de datos** del computador está formado por el conjunto de elementos de memoria y de proceso necesarios para ejecutar las instrucciones del repertorio, junto con los enlaces que los interconectan.
- Por su parte, la **estrategia de control** es la técnica utilizada para construir la unidad de control que gobierna a los elementos del camino de datos.
- Dado que los programas en bajo nivel habitualmente se crean a través de un proceso de compilación, el recuento dinámico de instrucciones viene dado por el repertorio de instrucciones de la máquina y por el compilador utilizado en la traducción.
- Por su parte, el repertorio de instrucciones es fundamental a la hora de diseñar el camino de datos del procesador, ya que los elementos e interconexiones que componen el circuito deben soportar la ejecución de las instrucciones y estas deben ejecutarse de forma eficiente.
- Al mismo tiempo, la estrategia de control del camino de datos también está fuertemente influida por el repertorio de instrucciones y sus requisitos.
- El camino de datos propiamente dicho, junto con el circuito de control, componen la **materialización** o **realización** del procesador. Esta realización tiene un impacto muy grande sobre el rendimiento global del computador, ya que determina tanto la duración del ciclo de reloj como el CPI.

Universidad Rey Juan Carlos

INTRODUCCIÓN

- Repertorio de instrucciones MIPS seleccionado:
  - Instrucciones aritméticas y lógicas
  - Instrucciones de acceso a memoria
  - Instrucciones para control de flujo

- Fases de la ejecución de instrucciones:
  - Lectura de la instrucción (*instruction fetch*, IF)
  - Decodificación de la instrucción y lectura de registros (*instruction decode*, ID)
  - Fase de ejecución de una operación de la ALU (*execution*, EX)
  - Fase de acceso a memoria (MEM)
  - Escritura del resultado en registro (*write back*, WB)

  - A veces varias de estas etapas se pueden ejecutar de forma simultánea.
  - Las fases de lectura y decodificación de la instrucción son comunes a todas las instrucciones.
  - No todas las instrucciones pasan por las mismas etapas.

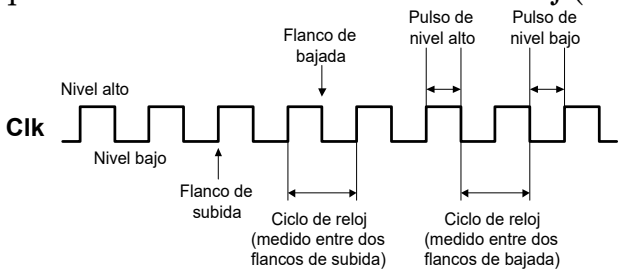
- **Contador de programa (PC, *program counter*)**: registro que contiene la dirección de memoria donde se encuentra la próxima instrucción que se va a leer.
  - Todas las instrucciones producen un incremento del PC.

9

- El repertorio de instrucciones seleccionado lo componen varias instrucciones de tipo R (**add**, **sub**, **and**, **or**, **nor** y **slt**), dos instrucciones de acceso a memoria (**lw** y **sw**), una instrucción de ramificación condicional (**beq**) y otra de salto incondicional (**j**).
- La ejecución de las instrucciones pasa por varias etapas:
  1. **Lectura de la instrucción**, que está en memoria. Esta fase es común a todas las instrucciones.
  2. **Interpretación (decodificación) de la instrucción y lectura de operandos fuente**. Esta fase también es común a todas las instrucciones y en ella se averigua cuál es la instrucción concreta que se acaba de leer, y determinándose sus operandos fuente y las señales de control que la UC activará para que los elementos del camino de datos colaboren de forma coordinada en la ejecución de las distintas fases de la instrucción.
  3. **Ejecución de la operación en la ALU.**
  4. **Acceso a memoria.**
  5. **Escritura del resultado en un registro.**
- Según se haya realizado la materialización del camino de datos, puede que en algunos casos haya varias etapas que se puedan ejecutar de forma simultánea, ahorrando así tiempo de procesador.
- Es preciso tener en cuenta que no todas las instrucciones necesitan realizar las mismas etapas.
  - Por ejemplo, una instrucción de suma (**add**) tiene fase de lectura de operandos, ejecución y escritura del resultado, mientras que la instrucción de no operación (**nop**) no necesita ninguna de esas tres cosas.
- Un elemento fundamental en el camino de datos es el registro **contador de programa** (**PC, *program counter***):
  - Siempre contiene la dirección de memoria en la que se encuentra la próxima instrucción que se va a leer (y decodificar y ejecutar).
  - Como en un programa las instrucciones van una detrás de otra, todas deberán realizar una operación de incremento del contador de programa, cosa que normalmente se hace en la fase de lectura de la instrucción (se lee la instrucción apuntada por el **PC**.
  - Al final del ciclo se modifica el contenido del **PC** con el nuevo valor incrementado).

METODOLOGÍA DE SINCRONIZACIÓN

- La metodología de sincronización define cuándo pueden leerse y escribirse las diferentes señales.
- La sincronización requiere contar con una señal de reloj (CLK).



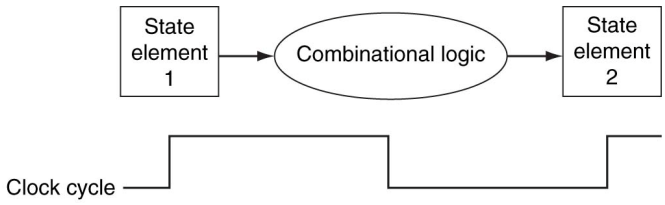
- Se elegirá **sincronismo por flanco**.
- **Elementos de estado: biestables de tipo D**.

10

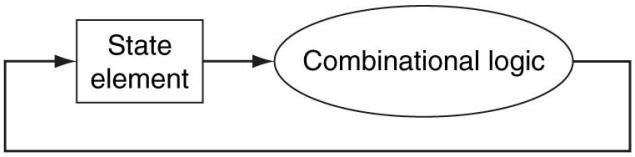
- El **reloj** es una señal periódica con un ciclo de duración fija. El ciclo de reloj está dividido en dos partes, el **nivel alto** y el **nivel bajo**. Los cambios de nivel se denominan **flancos**, y pueden ser **de subida** o **de bajada**.
- La metodología de sincronización:
  - Define cuándo pueden leerse y escribirse las diferentes señales en los **elementos de estado** (los **biestables**).
  - Se basa en la utilización de uno o varios relojes.
  - Puede realizarse sincronización **por nivel** o **por flanco**.
    - Utilizaremos sincronización por flanco, dado que es más sencilla de entender y controlar.
    - Entonces, los cambios de estado se producirán en el **flanco activo** de reloj, que puede ser el flanco de subida o el de bajada.
- El reloj actúa como una señal de muestreo, que causa que el valor de la entrada de datos de los biestables se lea y almacene en los mismos justo en el flanco activo.
  - La sincronización por flanco implica que el proceso de muestreo es instantáneo.
  - En el momento del flanco activo, las señales en la entrada de datos de los biestables deben ser **válidas**, es decir, estables, o sea, que no cambiarán a no ser que cambien las señales a partir de las cuales se calculan (normalmente a través de circuitos combinacionales).
- Como elementos de estado se utilizarán **biestables de tipo D activos por flanco** (*edge triggered D flip-flops*), habitualmente agrupados en **registros**.

METODOLOGÍA DE SINCRONIZACIÓN

- Fuente y destino diferentes:



- Fuente y destino coincidentes:

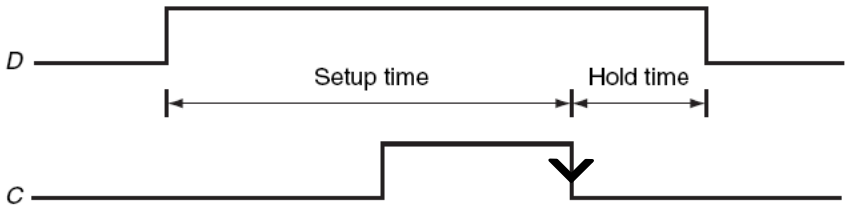


11

- En los sistemas con metodología de sincronización por flanco, los biestables proporcionan información válida a los bloques combinacionales, que son quienes realizan los cálculos.
- Al final del ciclo, los valores calculados por los bloques combinacionales se graban de nuevo en biestables.
- El ciclo de reloj debe ser suficientemente largo para que, en el momento del flanco activo, las señales proporcionadas por los bloques combinacionales se encuentren estabilizadas, es decir, deben ser válidas.
- Es posible que el elemento de estado que proporciona las entradas al bloque combinacional sea a la vez el lugar donde se graben los resultados proporcionados por el mismo, ya que el muestreo del resultado se produce justo al final del ciclo.
    - Por ello, la metodología de sincronización por flanco permite leer y escribir en un elemento de estado en un mismo ciclo de reloj sin que se produzcan condiciones de carrera que puedan hacer indeterminado el contenido final de dicho elemento de estado.
    - El valor leído por el circuito combinacional es el contenido inicialmente en el elemento de estado origen.
    - Es en el flanco activo de reloj (al final del ciclo) cuando se actualiza el contenido del elemento de estado destino.
- En estas figuras se supone que el flanco activo es el flanco de subida.

METODOLOGÍA DE SINCRONIZACIÓN

- En cualquier caso, las entradas a los biestables deben permanecer estables al menos durante los tiempos de preestabilización (*set-up time*) y mantenimiento (*hold time*) anterior y posterior respectivamente a la llegada del flanco de reloj.
- Ejemplo con reloj activo por flanco de bajada:

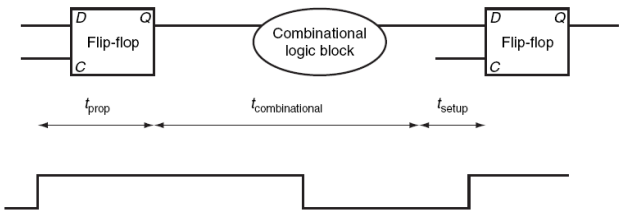


12

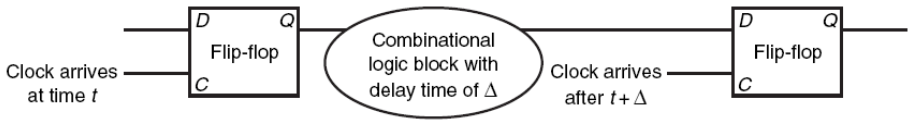
- Dado que la entrada **D** de los biestables se muestrea en un instante determinado (indicado por el flanco activo del reloj), su valor deber ser válido (permanecer estable) durante un cierto tiempo anterior y posterior al flanco de reloj.
  - El mínimo tiempo previo al flanco en el que la señal debe ser válida se llama **tiempo de preestabilización** (*setup time*).
  - El tiempo posterior al flanco en el que la entrada **D** debe seguir siendo válida se denomina **tiempo de mantenimiento** (*hold time*).
- Por tanto, las entradas de datos de los biestables activos por flanco deben ser válidas durante una ventana de tiempo alrededor del flanco activo, y cuya duración viene dada por la suma del tiempo de preestabilización más el de mantenimiento
- En estas figuras se supone que el flanco activo es el flanco de bajada.

METODOLOGÍA DE SINCRONIZACIÓN

- **Camino crítico**: camino más lento de un circuito (con mayor retardo).



- Problema: sesgo del reloj.



- Ciclo de reloj ≥ tiempo de propagación + retardo del camino crítico + tiempo de preestabilización + sesgo del reloj.

13

- Para que todo funcione bien, las señales deben ser válidas a la llegada del flanco activo en la entrada del elemento de estado en el que se van a grabar los resultados.
    - Para saber cuánto tiempo tarda el bloque combinacional en proporcionar salidas válidas, tenemos que encontrar el **camino crítico** en el mismo, que es el camino más lento que deben recorrer las señales hasta que se estabilizan las salidas.
- Además del camino crítico, también influyen sobre la duración del ciclo de reloj los tiempos de **propagación** y **preestabilización** del biestable.
    - El tiempo de propagación es el tiempo que tarda el biestable en presentar la salida correcta después de que se produzca el flanco activo. Se supone que, en cualquier caso, el tiempo de mantenimiento se respetará, lo cual sucederá si es menor que el tiempo de propagación del biestable.
- Sin embargo, hay un problema adicional, y es que el flanco de reloj no tiene por qué llegar al mismo tiempo a todos los elementos de estado del sistema.
    - Esta diferencia de tiempos se conoce como **sesgo del reloj** (*clock skew*).
    - Esto sucede porque la señal de reloj sigue diferentes caminos para llegar a los elementos de estado, y estos caminos pueden tener retardos diferentes.
    - Si el sesgo del reloj es suficientemente grande (si es mayor que el retardo del combinacional), entonces se producirán condiciones de carrera y el resultado se grabará mal.
    - Por tanto, los diseñadores deben tener este problema en cuenta y aumentar el ancho del ciclo de reloj sumándole el sesgo.
- En resumen, para que todo funcione, se debe cumplir lo siguiente:

Anchura del ciclo de reloj ≥ tiempo de propagación + retardo del camino crítico + tiempo de preestabilización + sesgo del reloj.

INSTRUCCIONES SELECCIONADAS

- Aritmético-lógicas:
  - Suma aritmética (`add`)
  - Resta aritmética (`sub`)
  - Producto lógico (`and`)
  - Suma lógica (`or`)
  - Suma lógica negada (`nor`)
  - Set less than (`slt`)
- Carga/almacenamiento:
  - `lw`
  - `sw`
- Salto condicional:
  - `beq`
- Salto incondicional:
  - `j`

14

- Esta es la selección de instrucciones que vamos a estudiar en nuestro camino de datos simplificado. Abarca todo tipo de instrucciones:
  - Aritmético-lógicas
  - De acceso a la memoria (carga y almacenamiento)
  - De control de flujo, es decir, de salto (tanto condicional como incondicional)

- El procesador MIPS real tiene muchas más instrucciones, que pueden encontrarse en la hoja de resumen de instrucciones colgada en el Aula Virtual y sacada del libro de Patterson & Hennessy.

- Intentaremos que todas las cuestiones del diseño de este camino de datos reducido sean compatibles con el camino de datos completo de MIPS. Para ello, se usarán los formatos de instrucción, códigos de operación y modos de direccionamiento habituales del procesador MIPS.

- Obviamente quedarán "lagunas" porque habrá muchas instrucciones reales que no estarán implementadas en nuestro camino de datos. Los ejercicios de este tema tratan de realizar pequeñas modificaciones al camino de datos para incluir nuevas instrucciones al repertorio. Estas modificaciones serán lo suficientemente pequeñas y manejables como para solo necesitar añadir nuevas conexiones entre componentes, nuevos multiplexores o ampliar los existentes, añadir nuevas puertas lógicas y poco más, aparte de las modificaciones oportunas de la unidad de control.

Si quisiéramos realizar grandes modificaciones, como añadir de golpe muchas instrucciones nuevas, deberíamos realizar un nuevo diseño desde cero del camino de datos y no solo modificar el existente.

NOTACIÓN DE PSEUDOCÓDIGO UTILIZADA

- Asignación con operador flecha. Ej.: `A ← B`
- Igualdad/desigualdad con operadores = y ≠. Ej.: `A = B`
- Valor de señales de control con operador =. Ej.: `LeerMem = 1`
- Extensión de signo de 16 a 32 bits con `extSigno16a32(valor)`
- Desplazamiento de bits hacia la izquierda con operador `<<`
- Concatenación de bits con operador `||`. Ej.: `valor1||valor2`
- Memoria como un array: `M[dirección]`
- Acceso a ciertos bits con operador `[ ]`. Ej.: `PC[31-28]`, `IR[31-26]`
- Banco de registros como un array: `Reg[índice]`, `0 ≤ índice ≤ 31`
- Combinación de todo a la vez. Ej.: `Reg[M[PC][15-11]]`
- Operación genérica de la ALU: `op`
- Comentarios explicativos: `#`

15

En las siguientes transparencias usaremos la notación de pseudocódigo que se indica a continuación:

- La operación de asignación en un registro (o en una dirección de memoria) se escribe con el operador flecha ←, que apunta hacia donde se produce la asignación de un valor. Por ejemplo, tanto a ← b como b → a significan que el valor de b se escribe en a. Solo se utiliza este operador para indicar la escritura. Normalmente usaremos solo la flecha hacia la izquierda ←.
- El operador que indica la comparación de igualdad lo representaremos con el signo =. Por ejemplo, a = b representa la comparación del valor del registro a con el del registro b. El resultado puede ser verdadero (si son iguales) o falso (si son diferentes). No confundirlo con el operador de asignación ←. La desigualdad se indicará con el signo ≠.
- También usaremos el operador igualdad para indicar el valor de una señal de control. Por ejemplo, LeerMem = 1 cuando haya que realizar una operación de memoria. LeerMem es una salida de la unidad de control y no un registro. Por eso no utilizaremos el operador flecha.
- La operación extSigno16a32(valor) recibe un valor numérico en complemento a 2 con 16 bits y realiza la extensión de signo para expresar el mismo valor, pero con 32 bits. Por tanto, se repite el bit 15 de valor desde el bit 16 hasta el 31.
- La operación de desplazamiento de bits hacia la izquierda se indica con el operador << seguido de un número que representa cuántos bits se mueven hacia la izquierda.
- La operación de concatenación de bits se representa con el operador ||, como, por ejemplo, valor1 || valor2. Este operador coloca valor1 en la parte izquierda (más significativa) del resultado, mientras que valor2 se sitúa en la parte derecha (menos significativa).
- La memoria se representa con la letra M. Cuando hacemos referencia a una dirección concreta de memoria, usaremos el operador corchete [ ]. Por ejemplo, escribiremos M[dirección] para referirnos al dato ubicado en dicha dirección de memoria.
- Para seleccionar solo algunos bits de un registro, también usaremos el operador corchete [ ]. Si PC es el registro contador de programa (tamaño 32 bits), los bits 31 a 28 del mismo son PC[31-28].
- El banco de registros se representa con Reg[índice], donde índice es un número de 0 a 31 que representa el número del registro al que se quiere acceder.
- Los tres últimos casos pueden aparecer conjuntamente. Así, por ejemplo, Reg[M[PC][15-11]] representa en el caso uniciclo acceder a la posición de memoria a la que apunta el contador de programa, M[PC], que corresponde a la instrucción actual. De los 32 bits de la instrucción, nos quedamos solo con los bits del 15 al 11, es decir, 5 bits, que representan un número en binario puro de 0 a 31. Este es el índice que se le da al banco de registros y representa el registro al que queremos acceder.
- La operación realizada por la ALU (de las 6 que aparecen en esta transparencia) se representa de manera genérica como op.
- Los comentarios explicativos se indicarán con el símbolo de almohadilla #.

INSTRUCCIONES ARITMÉTICO-LÓGICAS

- Dos operandos fuente en registros `s` y `t`
- Operando destino en registro `d`
- Comparten código de operación (`000000`) y se distinguen por los bits 5-0 (funct)
- Ej.: `add $s0, $s1, $s2  # $s0 ← $s1 + $s2`

```
100000 add
100010 sub
100100 and
100101 or
100111 nor
101010 slt
```

Formato R

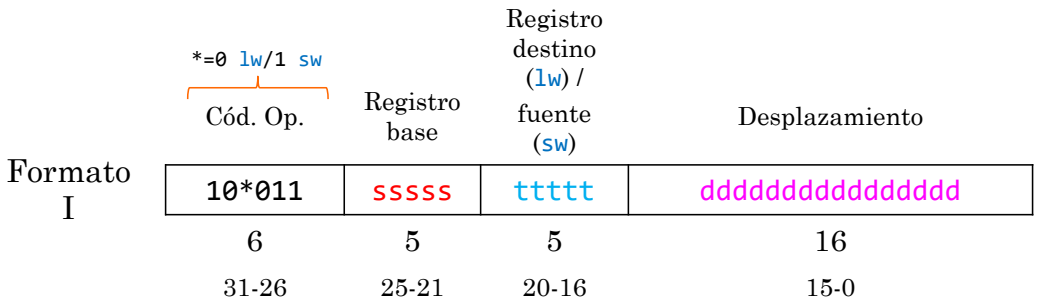
Cód. Op.	Registro fuente 1	Registro fuente 2	Registro destino	shamt	Funct
000000	sssss	ttttt	ddddd	00000	******
6	5	5	5	5	6
31-26	25-21	20-16	15-11	10-6	5-0

16

- Vamos a estudiar las instrucciones aritmético-lógicas de formato R (las que usan 2 operandos en sendos registros y almacenan el resultado en otro registro).
  - Todas estas instrucciones tienen el código de operación 000000 y se diferencian por los 6 bits del campo de función.
- FALACIA: "Todas las instrucciones aritmético-lógicas comparten el código de operación 000000 y son de formato R".
- Existen instrucciones aritmético-lógicas de formato I (ej.: addi, andi, etc.).
  - Estas instrucciones no usan el código de operación 000000 ni tienen todas ellas el mismo código de operación.
  - Al ser de formato I, solo codifican 2 registros en la instrucción y disponen además de un inmediato de 16 bits.
  - No las vamos a estudiar.

INSTRUCCIONES DE CARGA `lw`/ ALMACENAMIENTO `sw`

- Dirección = Registro base `s` + `extSigno16a32(desplazamiento)`
- Carga (`lw`): Registro destino `t` ← Memoria[dirección]
- Almacenamiento (`sw`): Memoria[dirección] ← Registro fuente `t`
- Ej.: `lw $s0, 4($s1)` `# $s0 ← M[$s1 + extSigno16a32(4)]`
- Ej.: `sw $s0, 4($s1)` `# M[$s1 + extSigno16a32(4)] ← $s0`



17

- Carga: lectura de un dato en memoria.
- Almacenamiento: escritura de un dato en memoria.
- En ambas instrucciones, hay un operando en memoria, cuya posición se indica a través de un registro y un desplazamiento (modo de direccionamiento indirecto a registro con desplazamiento).
- El otro operando es un registro y utiliza un modo de direccionamiento directo a registro.

INSTRUCCIÓN DE SALTO CONDICIONAL beq

- Si se cumple la condición de que dos registros s y t son iguales, su resta será 0 y se activará a 1 el indicador de resultado Cero en la ALU. En ese caso, el programa continúa ejecutándose en la instrucción en la dirección `PC + 4 + extSigno16a32(desplazamiento)<<2`.
- Si no se cumple la condición de igualdad, Cero vale 0 y la siguiente instrucción es la habitual (en `PC + 4`).
- Ej.: `beq $s0, $s1, etiq # Si ($s0=$s1), PC ← PC + 4 + + extSigno16a32(etiq)<<2; si no, PC ← PC + 4`

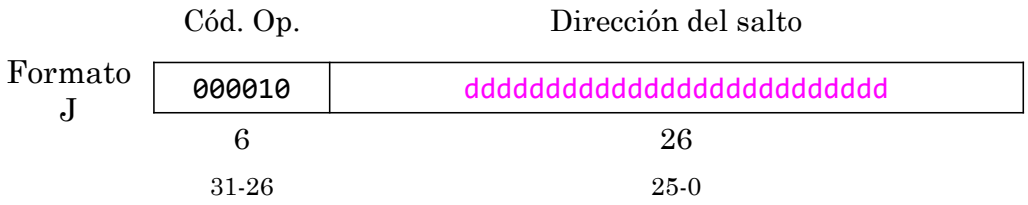
	Cód. Op.	Registro fuente 1	Registro fuente 2	Desplazamiento
Formato I	000100	sssss	ttttt	dddddddddddddddd
	6	5	5	16
	31-26	25-21	20-16	15-0

18

- La instrucción beq permite realizar un salto a una determinada instrucción, pero solo si se cumple una determinada condición.
  - En este caso particular, se debe cumplir que dos registros son iguales, al que se produce si su resta se anula.
- En ensamblador se usa una etiqueta para identificar la instrucción de destino del salto.
- En código máquina, el salto se codifica con 16 bits.
- Este valor es un desplazamiento relativo respecto de la posición actual del PC.
- El desplazamiento de 16 bits se extiende a 32 bits y se multiplica por 4 (desplazamiento de 2 posiciones hacia la izquierda).
  - Aunque el resultado tendría 34 bits, nos quedamos con los 32 bits menos significativos.
- Este valor se suma al registro PC, al que previamente se le ha sumado 4, y el resultado es la dirección de destino del salto en el caso de que hay que tomarlo.
- Si no hay que realizar el salto, la siguiente instrucción en ejecutarse es la esperable (dirección PC + 4).
- Se ha usado la notación del lenguaje C para indicar la operación de desplazamiento hacia la izquierda (<<).
- La dirección del destino del salto, representada en el ejemplo con la etiqueta etiq, utiliza un modo de direccionamiento relativo a PC.

INSTRUCCIÓN DE SALTO INCONDICIONAL j

- La siguiente instrucción ejecutada es la indicada por la dirección calculada así:
  - Se lee el campo `dirección` de 26 bits.
  - Se multiplica por 4 (`dirección<<2`), ya tenemos 28 bits.
  - Los 4 bits superiores se copian del valor `PC + 4`.
- Ej.: `j bucleIni  # PC ← (PC+4)[31-28]||(bucleIni<<2)`



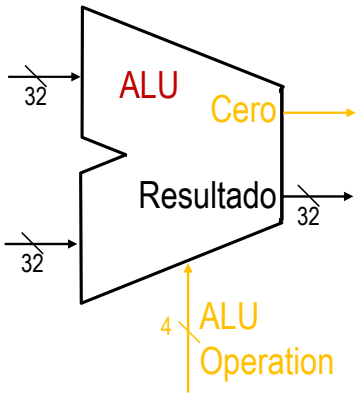
- La instrucción j corresponde a un salto a otra instrucción, salto que se produce siempre, sin verificar el cumplimiento de una condición.
- En ensamblador se usa una etiqueta para identificar la instrucción de destino del salto.
- En código máquina, el salto se codifica con 26 bits.
- Este valor de 26 bits se multiplica por 4 (desplazamiento de 2 posiciones hacia la izquierda), con lo que añadimos 00 por la derecha.
- Los 4 bits que nos faltan para llegar a 32 bits se toman del valor del registro PC al que previamente se le ha sumado 4.
- Hemos representado la operación de concatenación de bits con el símbolo ||. El operando de la izquierda se concatena en la parte más significativa del resultado, mientras que el de la derecha se coloca en la parte menos significativa del mismo.
- La dirección del destino del salto, representada en el ejemplo con la etiqueta bucleIni, utiliza un modo de direccionamiento pseudodirecto.

IMPLEMENTACIÓN UNICICLO

- Construcción del camino de datos uniciclo
- Funcionamiento para las instrucciones seleccionadas
- Unidad de control del camino de datos uniciclo

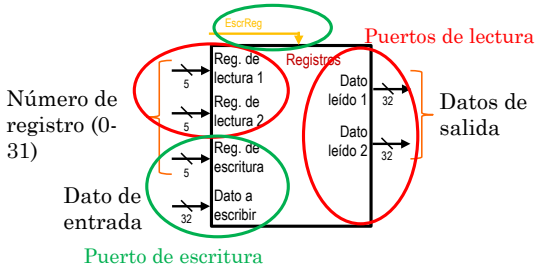
CONSTRUCCIÓN DEL CAMINO DE DATOS UNICICLO

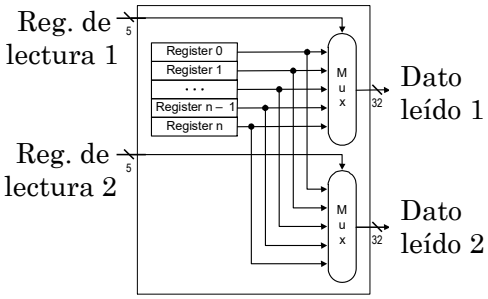
- **Unidad aritmético-lógica** (**ALU**: circuito combinacional para realizar los cálculos.
  - Usaremos la ALU diseñada modularmente en el tema 2.

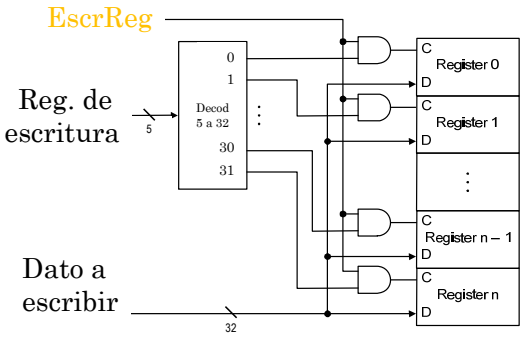


- Camino de datos uniciclo:
  - Circuito del procesador capaz de ejecutar las instrucciones del programa de tal forma que cada una de ellas se completa en **1 ciclo de reloj**.
  - La duración del periodo de reloj debe adaptarse a la **instrucción más lenta** del repertorio.
- Para realizar cálculos, será necesario contar con una unidad aritmético-lógica (ALU), que será la misma que se diseñó de forma modular en el tema anterior.
  - Está gobernada por una señal de control de 4 bits (**ALU Operation**) y será capaz de realizar las siguientes operaciones:
    - Suma.
    - Resta.
    - Asignar si mayor que (slt).
    - Operación lógica OR.
    - Operación lógica AND.
    - Operación lógica NOR.

CONSTRUCCIÓN DEL CAMINO DE DATOS UNICICLO





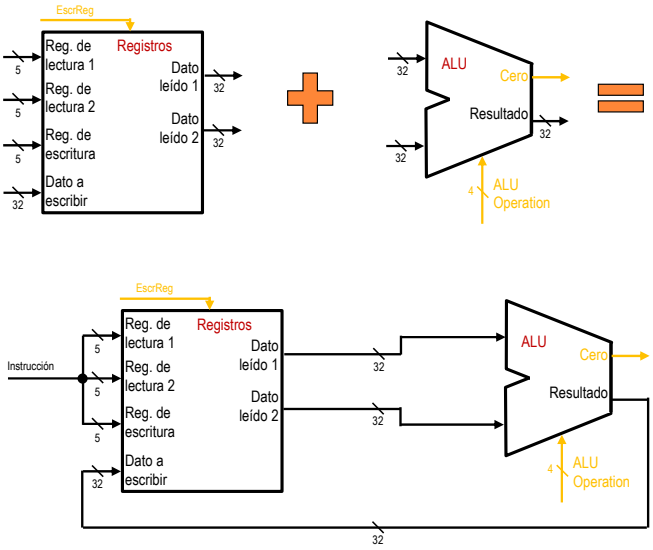


Banco de registros: contiene 32 registros de propósito general para almacenar datos, actuar como punteros, etc.

- Los operandos utilizados por la ALU proceden del banco de registros (arquitectura de carga-almacenamiento).
  - El banco de registros es el lugar de donde procederán habitualmente los operandos fuente de las instrucciones aritméticas o lógicas y también es donde se grabará el resultado de la mayoría de las instrucciones ejecutadas.
  - El banco de registros que necesitamos tendrá 32 registros de 32 bits. Estos registros estarán construidos mediante biestables **D** activos por flanco con habilitación de carga. Para seleccionar un registro, tanto para leerlo como para escribir sobre él, será necesario dar un código de 5 bits.
- Como muchas instrucciones de **MIPS** necesitan leer dos registros, el banco de registros tiene dos puertos de lectura, marcados en rojo en la figura. Las entradas **Reg. de lectura 1** y **Reg. de lectura 2** sirven para introducir los códigos de los registros que queremos leer. Sus contenidos se presentan por los puertos de salida **Dato leído 1** y **Dato leído 2**. Ambos puertos tienen 32 bits de ancho, suficientes para presentar a la salida el contenido de un registro por cada uno de ellos, y vienen de la salida de sendos mutiplexores de 32 entradas de 32 bits de ancho cada una. Cada uno de estos multiplexores selecciona el contenido de uno de los registros del banco, utilizando como señales de selección **Reg. de lectura 1** y **Reg. de lectura 2** respectivamente, y en realidad estaría formado por 32 multiplexores de 32:1, todos ellos con la misma señal de control de 5 bits que identifica al registro seleccionado.

- El banco de registros cuenta con un único puerto de entrada, marcado en verde en la figura, que permite escribir sobre uno de los registros. Para ello, introduciremos el dato que queremos escribir por los 32 bits de la entrada **Dato a escribir**, y el número del registro seleccionado a través de los 5 bits de la entrada **Reg. de escritura**, que está conectada a la entrada de datos de un decodificador. Cada salida del decodificador se encamina hacia la habilitación de escritura de uno de los registros. Para controlar en qué ciclos queremos escribir sobre un registro y en qué ciclos no, necesitaremos la señal **EscrReg**, que pondremos a 1 cuando queramos realizar una escritura. Con puertas lógicas AND que realizan el producto lógico entre **EscrReg** y una de las salidas del decodificador, tendremos las señales de habilitación de carga (**C**) que permitirán escribir sobre uno de los registros en los ciclos requeridos. El dato que queremos escribir sobre el registro lo introduciremos a través de los 32 bits de la entrada **Dato a escribir**, que están conectados a las entradas de datos de todos los registros. En resumen, activando **EscrReg** e introduciendo simultáneamente el código **k** por **Reg. de escritura**, escribiremos el dato que entra por **Dato a escribir** en el registro **k**. La escritura se realizará normalmente al final del ciclo, dado que los registros son síncronos por flanco.

CONSTRUCCIÓN DEL CAMINO DE DATOS UNICICLO

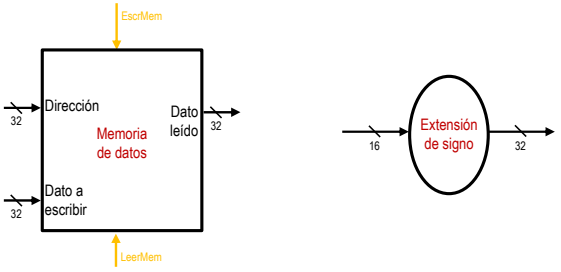


23

- La **ALU** y el banco de registros son los dos elementos básicos para realizar las fases de lectura de operandos, ejecución y escritura de resultados en las instrucciones de **tipo R**.
- ¿Cómo conectar el banco de registros con la ALU?
  - Conectar los puertos de salida del banco de registros **Dato leído 1** y **Dato leído 2** con las entradas para operandos fuente de la ALU.
  - Conectar la salida de la misma a la entrada **Dato a escribir** del banco de registros.
- Para que todo esto funcione, la unidad de control tendrá que introducir los bits adecuados por la señal **ALU Operation**, al tiempo que activará **EscrReg**.
- Los códigos de los tres registros, introducidos por las entradas **Reg. de lectura 1**, **Reg. de lectura 2** y **Reg. de escritura**, cada una de 5 bits, procederán de los campos de operandos fuente y destino de la propia instrucción.

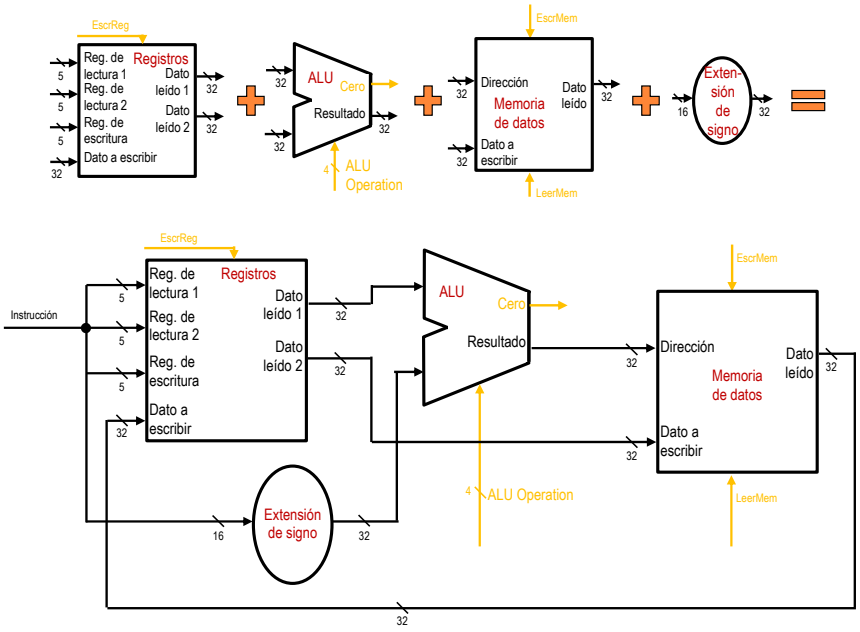
CONSTRUCCIÓN DEL CAMINO DE DATOS UNICICLO

- **Memoria de datos**: almacena las variables del programa.
  - **Lectura:** se activa **LeerMem** y sale por **Dato leído** el dato cuya dirección entra por **Dirección**.
  - **Escritura:** se activa **EscrMem** y el dato introducido por **Dato a escribir** se escribe en la dirección dada por **Dirección**.
- **Circuito para extensión de signo**: extiende el campo de desplazamiento de 16 a 32 bits antes de sumárselo al registro base para calcular la dirección del dato.



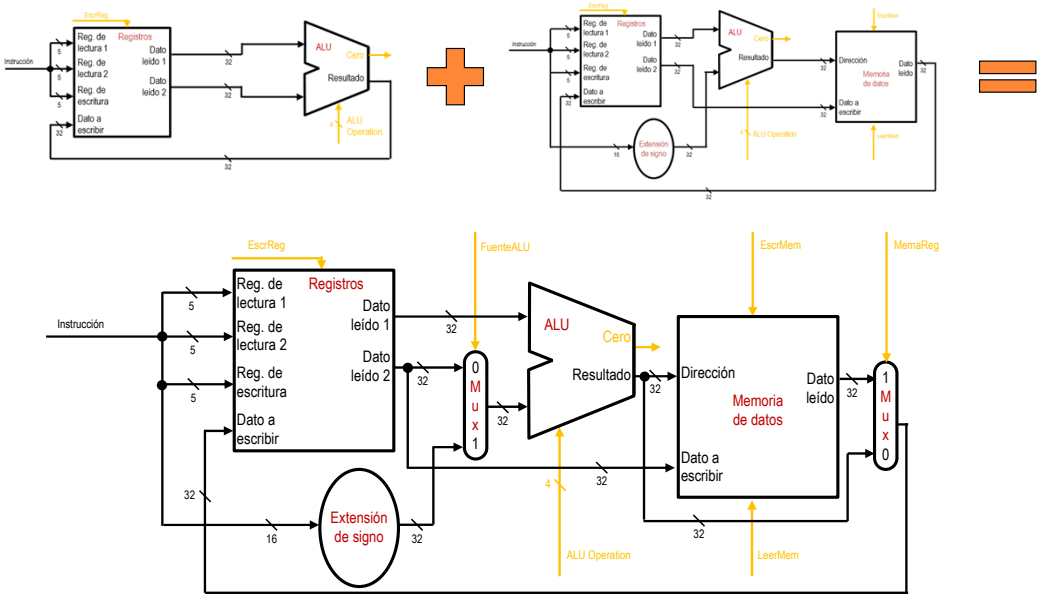
- Incluyamos ahora las operaciones de memoria (**lw** y **sw**):
  - **lw**:
    - Necesitaremos leer un dato residente en memoria y copiarlo en un registro de propósito general del banco de registros.
  - **sw**:
    - Tomaremos el contenido de un registro de propósito general y lo copiaremos en una posición de memoria.
- En ambos casos, la dirección del dato en memoria se calcula mediante la **suma** de un registro base (cuyo código viene dado en la propia instrucción) con un desplazamiento (que también está contenido en uno de los campos de la instrucción).
  - Esta suma se realizará en la propia **ALU**.
- Los elementos necesarios para las instrucciones de acceso a memoria son:
  - El banco de registros, porque de ahí sacamos el registro base, y el operando fuente en **sw**, y en él grabamos el dato leído de la memoria en **lw**.
  - El campo de desplazamiento de la instrucción, que tiene 16 bits y debe ser extendido en signo para poder sumárselo al registro base.
  - La **ALU**, para realizar el cálculo de la dirección mediante la suma del registro base más el desplazamiento, previamente extendido a 32 bits.
  - La propia memoria de datos, para leer o escribir el dato.
- El circuito de extensión de signo que transforma datos de 16 a 32 bits se estudió en el tema anterior.
- Memoria de datos:
  - Debe permitir realizar operaciones de lectura y de escritura.
    - Lectura:
      - Se introduce la dirección por la entrada **Dirección** (de 32 bits) y se activa la señal de control **LeerMem**.
      - Al cabo de un cierto tiempo (**latencia de lectura**), el dato solicitado aparecerá en los 32 bits de la salida **Dato leído**.
    - Escritura:
      - Se introduce simultáneamente la dirección del mismo por la entrada **Dirección** y el valor que queremos escribir por los 32 bits de la entrada **Dato a escribir**, al tiempo que activaremos la señal **EscrMem**.
      - Transcurrido el retardo correspondiente (**latencia de escritura**), el dato quedará grabado en la misma.

CONSTRUCCIÓN DEL CAMINO DE DATOS UNICICLO



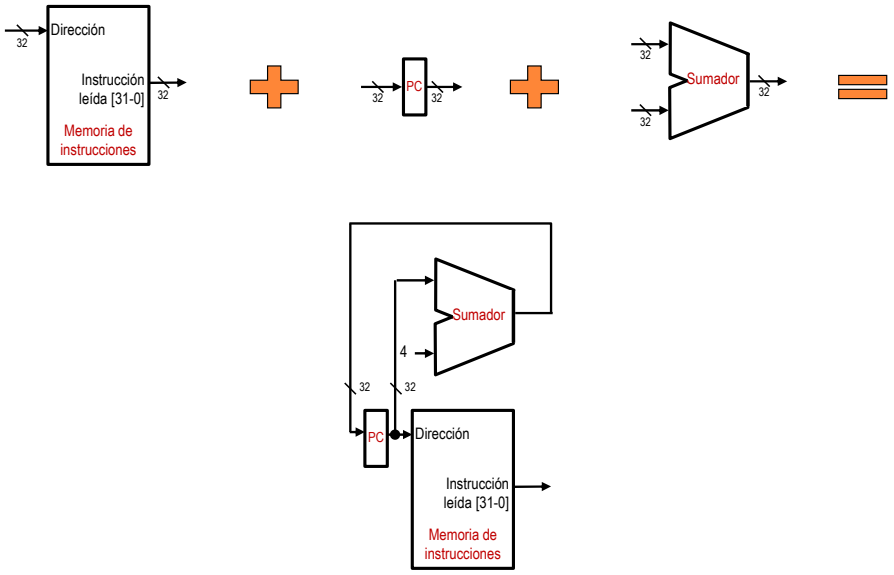
- ¿Cómo conectar el banco de registros, la ALU, la memoria de datos y el extensor de signo?
  - **Cálculo de la dirección:**
    - Se introduce el contenido del registro base (ofrecido por el banco de registros a través de su salida **Dato leído 1**) por el primer operando de la **ALU.**
    - Por el segundo operando introduciremos el desplazamiento (procedente de la instrucción) extendido en signo.
    - La **ALU** recibe la orden de sumar a través de **ALU Operation** y, como resultado, se obtendrá la dirección requerida. Por ello, es preciso conectar la salida de la **ALU** a la entrada **Dirección** de la memoria de datos.
  - **Instrucción sw:**
    - Cuando la salida de la **ALU** sea válida, es decir, cuando la dirección esté ya calculada, lo siguiente será realizar la escritura en memoria.
    - El dato fuente está en el banco de registros, que lo presenta al exterior a través de la salida **Dato leído 2**, que se encuentra conectada a la entrada **Dato a escribir de la memoria**.
    - Como la dirección ya está en **Dirección**, basta con activar **EscrMem**, y el dato se grabará en memoria, transcurrido un tiempo equivalente a la latencia de escritura.
  - **Instrucción lw:**
    - Cuando el cálculo de la dirección de memoria sea válido, habrá que realizar una lectura en memoria, lo cual haremos activando **LeerMem.**
    - Después de esperar un tiempo dado por la latencia de lectura, el dato leído válido estará presente en la salida **Dato leído de la memoria**.
    - La conexión de dicha salida de la memoria a la entrada **Dato a escribir del banco de registros** hace posible la escritura del dato válido en el banco sin más que activar **EscrReg**, dado que por la entrada **Reg. de escritura** ya tenemos disponible el código del registro destino.

CONSTRUCCIÓN DEL CAMINO DE DATOS UNICICLO



- Tenemos dos variantes del circuito:
  - Una sirve para las operaciones aritmético-lógicas de formato R.
  - La otra sirve para las operaciones de carga y de almacenamiento.
- Es preciso juntarlas para construir una primera versión del camino de datos global. Aquí nos surgen dos problemas:
  - Problema 1:
    - El segundo operando de la **ALU** puede proceder de dos sitios:
      - La salida **Dato leído 2** del banco de registros (en instrucciones tipo R).
      - El desplazamiento extendido en signo (en instrucciones de acceso a memoria).
    - Para resolverlo, es preciso incluir un multiplexor de dos entradas de 32 bits de ancho, que estará regulado por la señal de control **FuenteALU**, que tendrá que valer 0 en instrucciones de tipo R y 1 en las instrucciones **lw** y **sw**.
  - Problema 2:
    - El dato escrito en el banco de registros puede proceder de dos sitios:
      - La salida de la **ALU**.
      - La salida **Dato leído de la memoria**.
    - Por tanto, es preciso incorporar un multiplexor de dos entradas de 32 bits, que estará regulado por la señal de control **MemaReg**. En instrucciones de tipo R, **MemaReg** valdrá 0, y en la instrucción **lw**, **MemaReg** valdrá 1.
- Los multiplexores mencionados en realidad tienen dos entradas de datos de 1 bit de ancho.
  - Cuando hablamos de un multiplexor de dos entradas de 32 bits de ancho, en realidad estamos hablando de un conjunto de 32 multiplexores de 2 a 1, tal que cada uno deja pasar un bit de una de las dos entradas de datos con las que cuenta, dependiendo de la señal de control, que es común a todos los multiplexores individuales.

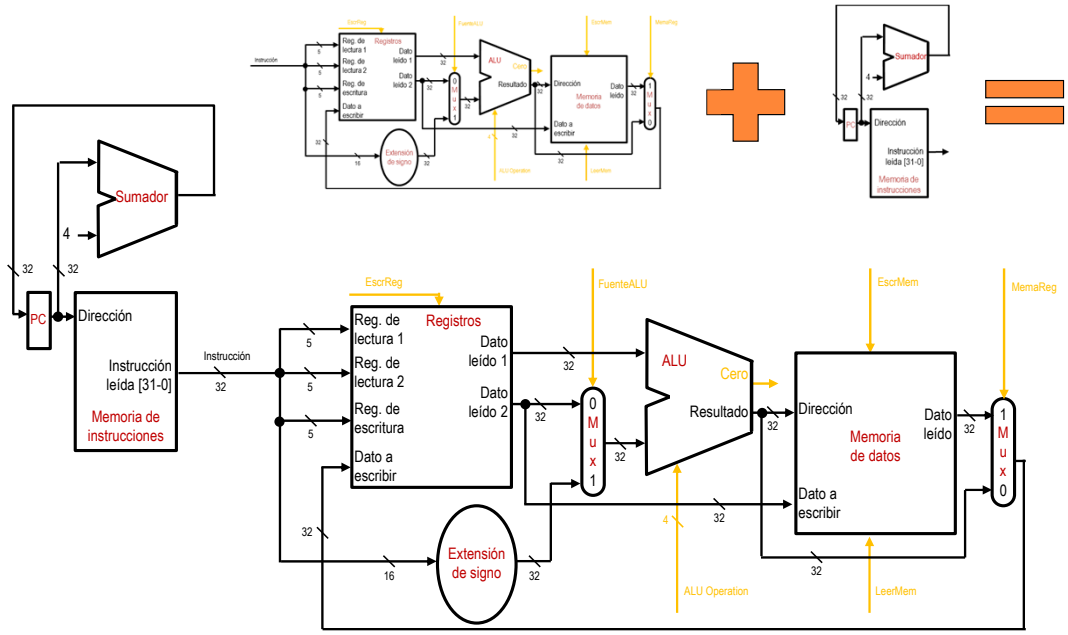
CONSTRUCCIÓN DEL CAMINO DE DATOS UNICICLO



27

- Incluyamos ahora la circuitería para búsqueda y secuenciación de las instrucciones del programa:
  - Las instrucciones del programa se encuentran en la memoria de instrucciones, una memoria separada e independiente de la memoria de datos (arquitectura Harvard).
    - Esto es así porque en el camino de datos uniciclo no podemos usar dos veces la memoria en el mismo ciclo de reloj.
    - La memoria es direccionable a nivel de bytes.
  - Todas las instrucciones en MIPS ocupan 32 bits, es decir, 4 bytes.
  - Necesitamos poder sumar 4 a la dirección actual.
- Necesitaremos entonces:
  - Una **memoria de instrucciones**, que contendrá las instrucciones del programa completo.
  - Un registro **contador de programa** (**PC**, *program counter*), que en todo momento contiene la dirección en memoria de la **siguiente instrucción** que se va a leer y ejecutar.
  - Un circuito **incrementador**, que servirá para actualizar el valor del contador de programa, y así realizar la secuenciación de las instrucciones: cada vez que se lea una instrucción, se incrementará el **PC** para que pase a apuntar a la siguiente.
- Esta memoria de instrucciones:
  - No tiene señales de control.
  - Lee la posición solicitada en cuanto ponemos una dirección en su entrada (cosa que haremos en el flanco de reloj que marca el principio del ciclo).
  - Transcurrida la latencia de lectura, presenta a la salida los 32 bits de la instrucción completa.
  - Para conseguir leer la instrucción, basta con conectar la salida del **PC** a la entrada **Dirección** de la memoria para realizar la lectura.
- La secuenciación se consigue conectando la salida del **PC** a la entrada de un sumador, cuya segunda entrada tiene un valor fijo 4.
- La salida del sumador contendrá el valor del **PC** incrementado, que se cargará en el propio registro cuando llegue el flanco activo (al final del ciclo de reloj).

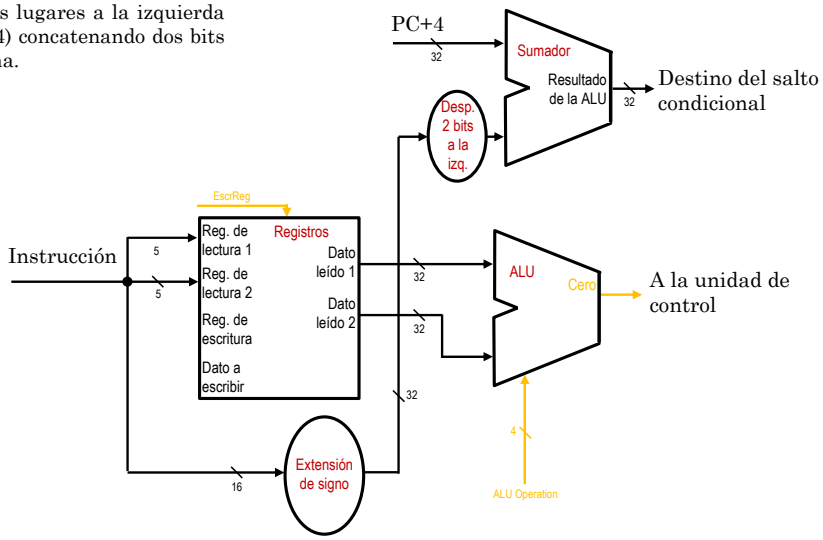
CONSTRUCCIÓN DEL CAMINO DE DATOS UNICICLO



- Incorporamos ahora el circuito que permite ir leyendo una a una las instrucciones con el que permite las operaciones aritmético-lógicas, las de carga y las de almacenamiento.
    - Ambos circuitos se conectan por los hilos por los que van los 32 bits de la instrucción, que salen de la memoria de instrucciones y son encaminados a los lugares correspondientes, que de momento son:
        - Las tres entradas de registros del banco de registros (de 5 bits cada una).
        - Los 16 bits del campo de desplazamiento, utilizados para calcular direcciones de memoria, que se envían al circuito extensión de signo.

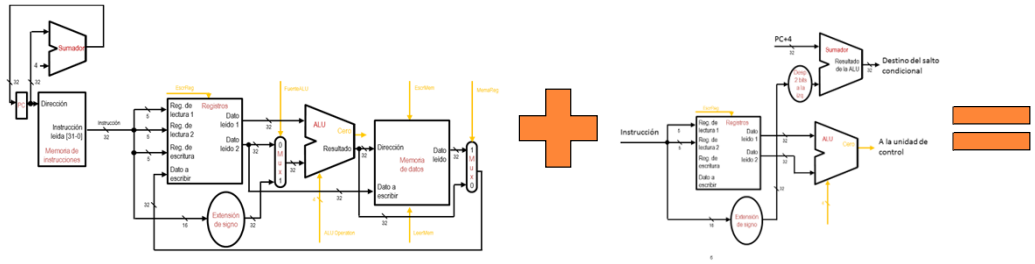
CONSTRUCCIÓN DEL CAMINO DE DATOS UNICICLO

Despl. 2 bits a la izquierda: circuito que desplaza dos lugares a la izquierda (multiplica por 4) concatenando dos bits a 0 por la derecha.

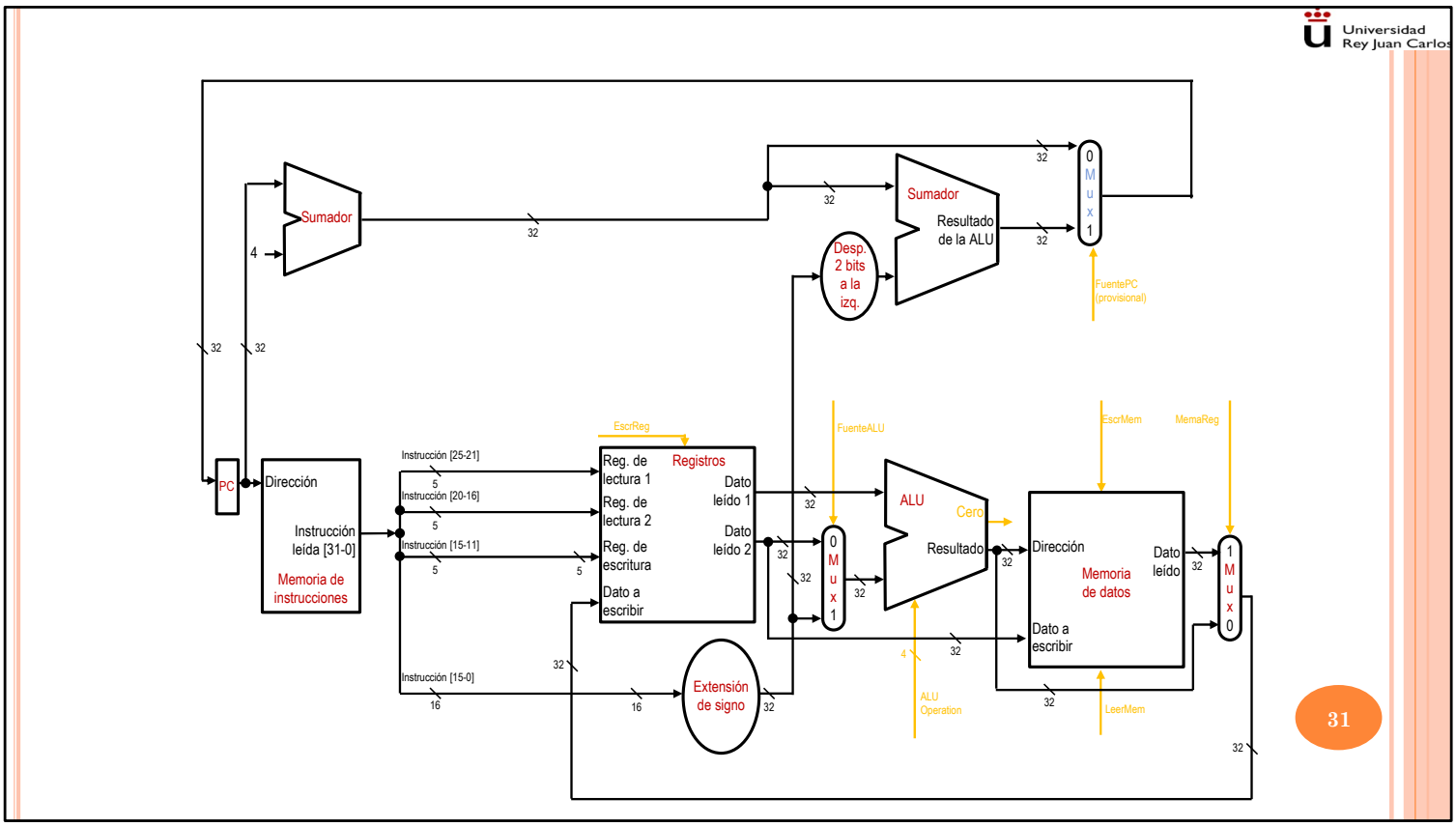


- Añadiremos ahora la instrucción de salto condicional **beq**:
  - Compara el contenido de dos registros, y en función de que sean iguales o no se decide si se realiza o no una ramificación (salto) en la secuencia de ejecución de instrucciones.
  - Si el contenido de los dos registros es distinto, la próxima instrucción es la que va detrás en la secuencia (no hay ramificación).
  - Si el contenido de ambos es idéntico, entonces la siguiente instrucción se encuentra en una posición de memoria situada a **n** instrucciones de distancia de la posición actual, siendo **n** el valor del campo **desplazamiento** de la propia instrucción.
- La comparación entre registros puede realizarse en la **ALU** restándolos y observando el indicador de resultado nulo (bit **Cero**) devuelto por la misma.
  - Si **Cero=1**, los dos registros tienen el mismo contenido, y sí se realiza la ramificación.
  - Si **Cero=0**, entonces los contenidos de los registros son distintos y no hay que ramificar.
- Denominaremos **PC incrementado** a la dirección de la instrucción siguiente en la secuencia, y **PC ramificado** a la dirección de la instrucción a la que hay que saltar si es preciso realizar la ramificación.
  - El valor de **PC incrementado** se calcula mediante la circuitería que hemos analizado previamente (el **PC** y el incrementador).
  - Para calcular el valor del **PC ramificado**, es preciso añadir un sumador nuevo.
    - Dado que el campo **desplazamiento** de la instrucción tiene 16 bits y contiene un valor **n** equivalente al número de instrucciones que hay que saltarse hacia atrás o hacia adelante (**desplazamiento** puede ser positivo o negativo), es preciso extenderlo en signo a 32 bits (circuito **Extensión de signo**) y multiplicarlo por 4 (circuito **Despl. 2 bits a la izquierda**) para ajustar el resultado de la suma a direcciones de byte.
- El nuevo valor del **PC** podrá ser el incrementado o el ramificado. Por lo tanto, cuando se incorpore esta circuitería al camino de datos global será preciso seleccionar entre ambos.

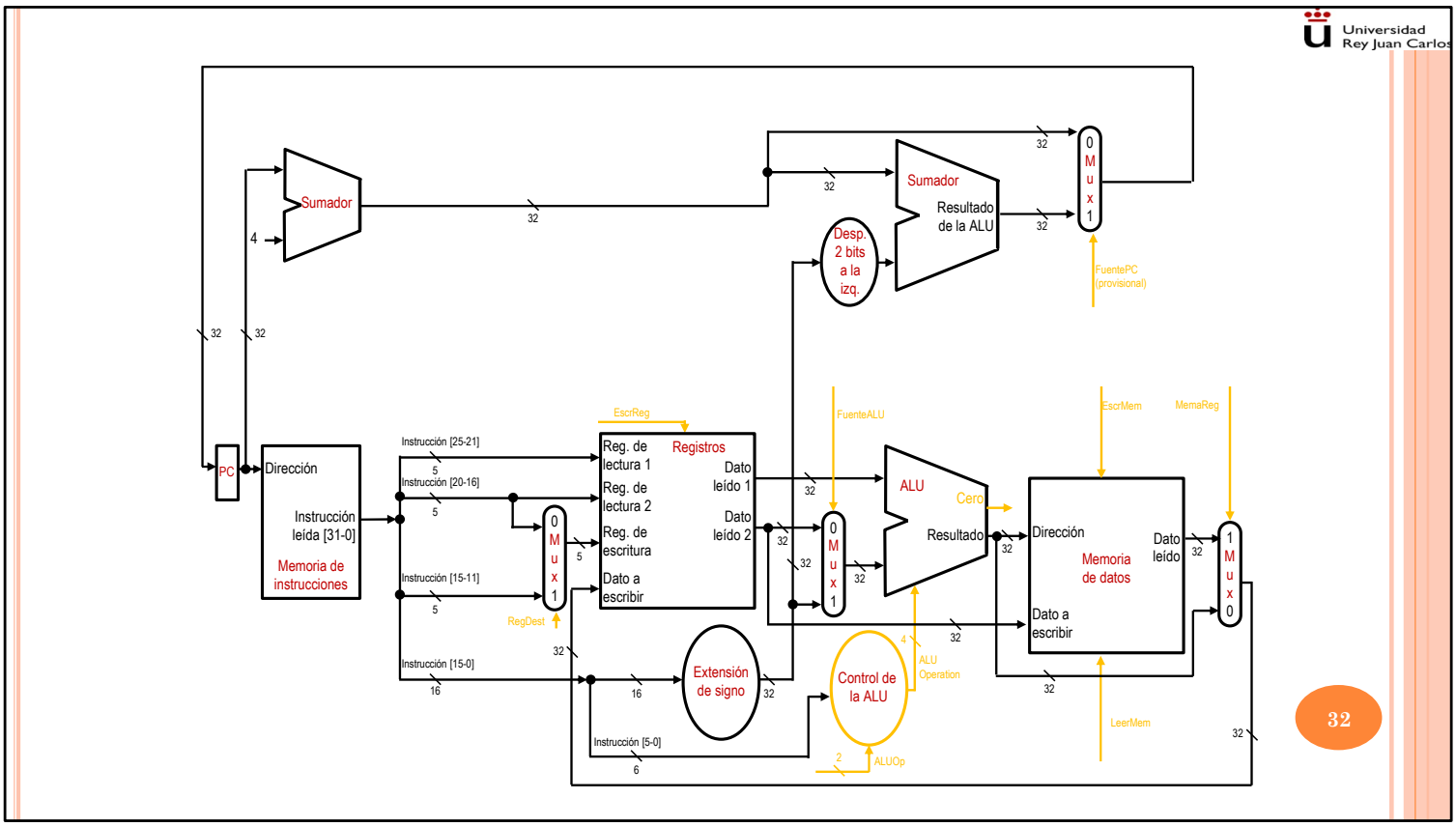
CONSTRUCCIÓN DEL CAMINO DE DATOS UNICICLO



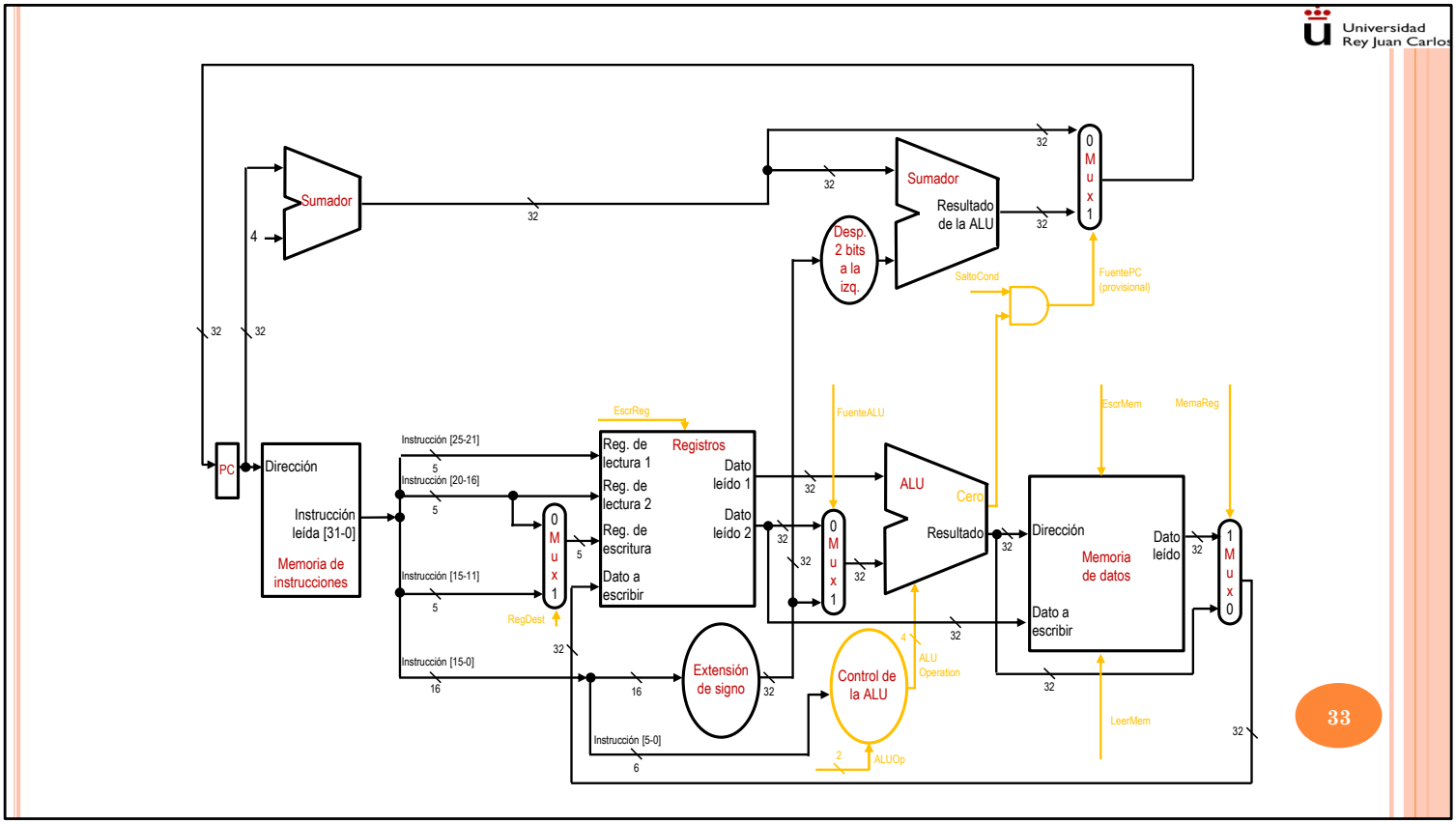
- Versión 3 del camino de datos:
  - Añadimos un multiplexor para elegir como nuevo valor del **PC** entre el **PC incrementado** y el **PC ramificado**.
    - Este multiplexor estará controlado por una señal llamada **FuentePC (provisional)**, que se diferencia de las restantes señales de control en que no puede ser generada directamente por la unidad de control, sino que depende del código de operación de la instrucción y del valor del indicador **Cero** generado por la **ALU**.
  - Cálculo de la dirección de ramificación:
    - Aprovecharemos el circuito de extensión de signo existente previamente.
  - La operación de comparación se realizará mediante una resta en la propia **ALU**, tomando como fuentes los dos registros ofrecidos por el banco de registros a través de sus dos puertos de lectura.
    - No nos interesa el resultado concreto de la resta, sino solo si es cero o no.



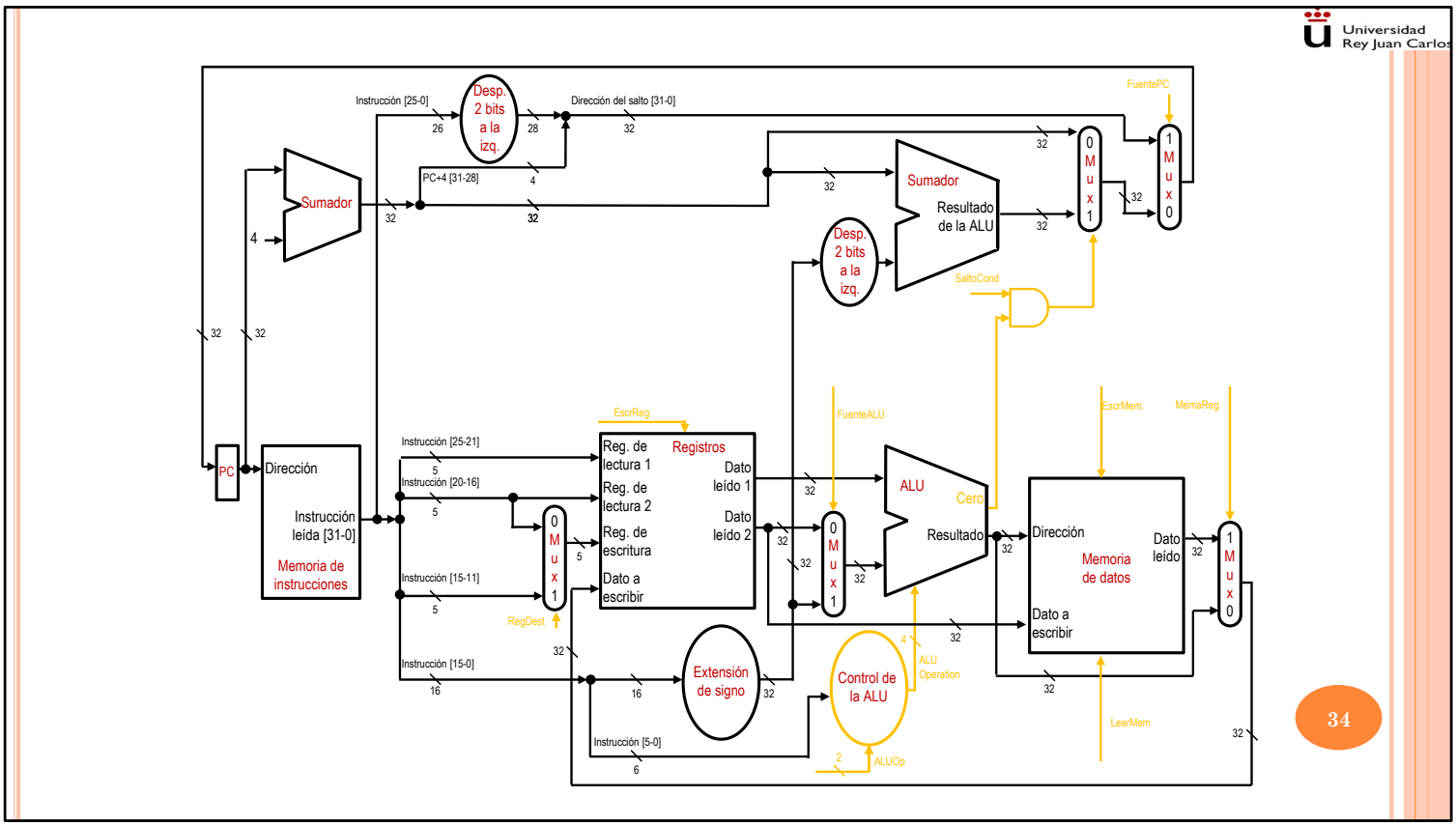
- Aquí hemos añadido los elementos adecuados para poder ejecutar:
  - Las instrucciones aritmético-lógicas.
  - Las de carga y almacenamiento.
  - La de salto condicional beq.
- De momento, la señal FuentePC es provisional.



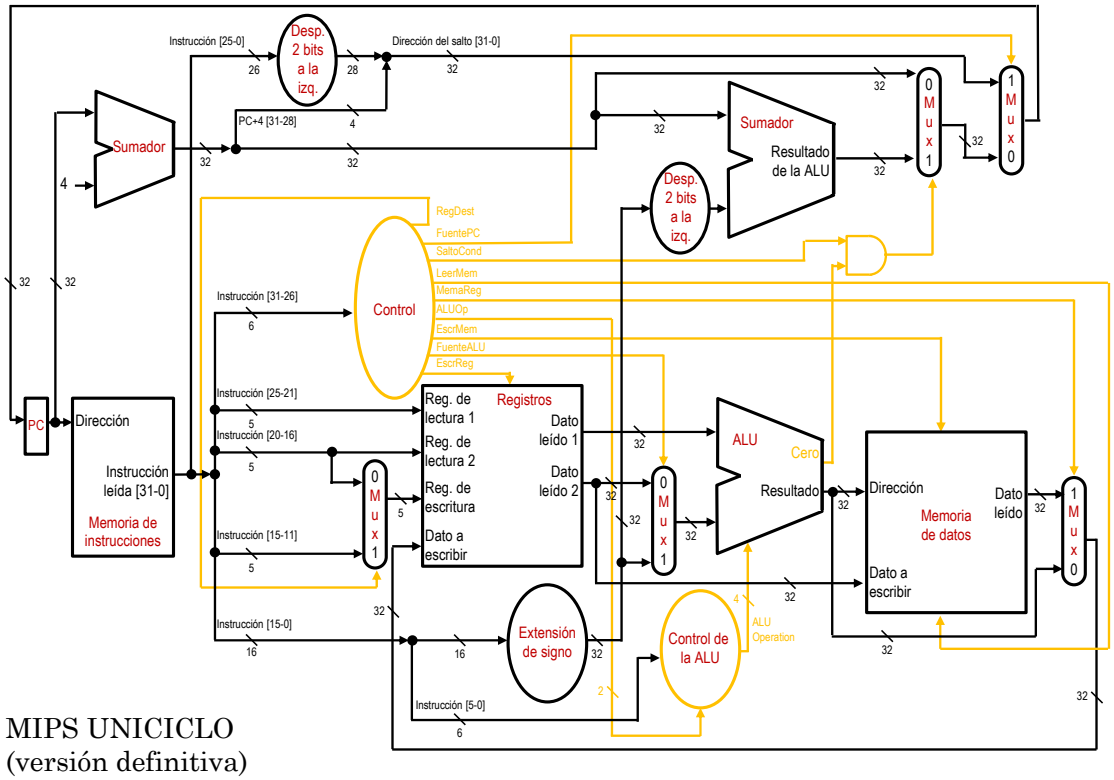
- Versión 4 del camino de datos:
  - Se resuelve el conflicto causado por las dos posibles ubicaciones del campo identificador del registro destino añadiendo un multiplexor, controlado por la señal **RegDest**, que permite seleccionar entre las dos opciones.
  - En cuanto a las instrucciones aritmético-lógicas de tipo R:
    - Tienen el mismo código de operación, pero distinto campo de función.
    - Realizan los mismos pasos en su ejecución.
    - Solo se distinguen por la operación realizada en la ALU.
    - Las señales que la unidad de control debería activar serían iguales para todas ellas, salvo los 4 bits de **ALU Operation**.
      - Si hacemos que la unidad de control genere **ALU Operation**, entonces debería tener 12 bits de entrada en total (los 6 del código de operación más los 6 del campo de función), lo cual puede complicar el diseño.
      - Para mantener un número reducido de entradas en la unidad de control, se ha optado por unificar el tratamiento de las instrucciones de tipo R, y dejar la generación de los cuatro bits de **ALU Operation** a un circuito separado llamado **Control de la ALU**.
  - Analizando las instrucciones se observa que hay tres casos de uso para la **ALU**:
    - Con **lw** o **sw**, sumará el registro base más el desplazamiento para obtener la dirección del operando residente en memoria.
    - Con **beq** realizará una resta de dos registros para saber si son iguales.
    - En las instrucciones de **tipo R** se realizará la operación indicada en el campo de función.
  - Basta con que la unidad de control genere una señal de dos bits, llamada **ALUOp**, que distinga entre los tres casos.
    - **ALUOp** y los bits del campo de función de la instrucción servirán de entradas para el circuito **Control de la ALU**, que generará los cuatro bits de **ALU Operation**.
    - Así, **Control de la ALU** es un circuito que sería una parte de la propia unidad de control, pero estaría desgajado de ella con objeto de mantener una unidad de control "central" sencilla con pocas entradas.
  - En concreto, los valores de **ALUOp** serán:
    - **00**: la ALU hace forzosamente una suma (instrucciones **lw** y **sw**).
    - **01**: la ALU hace forzosamente una resta (instrucción **beq**).
    - **10**: la ALU hace la operación indicada por el campo de función (instrucciones de **tipo R**).



- La diapositiva muestra la versión del camino de datos uniciclo que puede ejecutar los saltos condicionales **beq**.
    - Se muestra el circuito que calcula el valor de la señal **FuentePC provisional**, que no puede estar generada directamente por la unidad de control, ya que depende también del indicador **Cero** generado por la **ALU** en la comparación de registros fuente.
        - **FuentePC provisional** es el producto lógico de **Cero** por una nueva señal llamada **Salto Cond**, que valdrá 1 si estamos ejecutando la instrucción **beq**, y 0 en el resto de los casos.
        - Tres situaciones que pueden darse para decidir cuál es el próximo valor del **PC**:
            - Si la instrucción es aritmético-lógica de **tipo R**, **lw** o **sw**, la unidad de control podrá **Salto Cond** a 0, y en el **PC** se grabará la dirección incrementada.
            - Si la instrucción es **beq**, la unidad de control pondrá **Salto Cond** a 1, y entonces la dirección grabada en el **PC** dependerá del valor de la señal **Cero**:
                - Si los operandos fuente son distintos (señal **Cero=0**), se seguirá grabando la dirección incrementada.
                - Si los operandos fuente son iguales (señal **Cero=1**), se grabará la dirección de ramificación.



- Aquí hemos agregado la instrucción **j**, que realiza un salto incondicional.
  - Llamamos **FuentePC** a la señal que controla el multiplexor de más arriba a la derecha:
    - Permite elegir entre el valor PC + 4 o el del salto beq (**FuentePC = 1**).
    - Y el del salto incondicional (**FuentePC = 0**).



MIPS UNICICLO
(versión definitiva)

- Versión definitiva del camino de datos uniciclo, donde hemos insertado la unidad de control con todas las señales.

CONSTRUCCIÓN DEL CAMINO DE DATOS UNICICLO

Señal de control	Efecto cuando vale 0	Efecto cuando vale 1
RegDest	El identificador de registro destino está en **rt** (bits 20-16)	El identificador de registro destino está en **rd** (bits 15-11)
EscrReg	Ninguno	El registro destino se escribe con el valor correspondiente (el que entra por **Reg. de escritura**)
FuenteALU	El segundo operando de la ALU proviene del segundo registro leído	El segundo operando de la ALU es el desplazamiento extendido a 32 bits
ALUOp	Señal de 2 bits (valores 00/01/10). Indica qué operación debe realizar la ALU en función de la instrucción (sumar, restar, Funct), y junto con Funct sirve para calcular los 3 bits de operación de la ALU	
SaltoCond	El PC se incrementa normalmente (sumándole 4)	Si la condición de ramificación es cierta, el PC se carga con la dirección de ramificación, dada por la salida del sumador de cálculo de direcciones
FuentePC	Es el producto lógico de SaltoCond con el indicador Cero generado por la ALU al comparar (restar) dos registros en la instrucción de ramificación	
	O bien se incrementa el PC de la manera habitual o bien se carga en el PC la dirección de ramificación	Se carga en el PC la dirección de un salto incondicional
LeerMem	Ninguno	Se lee una posición de memoria y su contenido se coloca a la salida de datos
EscrMem	Ninguno	Se escribe una posición de memoria con el valor dado en la entrada de datos
MemaReg	El valor en la entrada del banco de registros procede de la ALU	El valor de la entrada del banco de registros procede de la memoria

36

- Resumen de las señales de control presentes en el camino de datos y la acción que producen dependiendo de que la unidad de control las active o no.
  - El valor no activo (**0**) de las señales de control debería tomarse como un valor "por defecto", que sería el aplicable cuando la instrucción no requiere usar el módulo controlado por las mismas.
    - Por ejemplo, cuando no sea preciso escribir en el banco de registros, entonces **EscrReg** debe valer 0.
    - El valor de ciertas señales puede comportar que el valor de otras sea indiferente.
      - Fijándonos en **EscrReg**, si vale 0, entonces el valor de **MemaReg** y **RegDest** es irrelevante, ya que su efecto es seleccionar el número y contenido del registro que vamos a grabar, y como no se va a grabar ninguno, entonces da igual el valor que tomen (podemos suponer que valen "**X**").
    - Las señales **LeerMem** y **EscrMem** deben valer 0 cuando no se va a realizar ningún acceso a la memoria de datos.
    - **EscrReg** debe valer 0 cuando no haya escritura en el banco de registros.
    - Asimismo, **SaltoCond** debe valer 1 en ramificaciones, y 0 en el resto, porque está controlando el valor que se va a escribir en el **PC** (recuérdese que el **PC** no tiene habilitación de escritura, por lo que se graba un nuevo valor en él en cada ciclo de reloj).
    - Todas las demás señales están controlando multiplexores (incluso **ALUOp**, que de alguna forma controla los multiplexores internos de la ALU), con lo que pueden valer "**X**" si el efecto que producen no es relevante en la instrucción en curso.

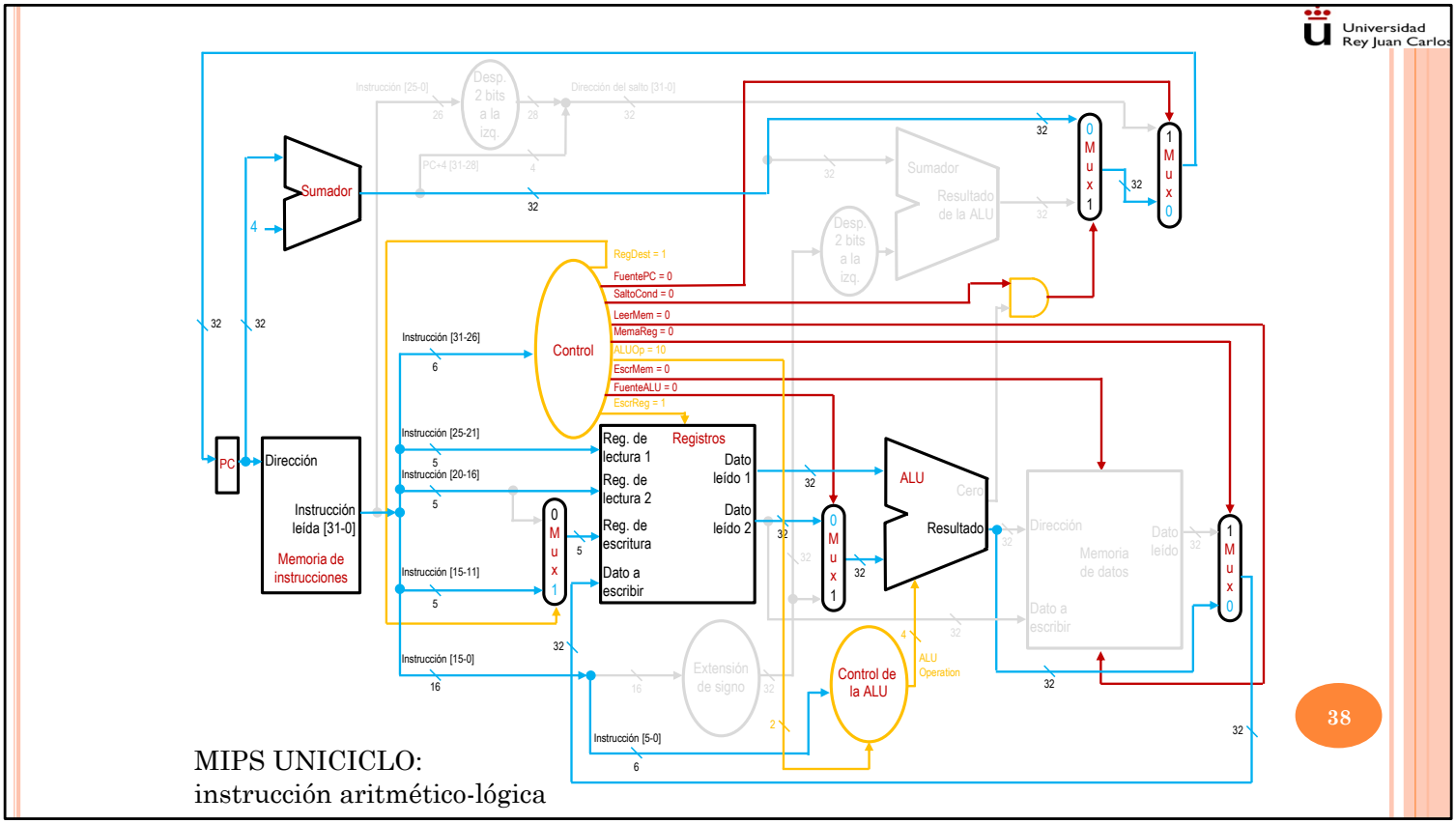
Universidad Rey Juan Carlos

FUNCIONAMIENTO PARA LAS INSTRUCCIONES SELECCIONADAS

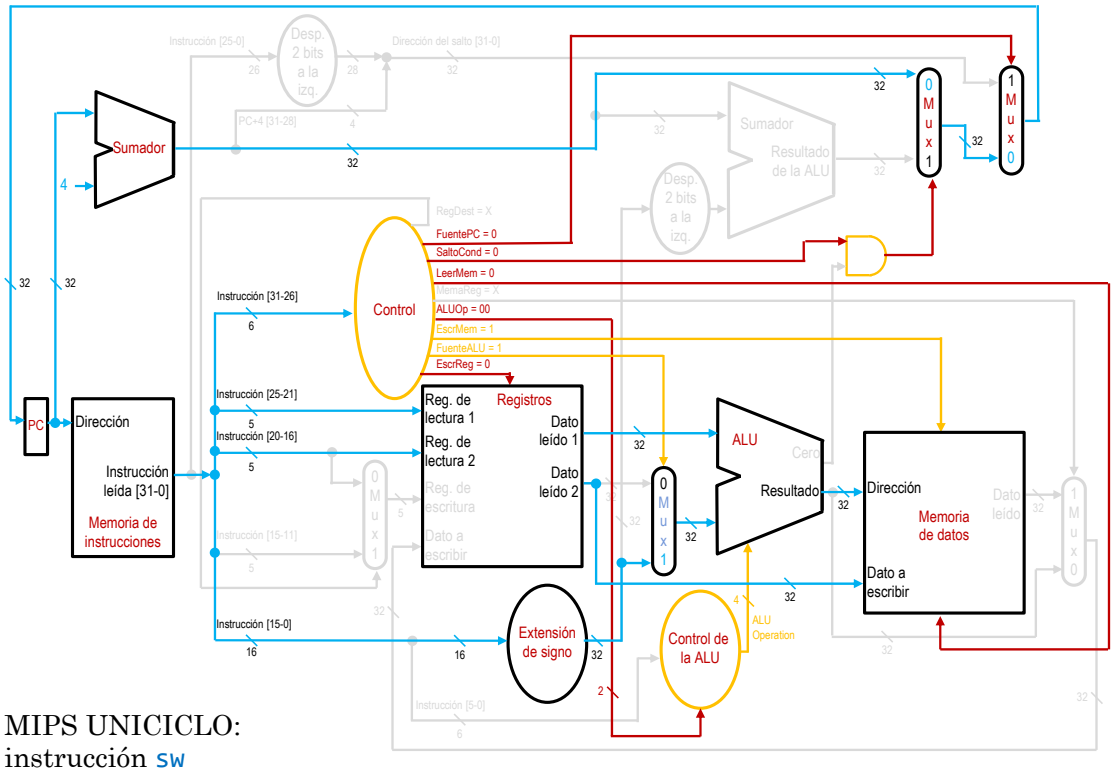
- Aritmético-lógica
- Almacenamiento `sw`
- Carga `lw`
- Ramificación condicional `beq` (con/sin salto)
- Salto incondicional `j`

37

- En las próximas transparencias veremos cómo se ejecutan las diferentes instrucciones.
- El convenio de colores es el siguiente:
  - Los cables por los que fluyen los datos para una determinada instrucción aparecen en azul, mientras que los que no intervienen en la instrucción aparecen en gris claro.
  - Las unidades funcionales que están activas aparecen con el borde en negro, mientras que las que no intervienen están en gris claro. ¡OJO! Que no intervengan en la instrucción no significa que no estén realizando alguna operación. Por ejemplo, la ALU SIEMPRE está calculando algo, incluso en la instrucción j, aunque el cálculo no sirve para nada.
  - Las señales de control que valen 0 aparecen en marrón oscuro, mientras que las que valen 1 aparecen en naranja. Si el valor es irrelevante (X), están en gris claro.
  - De igual forma, los componentes de control activos aparecen en naranja y los inactivos en marrón oscuro.



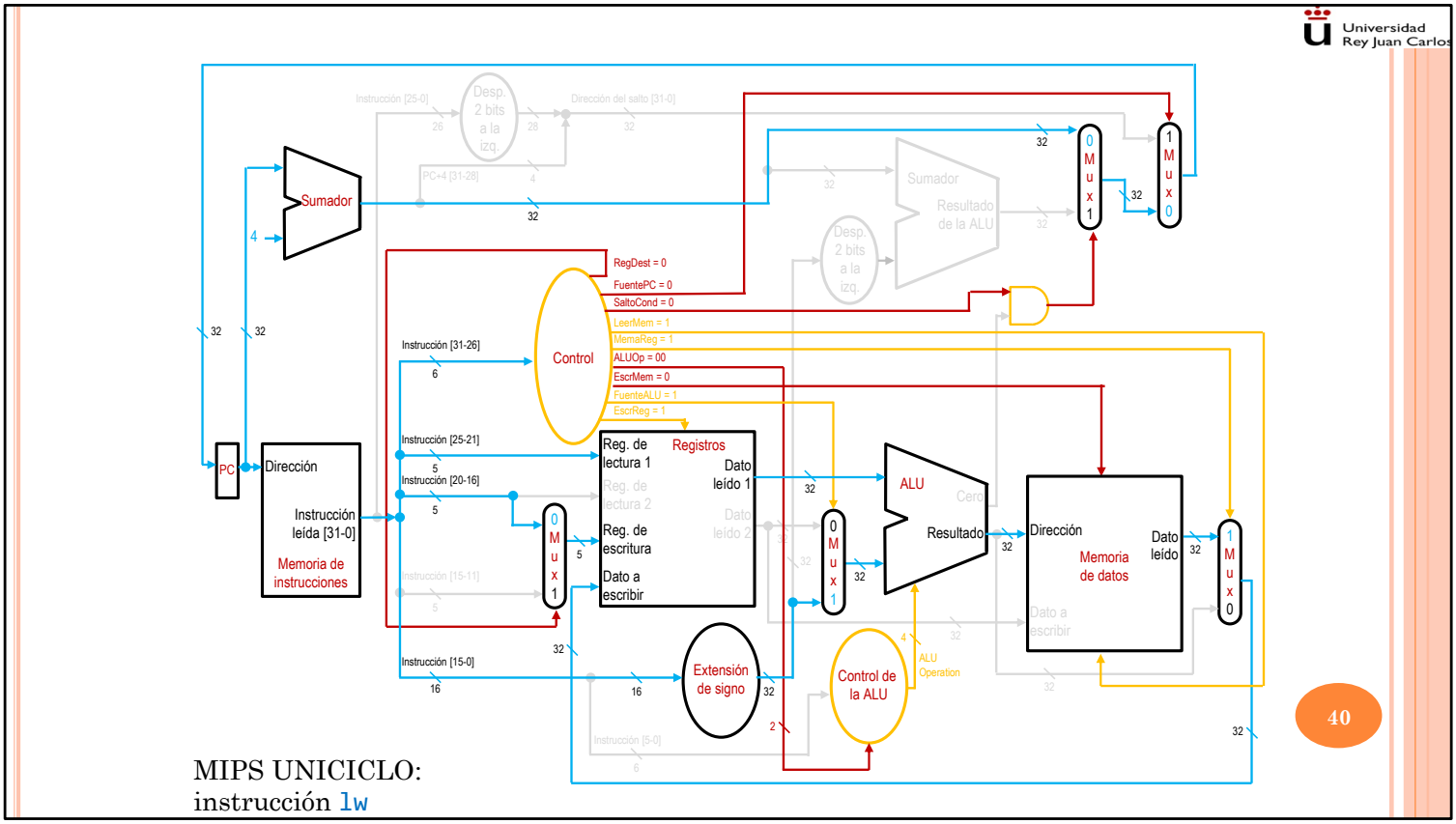
- Microoperaciones realizadas por esta instrucción:
  - PC ← PC + 4
  - Reg[M[PC][15-11]] ← Reg[M[PC][25-21]] op Reg[M[PC][20-16]]
- Vídeo explicativo: https://youtu.be/K7izja8h-Lk



MIPS UNICICLO:
instrucción `sw`

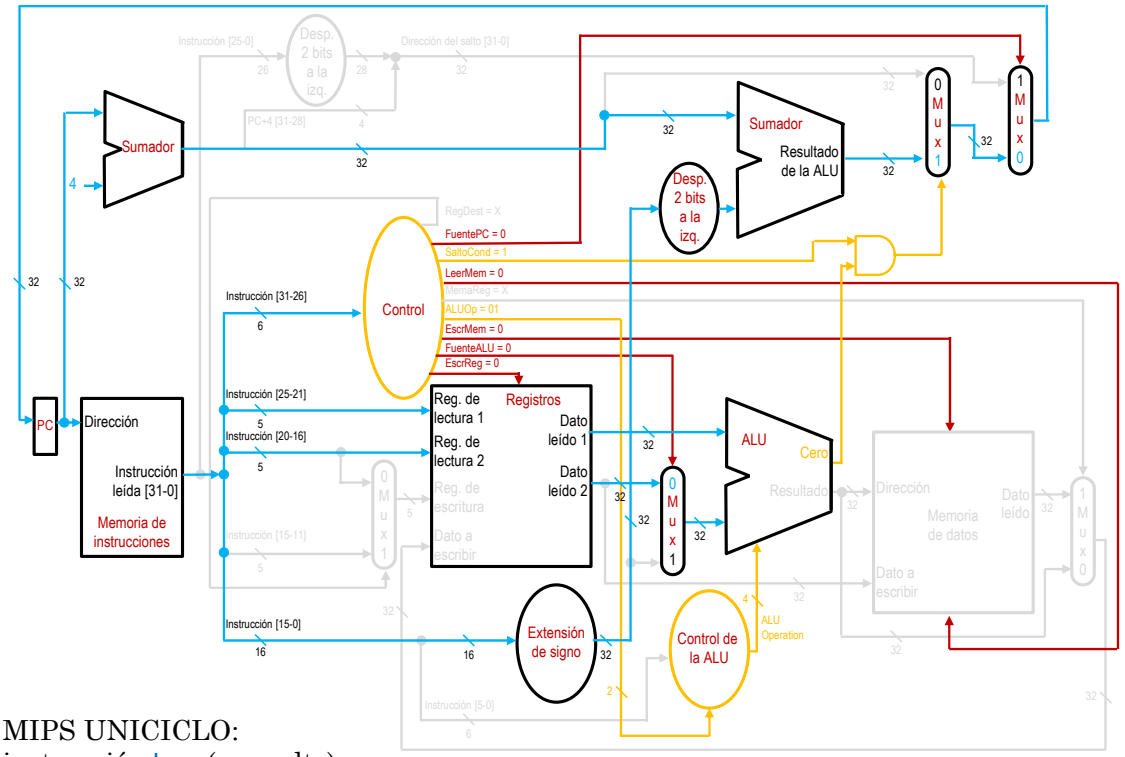
39

- Microoperaciones realizadas por esta instrucción:
  - PC ← PC + 4
  - M[Reg[M[PC][25-21]] + extSigno16a32(M[PC][15-0])] ← Reg[M[PC][20-16]]
- Vídeo explicativo: https://youtu.be/g6Zpor6g2tA



MIPS UNICICLO:
instrucción `lw`

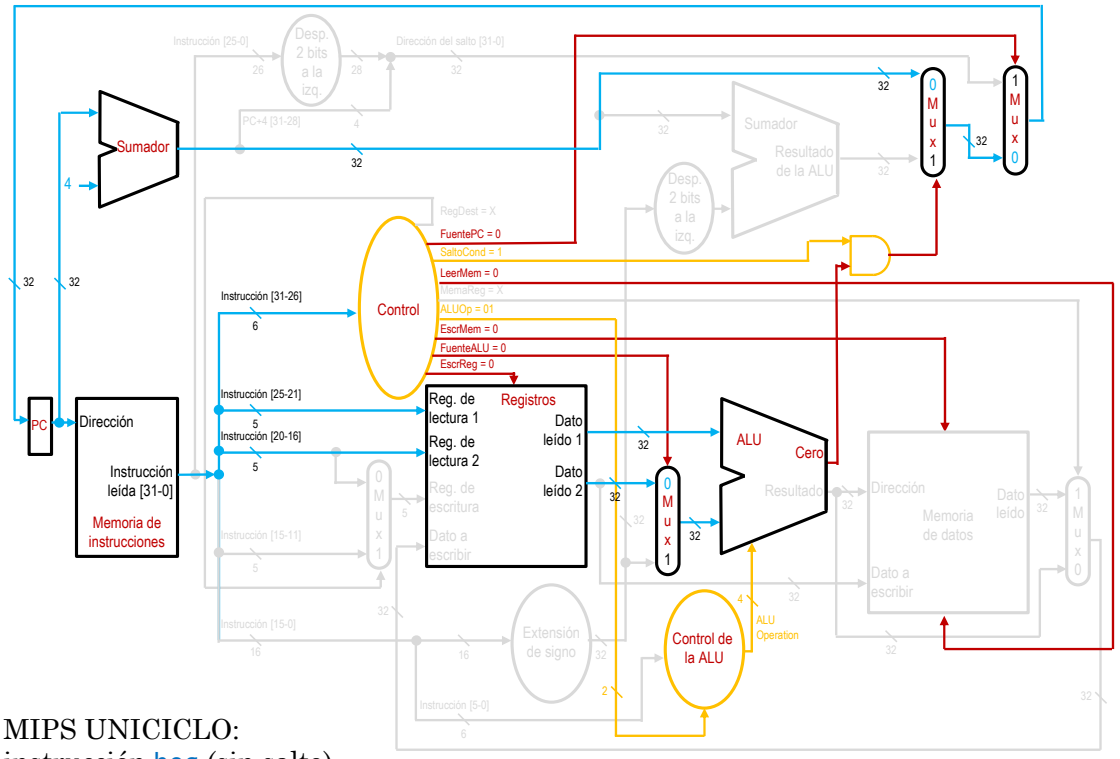
- Microoperaciones realizadas por esta instrucción:
  - PC ← PC + 4
  - Reg[M[PC][20-16]] ← M[Reg[M[PC][25-21]] + extSigno16a32(M[PC][15-0])]
- Vídeo explicativo: https://youtu.be/mF5szbqpeu0



41

MIPS UNICICLO:
instrucción beq (con salto)

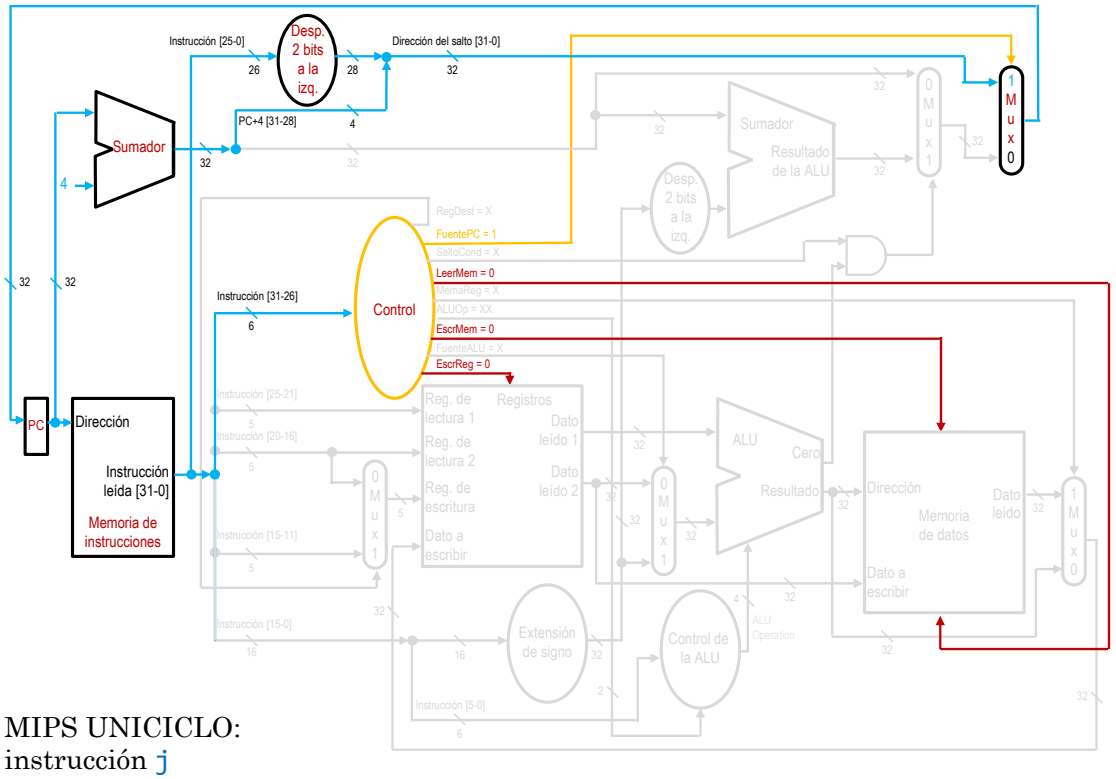
- Microoperaciones realizadas por esta instrucción:
  - Si (Reg[M[PC][25-21]] = Reg[M[PC][20-16]]) entonces PC ← PC + 4 + extSigno16a32(M[PC][15-0])<<2
- Vídeo explicativo: https://youtu.be/yUeIaziNMaw



MIPS UNICICLO:
instrucción beq (sin salto)

42

- Microoperaciones realizadas por esta instrucción:
  - Si (Reg[M[PC][25-21]] ≠ Reg[M[PC][20-16]]) entonces PC ← PC + 4
- Vídeo explicativo: https://youtu.be/yUeIaziNMaw

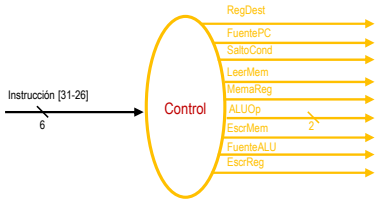


MIPS UNICICLO:
instrucción j

- Microoperaciones realizadas por esta instrucción:
  - PC ← (PC + 4)[31-28] || (M[PC][25-0]<<2)
- Vídeo explicativo: https://youtu.be/C_hSDkLd2KU

UNIDAD DE CONTROL DEL CAMINO DE DATOS UNICICLO

- Control sencillo.
  - Instrucciones con longitud fija
  - Código de operación en posición fija
  - Instrucciones con codificación homogénea
- Entrada de la unidad de control: **Instrucción[31-26]**



44

- Los formatos de las instrucciones de **MIPS** utilizadas son muy homogéneos y de longitud fija, lo que facilita enormemente el diseño de la unidad de control.
  - El campo del código de operación tiene siempre el mismo tamaño (6 bits) y está siempre en el mismo lugar (**Instrucción[31-26]**).
  - Solo se usan unas pocas de las combinaciones posibles para el código de operación.
- Para esta versión reducida de MIPS, el control es extremadamente sencillo y la mejor opción es implementar la unidad de control mediante una pequeña red de puertas.
- Si se incluyesen todas las instrucciones del repertorio de MIPS, la función de control se especificaría mediante una gran tabla y su síntesis se realizaría mediante otras técnicas, por ejemplo, mediante decodificadores, una **ROM**, o una **PLA**.
  - Esto está fuera de los límites de este curso y no lo vamos a ver.

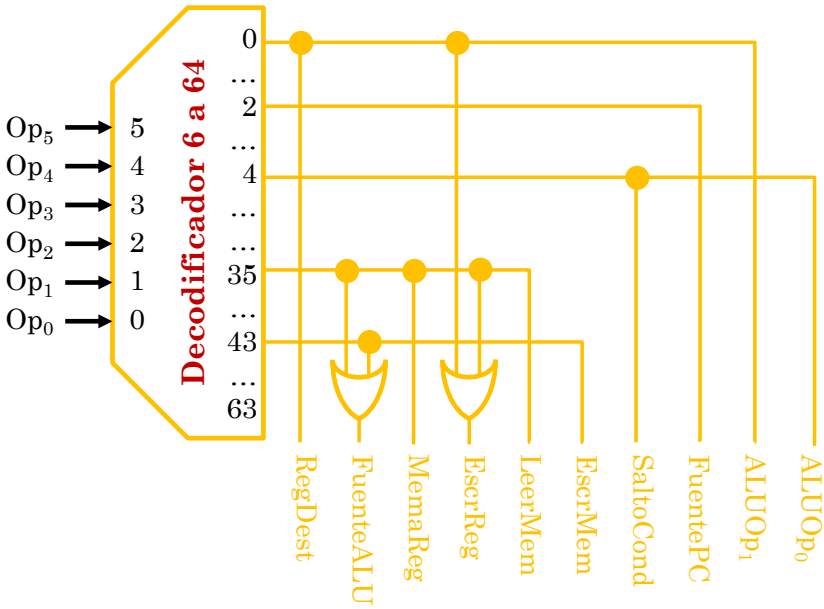
UNIDAD DE CONTROL DEL CAMINO DE DATOS UNICICLO

	Entradas						Salidas									
	Instrucción[31] = Op_5	Instrucción[30] = Op_4	Instrucción[29] = Op_3	Instrucción[28] = Op_2	Instrucción[27] = Op_1	Instrucción[26] = Op_0	RegDest	FuenteALU	MemaReg	EscrReg	LeerMem	EscrMem	SaltoCond	FuentePC	$ALUOp_1$	$ALUOp_0$
Aritm.-lóg.	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0
j	0	0	0	0	1	0	X	X	X	0	0	0	X	1	X	X
beq	0	0	0	1	0	0	X	0	X	0	0	0	1	0	0	1
lw	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0	0
sw	1	0	1	0	1	1	X	1	X	0	0	1	0	0	0	0

45

- Tabla de verdad del circuito de la unidad de control.
  - Entradas:
    - Código de operación (6 bits).
  - Salidas:
    - Señales de control que gobiernan los restantes elementos del circuito (10 bits).
- La tabla de verdad tiene 6 entradas, luego tiene 2^6=64 filas, de las cuales solo hemos dibujado las 5 que son relevantes.
- Hay que calcular la función de conmutación de cada una de las salidas por el método que se crea más conveniente.

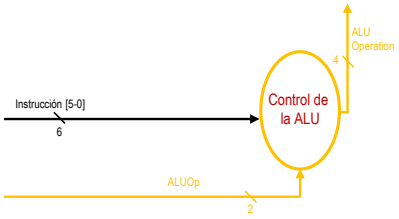
UNIDAD DE CONTROL DEL CAMINO DE DATOS UNICICLO



46

- Una manera muy sencilla de construir el circuito de la unidad de control del caso uniciclo es mediante un decodificador de 6 entradas y 64 salidas (numeradas de 0 a 63), que permite implementar la primera forma canónica (suma de minitérminos) de las salidas de manera directa.
- Las señales que queremos generar son las salidas del decodificador que corresponden a las filas de la tabla que contienen un 1.
    - En dos casos hay que usar una puerta OR de 2 entradas, ya que son dos filas las que valen 1.
- La ventaja de usar el decodificador es que permite añadir nuevas instrucciones de manera inmediata, simplemente conectando las salidas del mismo con puertas OR para producir las nuevas señales de control.
- Otra posibilidad para implementar este circuito es simplificar la tabla de verdad y utilizar las puertas lógicas AND, OR y NOT que fueran necesarias, eliminando así el decodificador.
    - Sin embargo, en este caso habría que volver a simplificar todo el circuito si añadimos nuestras instrucciones.

UNIDAD DE CONTROL DEL CAMINO DE DATOS UNICICLO



Operaciones efectuadas por la ALU:

ALU Operation (Ainvert Bnegate op1 op0)	Operación
0000	and
0001	or
0010	add
0110	sub
0111	slt
1100	nor

Ver tema de la ALU

Control de la ALU: dos entradas.

1. **ALUOp**
   - Generada por la unidad de control.
   - Dos bits.
     - `00`: suma (`lw` y `sw`).
     - `01`: resta (`beq`).
     - `10`: operación indicada por el campo **Funct** (**tipo R**).
2. **Funct:**
   - Procede de los bits 5-0 de la propia instrucción.
   - Solo se usa si la instrucción actual es de tipo R.

47

- El control de la **ALU** se realiza mediante dos señales, **ALUOp** y **Instrucción[5-0]**, que actúan como entradas en el circuito **Control de la ALU**, que genera los cuatro bits de la señal **ALU Operation** utilizados para gobernar la operación realizada por la **ALU**.
  - **ALUOp** es una señal generada por la unidad de control y consta de dos bits, que sirven para distinguir entre las tres situaciones posibles en que podemos encontrarnos para decidir qué operación realiza la **ALU** dependiendo de cuál sea la instrucción actual.
    - **00**: fuerza una suma, lo que es útil cuando nos encontramos con instrucciones como **lw** y **sw**, que usan la **ALU** para calcular una dirección de un dato residente en la memoria de datos.
    - **01**: fuerza una resta, que es lo que necesita la instrucción de ramificación **beq** para comparar dos registros y ver si su contenido coincide o no.
    - **10**: la operación realizada depende el campo de función de la instrucción, utilizado para distinguir entre las diferentes instrucciones de **tipo R**.
  - **Instruction[5-0]** son los seis bits de menor peso de la instrucción y corresponden con los bits del **campo de función**, presente en las instrucciones de **tipo R**. En las instrucciones aritmético-lógicas de formato R el código de operación es siempre el mismo (000000) y, por tanto, este campo de función permite distinguir unas instrucciones de otras y es decisivo para generar los valores de la señal **ALU Operation** necesarios para su correcta ejecución.

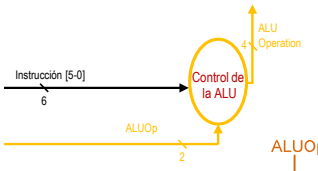
UNIDAD DE CONTROL DEL CAMINO DE DATOS UNICICLO

Instruc-ción	Operación	Acción en ALU	ALUOp	Funct F5 F4 F3 F2 F1 F0	ALU Operation
lw	Cargar palabra	Sumar	00	XXXXXX	0010
sw	Almacenar palabra	Sumar	00	XXXXXX	0010
beq	Ramificar si iguales	Restar	01	XXXXXX	0110
add	Sumar (tipo R)	Sumar	10	100000	0010
sub	Restar (tipo R)	Restar	10	100010	0110
and	And (tipo R)	And	10	100100	0000
or	Or (tipo R)	Or	10	100101	0001
nor	Nor (tipo R)	Nor	10	100111	1100
slt	Activar si menor que (tipo R)	Activar si menor que	10	101010	0111

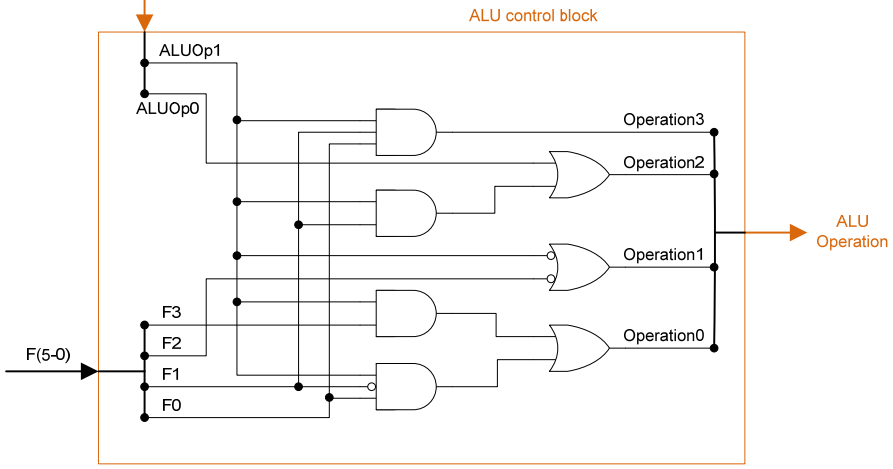
48

- En esta tabla se resumen todas las situaciones posibles en que se puede encontrar la **ALU** en el repertorio de instrucciones reducido elegido para esta implementación.
  - La tabla contiene una línea por cada instrucción ejecutada, indicándose en cada caso:
    - Qué operación realiza la instrucción.
    - El valor requerido para **ALUOp**.
    - El contenido el campo de función (solo en instrucciones de **tipo R**).
    - La acción necesaria en la **ALU.**
    - Las entradas de control que hay que suministrar a la propia **ALU**, que en la tabla de operaciones de la **ALU** aparecen en la columna **ALU Operation**.
- Entonces, el circuito **Control de la ALU** tendrá como:
  - Entradas:
    - **ALUOp**.
    - **Campo de función (funct)**.
  - Salida:
    - **ALU Operation**.
- Se observa que las dos primeras filas (que corresponden con las instrucciones **lw** y **sw**) coinciden en cuanto al valor de dichas señales, y darán lugar a una única fila en la tabla de verdad del circuito.
- Cuando **ALUOp=00** o **ALUOp=01**, el campo de función no se utiliza para determinar la operación realizada por la **ALU**. Sin embargo, cuando **ALUOp=10**, el valor de **ALU Operation** depende de los bits **F5-F0**.
- Se observa que, con las instrucciones de tipo R seleccionadas, **F5=1** y **F4=0**, con lo que no tienen influencia en la simplificación final de la función.
  - Otra cosa sería si se incluyesen nuevas instrucciones que tuvieran valores diferentes en dichos bits.

UNIDAD DE CONTROL DEL CAMINO DE DATOS UNICICLO



Circuito mínimo para el control de la ALU



49

- Aquí se muestra el circuito resultante que genera **ALU Operation** a partir del campo de función **Instrucción[5-0]** y de **ALUOp**.
  - Como puede verse, ni **F5** ni **F4** aparecen en el resultado final de la simplificación.

Universidad Rey Juan Carlos

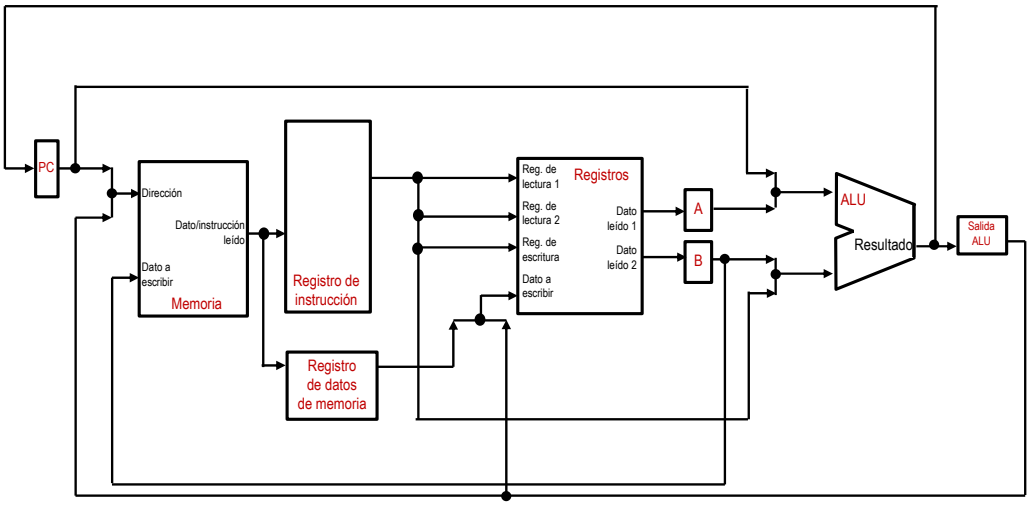
IMPLEMENTACIÓN MULTICICLO

- Construcción incremental del camino de datos multiciclo
- Funcionamiento para las instrucciones seleccionadas
- Unidad de control del camino de datos multiciclo

50

- Camino de datos multiciclo:
  - Circuito del procesador capaz de ejecutar las instrucciones del programa de tal forma que cada una de ellas se completa en **un número entero de ciclos de reloj**, si bien cada instrucción puede necesitar **más o menos ciclos**.
  - La duración del periodo de reloj debe adaptarse a la **fase más lenta** que puede necesitar una instrucción del repertorio.
- En la implementación multiciclo dividiremos las instrucciones en diversos ciclos de reloj, más breves que en el caso uniciclo.
  - Consideraremos que el ciclo comienza típicamente leyendo el valor de un registro y termina escribiendo otro valor en otro registro. En el tiempo que dura el ciclo solo se puede hacer una operación elemental, que en nuestro caso pueden ser:
    - Un acceso a la memoria, ya sea para leer o escribir en ella.
    - Dos accesos de lectura y uno de escritura (este opcional) en el banco de registros.
    - Una operación en la ALU.
  - Por ejemplo, NO estaría permitido que en el mismo ciclo de reloj se leyera un valor en el banco de registros y el resultado sirviera como operando en la ALU, ya que eso implicaría dos operaciones elementales.
  - Consideraremos que las operaciones de extensión de signo, desplazamiento de bits y la selección realizada por los multiplexores se realizan en un tiempo despreciable frente a las operaciones que hemos denominado elementales.

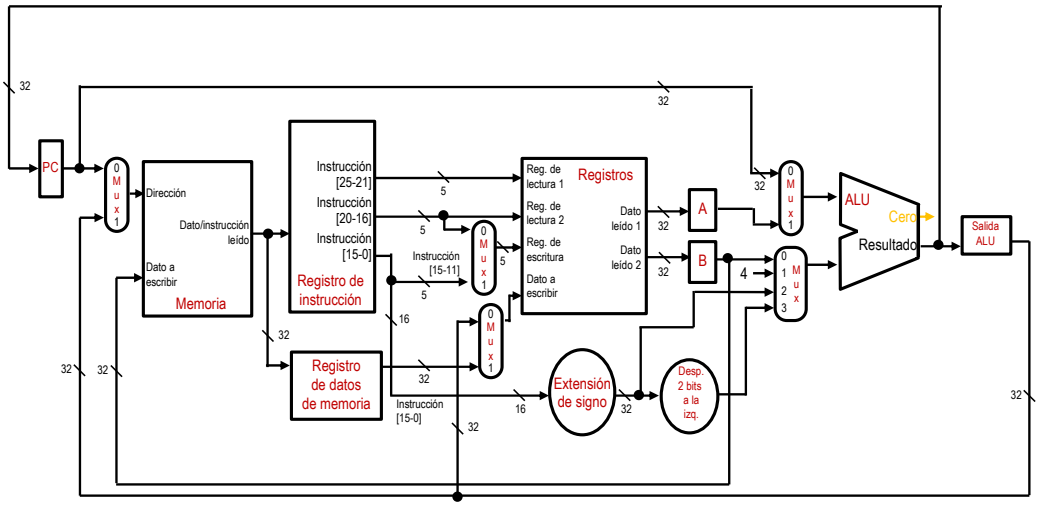
CONSTRUCCIÓN DEL CAMINO DE DATOS MULTICICLO



51

- Esta diapositiva presenta una visión de alto nivel del camino de datos multiciclo, con sus elementos principales:
  - La unidad de memoria.
  - El banco de registros.
  - La unidad aritmético-lógica.
  - Las conexiones necesarias entre ellos.
- En este modelo no hace falta replicar elementos, pues estos pueden utilizarse varias veces a lo largo de la ejecución de la instrucción, con la única restricción de no utilizar la misma unidad funcional dos veces en el mismo ciclo.
- Compartir las unidades funcionales causará que los multiplexores sean más anchos y además ahora será preciso añadir una serie de **registros transparentes** de propósito específico encargados de conservar los datos entre los ciclos de la instrucción. Estos registros son:
  - Registro de instrucción (*instruction register*, **IR**): para almacenar la última instrucción leída en memoria.
  - Registro de datos de memoria (*memory data* register, **MDR**): para almacenar el último dato leído de memoria.
  - Registros para operandos en banco de registros: uno en cada salida del banco de registros para lecturas (**A** y **B**).
  - Registro para la salida de la **ALU** (**Salida ALU**): para guardar el resultado de la **ALU** (también se le llama en ocasiones **registro acumulador**).
- Quitando el **IR**, todos los demás registros cargan el valor de su entrada en todos los ciclos de reloj y, por tanto, la unidad de control no necesitará generar para ellos señales de habilitación de carga.
  - Sin embargo, el **IR** se cargará una sola vez a lo largo de la ejecución de la instrucción (en la fase de lectura o *fetch*) y, por tanto, para él sí será preciso que la unidad de control genere la pertinente señal de escritura.

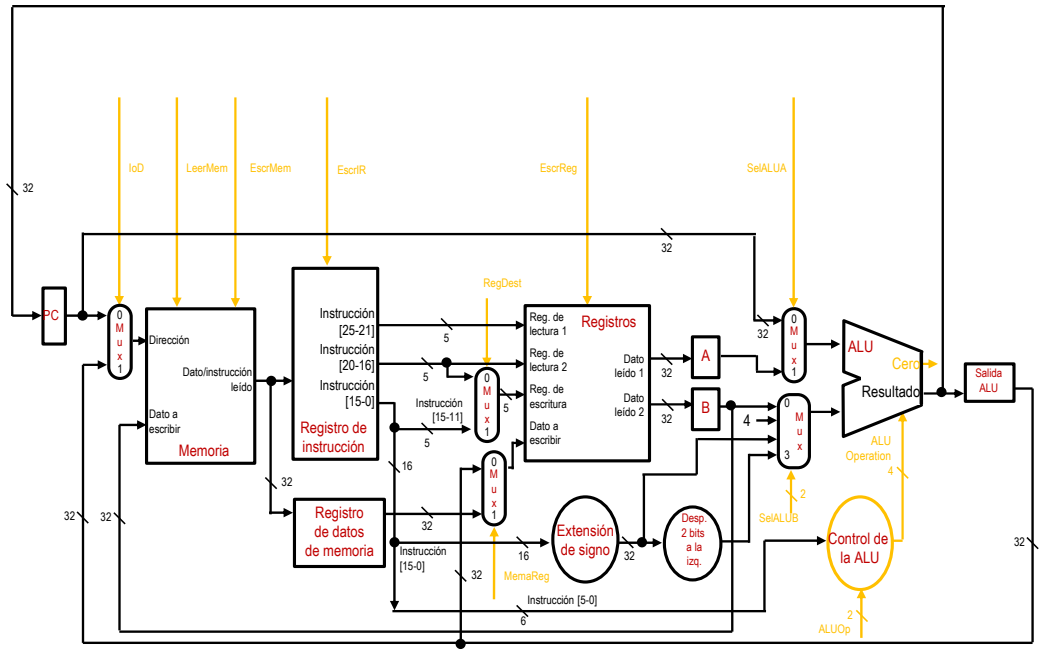
CONSTRUCCIÓN DEL CAMINO DE DATOS MULTICICLO



52

- La reutilización de las unidades funcionales en diferentes fases provoca que aumente el ancho de los multiplexores existentes y que además sean necesarios multiplexores donde antes no los había.
    - Reutilización de la **ALU**,:
        - Dos opciones para el primer operando:
            - El **PC**, para obtener la dirección incrementada o la dirección de ramificación.
            - El primer registro de lectura del banco de registros.
        - Por tanto, hay que añadir un multiplexor para seleccionar entre ambos, más una señal de control para gobernarlo.
        - Cuatro para el segundo operando de la ALU:
            - El segundo registro de lectura.
            - Un 4 (para incrementar el **PC**).
            - El campo *desplazamiento* extendido en signo (para calcular direcciones o para operaciones con inmediatos).
            - El mismo campo *desplazamiento* extendido en signo y desplazado dos lugares a la izquierda (para calcular la dirección de ramificación).
        - Por ello, el multiplexor para seleccionar el segundo operando de la **ALU** tiene ahora cuatro entradas, y la señal para controlarlo es de dos bits.
    - También se añade un multiplexor (con su señal de control) para seleccionar la opción correcta en la entrada de direcciones de la memoria:
        - Cuando se acceda a instrucciones, la dirección debe venir dada por el contenido del **PC**.
        - Cuando se acceda a datos, la dirección provendrá del registro **Salida ALU**.

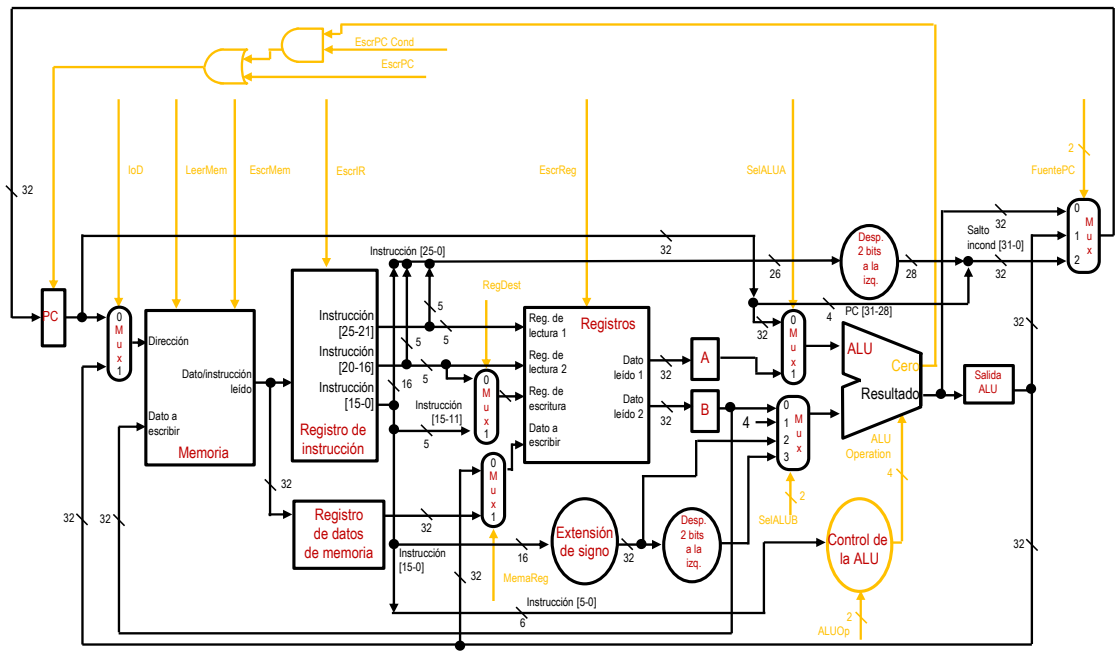
CONSTRUCCIÓN DEL CAMINO DE DATOS MULTICICLO



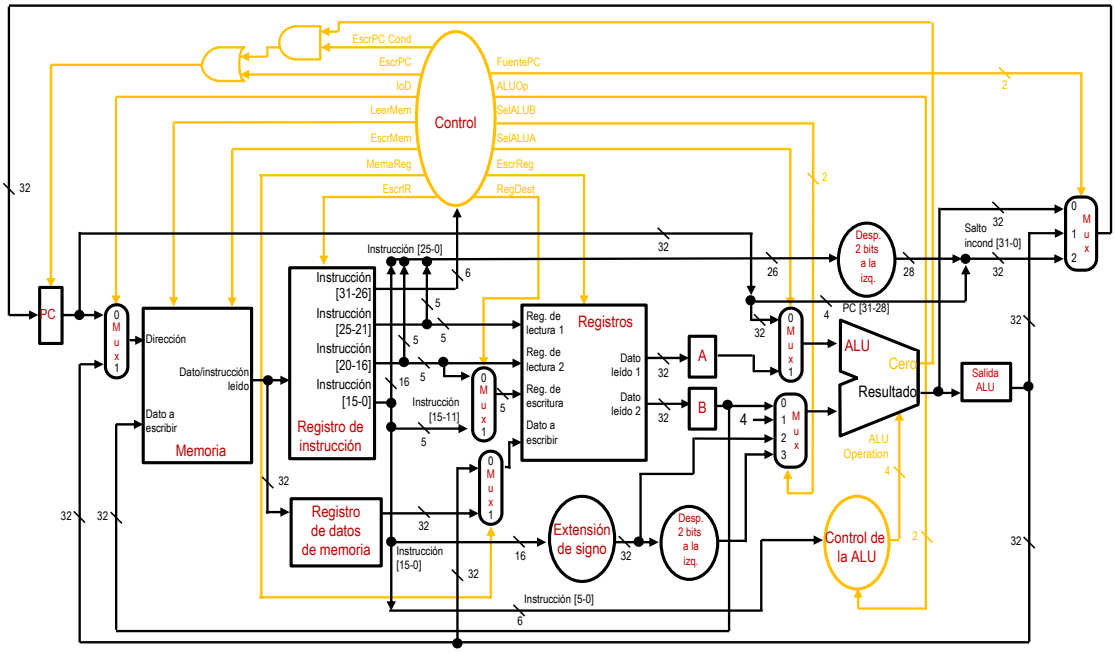
53

- Añadimos las señales de control, más el circuito de control de la **ALU** (que será idéntico al del caso uniciclo).
- El registro de instrucción lleva una señal de habilitación de escritura (**EscrIR**), puesto que solo se escribe en el ciclo de lectura de la instrucción (*fetch*).
- Los registros **A**, **B**, **Salida ALU** y **MDR** se escriben incondicionalmente en todos los ciclos de reloj y, por tanto, no necesitan una señal de habilitación de escritura.
- Falta la señal de habilitación de escritura del **PC**, que solo será modificado en los siguientes casos:
    - Cuando se calcule el **PC** incrementado (**PC+4**) en la fase de *fetch*.
    - Cuando se ejecute una instrucción de salto (aún no contemplada en el dibujo).
    - Cuando se ejecute una instrucción de ramificación condicional y la condición de ramificación sea cierta (aún tampoco contemplado en el dibujo).
- En los dos primeros casos la escritura sobre el **PC** será incondicional y en el siguiente la escritura se realizará o no dependiendo del valor de la señal **Cero** emitida por la **ALU**.

CONSTRUCCIÓN DEL CAMINO DE DATOS MULTICICLO



- En esta versión del camino de datos se muestra la unidad de control con las señales que genera y se añade la circuitería necesaria para implementar las instrucciones de salto y de ramificación condicional.
- Los elementos añadidos son los siguientes:
  - Un circuito para construir la dirección de salto, similar al utilizado en el camino de datos uniciclo.
  - Un multiplexor para elegir entre las tres opciones: dirección incrementada (0), dirección de ramificación (1) y dirección de salto (2).
  - Una señal (**EscrPC**) para escribir incondicionalmente en el **PC**.
  - Una señal (**EscrPC Cond**) para escribir condicionalmente en el **PC**. Esta señal se activará en la última fase de la ejecución de las instrucciones de ramificación condicional.
  - Un pequeño circuito que generará la señal de habilitación de escritura sobre el **PC**.
- La señal de habilitación de escritura sobre el **PC** se activará en dos situaciones:
  - Siempre que la unidad de control active la señal **EscrPC**, lo que sucederá cuando se quiera actualizar el **PC** con el valor incrementado en la fase de *fetch*, o bien cuando nos encontremos en la última fase de ejecución de las instrucciones de salto incondicional.
  - En la última fase de las instrucciones de ramificación condicional, cuando además se cumpla que la condición sea cierta. En la instrucción **beq** la unidad de control activará **EscrPC Cond**, con lo cual la escritura en el **PC** se realizará si la señal **Cero** emitida por la **ALU** es igual a 1. Como en la **ALU** se habrá realizado una resta entre dos registros, la señal **Cero** se activará únicamente cuando el contenido de ambos sea idéntico. Si el contenido de ambos registros es distinto, la señal **Cero** valdrá 0, y la escritura sobre el **PC** no se habilitará.



- Versión definitiva del camino de datos multiciclo.

CONSTRUCCIÓN DEL CAMINO DE DATOS MULTICICLO

Señal de control	Efecto cuando vale 0	Efecto cuando vale 1
RegDest	El identificador de registro destino está en **rt** (bits 20-16)	El identificador de registro destino está en **rd** (bits 15-11)
EscrReg	Ninguno	El registro destino se escribe con el valor correspondiente
SelALUA	El primer operando de la **ALU** proviene del **PC**	El primer operando de la **ALU** proviene del registro **A**
LeerMem	Ninguno	Se lee una posición de memoria y su contenido se coloca a la salida de datos
EscrMem	Ninguno	Se escribe una posición de memoria con el valor dado en la entrada de datos
MemaReg	El valor en la entrada del banco de registros procede de la **ALU**	El valor de la entrada del banco de registros procede del registro **MDR**
IoD	El **PC** suministra la dirección para acceder a memoria	**Salida ALU** suministra la dirección para acceder a memoria
EscrIR	Ninguno	La salida de memoria se escribe en el registro **IR**
EscrPC	Ninguno	Escribir el **PC** un valor que depende de la señal FuentePC
EscrPC Cond	Ninguno	Escribir el **PC** cuando la señal de resultado nulo de la **ALU** está también activa

- Esta tabla muestra el resumen de las señales de control presentes en el camino de datos y la acción que producen dependiendo de que la unidad de control las active o no.
    - El valor no activo de las señales de control debería tomarse como un valor "por defecto", que sería el aplicable cuando la instrucción no requiere usar el módulo controlado por las mismas.
        - Por ejemplo, cuando no sea preciso escribir en el banco de registros, entonces **EscrReg** debe valer 0.
        - Lo que sucede es que el valor de ciertas señales puede comportar que el valor de otras sea indiferente.
        - Fijándonos en **EscrReg**, si vale 0, entonces el valor de **MemaReg** y **RegDest** es irrelevante, ya que su efecto es seleccionar el número y contenido del registro que vamos a grabar, y como no se va a grabar ninguno, entonces da igual el valor que tomen (podemos poner que valen "**X**").
- Las señales **LeerMem** y **EscrMem** deben valer 0 cuando no se va a realizar ningún acceso a memoria. Como hemos visto, **EscrReg** debe valer 0 cuando no haya escritura en el banco de registros.
- **EscrIR** solo se activará en la fase de lectura de la instrucción.
- **EscrPC** se activa en la fase de lectura y en la de ejecución de las instrucciones de salto.
- **EscrPC Cond**, por su parte, se activará solo en la fase de ejecución de las instrucciones de ramificación.

CONSTRUCCIÓN DEL CAMINO DE DATOS MULTICICLO

Señal de control	Valor	Efecto
ALUOp	00	La ALU realiza una operación de suma
	01	La ALU realiza una operación de resta
	10	La operación realizada por la ALU viene dada por el campo **Funct** de la instrucción
SelALUB	00	El segundo operando de la ALU proviene del registro B
	01	El segundo operando de la ALU es la constante 4
	10	El segundo operando de la ALU son los 16 bits menos significativos del IR extendidos en signo a 32 bits
	11	El segundo operando de la ALU son los 16 bits menos significativos del IR extendidos en signo a 32 bits y desplazados dos lugares hacia la izquierda
FuentePC	00	La salida de la ALU (PC+4) se envía para ser escrita en el PC
	01	El contenido del registro Salida ALU (destino de la ramificación condicional) se envía para ser escrito en el PC
	10	La dirección de destino de salto en una instrucción **J** desplazada dos bits hacia la izquierda y concatenada con los 4 bits más significativos del PC se envía para ser escrita en el PC

- La unidad de control de la ALU se realizará igual que en el camino de datos uniciclo.

57

- **SelALUA** se usa para seleccionar el primer operando entrante en la **ALU**, mientras que **SelALUB** selecciona el segundo. **ALUOp** coincide con la señal homónima del camino de datos uniciclo.
- **FuentePC** permite determinar qué valor se encamina hacia la entrada de datos del **PC** y se elige entre tres opciones:
  - El **PC** incrementado (procedente de la **ALU**).
  - El **PC** de ramificación (procedente del registro **Salida ALU**, es preciso haberlo calculado en el ciclo anterior).
  - El **PC** de salto (procedente de la concatenación de la salida del desplazador de 2 bits a la izquierda con los cuatro bits más significativos del **PC**).
- La señal de habilitación de escritura sobre el **PC** es calculada, y corresponde con la siguiente expresión booleana: **EscrPC + (EscrPC Cond) · Cero**.

Nombre de la etapa		Acción en instruc-ción aritm.-lógica	Acción en instrucción de memoria	Acción en instruc-ción de ramifica-ción condicio-nal	Acción en instrucción de salto
IF	Carga de instrucción	IR ← M[PC] PC ← PC + 4			
ID	Decodificación de instrucción / carga de registros	A ← Reg[IR[25-21]] B ← Reg[IR[20-16]] Salida ALU ← PC + extSigno16a32(IR[15-0])<<2			
EX	Ejecución / cálculo de dirección / finalización de salto y ramificación	Salida ALU ← A op B	Salida ALU ← A + extSigno16a32(IR[15-0])	Si (A = B) entonces PC ← Salida ALU	PC ← PC[31-28] || (IR[25-0]<<2)
MEM	Acceso a memoria	–	lw: MDR ← M[Salida ALU] sw: M[Salida ALU] ← B	–	–
WB	Escritura del resultado	Reg[IR[15-11]] ← Salida ALU	lw: Reg[IR[20-16]] ← MDR	–	–

Universidad Rey Juan Carlos

58

1. Carga de la instrucción (*instruction fetch*, IF)

Se lee una instrucción de memoria y se carga en el **IR**, y en paralelo se incrementa el **PC**.

2. Decodificación de la instrucción (*instruction decoding*, ID)

La decodificación consume un ciclo de reloj completo, porque a veces requiere una circuitería compleja. A la vez pueden realizarse otras operaciones:

- Operaciones comunes a todas las instrucciones y que puedan adelantar su ejecución.
- Operaciones que puedan ser útiles para acelerar la ejecución de ciertas instrucciones (lectura de registros en el banco o cálculo de dirección de ramificación, por ejemplo).

Las siguientes etapas dependen de cuál sea la instrucción leída.

3. Ejecución ALU, cálculo de la dirección de memoria o finalización de salto (EX)

La ALU realiza una operación:

- En instrucciones R se ejecuta la operación pedida.
- En instrucciones **lw** o **sw** se calcula la dirección del operando de memoria.
- En ramificaciones se comparan los registros para conocer el valor de la condición, y en su caso se carga la dirección de ramificación (calculada en la fase anterior) en el **PC**.
- En saltos incondicionales se escribe la dirección de salto en el **PC**.

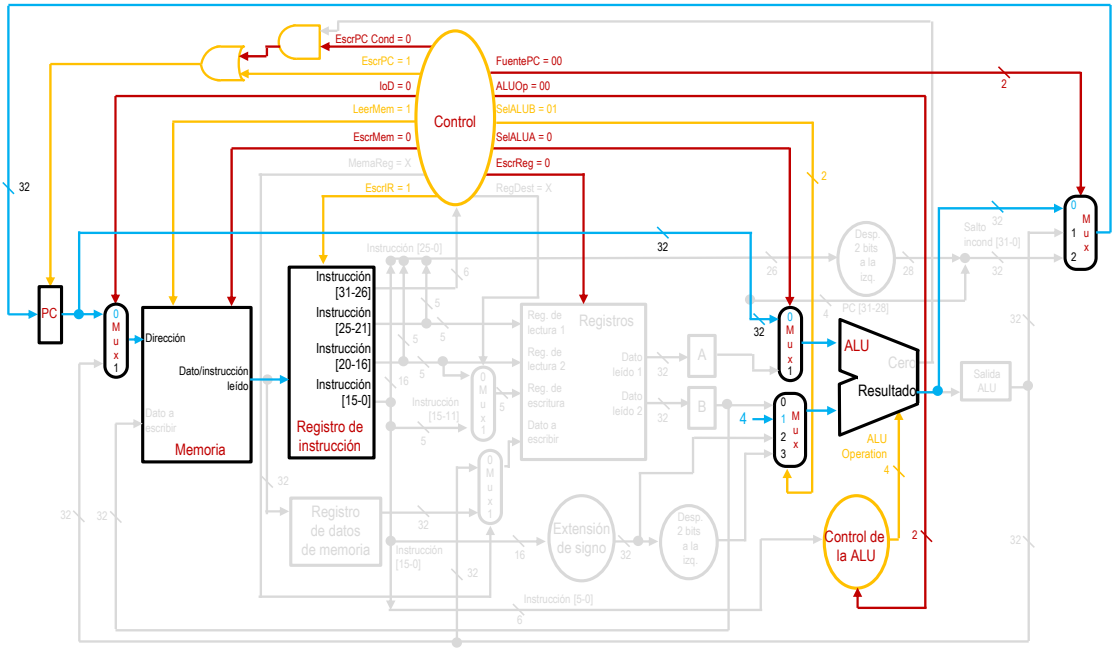
4. Acceso a memoria (MEM)

Se efectúa el acceso a memoria:.

- **lw**: se lee el dato de memoria.
- **sw**: se escribe el dato en memoria.

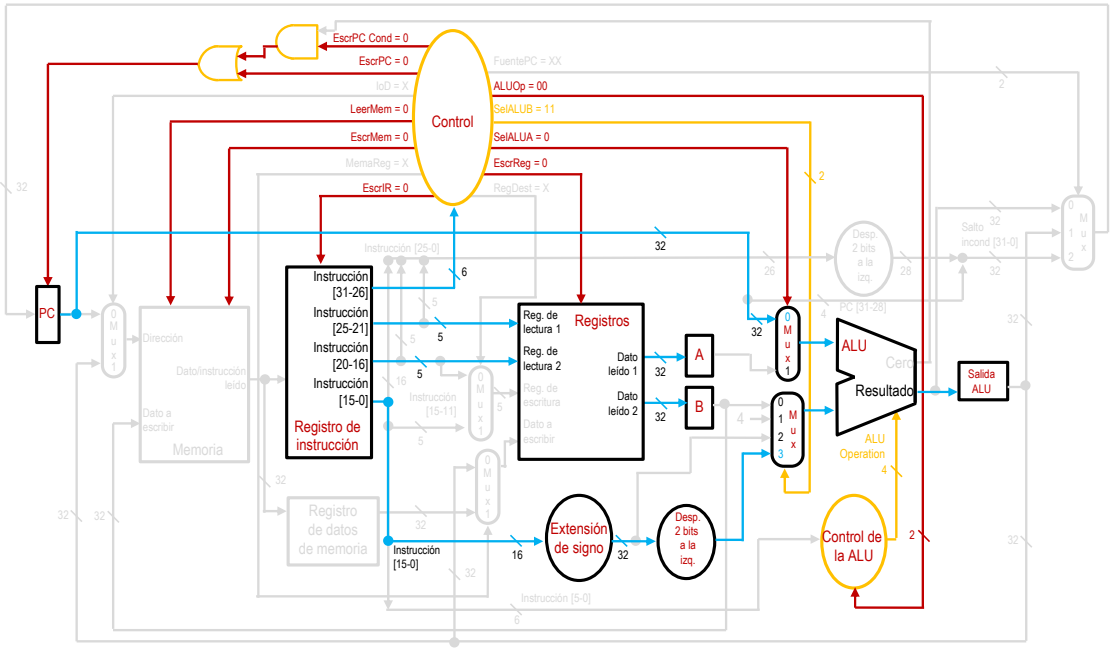
5. Escritura del resultado en el banco de registros (WB)

- En instrucciones **lw** se escribe en el banco de registros el dato leído en la memoria.
- En instrucciones R se escribe en el banco de registros el resultado de la operación.

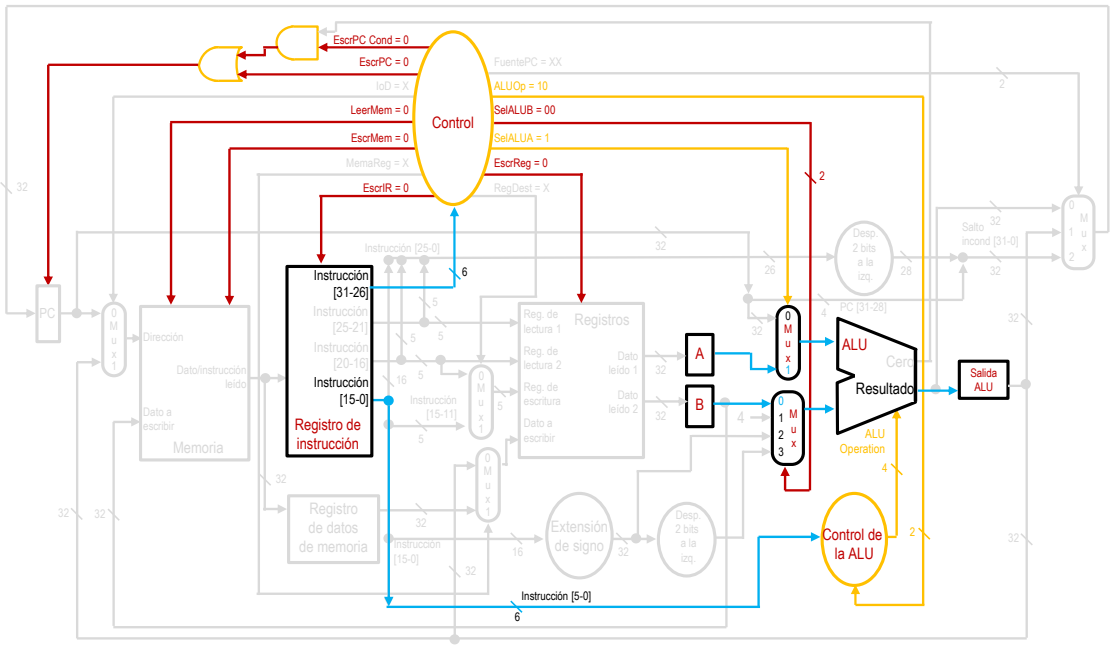


59

- Microoperaciones realizadas por esta instrucción:
  - IR ← M[PC]
  - PC ← PC + 4

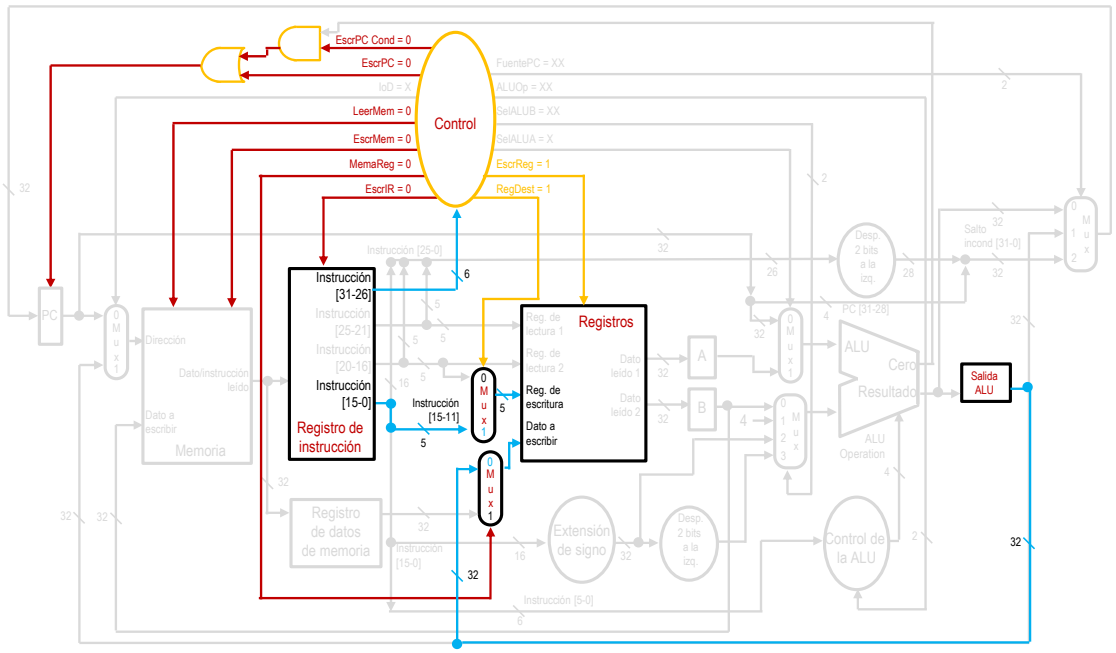


- Microoperaciones realizadas por esta instrucción:
  - A ← Reg[IR[25-21]]
  - B ← Reg[IR[20-16]]
  - Salida ALU ← PC + extSigno16a32(IR[15-0])<<2

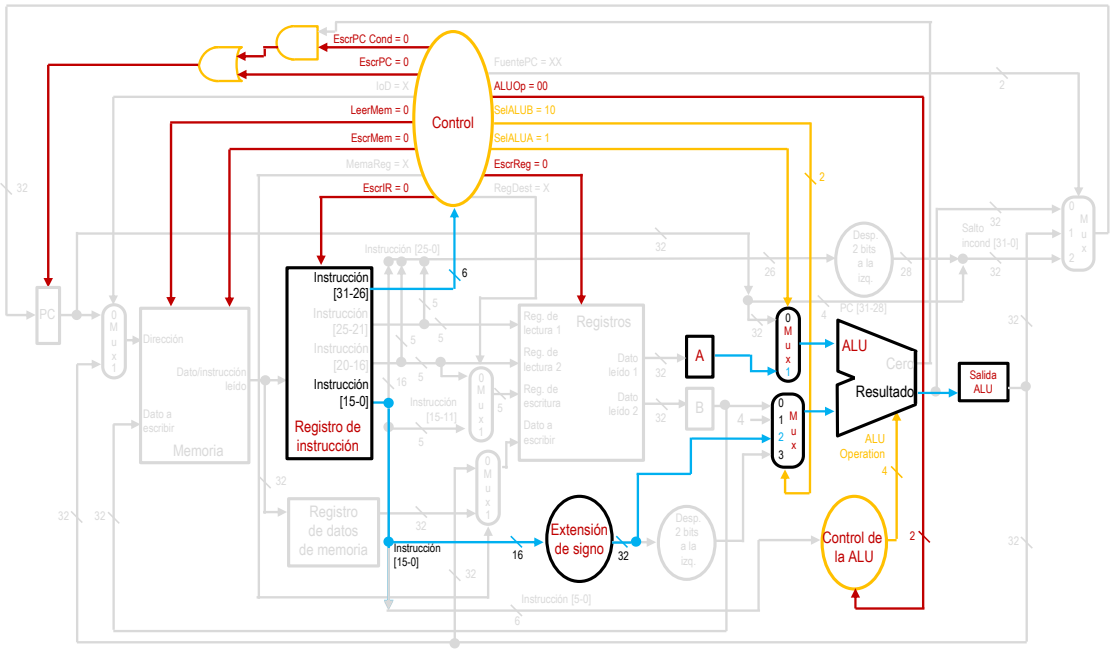


61

- Microoperaciones realizadas por esta instrucción:
  - Salida ALU ← A op B
- Vídeo explicativo: https://youtu.be/ALqlERsoZF0

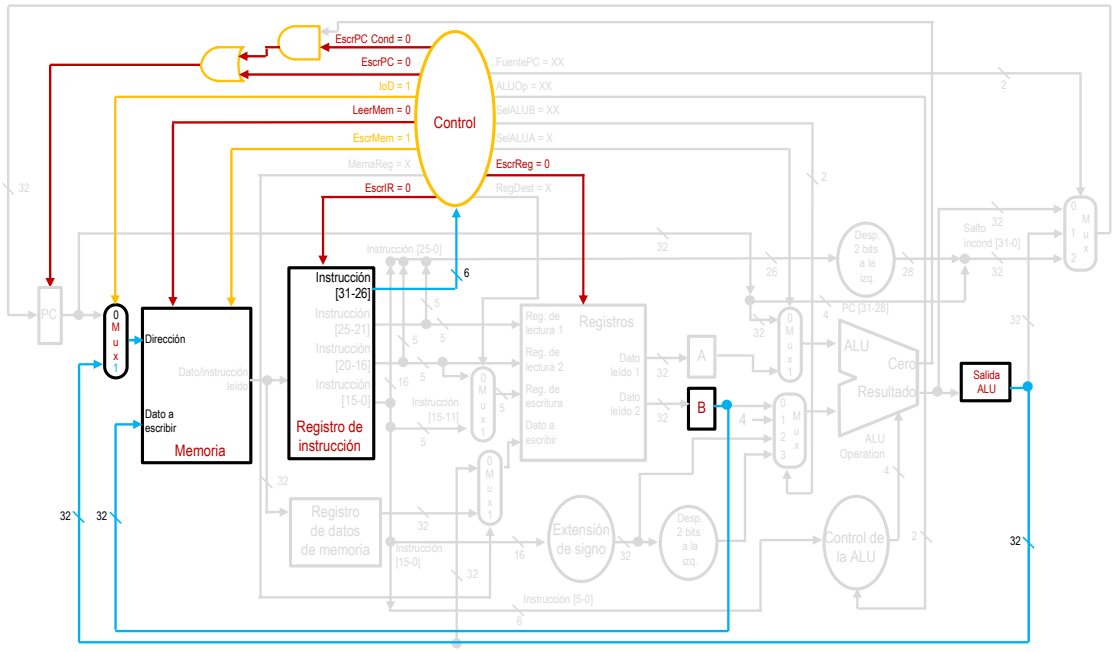


- Microoperaciones realizadas por esta instrucción:
  - Reg[IR[15-11]] ← Salida ALU
- Vídeo explicativo: https://youtu.be/ALqlERsoZF0

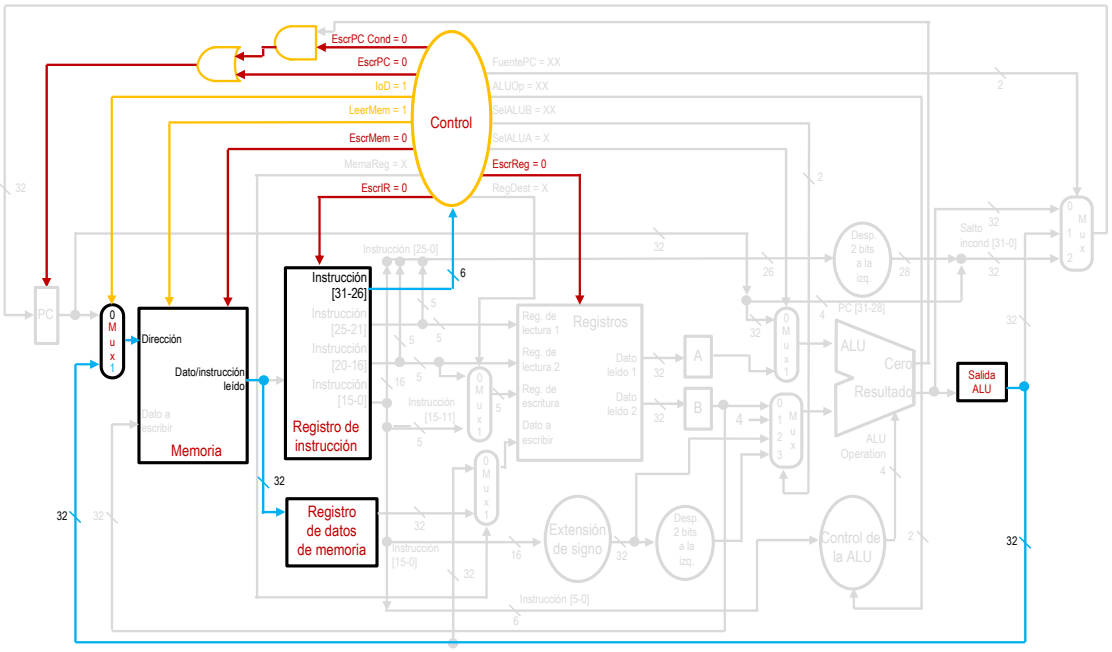


63

- Microoperaciones realizadas por esta instrucción:
  - Salida ALU ← A + extSigno16a32(IR[15-0])

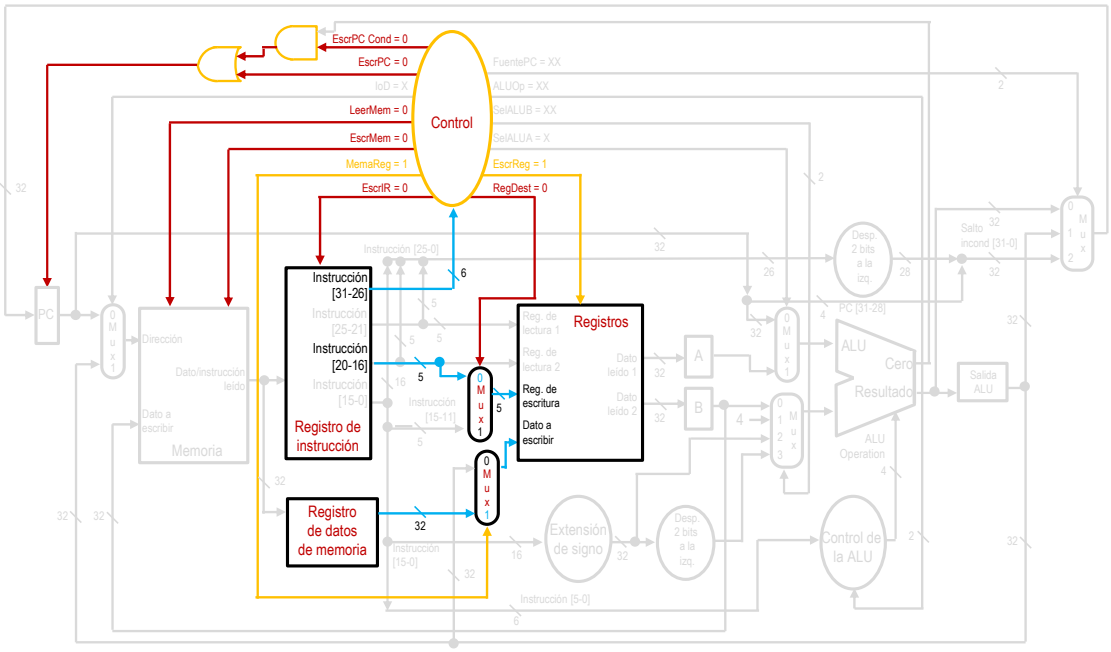


- Microoperaciones realizadas por esta instrucción:
  - M[Salida ALU] ← B
- Vídeo explicativo: https://youtu.be/WvPy_An8p54

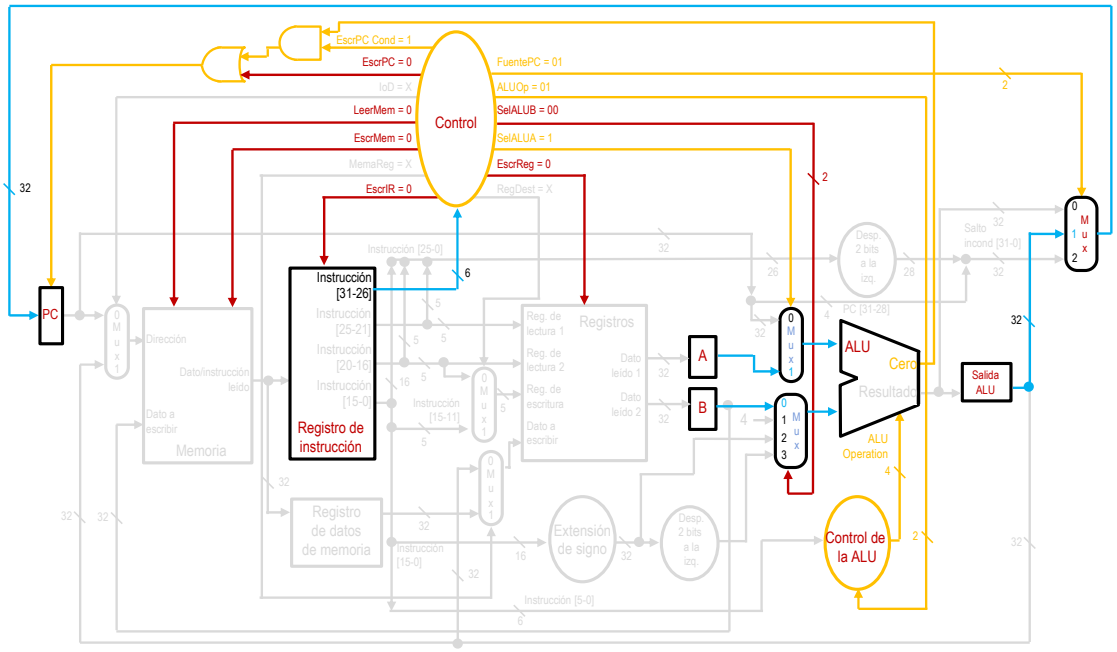


65

- Microoperaciones realizadas por esta instrucción:
  - MDR ← M[Salida ALU]
- Vídeo explicativo: https://youtu.be/2jkwrNUz9mM

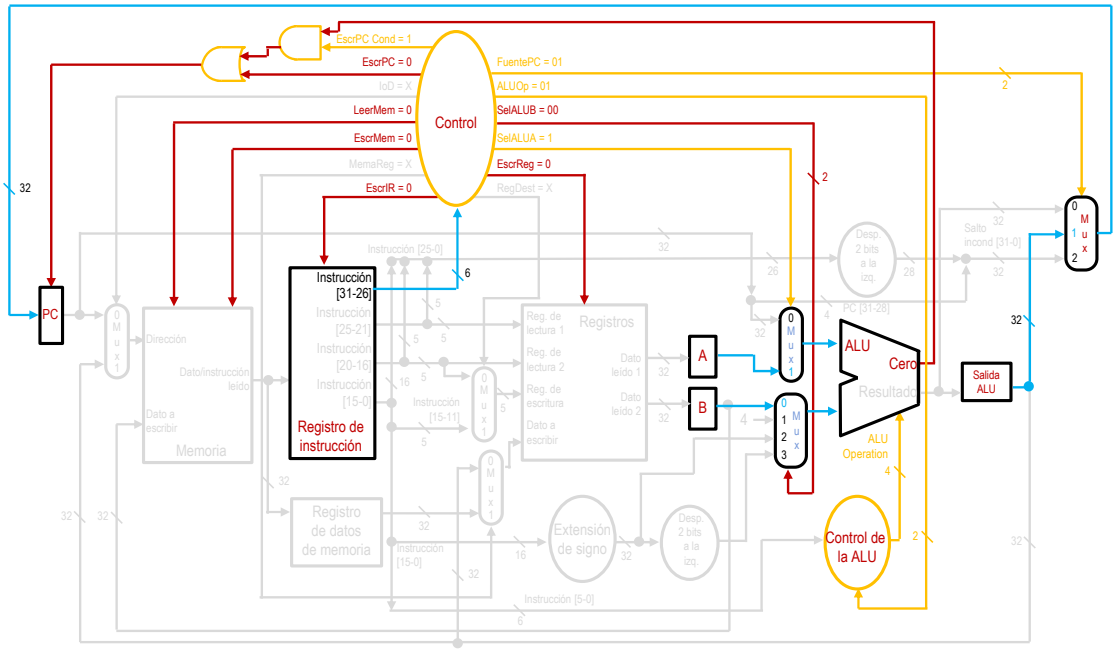


- Microoperaciones realizadas por esta instrucción:
  - Reg[IR[20-16]] ← MDR
- Vídeo explicativo: https://youtu.be/2jkwrNUz9mM



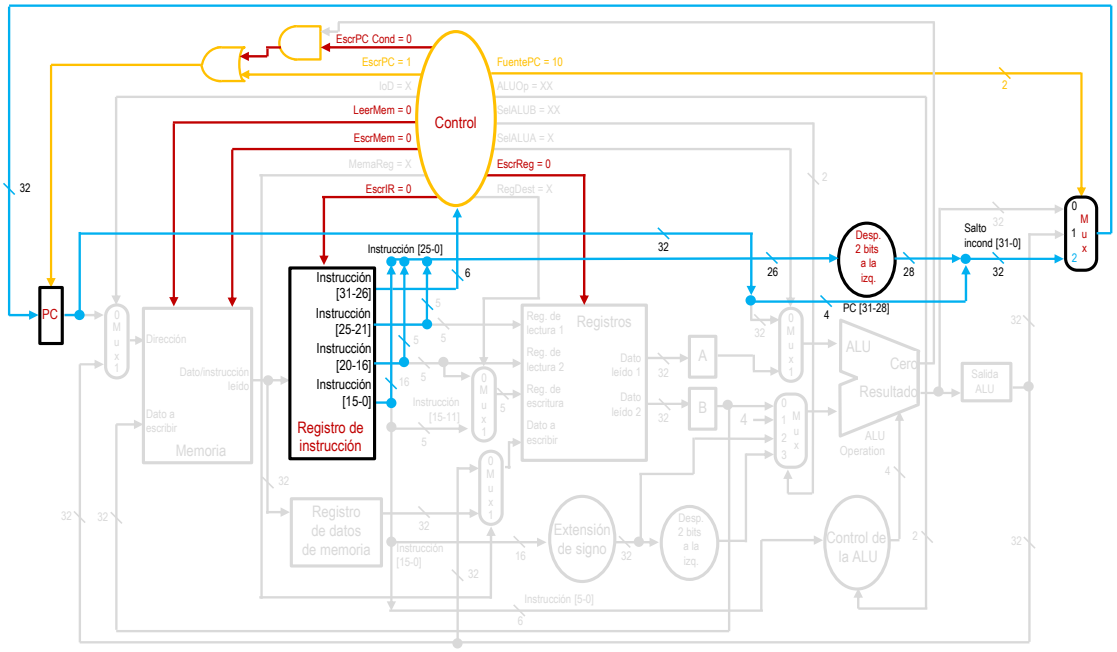
67

- Microoperaciones realizadas por esta instrucción:
  - Si (A = B) entonces PC ← Salida ALU
- Vídeo explicativo: https://youtu.be/A6FLQR08IsI



68

- Microoperaciones realizadas por esta instrucción:
  - Si (A ≠ B) entonces no hacemos nada
- Vídeo explicativo: https://youtu.be/A6FLQR08IsI

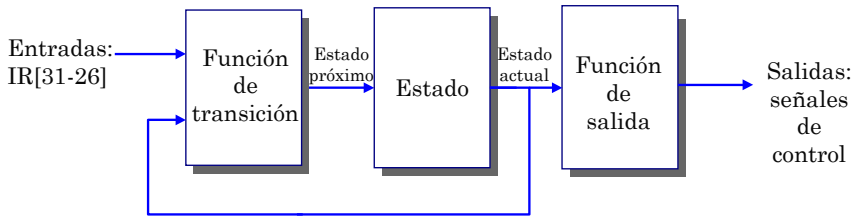


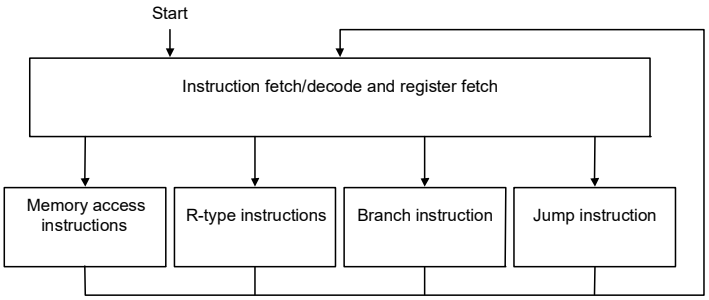
69

- Microoperaciones realizadas por esta instrucción:
  - PC ← PC[31-28] || (IR[25-0]<<2)
- Vídeo explicativo: https://youtu.be/sm2DLfRQrJo

UNIDAD DE CONTROL DEL CAMINO DE DATOS MULTICICLO

- La unidad de control se diseñará como una máquina de estados finitos (**FSM**, ***finite state machine***) con el modelo de Moore (circuito secuencial).

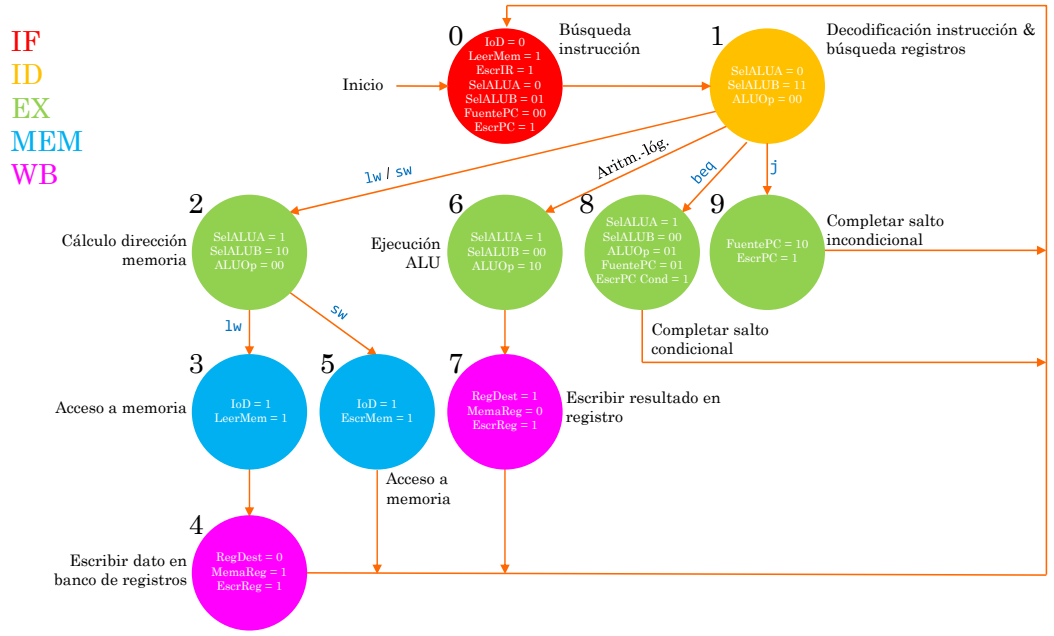




70

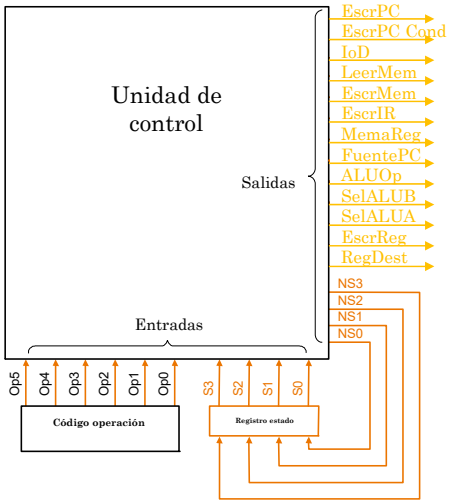
- La unidad de control se diseñará como una máquina de estados finitos (MEF, FSM en inglés) con el modelo de Moore:
  - Sus entradas están constituidas por los bits [31-26] de la instrucción, es decir, por el **código de operación**.
  - Las salidas del circuito de control son las señales de control.
- En el modelo de Moore, las salidas del circuito no dependen de las entradas, sino del estado actual.
  - Por ello, las salidas se generan muy rápido al principio del ciclo de reloj.
  - Las señales de control generadas son estables durante todo el ciclo de reloj.
- Por su parte, cada estado de la máquina corresponderá con una etapa de la ejecución de la instrucción.
  - Las dos primeras etapas de cada instrucción son la lectura de la instrucción (*fetch*) y la decodificación.
    - Estas dos etapas son comunes a todas las instrucciones.
  - Una vez la instrucción actual ha sido leída y decodificada, pasará por una secuencia de estados propia, que es distinta de las secuencias correspondientes a otras instrucciones.
  - Al final de la secuencia de estados propia de la instrucción, se regresa a la fase de *fetch* para leer una instrucción nueva y se repite el proceso.

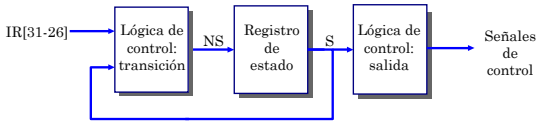
UNIDAD DE CONTROL DEL CAMINO DE DATOS MULTICICLO



- Esta diapositiva presenta el diagrama de estados que conforman todas las fases para las instrucciones soportadas en este camino de datos.
  - Las salidas especificadas en cada estado son solo las señales activas.
  - Las señales no especificadas permanecen inactivas en el ciclo correspondiente.
  - Las flechas indican las transiciones entre estados que se producen en los flancos de reloj del procesador.
- El estado 0 corresponde con la fase de lectura de la instrucción (*instruction fetch*).
  - Esta fase comprende la lectura de la instrucción en memoria (**IoD**=0, **LeerMem=1**) y su grabación en el registro de instrucción (**EscrIR=1**) al final del ciclo.
  - Además, en este ciclo se incrementa el **PC** (**SelALUA**=0, **SelALUB**=01, **ALUOp**=00, **FuentePC**=00, **EscrPC**=1).
- El estado 1 corresponde con la fase de decodificación.
  - Se leen los registros del banco de registros (lo cual se realiza de forma automática gracias a las conexiones existentes entre el registro de instrucción y las entradas **Reg. de lectura 1** y **2** del banco de registros).
  - También se calcula una hipotética dirección de ramificación por si acaso la instrucción leída es **beq** (**SelALUA**=0, **SelALUB**=11, **ALUOp**=00).
  - Al final de esta fase, se transita hacia un estado diferente dependiendo del código de operación de la instrucción leída.
- Si la instrucción implica el acceso a un dato residente en memoria (**lw** o **sw**), desde el estado 1 se pasa al 2, en el que se calcula la dirección de memoria del dato sumando un registro base más un desplazamiento extendido en signo (**SelALUA**=1, **SelALUB**=10, **ALUOp**=00).
  - La dirección calculada en la **ALU** queda grabada en **Salida ALU**.
  - Si la instrucción es una carga (**lw**):
    - Del estado 2 se pasa al 3, en el que se lee el dato en memoria (**IoD**=1, **LeerMem**=1).
    - El dato leído se graba en el **Registro de datos de memoria** (**MDR**).
    - Después se pasa al estado 4, donde el contenido del **MDR** se graba en el banco de registros (**MemaReg**=1, **RegDest**=0, **EscrReg**=1).
    - A continuación, se transita al estado 0 para leer la siguiente instrucción.
  - Si la instrucción es un almacenamiento (**sw**):
    - Del estado 2 se pasa al 5, donde se graba en memoria el contenido del registro **B** (**IoD**=1, **EscrMem**=1).
    - Después se transita al estado 0.
- Si la instrucción leída es de tipo R:
  - Del estado 1 se pasa al estado 6, en el que se realiza la ejecución de la operación propiamente dicha (**SelALUA**=1, **SelALUB**=00, **ALUOp**=10).
  - El resultado de la operación realizada en la **ALU** se graba en **Salida ALU**.
  - Al final del ciclo se transita al estado 7, en el que el dato contenido en **Salida ALU** se graba en el banco de registros (**MemaReg**=0, **RegDest**=1, **EscrReg**=1).
  - Después se pasa al estado 0 para leer la siguiente instrucción.
- Si la instrucción es una ramificación (**beq**):
  - Del estado 1 se pasa al 8.
  - En el estado 1 se calculó la dirección de ramificación y se grabó en **Salida ALU**.
  - Ahora en el estado 8 hay que comparar los registros fuente para ver si su contenido coincide (**SelALUA**=1, **SelALUB**=00, **ALUOp**=01), y en su caso se grabará el contenido de **Salida ALU** en el **PC** (**FuentePC**=01, **EscrPC Cond**=1) dependiendo del resultado de la comparación.
- Por último, si la instrucción leída es de salto incondicional (**j**):
  - Desde el estado 1 se transitará al 9.
  - En este estado la dirección de salto se grabará en el **PC** (**FuentePC**=10, **EscrPC**=1).
  - A continuación, se transitará al estado 0 para una nueva fase de lectura de instrucción.
- La presente diapositiva muestra el diagrama de estados completo para nuestro modelo multiciclo. Se observa que el número total de estados es reducido (10), lo cual facilita el diseño de la unidad de control.

UNIDAD DE CONTROL DEL CAMINO DE DATOS MULTICICLO





- Entrada: código de operación **Op** (bits [31-26] de **IR**).
- Registro de estado (***state register***): almacena $S_3S_2S_1S_0$.
- Lógica de control:
  - Transición: calcula $NS_3NS_2NS_1NS_0$.
  - Salida: calcula señales de control (**EscrPC**, **EscrPC Cond**, etc.).

72

- Hemos diseñado la unidad de control como una FSM de acuerdo con el modelo de Moore.
  - Constará de la lógica de control necesaria para implementar las funciones de transición y salida.
  - También tendrá un registro para almacenar el estado actual de la máquina secuencial.
    - El registro de estado (***state register***) tiene 4 bits, suficientes para almacenar combinaciones para 10 estados diferentes. Sus salidas del registro son **S3 … S0**, y sus entradas **NS3 … NS0** proceden de la lógica de control encargada de calcular la función de transición.
  - La lógica de control para las transiciones toma como entradas el código de operación (contenido en los 6 bits más significativos del registro de instrucción) y el estado actual (**S3 … S0**), y generan como salida el número del estado siguiente (**NS3 … NS0**).
- La lógica de control para las salidas toma como entrada el estado actual (**S3 … S0**) y genera todas las señales de control (**EscrPC**, **EscrPC Cond**, **IoD**, **LeerMem**, **EscrMem**, **EscrIR**, **MemaReg**, **FuentePC**, **ALUOp**, **SelALUA**, **SelALUB**, **EscrReg** y **RegDest**).
- Existen diferentes formas de implementar la lógica de control.
  - Estudiaremos el caso de una memoria ROM (memoria de solo lectura).

UNIDAD DE CONTROL DEL CAMINO DE DATOS MULTICICLO

Estado actual $S_3S_2S_1S_0$	Código de operación ($Op_5Op_4Op_3Op_2Op_1Op_0$)				
	000000 (aritm.-lóg.)	000010 (j)	000100 (beq)	100011 (lw)	101011 (sw)
0000	0001	0001	0001	0001	0001
0001	0110	1001	1000	0010	0010
0010	XXXX	XXXX	XXXX	0011	0101
0011	XXXX	XXXX	XXXX	0100	XXXX
0100	XXXX	XXXX	XXXX	0000	XXXX
0101	XXXX	XXXX	XXXX	XXXX	0000
0110	0111	XXXX	XXXX	XXXX	XXXX
0111	0000	XXXX	XXXX	XXXX	XXXX
1000	XXXX	XXXX	0000	XXXX	XXXX
1001	XXXX	0000	XXXX	XXXX	XXXX

73

- Esta diapositiva muestra la tabla de verdad de la función de transición de la unidad de control del modelo multiciclo y equivale al diagrama de estados mostrado anteriormente.
  - Las entradas de la tabla son los 4 bits del estado actual (columna de la izquierda) y los 6 bits del código de operación (fila superior).
  - Las salidas son los 4 bits del estado siguiente (cada una de las celdas).
- Por simplicidad, no hemos incluido en la tabla los casos de códigos de operación "ilegales" o no reconocidos por la unidad de control, que deberían abortar la ejecución del programa y mostrar un mensaje de error.
- Las combinaciones de estado y código de operación que no tienen sentido se han marcado con valores 'X' (*don't care*).

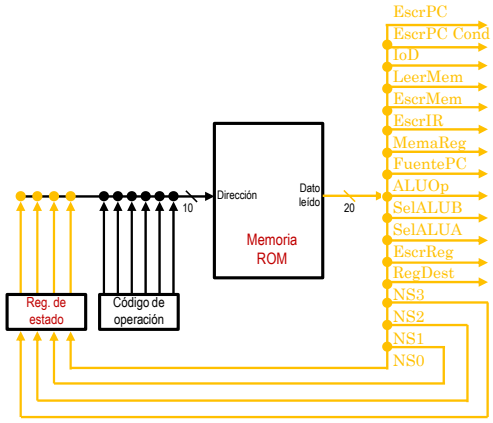
UNIDAD DE CONTROL DEL CAMINO DE DATOS MULTICICLO

Estado actual $S_3S_2S_1S_0$	Salidas de la unidad de control															
	EscrPC	EscrPCCond	IoD	LeerMem	EscrMem	EscrIR	MemaReg	FuentePC$_1$	FuentePC$_0$	ALUOp$_1$	ALUOp$_0$	SelALUB$_1$	SelALUB$_0$	SelALUA	EscrReg	RegDest
0000	1	0	0	1	0	1	X	0	0	0	0	0	1	0	0	X
0001	0	0	X	0	0	0	X	X	X	0	0	1	1	0	0	X
0010	0	0	X	0	0	0	X	X	X	0	0	1	0	1	0	X
0011	0	0	1	1	0	0	X	X	X	X	X	X	X	X	0	X
0100	0	0	X	0	0	0	1	X	X	X	X	X	X	X	1	0
0101	0	0	1	0	1	0	X	X	X	X	X	X	X	X	0	X
0110	0	0	X	0	0	0	X	X	X	1	0	0	0	1	0	X
0111	0	0	X	0	0	0	0	X	X	X	X	X	X	X	1	1
1000	0	1	X	0	0	0	X	0	1	0	1	0	0	1	0	X
1001	1	0	X	0	0	0	X	1	0	X	X	X	X	X	0	X

74

- Esta diapositiva muestra la tabla de verdad de las salidas (que son las 16 señales de control que tenemos que generar).
  - Al haber implementado la unidad de control mediante una máquina de Moore, los valores de las salidas solo dependen del estado actual de la FSM, el cual se representa con 4 bits.
  - Por lo tanto, esta tabla tiene 16 filas en total, de las cuales solo son relevantes de la fila 0 a la 9.
  - La tabla tiene 16 columnas, una por cada bit de salida, es decir, el circuito que implementa la función de salida tiene que devolver 16 bits.
  - No se han dibujado las filas correspondientes a los estados que van del 10 al 15, que no están contemplados para esta versión de la unidad de control.
    - Si hiciera falta establecer los valores de las salidas para dichos estados hipotéticos, se podrían haber puesto a 0 explícitamente aquellas señales que podrían dar lugar a escrituras indeseadas.
      - En particular, habría que proteger los registros PC e IR (EscrPC = 0, EscrPr Cond = 0 y EscrIR = 0), así como la memoria (LeerMem = 0 y EscrMem = 0) y el banco de registros (EscrReg = 0).
      - El resto de las señales de control son irrelevantes y se podrían poner con valores 'X' (*don't care*).

UNIDAD DE CONTROL DEL CAMINO DE DATOS MULTICICLO



- Memoria ROM: puede materializar las funciones de transición y salida sin simplificar.
- Número de entradas de dirección: 10
  - Entradas 9-6: bits del registro de estado (S).
  - Entradas 5-0: bits del código de operación (Op).
- Número de palabras de la memoria: 2^{10} = 1024.
- Ancho de cada palabra: 20 bits
  - Salidas 19-16: próximo estado (NS).
  - Salidas 15-0: señales de control.
- Tamaño total de la memoria:
  2^{10} entradas × 20 bits = 20 kb = 2,5 kB
- Más barato:
  - Usar dos memorias ROM separadas: una para la función de transición y otra para la función de salida.
  - Otras técnicas: PLA, microprogramación…

75

- La implementación mediante una memoria **ROM** (read-only memory) materializa la tabla de verdad completa de las funciones de transición y salida.
  - En la **ROM** grabaríamos las tablas de verdad completas.
- Usar una **ROM** única para transiciones y salidas es un despilfarro.
  - Al materializar la tabla de transiciones completa, tendríamos que contar con una **ROM** de 1024 palabras.
  - El ancho de palabra de la memoria coincidiría con el número de bits necesario para la función de transición (4 bits) más la función de salida (16 bits).
  - La **ROM** tendría 20 columnas × 1024 filas, o sea, ocuparía 20 kb (kilobits) o 2,5 kB (kilobytes).
- En nuestro modelo, la mayoría de las combinaciones (**estado, código de operación**) en la función de transición son imposibles:
  - Hay 15 transiciones relevantes, con lo cual nos sobrarían 1009 palabras en la **ROM**.
  - Además, como el control se está implementando como una máquina de Moore, al meter las transiciones y las salidas en una **ROM** única, el espacio ocupado por la salida es 2^6 veces mayor de lo necesario, pues la función de transición tiene 2^6 veces más filas que la función de salida.
  - Como el número de estados **s** no es potencia de 2, sobran las filas correspondientes a estados los inexistentes (estados 10 al 15).
- Para evitar tanto despilfarro, una opción posible es usar dos memorias **ROM** separadas (una para las transiciones y otra para las salidas), o bien, una PLA (programmable logic array) o incluso utilizar la técnica de microprogramación.
  - Ninguna de estas técnicas las vamos a estudiar, pero se pueden consultar en la bibliografía (Patterson & Hennessy, 3.ª edición).