Tema 9: Diseño del Procesador

Bibliografía recomendada:

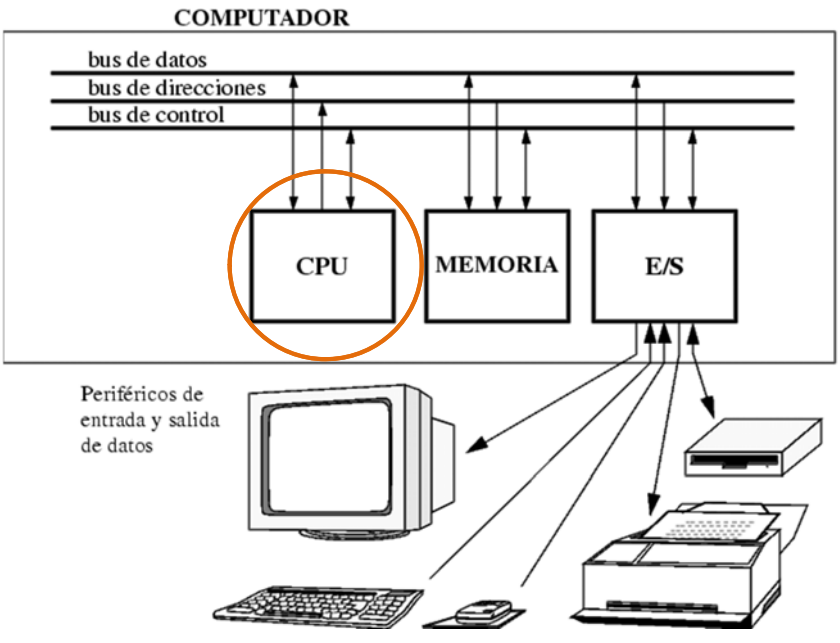
Estructura y Diseño de Computadores: Volumen 1

D. Patterson y J. Hennessy, ed. Reverté, 2000

Capítulo 5: "El procesador: camino de datos y control"

fc²

¿Qué vamos a estudiar en este tema?



Modelo Von Neumann

fc²

La CPU

- La CPU es un sistema digital cuya funcionalidad podríamos describir con la siguiente secuencia:
  1. Leer de memoria la siguiente instrucción
  2. Decodificar la instrucción
  3. Obtener los operandos de la instrucción
  4. Ejecutar la instrucción (realizar la operación)
  5. Almacenar el resultado

  Volver a 1

Decisiones que hay que tomar para diseñar la CPU

- **Para poder diseñar la CPU necesitamos tomar algunas decisiones:**
  - Qué **repertorio de instrucciones** queremos que pueda ejecutar el procesador
  - Qué **tipos de datos** puede manipular el procesador
  - Con qué **tamaño de palabra** va a trabajar el procesador
  - **TODOS los datos** que manipula un programa se almacenan en la **memoria o** se va a disponer de **un banco de registros**
    - **La memoria**
      - Tiene más capacidad de almacenaje
      - Acceso a los datos lento
    - **El banco de registros**
      - Tiene menos capacidad de almacenaje
      - Acceso a los datos muy rápido
  - Cuantos ciclos de reloj necesitan las **instrucciones**
    - **Multiciclo**: cada instrucción se ejecuta en varios ciclos de reloj
    - **Tiempo de ciclo⇔ frecuencia de reloj**

Decisiones que hay que tomar para diseñar la CPU

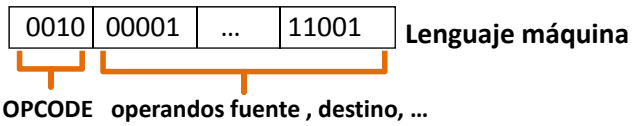
- Sobre el **Repertorio de instrucciones**
  - **¿ RISC o CISC ?**
    - RISC: *Reduced Instruction Set Computer*
    - CISC: *Complex Instruction Set Computer*
  - **Ancho de la instrucción:**
    - Número fijo o variable de bits que podemos utilizar para codificar la instrucción
  - **Tipos de instrucciones**
  - **Número de instrucciones de cada tipo**
  - Cómo se va a especificar en la instrucción dónde están los datos y cómo acceder a ellos
    - **Modos de direccionamiento**

Decisiones que hay que tomar para diseñar la CPU

- Sobre el **Repertorio de instrucciones: Tipo de instrucciones**
  - Un computador debe ser capaz de:
    - Realizar las operaciones matemáticas elementales
      - **Instrucciones aritmético-lógicas**
    - Obtener los datos que utiliza la instrucción y almacenar los resultados de las operaciones
      - **Instrucciones de acceso a memoria**
    - Modificar el flujo secuencial del programa
      - **Instrucciones de salto**
    - Otras dependiendo de las características particulares del procesador
  - **Recordad que un computador es un sistema digital** luego sólo puede hacer operaciones con datos y leer o escribir datos en memoria o registros

Decisiones que hay que tomar para diseñar la CPU

- Sobre el **Repertorio de instrucciones:** **Formato de las instrucciones**
  - Un computador **NO entiende los lenguajes de programación de alto nivel**
    - Sólo entiende ceros y unos (código máquina)
  - **La instrucción es una cadena de "0" y "1"** que contiene toda la información que necesita la CPU para poder ejecutarla
  - Las instrucciones tienen que contener la siguiente información:
    - **Tipo de instrucción** → OPCODE (Código de Operación)
    - **Dónde se encuentra el valor de cada operando** → Modos de direccionamiento (MDs)



  - ¿Cuántos operandos explícitos tiene cada instrucción?
    - 0
    - 1
    - 2 (1 fuente/destino y 1 fuente)
    - 3 (2 fuente y un destino)
    - Más de 3???

fc2

Decisiones que hay que tomar para diseñar la CPU

- Sobre el **Repertorio de instrucciones:** **Modos de direccionamiento**
  - Es la forma de especificar dentro de la instrucción:
    - **Dónde están las datos**
      - Memoria
      - Registros
      - En la propia instrucción
    - **Cómo acceder a ellos**
  - Cada familia de procesadores ofrece distintas posibilidades:
    - Inmediato
    - Absoluto
    - Directo de Registro
    - Indirecto de Registro
    - Indirecto de Registro con desplazamiento
    - ...

fc2

Arquitectura del procesador

- Como hemos visto, **hay muchas decisiones que tomar para poder diseñar la CPU de un procesador**
- **Todas estas decisiones** son las que **definen la arquitectura** de un procesador
- Definición de **Arquitectura del procesador**
  - Es el conjuntos de atributos de un computador que son visibles a:
    - El programador en lenguaje máquina
    - El sistema operativo
    - El compilador
  - Engloba los siguientes elementos:
    - Conjunto de **instrucciones máquina**
    - **Tipos básicos de datos soportados por las instrucciones máquina**
    - **Modos de direccionamiento**
    - Conjunto de **registros visibles al programador**
      - Registros de datos, direcciones, estado, contador de programa (PC)
    - Mecanismos de **E/S**

Familia de procesadores

- Se denomina familia al conjunto de computadores / procesadores con la **misma arquitectura pero distinta implementación**
- Las familias de computadores hacen posible que:
  - Existan máquinas de la misma familia con distinta
    - Tecnología
    - Velocidad
    - Prestaciones
    - Precio
  - Las máquinas de una misma familia sean compatibles entre sí
    - La compatibilidad suele ser hacia arriba (upward compatibility)
  - Todos los miembros de una misma familia pueden ejecutar los mismos programas

Familia de procesadores

- 68000 - CISC
- Intel (X86) - CISC
- **MIPS – RISC**
  - **Con esta arquitectura vamos a aprender el diseño de un procesador**
- **ARM – RISC**
  - **Con esta arquitectura vamos a aprender a programar en lenguaje ensamblador**
    - **La hemos visto en el Tema 8**
    - **Las practicas se realizan con el ARM**

fc2

Procesador MIPS

fc2

MIPS

- **MIPS = M**icroprocessor without **I**nterlocked **P**ipeline **S**tages
- **Creado en 1981 por Hennessy**
- Producido por Silicon Graphics (SGI)
- Utilizado por:
  - TiVo
  - Windows CE
  - Cisco routers
  - Nintendo 64
  - Sony PlayStation, PlayStation 2, PlayStation Portable

Evolución del MIPS

- **R2000** (1985)
  - Procesador con operaciones multiciclo de multiplicación y división, dentro de un coprocesador matemático
    - Los resultados de estas operaciones eran conseguidos a través de instrucciones particulares
  - Aunque los registros eran de 32 bits podían utilizarse como de 64 para doble precisión
- **R3000** (1988)
  - Añade una cache de 32KB (más tarde 64KB)
  - Añade una MMU para gestionar la memoria virtual
  - Entre otros , fue utilizado en la **Play Station**, además en los primeros portátiles que utilizaban **Windows CE**
  - Fue el primer procesador MIPS que tuvo éxito en el mercado
    - Se vendieron 1 millón de estos procesadores

Evolución del MIPS

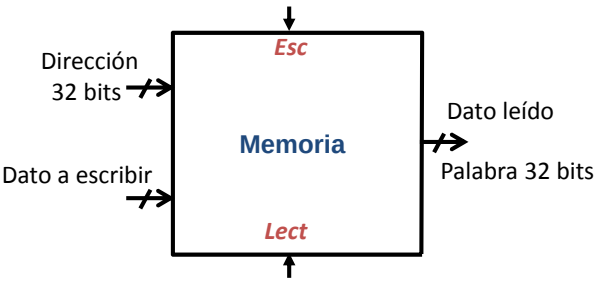
- **R4000** (1991)
  - Consigue un alto ciclo de reloj
    - Aumenta el conjunto de instrucciones a un procesador de 64 bits
    - Añade la unidad de punto flotante (FPU) dentro del chip
    - Reducen las caches a 8KB
    - Tiene una super-segmentación
  - **Nintendo64** y los primeros routers de **Cisco (36x0 y 7x00)** utilizan una CPU basada en este diseño
- **R8000** (1994)
  - Fue el primer procesador super-escalar que podía trabajar a la vez con dos ALUs y dos memorias
  - Su unidad FPU era perfecta para cálculos científicos
  - Duró en el mercado sólo un año
- **R10000** (1995):
  - Tenía caches de 32 KB y su mayor innovación fue la utilización de la ejecución fuera de orden
- **R20000** (2006)
  - 2 cores … no ha llegado a producirse
- **Loongson/Godson** (actualidad): Versiones multicore (16 cores).

Arquitectura MIPS

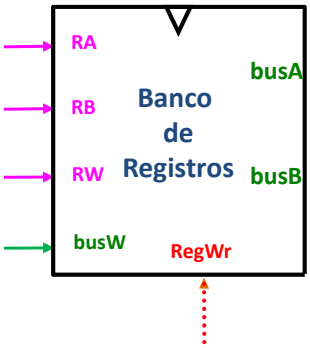
- **Antes de empezar a diseñar la CPU del MIPS necesitamos conocer su arquitectura**:
  - Vamos a trabajar con una **versión simplificada del MIPS**
- **Repertorio de instrucciones**
  - Tamaño de las instrucciones **32 bits** (4 bytes)
  - Todas las instrucciones tienen el **mismo tamaño**
  - **Tres tipos** de instrucciones
    - Aritmético-lógicas
    - Acceso a memoria
    - De salto condicional
  - Sólo existen **3 formatos** de instrucciones
- **Modos de direccionamiento**
  - **Tres tipos**
    - Directo a registro
    - Indirecto registro con desplazamiento
    - Inmediato
- **Temporalización de las instrucciones**
  - multiciclo

Arquitectura MIPS

- **Memoria**
  - **Direccionable por bytes**
  - **Dirección 32 bits**, (4 Gbyte)
  - La **palabra** de **32 bits** (4 bytes)
  - Memoria **alineada**
    - Siempre **accederemos a direcciones múltiplo de 4** porque tanto datos como instrucciones son de 4 bytes
  - Admite **little y big endian**
    - Tiene un bit de modo para seleccionarlo

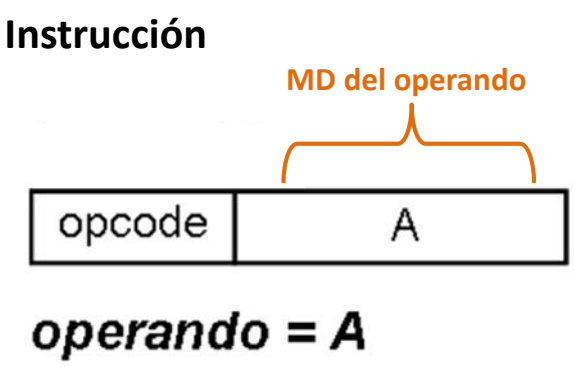


- **Banco de Registros**
  - **32 registros (5bits) de 32 bits (4 bytes)**
  - Lectura paralela de 2 registros
  - Escritura en un registro



Modos de direccionamiento del MIPS

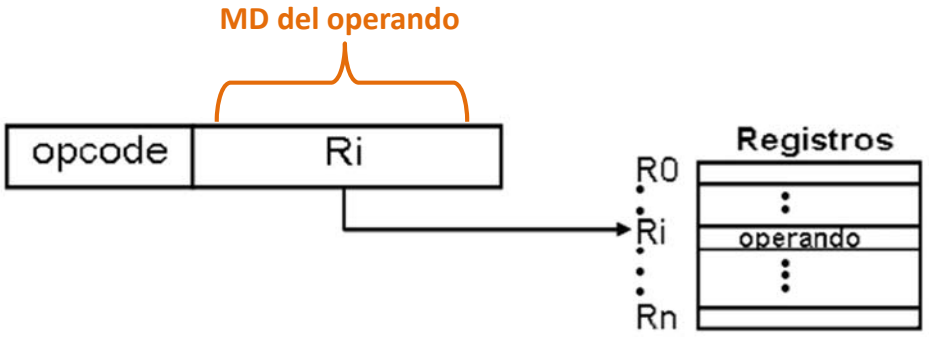
- **Inmediato**
  - **Dónde está el operando**
    - Está contenido en la propia instrucción:
  - **Cómo acceder a él**
    - **Accediendo los bits de la instrucción que lo contiene**



Modos de direccionamiento del MIPS

- **Directo registro**
  - **Dónde está el operando**
    - En un registro del banco de registros
  - **Cómo acceder a él**
    - **La instrucción nos indica el número del registro**

Instrucción

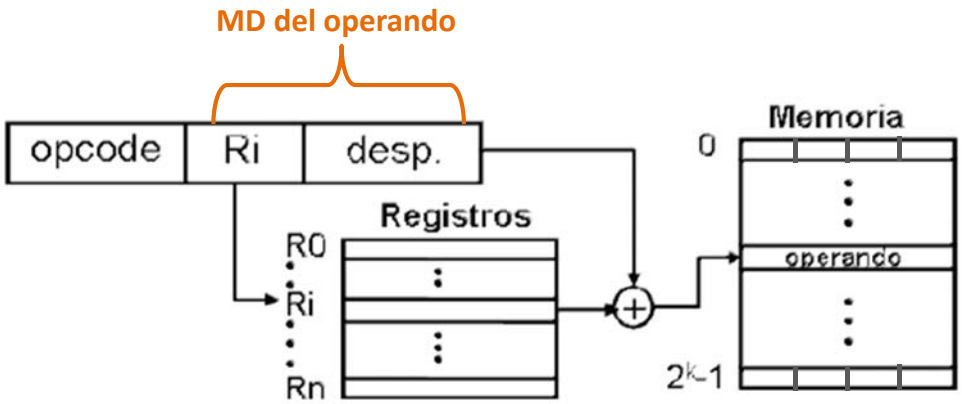


Modos de direccionamiento del MIPS

- **Indirecto registro con desplazamiento inmediato**
  - **Dónde está el operando**
    - En una posición de memoria
  - **Cómo acceder a él**
    - Primero hay que calcular la dirección de memoria (Efective Adresse)

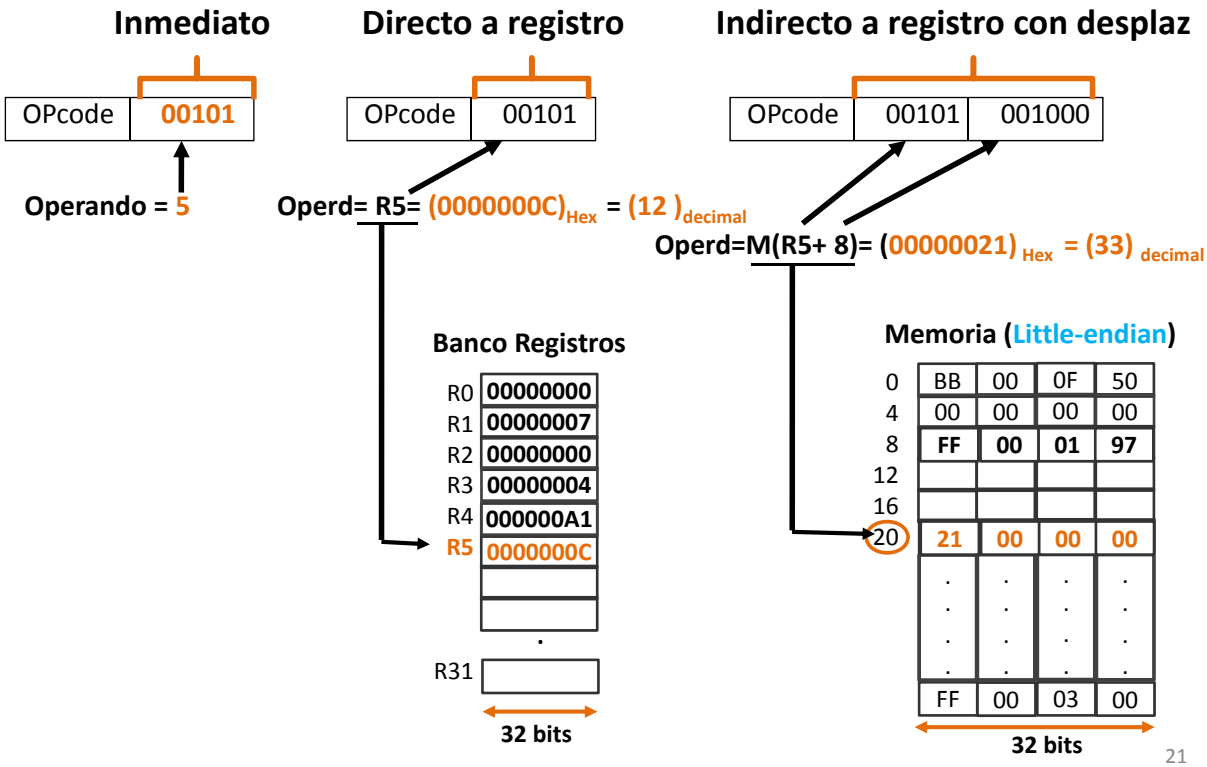
    Dir. Mem = **contenido del registro + desplazamiento**= Ri + desplazamiento

Instrucción



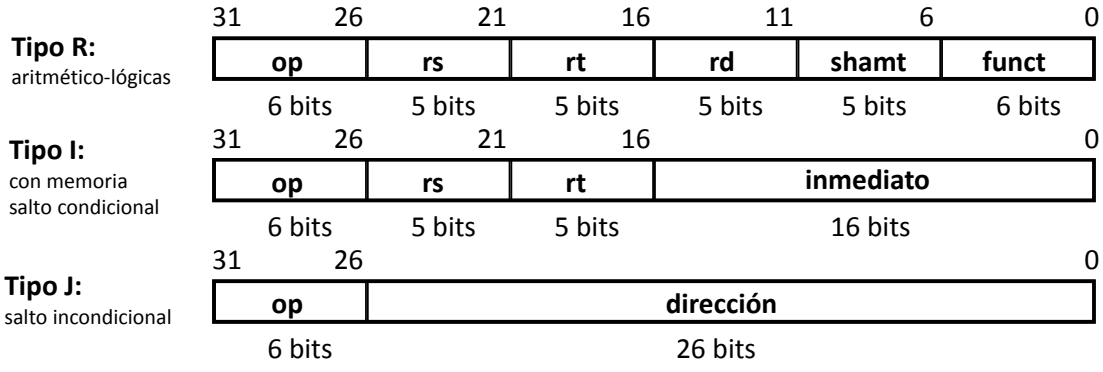
Modos de direccionamiento del MIPS

- Ejemplo



Repertorio de instrucciones del MIPS

- **Formato de las instrucciones**



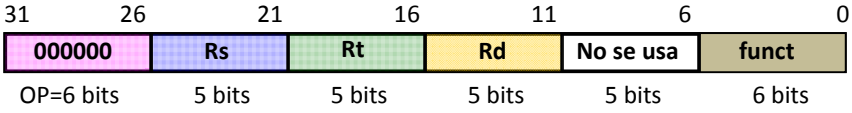
El significado de los campos es:

- **op**: identificador de instrucción
- **rs**, **rt**, **rd**: identificadores de los registros fuentes y destino
- **shamt**: cantidad a desplazar (en operaciones de desplazamiento)
- **funct**: selecciona la operación aritmética a realizar
- **inmediato**: es el valor del desplazamiento en direccionamiento a registro-base
- **dirección**: dirección destino del salto

Instrucciones formato tipo-R

Instrucciones aritmético-lógicas

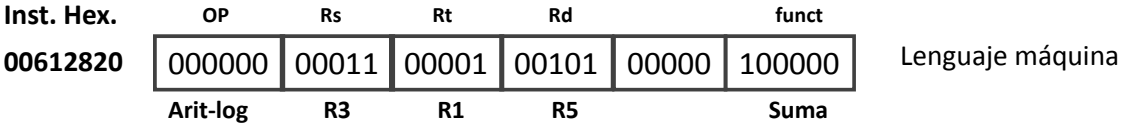
- Tiene tres operandos: **$Operando_{Destino}$ = $Operando_{fuente1}$ (funct) $Operando_{fuente2}$**
- Modo de direccionamiento de los operandos: **Directo a Registro**



- Campo **OP** (Código de Operación) = 000000
  - Indica que es una instrucción aritmético-lógica
- **Campo de los operandos**:
  - Indica donde se encuentran los operandos
    - **Rs** → Número del registro donde se encuentra **el primer operando fuente**
    - **Rt** → Número del registro donde se encuentra **el segundo operando fuente**
    - **Rd** → Número del registro donde se encuentra **el operando destino**
  - Se necesitan 5 bits porque el Banco de Registros tiene 32 registros
- Campo **funct**
  - Indica la operación aritmética o lógica que hay que realizar
    - add → funct=32 (100000)
    - sub → funct=34 (100010)
    - and → funct=36 (100100)
    - or → funct=37 (100101)

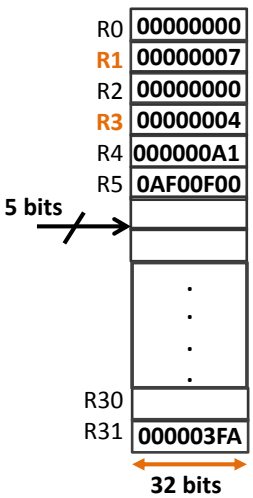
Instrucciones formato tipo-R

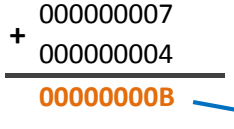
Instrucciones aritmético-lógica



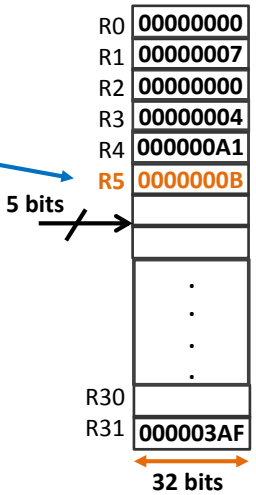
Lenguaje máquina

Banco de Registros
Antes de ejecutar la instrucción





Banco de Registros
Después de ejecutar la instrucción



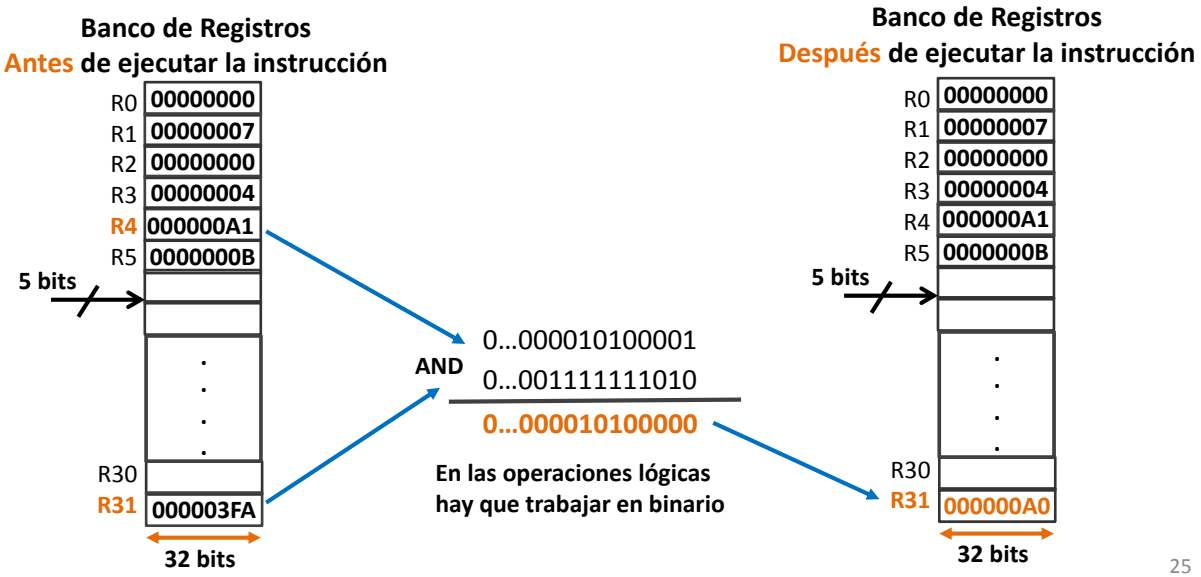
Las operaciones se pueden hacer en la base que queramos, pero en el banco de registros lo guardamos en Hex porque:
- ✓ Es como lo muestran los simuladores
- ✓ Es más cómodo que en binario

Instrucciones formato tipo-R

- **Instrucciones aritmético-lógica**

Inst. Hex.	OP	Rs	Rt	Rd		funct
00612824	000000	00100	11111	11111	00000	100100
	Arit-log	R4	R31	R31		AND

Lenguaje máquina



Instrucciones formato tipo-I

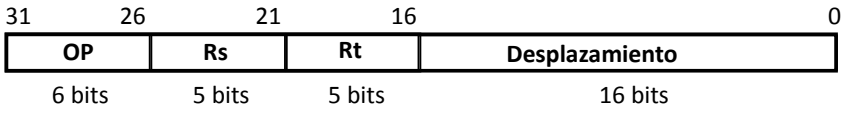
- **Instrucciones de acceso a memoria**
  - Tiene dos operandos: **$Operando_{Destino} \leftarrow Operando_{Fuente}$**
    - **Load** → lee un dato de **memoria** (**$Oper_{Fuente}$**) y lo escribe en un **registro** (**$Oper_{Destino}$**)

      R ← Memoria(Dirección)
    - **Store** → lee un dato de un **registro** (**$Oper_{Fuente}$**) y lo escribe en **memoria** (**$Oper_{Destino}$**)

      Memoria(Dirección) ← R
  - Modo de direccionamiento de los operandos:
    - **Directo a Registro**
      - Para los operandos que están en el banco de registros
    - **Indirecto a registro con desplazamiento inmediato**
      - Para los operandos que están en la memoria
      - Dirección de memoria = **R + SignExt(Desplazamiento)**
      - **El valor del desplazamiento está en la propia instrucción,** tiene 16 bits y está representado de complemento a 2

Instrucciones formato tipo-I

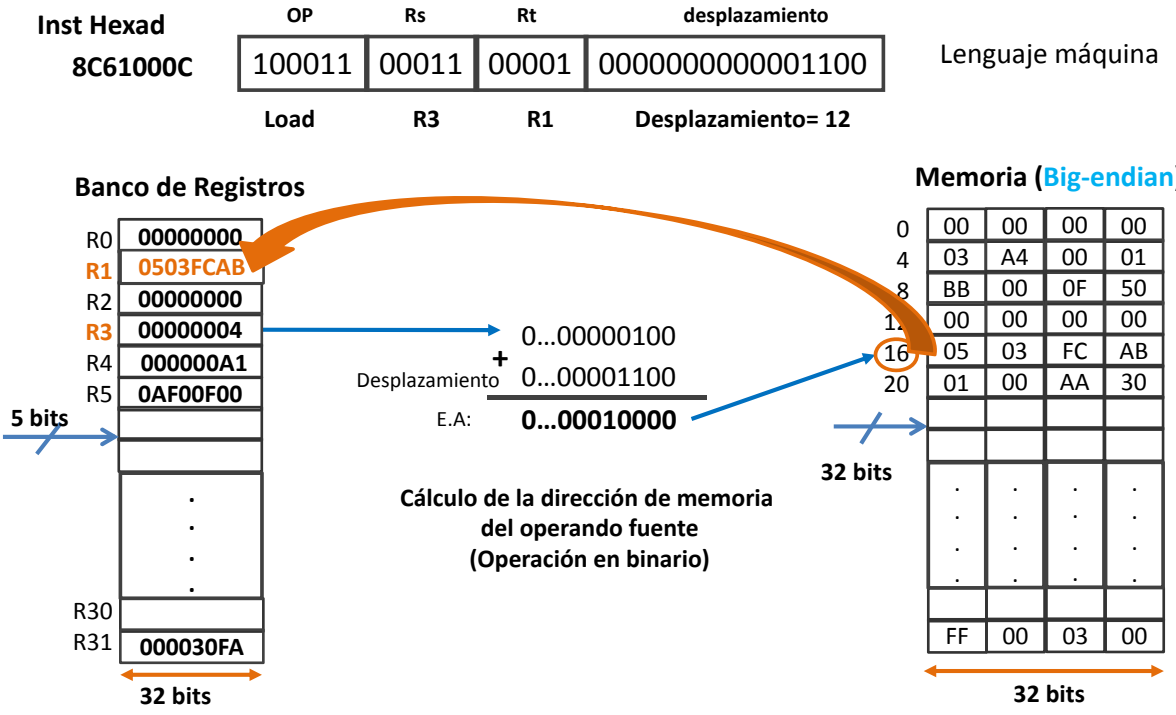
Instrucciones de acceso a memoria



- Campo **OP**
  - OP=35 (100011) → Indica que es una instrucción Load
  - OP= 43 (101011) → Indica que es una instrucción Store
- **Campo de los operandos**:
  - Indica donde se encuentran los operandos
    - **Rs**→ Número del registro que se necesita para calcular la dirección de memoria
    - **Rt**
      - » Número del registro donde se encuentra el **operando destino (**inst **Load**)
      - » Número del registro donde se encuentra el **operando fuente** (inst **store**)
    - **Desplazamiento**
      - » Contiene el valor que hay que sumarle al contenido del registro Rs para calcular la dirección de memoria
      - » Rango = $[-2^{15}, +(2^{15}-1)]$
  - Se necesitan 5 bits porque el Banco de Registros tiene 32 registros

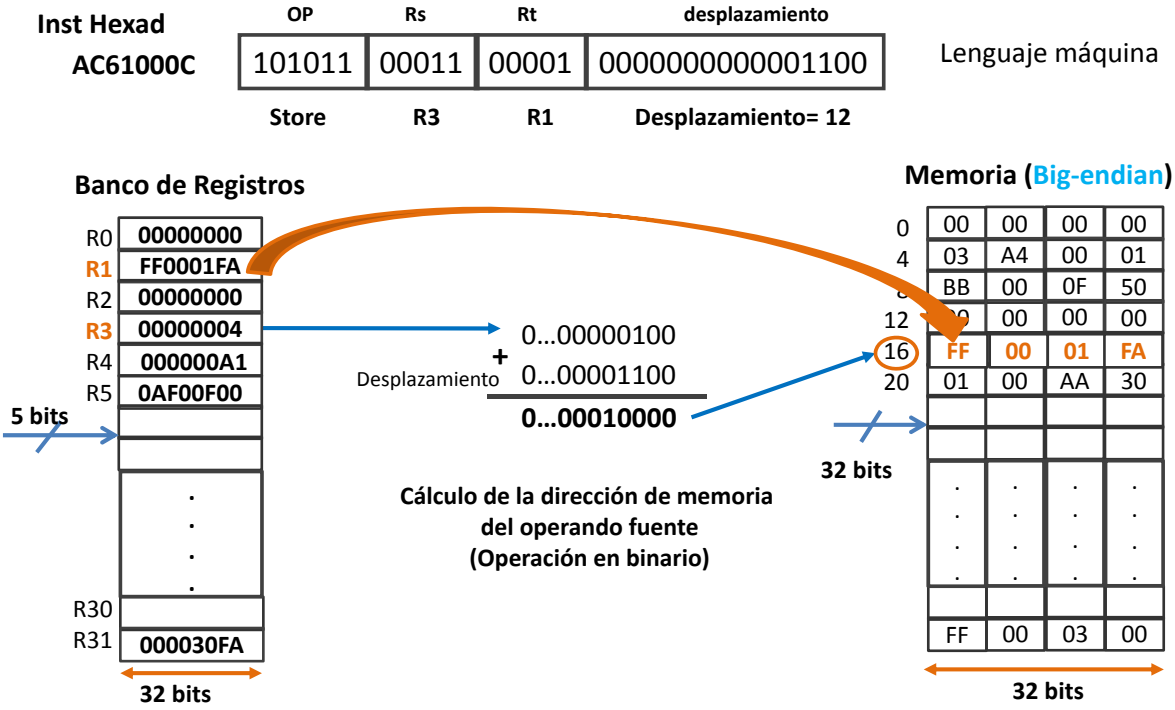
Instrucciones formato tipo-I

Instrucción Load



Instrucciones formato tipo-I

Instrucción Store



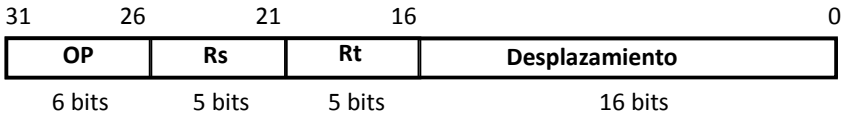
Instrucciones formato tipo-I

Instrucciones de salto condicional

- Tiene tres operandos: **Operando$_{fuente1}$, Operando$_{fuente2}$ y Desplazamiento**
  - **Si Operando$_{fuente1}$ = Operando$_{fuente2}$** hay que **calcular la dirección** de memoria donde está la instrucción **a la que hay que saltar**
  - **En caso contrario no hace nada**
- Modo de direccionamiento de los operandos:
  - **Directo a Registro**
    - Para los operandos que están en el banco de registros
  - **Inmediato**
    - El valor del desplazamiento está en la propia instrucción

Instrucciones formato tipo-I

- **Instrucciones de salto condicional**



  - Campo **OP**
    - OP= 4 (000100) → Indica que es una instrucción de salto
  - **Campo de los operandos fuentes**:
    - Indica donde se encuentran los operandos
      - **Rs**→ Número del registro donde se encuentra el primer operando fuente
      - **Rt** → Número del registro donde se encuentra el segundo operando fuente
      - **Desplazamiento**
        - » Contiene el valor del desplazamiento , este se necesita para calcular la dirección de memoria donde está la instrucción a la que hay que saltar
        - » **Si no hay que saltar este valor no se usa**
    - Se necesitan 5 bits porque el Banco de Registros tiene 32 registros

Sobre el lenguaje máquina

- El computador
  - No entiende las instrucciones de los lenguajes de alto nivel
    - Java, C, Pascal, Phyton …
  - Sólo entiende lenguaje máquina
- Hay que programar **en lenguaje máquina**?
  - En los primeros procesadores se programaba en este lenguaje
  - No es intuitivo
  - Es muy pesado tener que poner las instrucciones con "0" y "1"
- Se crea el **lenguaje ensamblador**
  - **Conjunto de nemotécnicos que representan a cada una de las instrucciones del lenguaje máquina**

Lenguaje máquina						Lenguaje ensamblador
000000	00011	00001	00101	00000	100000	**ADD R3, R1, R5**
100011	00011	00001	0000000000001100			**LW R1, 12(R3)**

Sobre el lenguaje máquina

- **Programar en alto nivel**
  - Existen muchos lenguajes
    - Java, C, Pascal, Phyton …
  - Estos lenguajes son independientes de la arquitectura del procesador donde se va a ejecutar el programa
  - La **arquitectura** del procesador es **transparente al programador**
  - **El compilador** se encarga de **traducir** el programa **a código máquina**

Asignatura Fundamentos de la programación, …

- **Programar en bajo nivel**
  - Hay que usar lenguaje ensamblador
  - Este lenguaje depende de la arquitectura del procesador donde se va a ejecutar el programa
  - **El programador tiene que conocer la arquitectura del procesador**
  - **El ensamblado traduce** los programas escritos en **lenguaje ensamblador a código máquina**

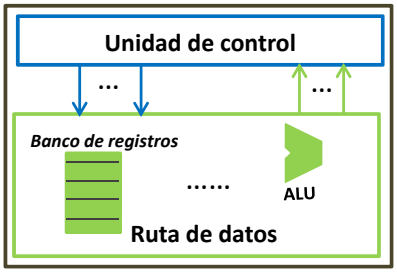
Lo veremos:
- En el tema 10 y en las prácticas
- En la asignatura de Estructura de Computadores

Lenguaje ensamblador del MIPS

- Sólo vamos a ver **un conjunto reducido de instrucciones** porque no vamos a programar con esta arquitectura

- **Instrucciones del MIPS en lenguaje ensamblador**
  - Instrucciones con referencia a memoria:
    - **lw Rt, desplaz(Rs)**
    - **sw Rt, desplaz(Rs)**
  - Instrucciones aritmético-lógicas con operandos en registros:
    - **add Rd, Rs, Rt**
    - **sub Rd, Rs, Rt**
    - **and Rd, Rs, Rt**
    - **or Rd, Rs, Rt**
  - Instrucciones de salto condicional:
    - **beq Rs, Rt, desplaz** si (rs = rt) salta

Unidad de proceso y unidad de control



- **Unidad de proceso (Ruta de datos o Data-path)**
  - Circuito que realiza las operaciones que indican las instrucciones del programa
  - Está formada por módulos combinacionales y secuenciales
    - ALU, Registros especiales, Multiplexores, Banco de registros …
- **Unidad de control**
  - Genera las señales necesarias para que la unidad de proceso realice en cada momento las operaciones correspondientes
  - Varias alternativas. En este curso estudiaremos su diseño como máquina de estados.

Procesador multiciclo

- Dividimos la ejecución de cada instrucción en 5 fases:

1. Búsqueda de la instrucción (*fetch*)
2. Decodificación (*deco*)
3. Ejecución (*ex*)
4. Acceso a memoria (*mem*)
5. Almacenamiento del resultado (*write back*)

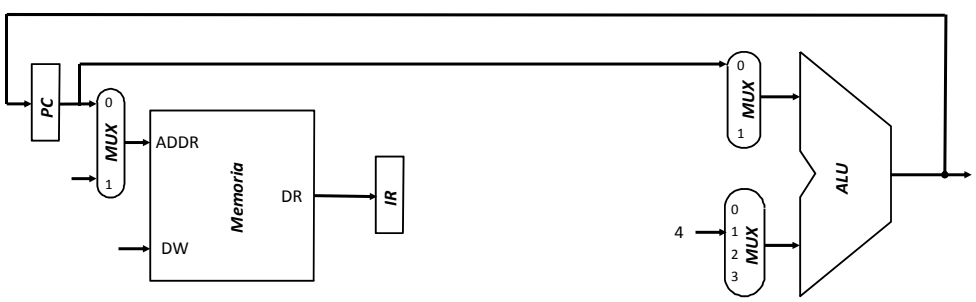
- El tiempo de ejecución de cada fase es 1 ciclo de reloj.
- Cada instrucción tarda un número distinto de ciclos, según las fases que deba ejecutar

Diseño de la ruta de datos multiciclo

- ¿Qué HW necesitamos para implementar cada una de las fases de ejecución de una operación?
  - Nótese que las distintas fases de la ejecución de una instrucción pueden compartir el HW de la ruta de datos ya que se ejecutan en ciclos distintos

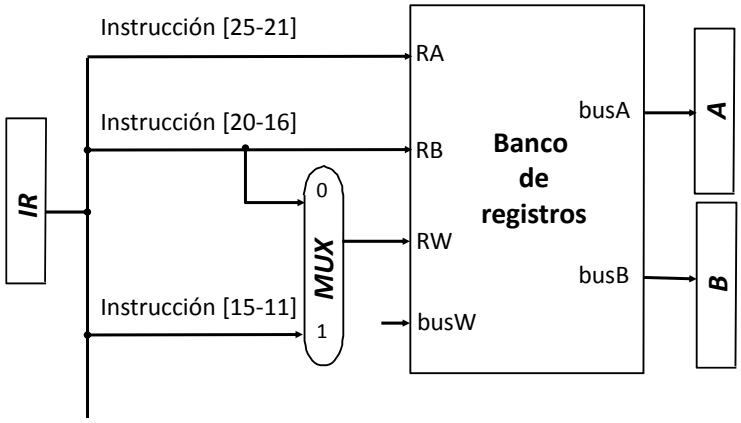
Fase de Búsqueda de Instrucción (Fetch)

- Lectura de la instrucción ubicada en la dirección de la memoria indicada por el contador de programa.
- Aprovechamos para incrementar PC, dejándolo listo para la siguiente instrucción
- Hardware necesario:
  - Contador de programa (PC)
  - Memoria
  - Registro de instrucción (IR)
  - Sumador (utilizamos la ALU usada para realizar op. aritmetico-lógicas)



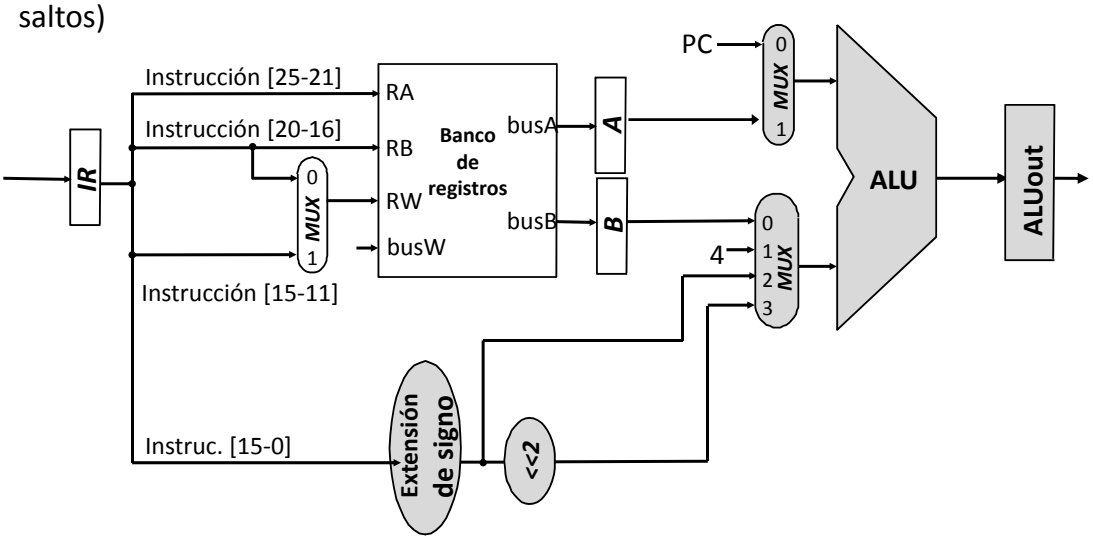
Fase de decodificación de Instrucción (Deco)

- Lectura de los **operandos** necesarios del **banco de registros**
- Hardware necesario:
  - **Banco de registros**
  - **2 registros de operandos *A* y *B***



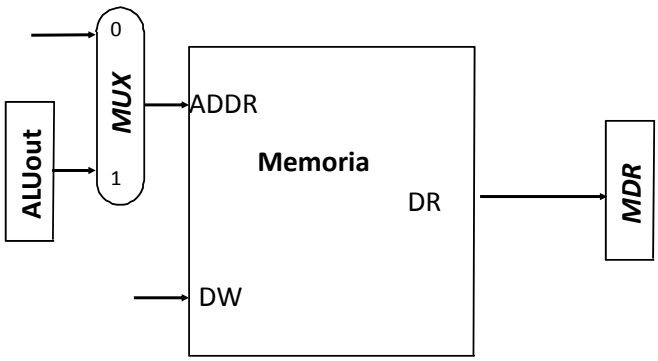
Fase de Ejecución de la Instrucción (Ex)

- Realizar la operación indicada por la instrucción
- Hardware necesario:
  - ALU
  - Registro para salvar el resultado de la ALU (ALUOut)
  - Extensor de signo (únicamente para lw y sw)
  - Desplazador a la izquierda para implementar la multiplicación por 4 (para saltos)



Fase de Acceso a Memoria (Mem)

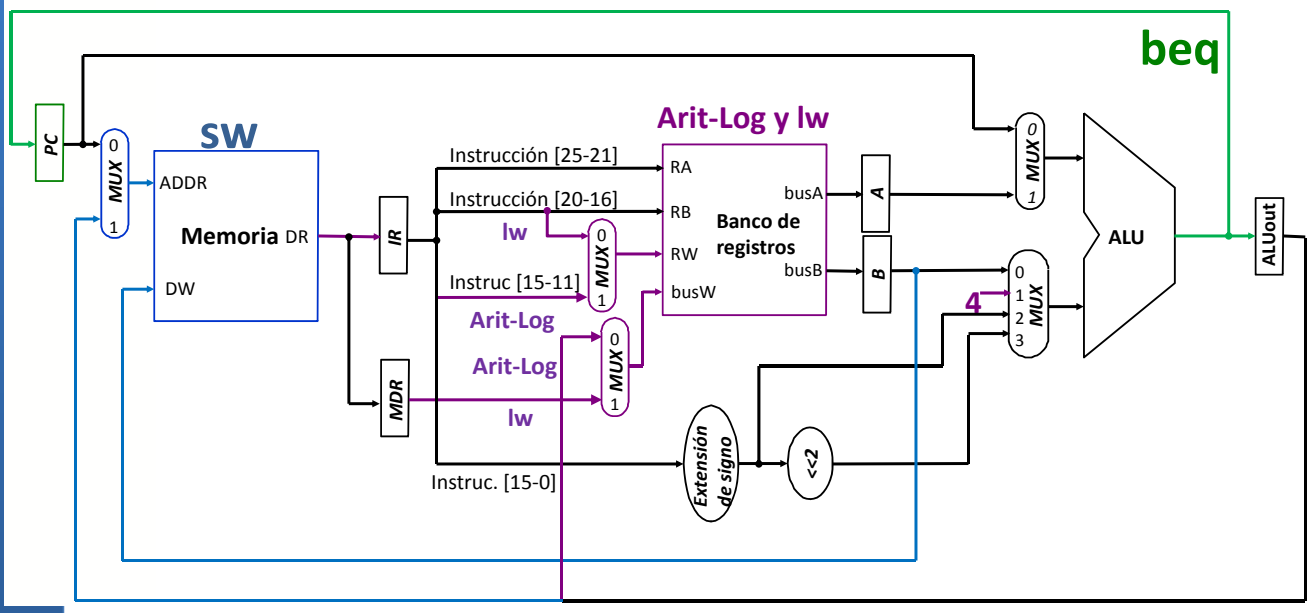
- Lectura de un dato de la memoria
- Hardware necesario:
  - Memoria
  - Registro para almacenar el dato leído (MDR)



Fase de Escritura del resultado (WB)

- Las instrucciones aritmético-lógicas y lw
  - Escritura del resultado en el banco de registros
- Las instrucciones sw
  - Escritura en la memoria
- La instrucción de salto
  - Actualiza el valor de PC si se cumple la condición del salto
- Hardware necesario:
  - Banco de registros
  - Memoria

Fase de escritura del resultado (WB)



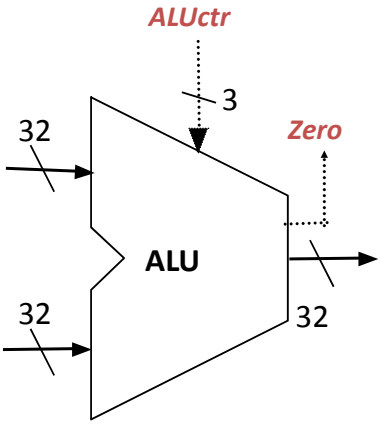
Relación de componentes HW necesarios

- **Memoria:** contiene instrucciones y de datos
- **Banco de registros** visibles al programador
- **Registros** no son visibles al programador
  - PC → Contiene la dirección de la siguiente instrucción que hay que ejecutar
  - IR → Contiene una instrucción leída de memoria
  - MDR → Contiene un dato leído de memoria
  - ALUout → Contiene el resultado de la ALU
  - A, B → Contienen datos leídos del banco de registros
- **ALU**
  - Capaz de realizar suma, resta, and, or, comparación de mayoría e indicación de que el resultado es cero (para realizar la comparación de igualdad mediante resta)
- **Extensor de signo**
  - Para adaptar el operando inmediato de 16 bits al tamaño de palabra
- **Desplazador a la izquierda**
  - Para implementar la multiplicación por 4 (necesaria en el salto)
- **Multiplexores**
  - Para poder seleccionar entre varias opciones

ALU

- Opera con datos de 32 bits
- Realiza las siguientes operaciones

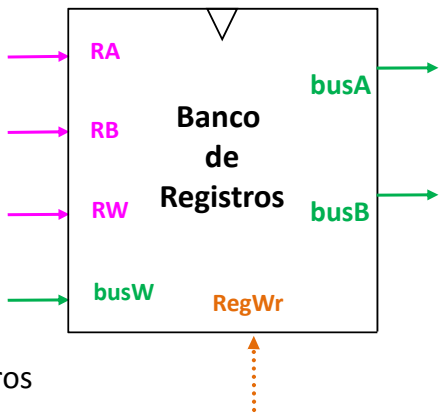
ALUctr	*función*
000	A and B
001	A or B
010	A + B
110	A – B
111	1 si (A<B), sino 0



fc²

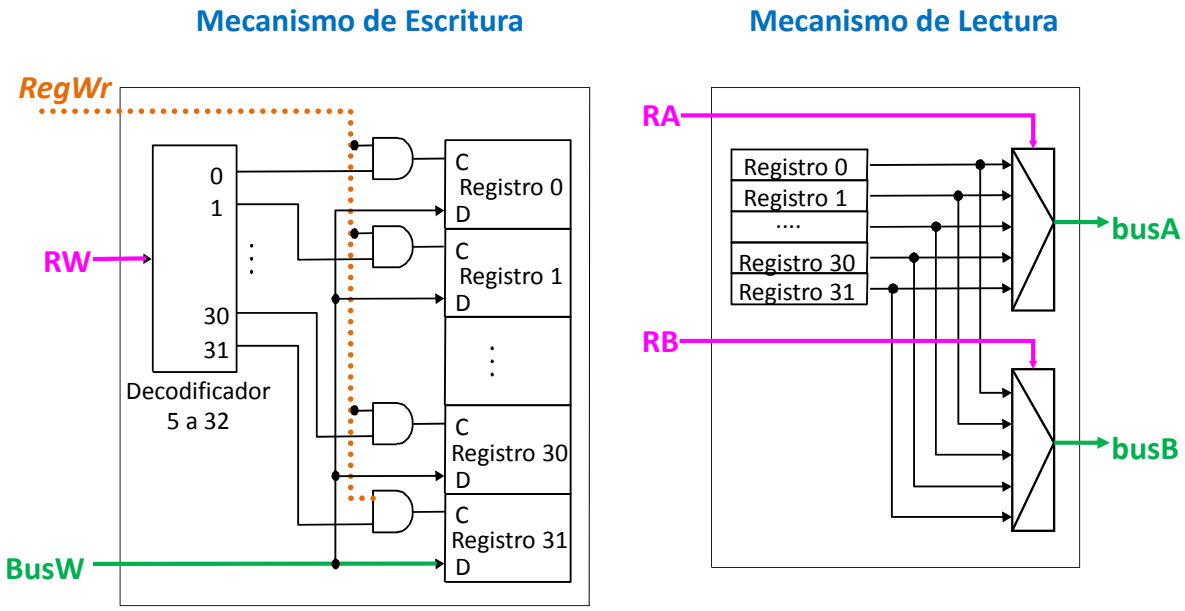
Banco de Registros

- Tiene 32 registros
- Se puede acceder simultáneamente a 3 registros (porque las instrucciones de tipo R lo requieren)
  - Dos registros de lectura
  - Un registro de escritura
- Para acceder a un registro se necesita:
  - Identificador del registro
  - 3 entradas de 5 bits cada una
- Salida/entrada de los datos
  - 2 salidas de datos de 32 bits
  - 1 entradas de datos de 32 bits
- 1 entrada de control
  - Para habilitar la escritura sobre uno de los registros
- 1 puerto de reloj
  - Sirve para las operaciones de escritura
  - Las de lectura son combinacionales



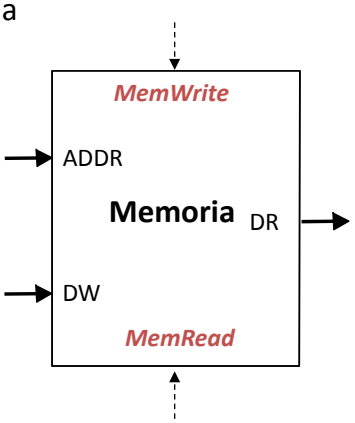
fc²

Banco de Registros: implementación

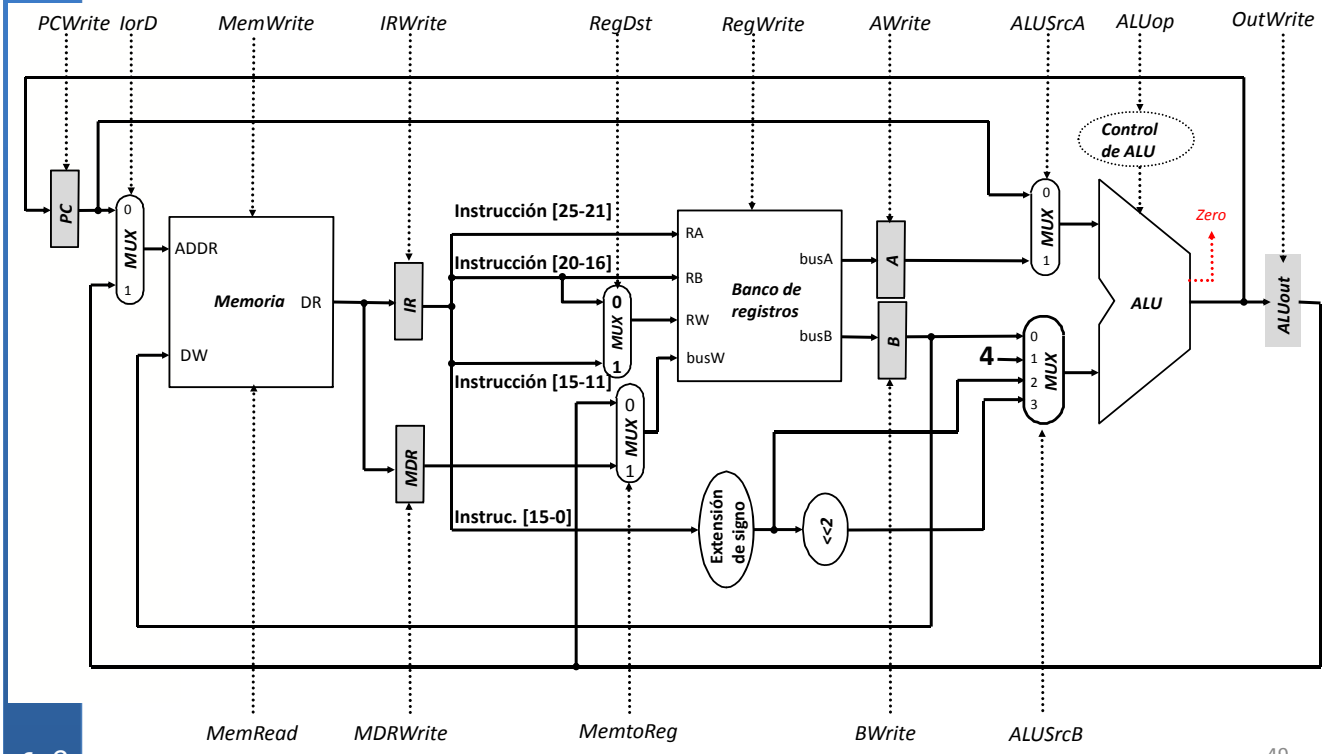


Memoria

- Suponemos un comportamiento idealizado
  - Se comporta temporalmente como el banco de registros (síncronamente)
  - Tiempo de acceso menor que el tiempo de ciclo
  - En el tema 11 se estudiará mas en detalle la memoria
- Unificada e "Integrada" dentro de la CPU
  - Una sola memoria para datos e instrucciones
- Direccionable por bytes
- Tamaño de palabra 4 bytes (32 bits)
  - Permite aceptar/ofrecer 4 bytes por acceso
- Para acceder a la memoria se necesita:
  - 1 entrada de dirección
  - 1 salida de datos de 32 bits
  - 1 entrada de datos de 32 bits
  - Entradas de control para seleccionar el tipo de operación (lectura/escritura)



Señales de Control



Diseño del controlador multiciclo

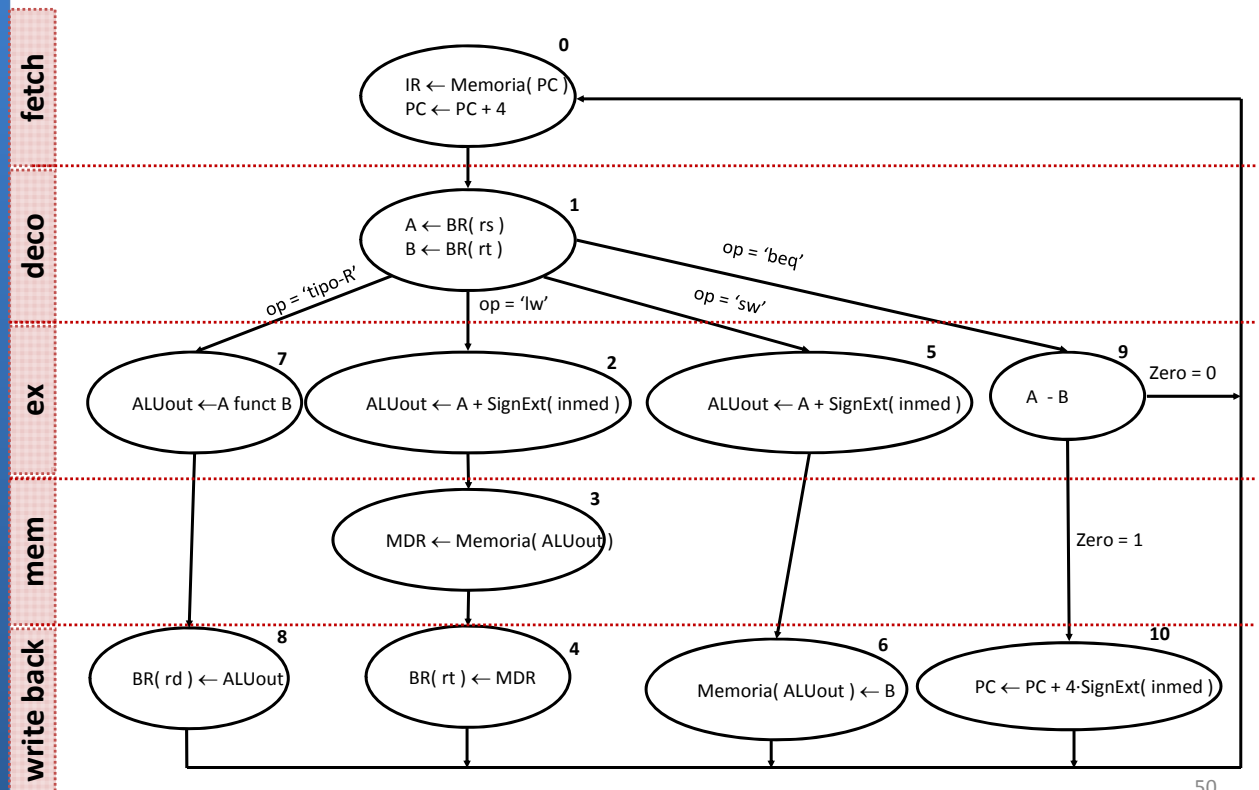
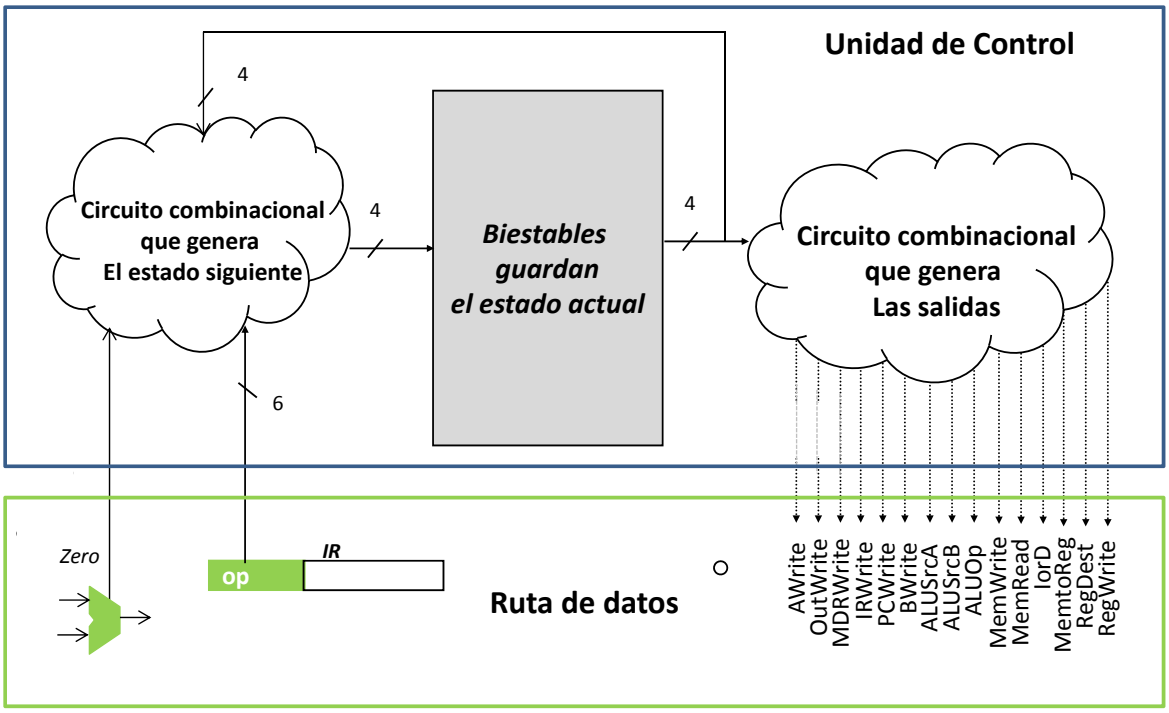
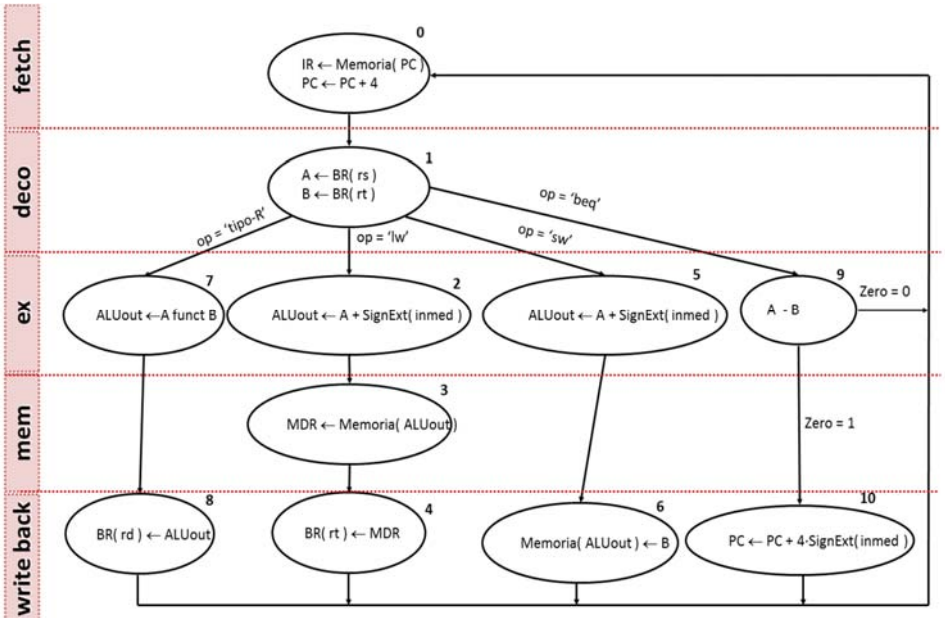


Diagrama conceptual



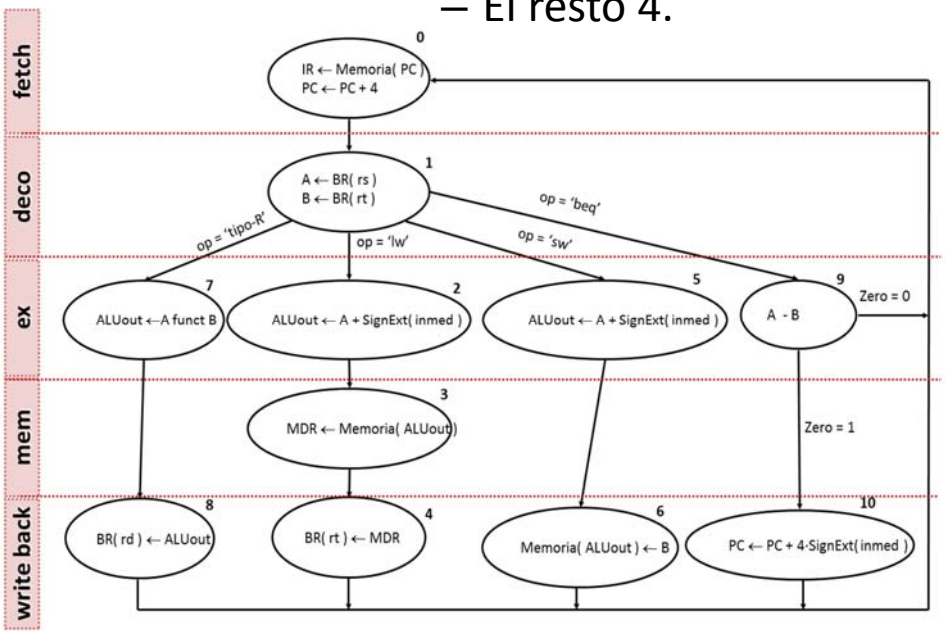
Diseño del controlador multiciclo

- Las fases de búsqueda de instrucción y decodificación son comunes a todas las operaciones
- Las fases de ejecución y almacenamiento del resultado son distintas.



Diseño del controlador multiciclo

- La fase de acceso a memoria sólo la realizan las operaciones lw.
- Ciclos de reloj:
  - La operación de Load tarda 5.
  - La de Salto 3 o 4.
  - El resto 4.

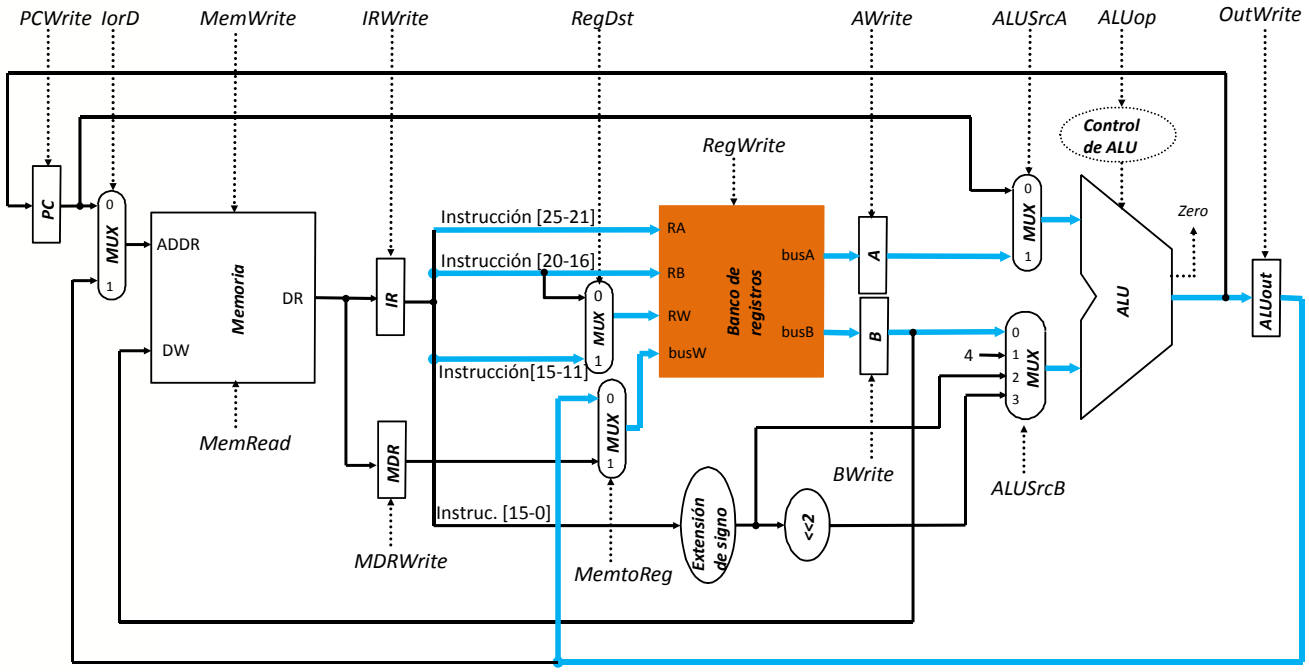


Ejecución de una instrucción Arit-Log

- **Instrucción Aritmetico-lógica**

OP=6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
000000	Rs	Rt	Rd	No se usa	funct

  - BR(Rd) ← BR(Rs) funct BR(Rt)



Diseño del controlador multiciclo

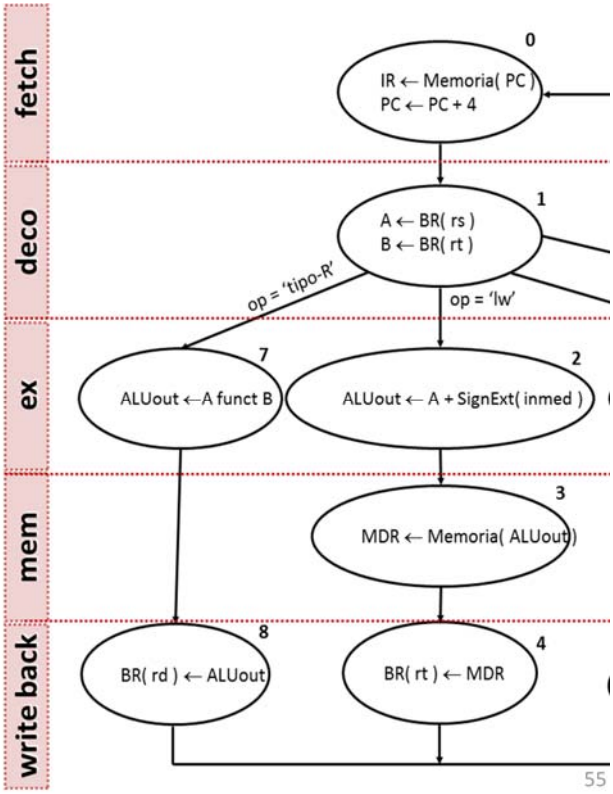
Instrucción Aritmético-lógica

Transferencias entre registros "lógicas"

BR(rd) ← BR(rs) funct BR(rt),
PC ← PC + 4

Transferencias entre registros "físicas"

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A funct B
4. BR(rd) ← ALUout



Diseño del controlador multiciclo

Transferencias entre registros "lógicas"

BR(rd) ← BR(rs) funct BR(rt), PC ← PC + 4

Transferencias entre registros "físicas"

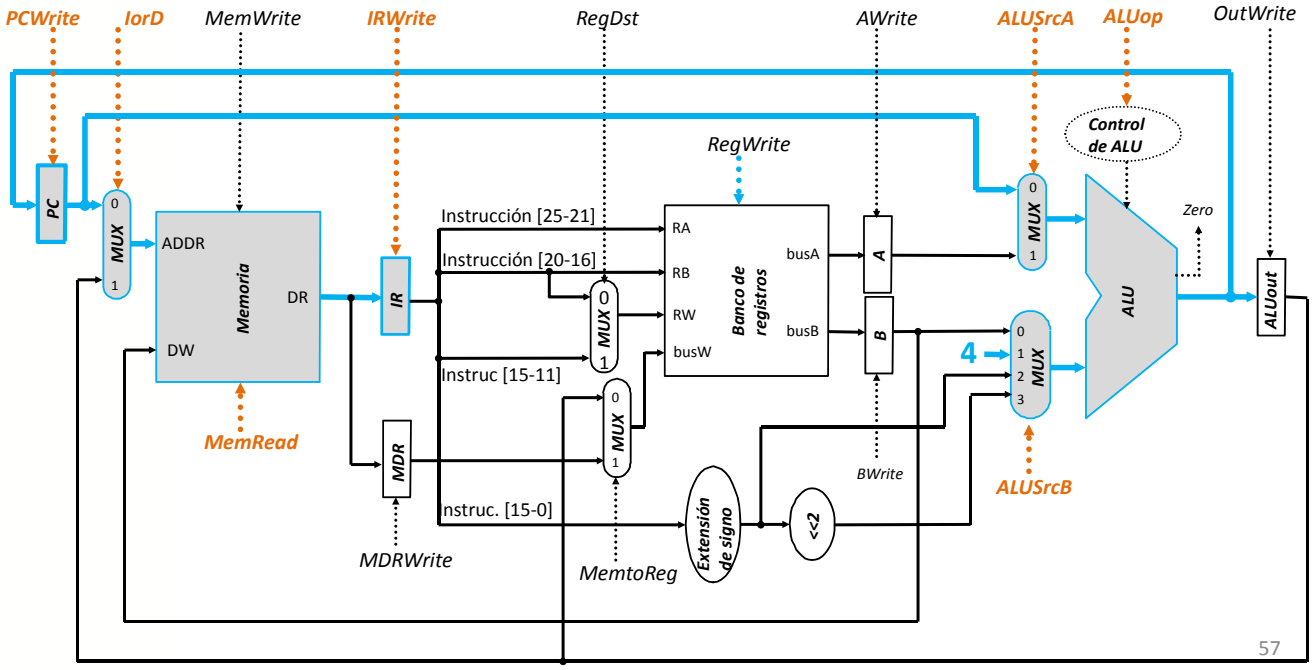
1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A funct B
4. BR(rd) ← ALUout

Instrucción Aritmético-lógica

PCWrite = 1
IorD = 0
MemWrite = 0
MemRead = 1
IRWrite = 1

Ejecución de una instrucción Arit-Log

- **Instrucción Artit-Log → S0: Búsqueda de la instrucción**
  - IR← Memoria(PC) y PC ← PC + 4



Diseño del controlador multiciclo

Transferencias entre registros "lógicas"

BR(rd) ← BR(rs) funct BR(rt), PC ← PC + 4

Transferencias entre registros "físicas"

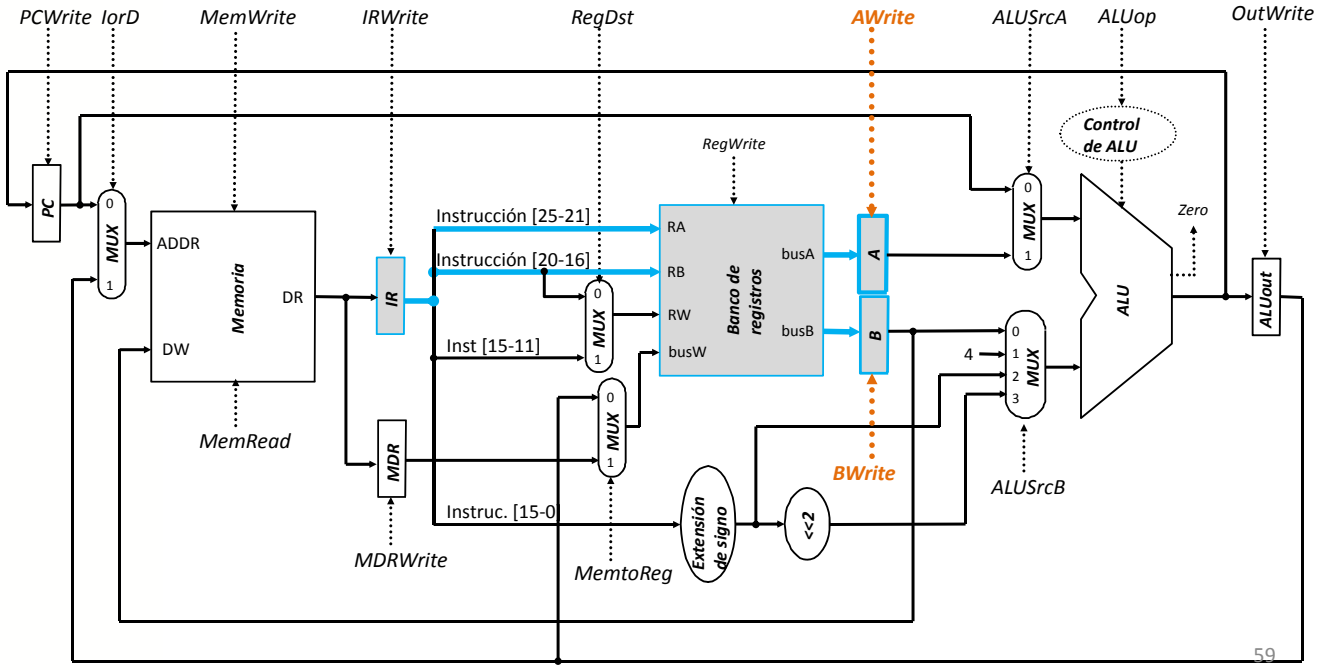
1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A funct B
4. BR(rd) ← ALUout

Instrucción Aritmético-lógica

AWrite = 1
BWrite = 1
RegWrite = 0

Ejecución de una instrucción Arit-Log

- **Instrucción Artit-Log → S1: Decodificación y lectura de registros**
  - A ← BR(Rs)
  - B ← BR(Rt)



Diseño del controlador multiciclo

Transferencias entre registros "lógicas"

BR(rd) ← BR(rs) funct BR(rt), PC ← PC + 4

Transferencias entre registros "físicas"

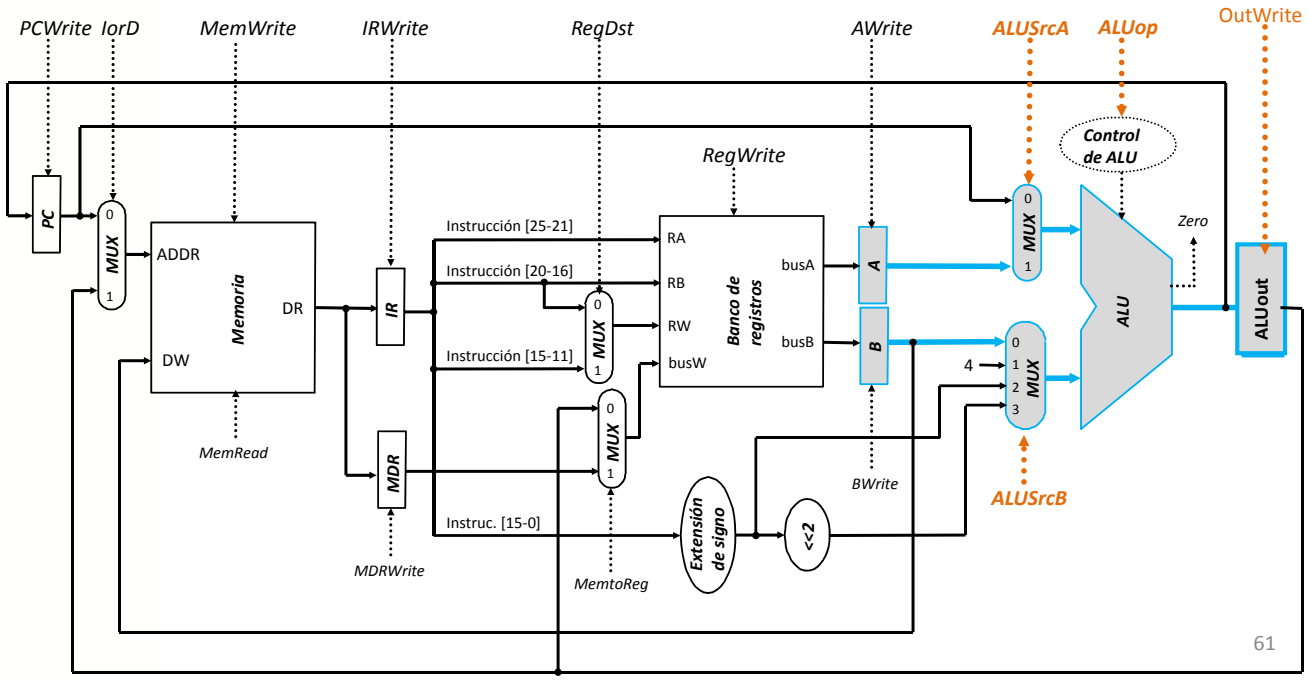
1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A funct B
4. BR(rd) ← ALUout

Instrucción Aritmético-lógica

ALUSrcA = 1
ALUSrcB = 0
ALUOp = *"function"*
OutWrite = 1

Ejecución de una instrucción Arit-Log

- **Instrucción Artit-Log → S7: realizar la función indicada en la instrucción**
  - Aluout ← A funct B
  - A contiene el dato de Rs
  - B contiene el dato de Rt



Diseño del controlador multiciclo

Transferencias entre registros "lógicas"

BR(rd) ← BR(rs) funct BR(rt), PC ← PC + 4

Transferencias entre registros "físicas"

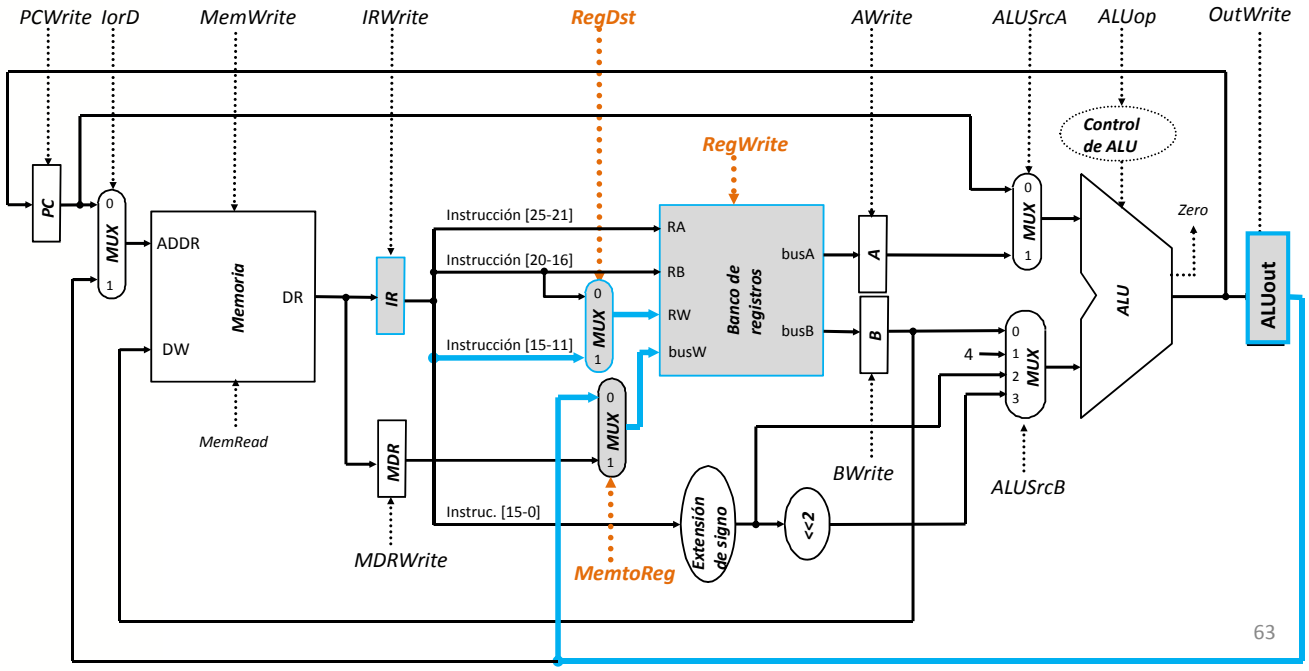
1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A funct B
4. BR(rd) ← ALUout

Instrucción Aritmético-lógica

MemtoReg = 0
RegDst = 1
RegWrite = 1

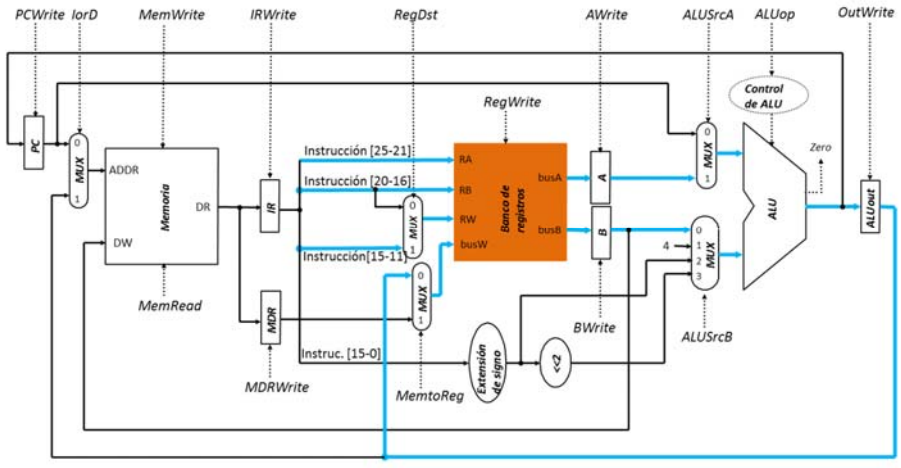
Ejecución de una instrucción Arit-Log

- **Instrucción Arit-Log → S8: Escritura en el banco de registros**
  - ALUout contiene el resultado de la operación
  - BR(Rd) ← ALUout



Diseño del controlador multiciclo

- **Instrucciones Aritmético-lógicas**
  - Todas las señales de carga de registros deberían estar a **cero** cuando no se estén cargando dichos registros.
  - La señal **RegWrite** debería estar a **uno** en el estado **writeback** y a cero en el resto de estados
  - La señal **MemWrite** debería estar a **cero** siempre
  - La señal **MemRead** debería estar a **uno** en el estado **fetch** y a cero en el resto de estados



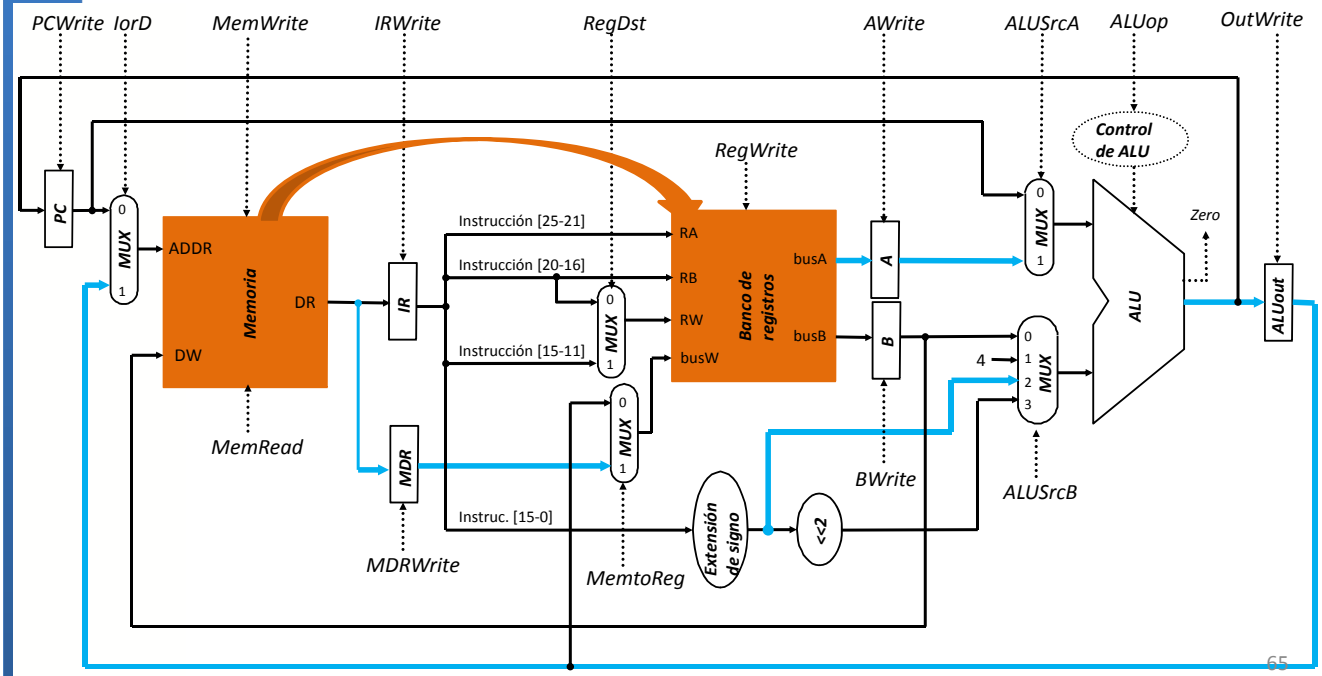
fc2

Ejecución de una instrucción Load

- **Instrucción Load:** lw Rt, desplaz(Rs)

6 bits	5 bits	5 bits	16 bits
OP	Rs	Rt	Desplazamiento

  - **BR(Rt) ← Memoria(BR(Rs) + SignExt(desplaz))**



Diseño del controlador multiciclo

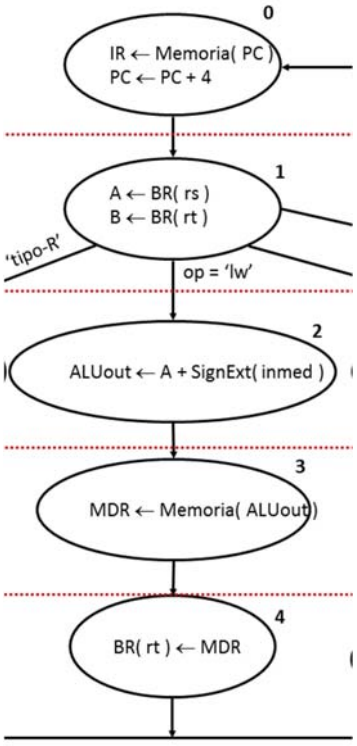
Instrucción de carga (lw)

Transferencias entre registros “lógicas”

BR(rt) ← Memoria(BR(rs) + SignExt(inmed)),
PC ← PC + 4

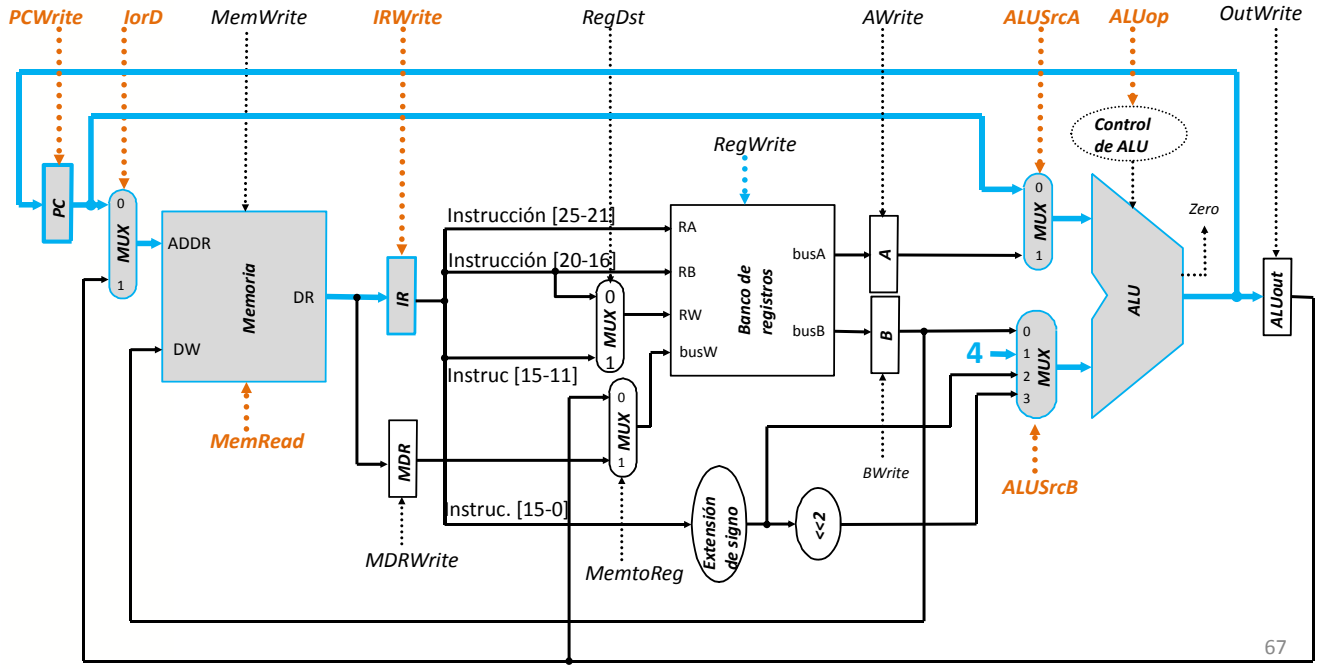
Transferencias entre registros “físicas”

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A + SignExt(inmed)
4. MDR ← Memoria(ALUout)
5. BR(rt) ← MDR



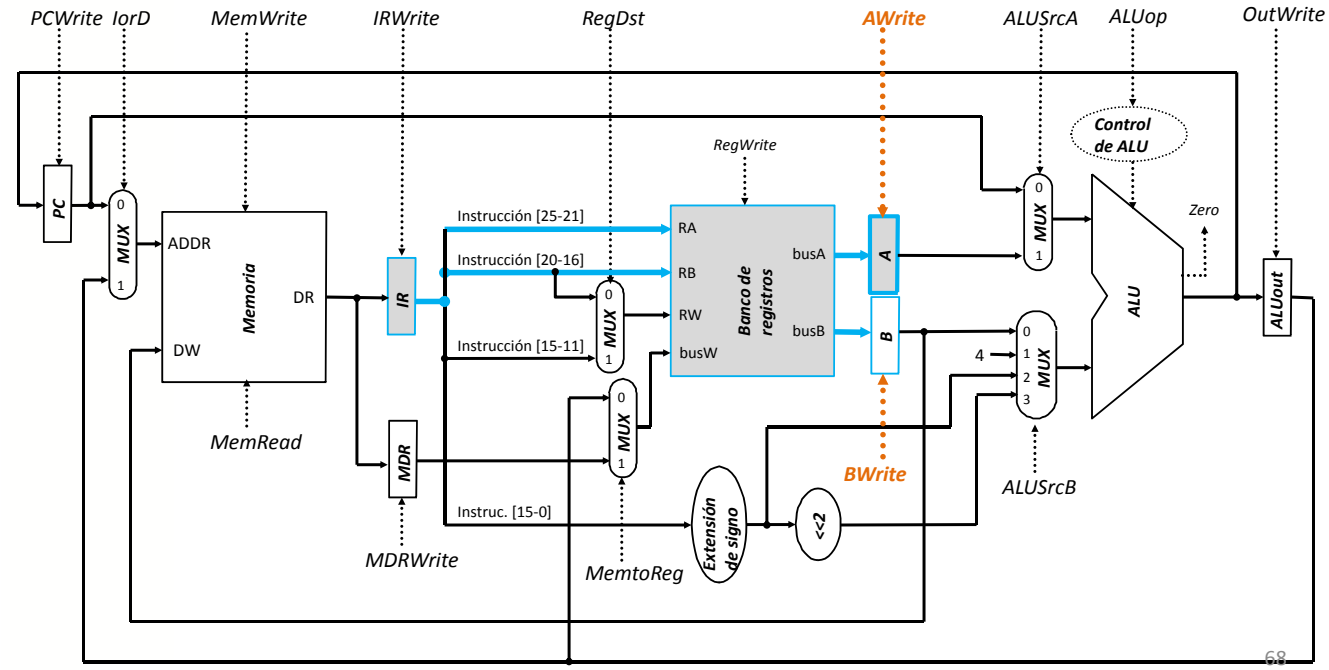
Ejecución de una instrucción Load

- **Instrucción Load → S0: Búsqueda de la instrucción**
  - IR← Memoria(PC) y PC ← PC + 4



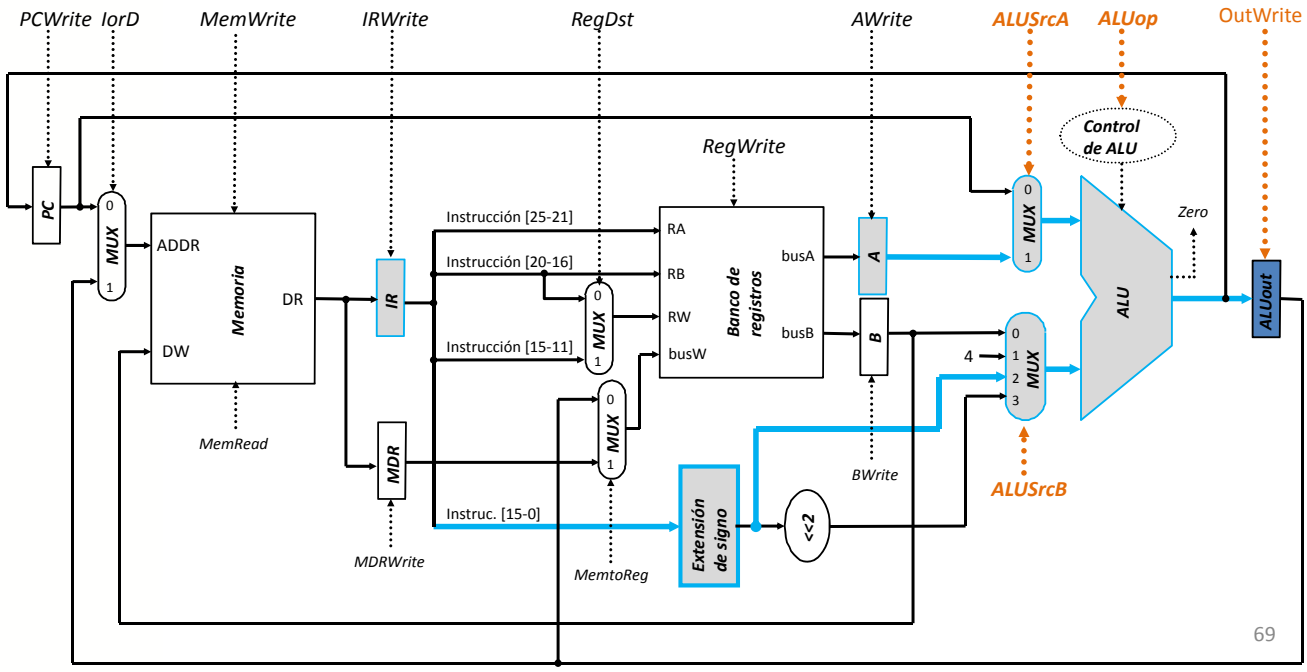
Ejecución de una instrucción Load

- **Instrucción Load → S1: Decodificación y lectura de registros**
  - A ← BR(Rs) este dato es para calcular la dirección
  - B ← BR(Rt) este dato no lo vamos a usar



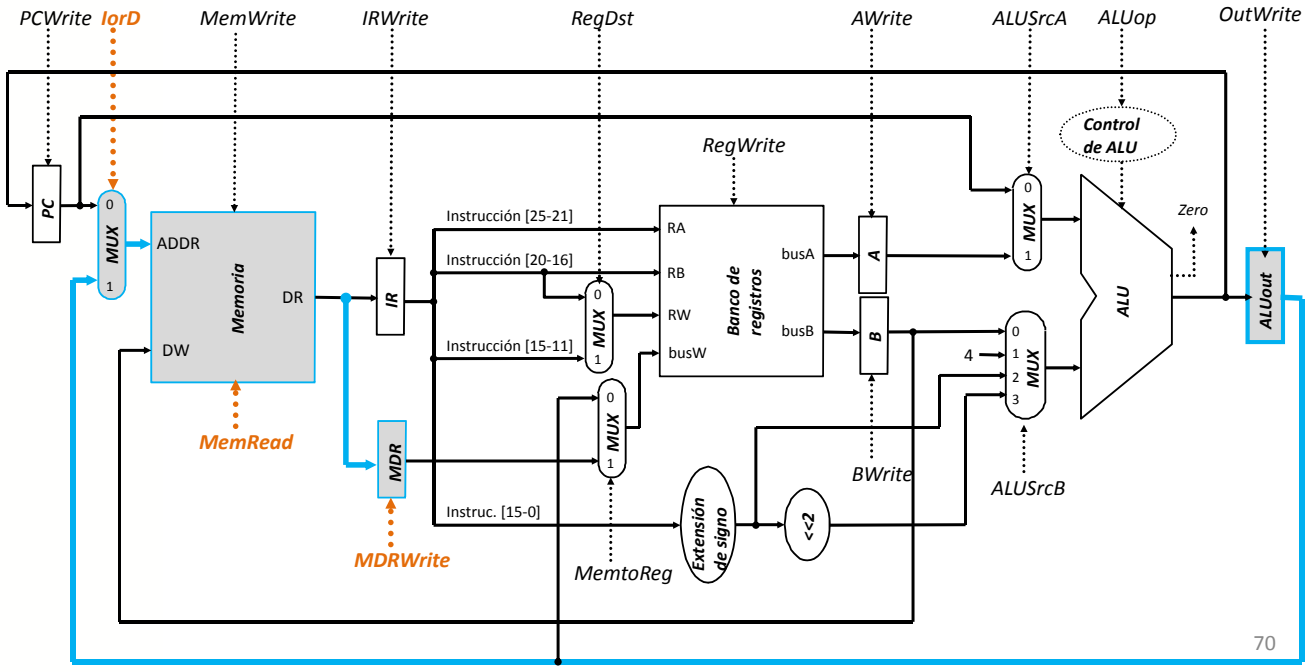
Ejecución de una instrucción Load

- **Instrucción Load → S2: Cálculo de la dirección de memoria**
  - Aluout ← A + SignExt(inmed)
  - A contiene el dato de Rs



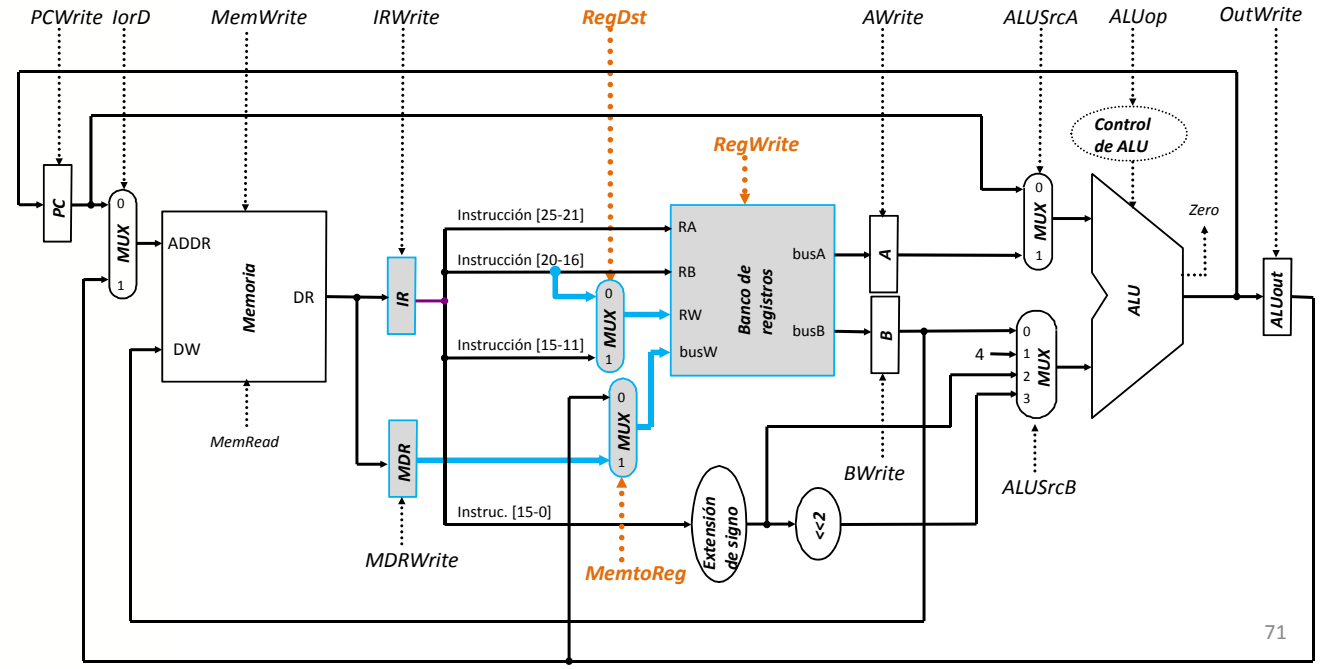
Ejecución de una instrucción Load

- **Instrucción Load → S3: Acceso a memoria**
  - MDR ← Memoria(ALUout)
  - ALUout contiene la dirección de memora



Ejecución de una instrucción Load

- **Instrucción Load → S4: Escritura en el banco de registros**
  - BR(Rt) ← MDR
  - MDR contiene el dato de memoria que se quiere guardar en Rt

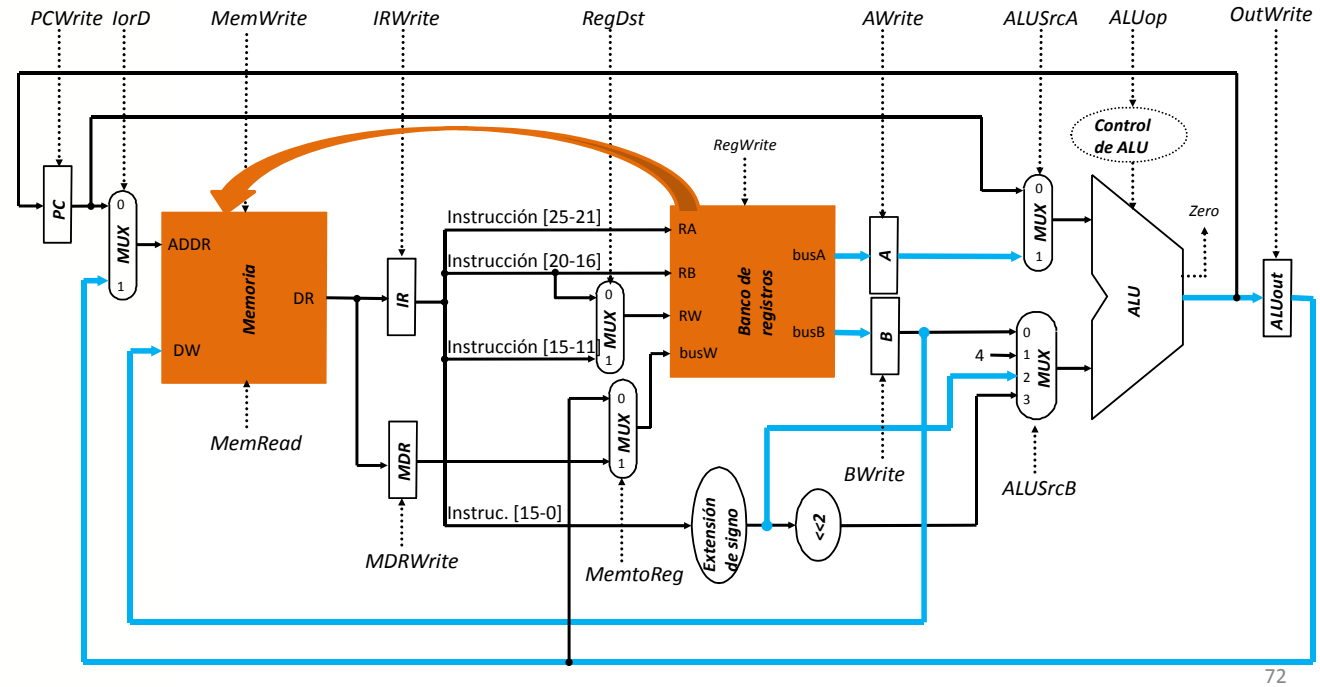


Ejecución de una instrucción Store

- **Instrucción Store:** Sw Rt, desplaz(Rs)

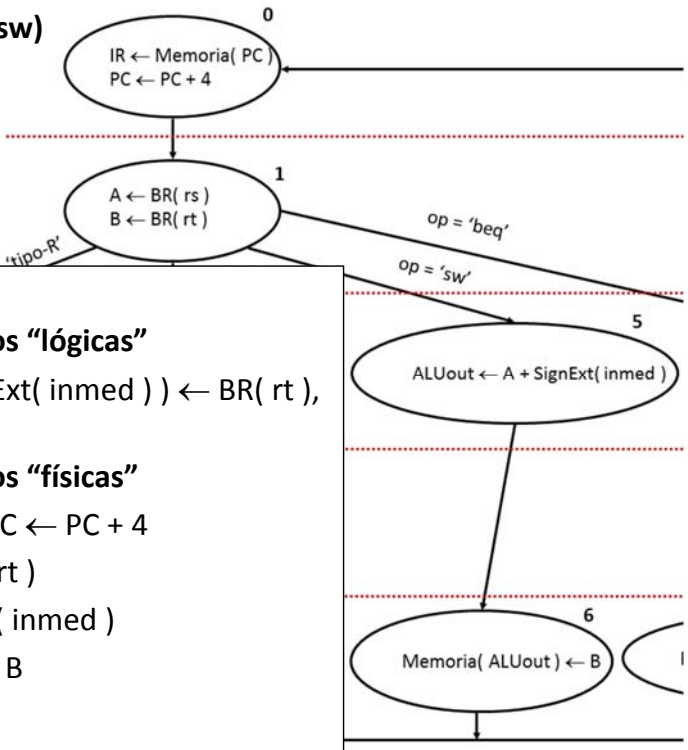
6 bits	5 bits	5 bits	16 bits
OP	Rs	Rt	Desplazamiento

  - Memoria(BR(Rs) + SignExt(desplaz)) ← Rt



Diseño del controlador multiciclo

Instrucción de almacenaje (sw)



Transferencias entre registros "lógicas"

Memoria(BR(rs) + SignExt(inmed)) ← BR(rt), PC ← PC + 4

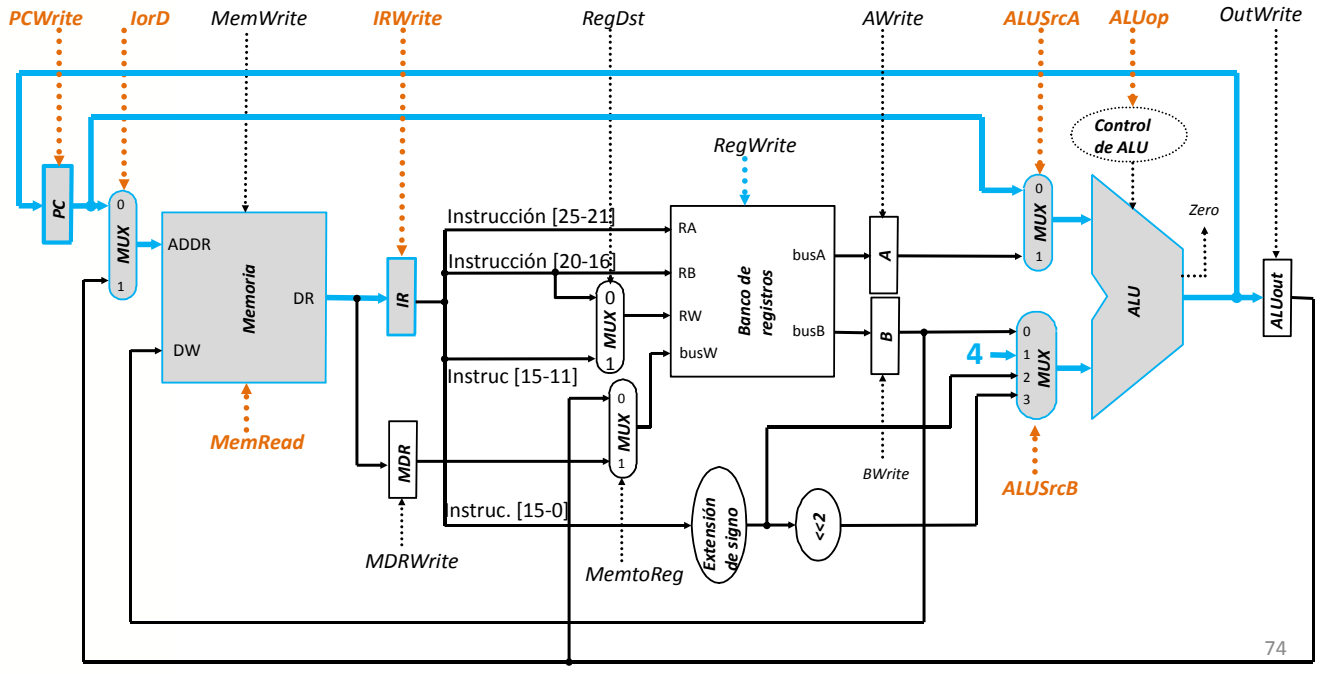
Transferencias entre registros "físicas"

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A + SignExt(inmed)
4. Memoria(ALUout) ← B

fc²

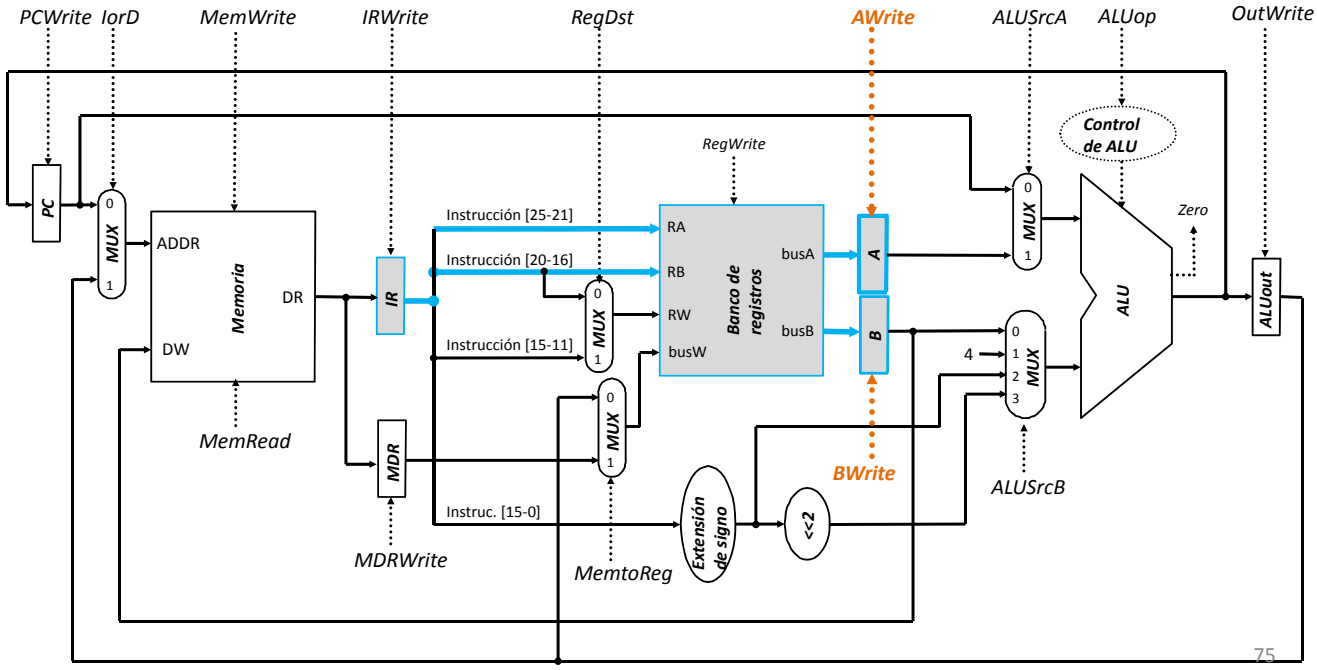
Ejecución de una instrucción Store

- **Instrucción Store → S0: Búsqueda de la instrucción**
  - IR← Memoria(PC) y PC ← PC + 4



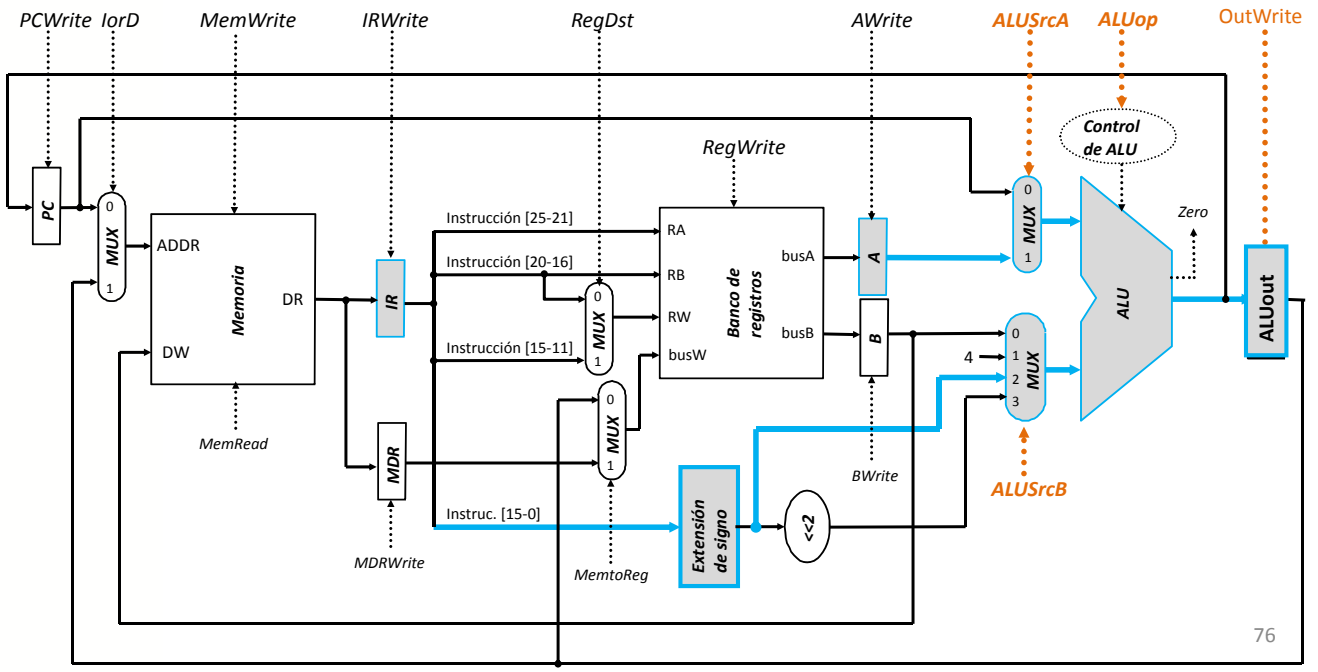
Ejecución de una instrucción Store

- **Instrucción Store → S1: Decodificación y lectura de registros**
  - A ← BR(Rs) este dato es para calcular la dirección
  - B ← BR(Rt) este dato es el que se guarda en memoria



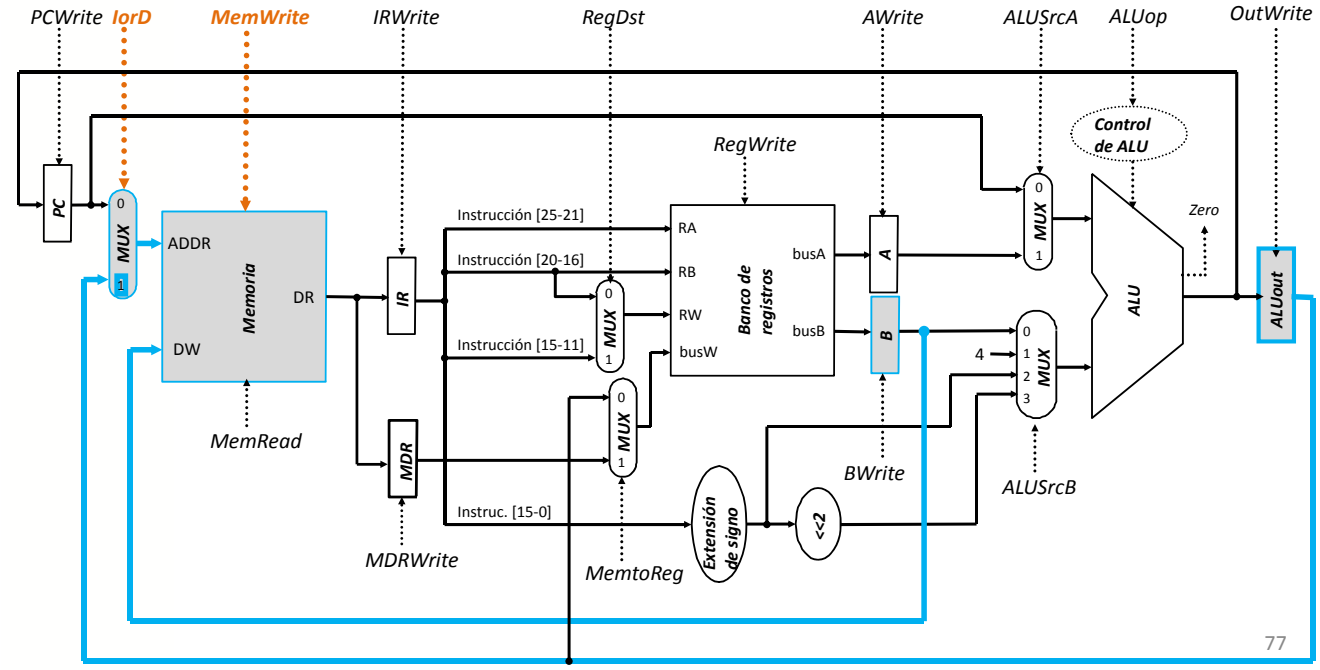
Ejecución de una instrucción Store

- **Instrucción Store → S5: Cálculo de la dirección de memoria**
  - Aluout ← A + SignExt(desplaz)
  - A contiene el dato de Rs



Ejecución de una instrucción Store

- **Instrucción Store → S6: Escritura en memoria**
  - Memoria(ALUout) ← B
  - B contiene el dato del registro Rt que se quiere guardar en memoria
  - ALUout contiene la dirección de memora

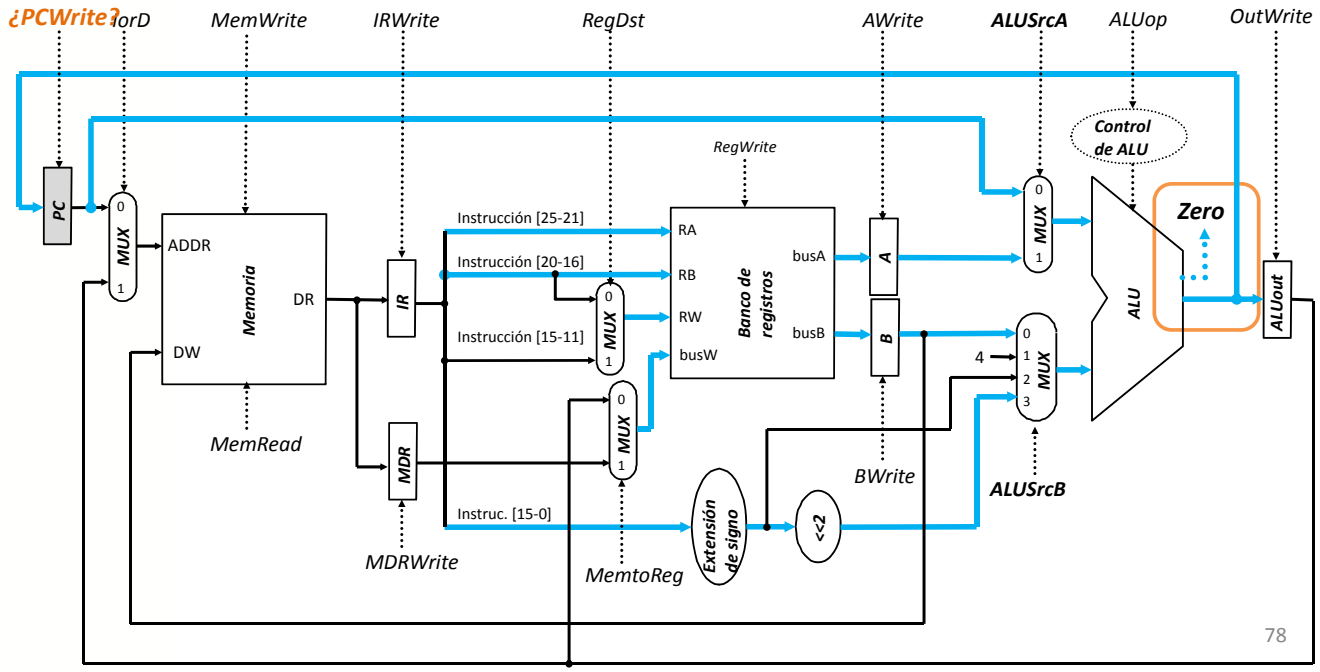


Ejecución de una instrucción Salto

- **Instrucción de salto:** Beq Rs,Rt, Desplz

6 bits	5 bits	5 bits	16 bits
OP	Rs	Rt	Desplazamiento

  - si (BR(Rs) = BR(Rt))
    - Entonces PC ← PC + 4·SignExt(desplaz)
    - Sino PC ← PC + 4



Diseño del controlador multiciclo

Observaciones: en todas las instrucciones las acciones 1. y 2. son iguales

Instrucción de salto condicional (beq)

Transferencias entre registros "lógicas"

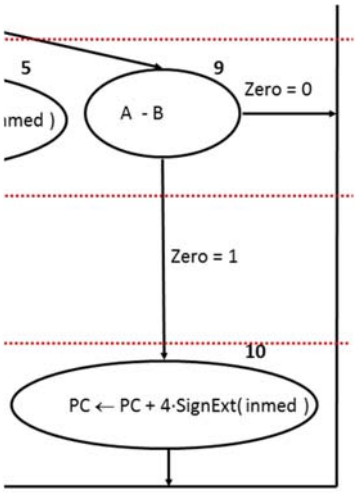
si (BR(rs) = BR(rt))

entonces PC ← PC + 4 + 4·SignExt(inmed)

sino PC ← PC + 4

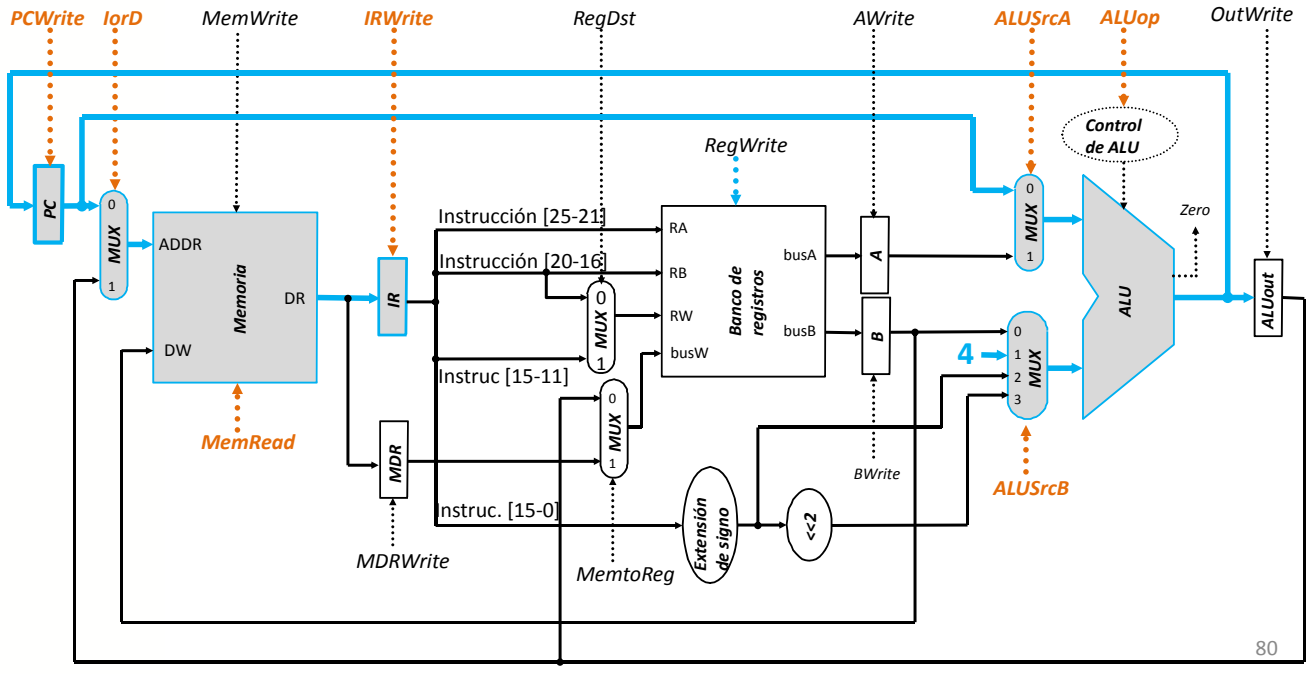
Transferencias entre registros "físicas"

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt),
3. A - B
4. si Zero entonces PC ← PC + 4·SignExt(inmed)



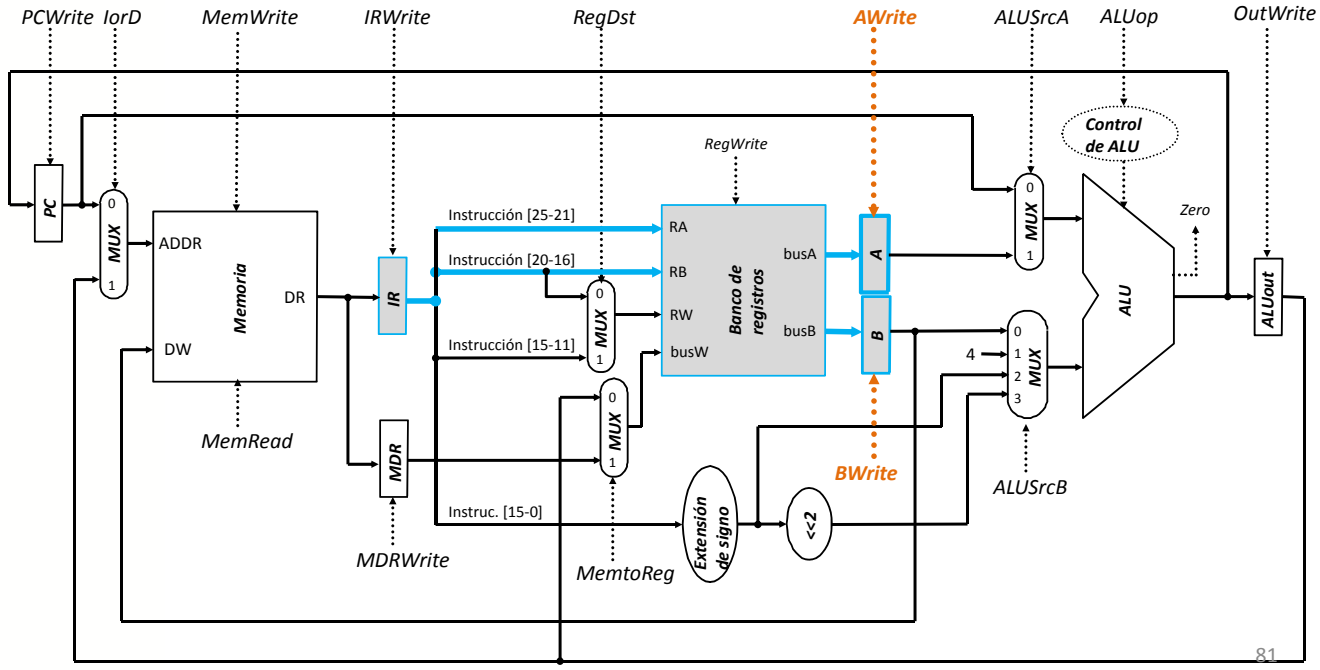
Ejecución de una instrucción Salto

- **Instrucción de Salto → S0: Búsqueda de la instrucción**
  - IR← Memoria(PC) y PC ← PC + 4



Ejecución de una instrucción de Salto

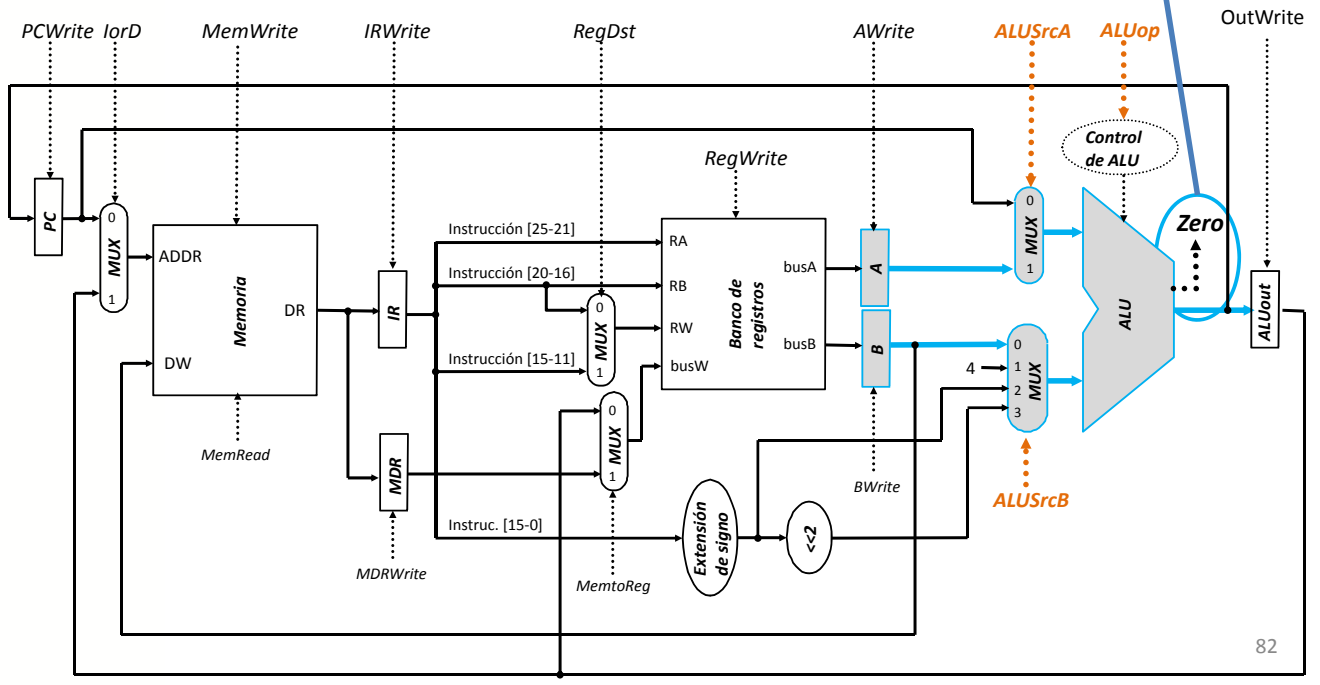
- **Instrucción de Salto → S1: Decodificación y lectura de registros**
  - A ← BR(Rs)
  - B ← BR(Rt)



Ejecución de una instrucción de Salto

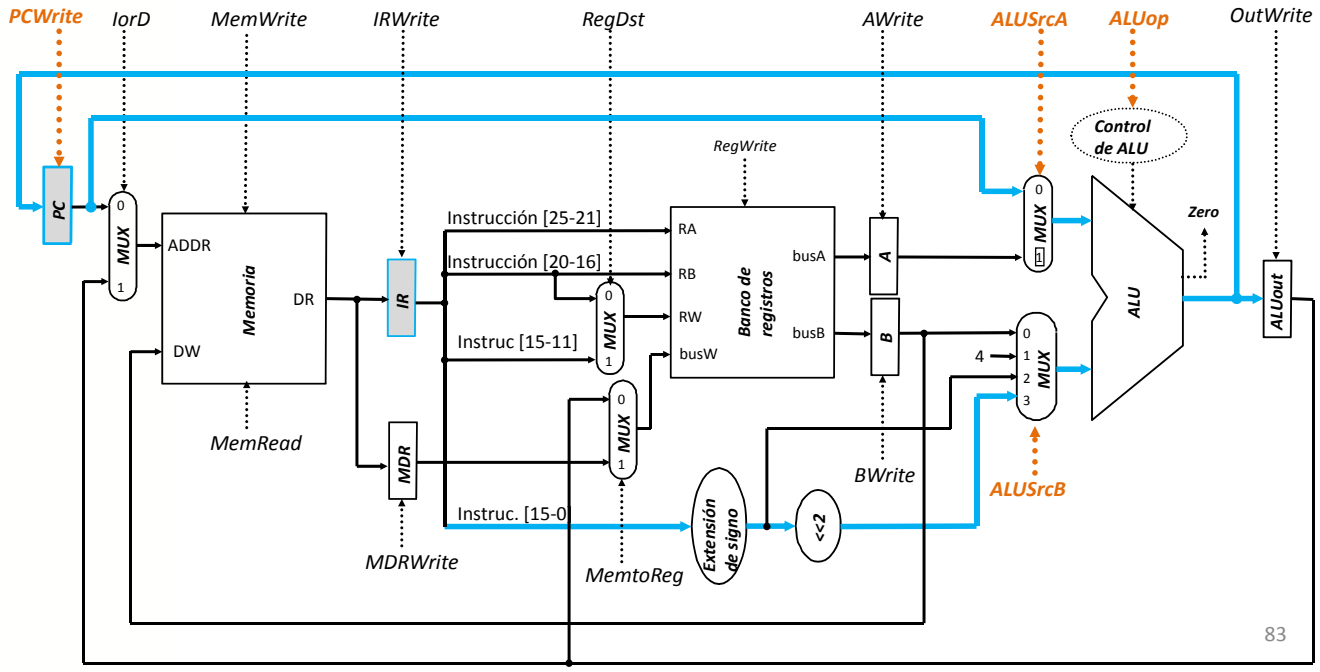
- **Instrucción de Salto → S9: Realiza una resta**
  - Aluout ← A - B
  - A contiene el dato de Rs
  - B contiene el dato de Rt

La unidad de control comprueba este valor
Si Z=1 salta, sino No salta



Ejecución de una instrucción Salto

- **Instrucción de salto → S10: Cálculo de la dirección de salto**
  - PC ← PC + 4 + 4·SignExt(desplaz)

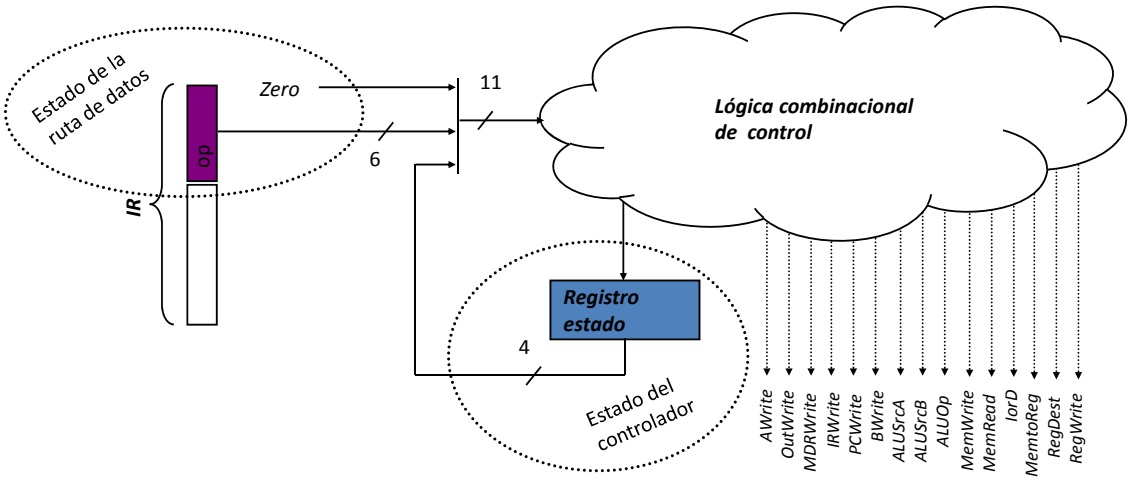


Diseño del controlador multiciclo

El controlador como FSM (finite state machine)

- 1. Se **traducen** las transferencias entre registros como **conjuntos de activaciones** de los puntos de control de la ruta de datos
- 2. Se **codifican** los estados
- 3. Mediante **tablas de verdad** se describen:
  - las **transiciones de estado** en función del código de operación y del estado de la ruta de datos
  - el **valor de las señales** de control en función del estado (controlador tipo Moore) y adicionalmente en función de algunas señales de la ruta de datos (controlador tipo Mealy).
- 4. La **estructura del controlador** estará formada por:
  - **Registro de estado**
  - Conjunto de **lógica combinacional de control** que implementa las anteriores tablas

Diseño del controlador multiciclo

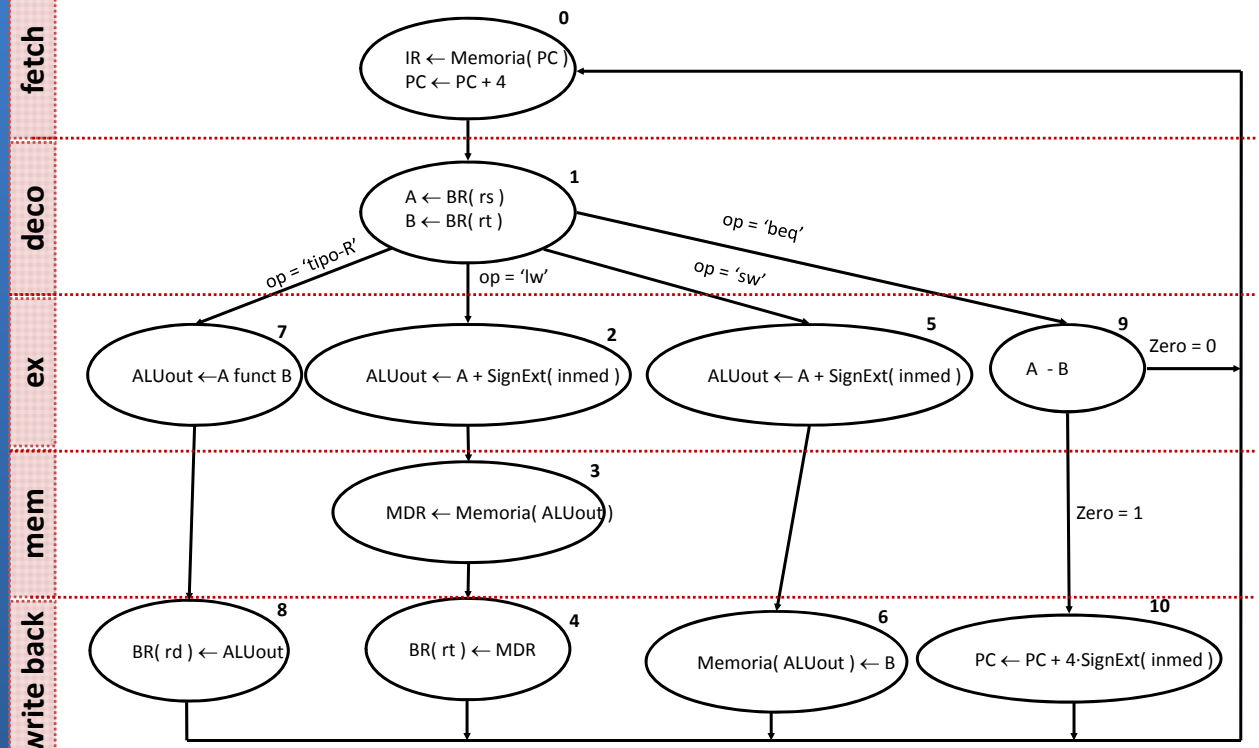


Diseño del controlador multiciclo

Tabla de verdad del controlador

Estado actual	op	Zero	Estado siguiente	IRWrite	PCWrite	AWrite	BWrite	ALUSrcA	ALUScrB	ALUOp	OutWrite	MemWrite	MemRead	IorD	MDRWrite	MemtoReg	RegDest	RegWrite
0000	XXXXXX	X	0001	1	1			0	01	00 (add)		0	1	0				0
0001	100011 (lw)	X	0010															
0001	101011 (sw)	X	0101	0	0	1	1					0	0					0
0001	000000 (tipo-R)	X	0111															
0001	000100 (beq)	X	1001															
0010	XXXXXX	X	0011	0	0			1	10	00 (add)	1	0	0					0
0011	XXXXXX	X	0100	0	0							0	1	1	1			0
0100	XXXXXX	X	0000	0	0							0	0			1	0	1
0101	XXXXXX	X	0110	0	0		0	1	10	00 (add)	1	0	0					0
0110	XXXXXX	X	0000	0	0							1	0	1				0
0111	XXXXXX	X	1000	0	0			1	00	10 (funct)	1	0	0					0
1000	XXXXXX	X	0000	0	0							0	0			0	1	1
1001	XXXXXX	0	0000	0	0			1	00	01 (sub)		0	0					0
1001	XXXXXX	1	1010															
1010	XXXXXX	X	0000	0	1			0	11	00 (add)		0	0					0

Diseño del controlador multiciclo



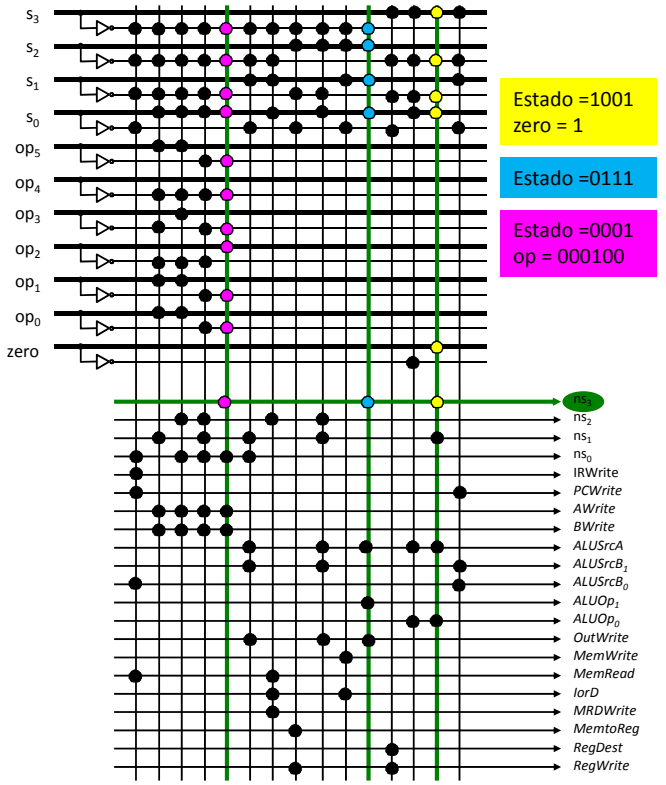
Diseño del controlador multiciclo

Estado actual	op	Zero	Estado siguiente	IRWrite	PCWrite	AWrite	BWrite	ALUSrcA	ALUScrB	ALUOp	OutWrite	MemWrite	MemRead	IorD	MDRWrite	MemtoReg	RegDest	RegWrite
$s_3\ s_2\ s_1\ s_0$	$op_5\ op_4\ op_3\ op_2\ op_1\ op_0$		**ns_3** $ns_2\ ns_1\ ns_0$															
0 0 0 0	X X X X X X	X	**0** 0 0 1	1	1			0	01	00 (add)		0	1	0				0
0 0 0 1	1 0 0 0 1 1	X	**0** 0 1 0															
0 0 0 1	1 0 1 0 1 1	X	**0** 1 0 1															
0 0 0 1	0 0 0 0 0 0	X	**0** 1 1 1	0	0	1	1					0	0					0
0 0 0 1	0 0 0 1 0 0	X	**1** 0 0 1															
0 0 1 0	X X X X X X	X	**0** 0 1 1	0	0			1	10	00 (add)	1	0	0					0
0 0 1 1	X X X X X X	X	**0** 1 0 0	0	0							0	1	1	1			0
0 1 0 0	X X X X X X	X	**0** 0 0 0	0	0							0	0			1	0	1
0 1 0 1	X X X X X X	X	**0** 1 1 0	0	0		0	1	10	00 (add)	1	0	0					0
0 1 1 0	X X X X X X	X	**0** 0 0 0	0	0							1	0	1				0
0 1 1 1	X X X X X X	X	**1** 0 0 0	0	0			1	00	10 (funct)	1	0	0					0
1 0 0 0	X X X X X X	X	**0** 0 0 0	0	0							0	0			0	1	1
1 0 0 1	X X X X X X	0	**0** 0 0 0															
1 0 0 1	X X X X X X	1	**1** 0 1 0	0	0			1	00	01 (sub)		0	0					0
1 0 1 0	X X X X X X	X	**0** 0 0 0	0	1			0	11	00 (add)		0	0					0

Diseño del controlador multiciclo

- Lógica discreta:
  - 21 funciones de conmutación
  - 11 variables diferentes
- 1 PLA
  - 11 entradas
  - 21 salidas
  - 15 términos producto
- 1 ROM (~42 Kbits):
  - 11 bits de dirección (2^{11} palabras)
  - palabras de 21 bits
- 2 ROM (~10 Kbits)
  - ROM de control:
    - 4 bits de dirección (2^4 palabras)
    - palabra de 17 bits
  - ROM de siguiente estado:
    - 11 bits de dirección (2^{11} palabras)
    - palabras de 4 bits

- Ventajas de la lógica discreta:
  - velocidad y coste
- Ventajas de la lógica almacenada:
  - facilidad de diseño
  - adaptabilidad

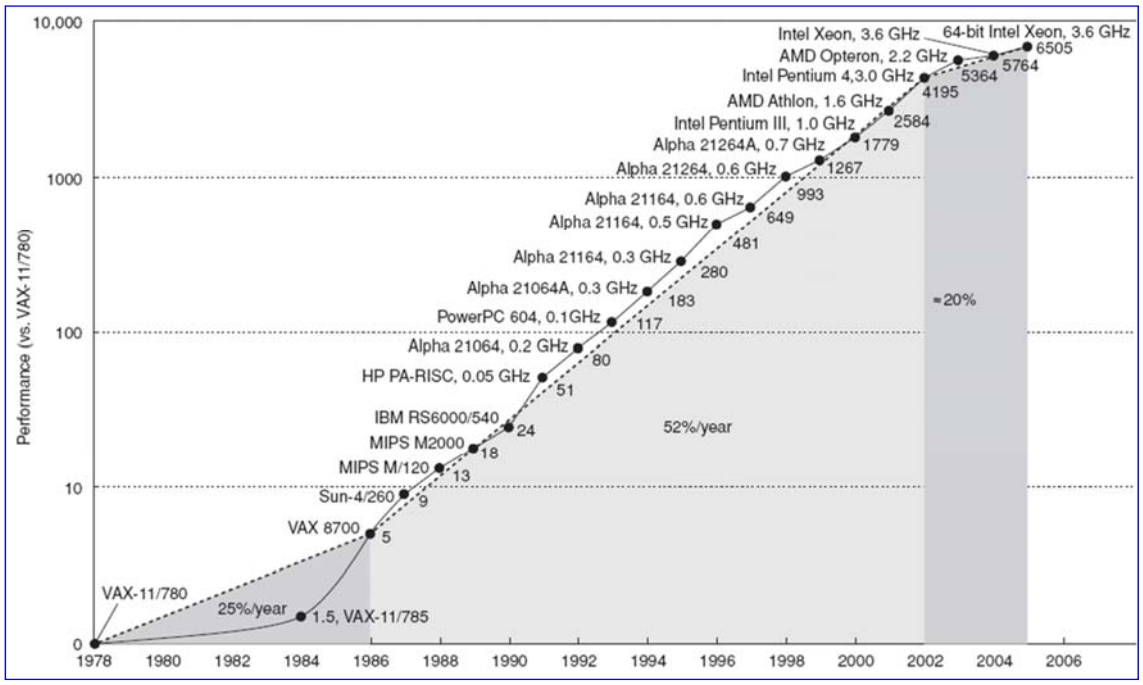


El papel del rendimiento

"Los buenos programadores se han preocupado siempre por el rendimiento de sus programas porque la rápida obtención de resultados es crucial para crear programas de éxito"

D. A. Patterson y J. L. Henessy

Crecimiento del rendimiento de los procesadores

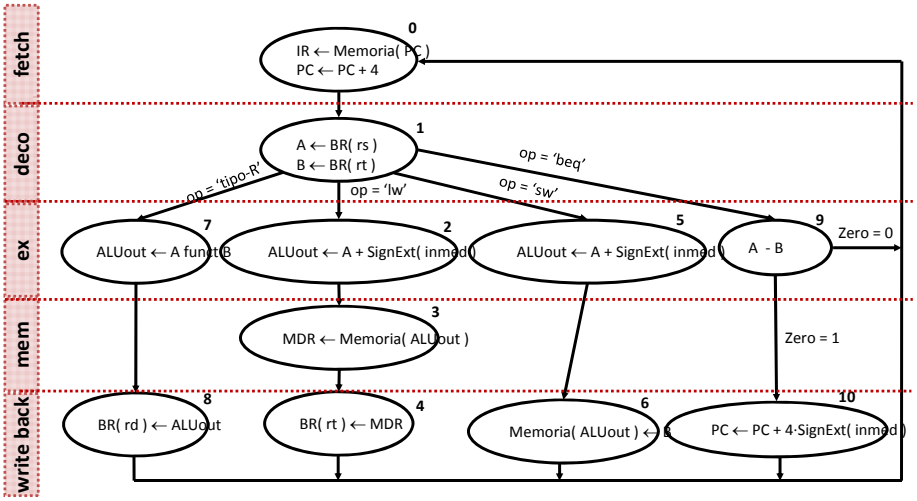


fc²

Rendimiento de los procesadores

- **¿Cuántos ciclos tarda en ejecutarse este programa?**
  - **Depende del diseño del procesador**: por ejemplo en el MIPS multiciclo

lw r1, 0(r0) → 5
lw r2, 4(r0) → 5
add r3, r1, r2 → 4
beq r3, r5, 1 → 4!
sub r3, r3, r5 → 4
sw r3, 8(r0) → 4

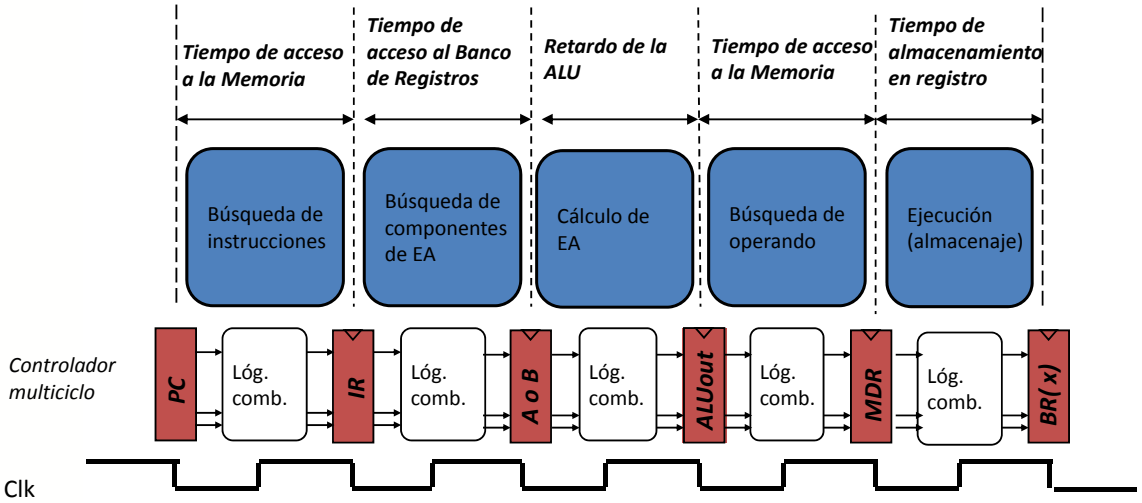


- **Y cuánto Tiempo**
  - **Depende de la frecuencia del procesador**

fc²

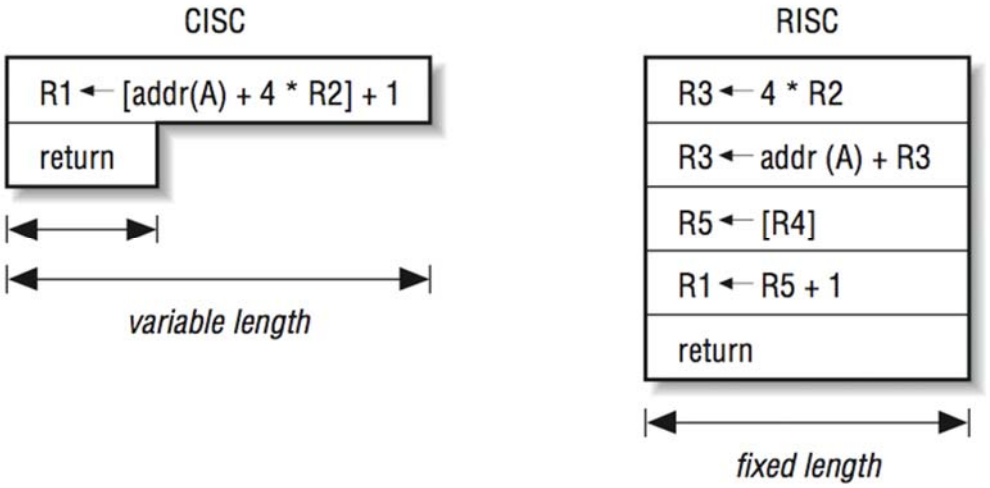
Rendimiento de los procesadores

- Temporización:
  - La etapa más lenta limita el periodo de reloj.



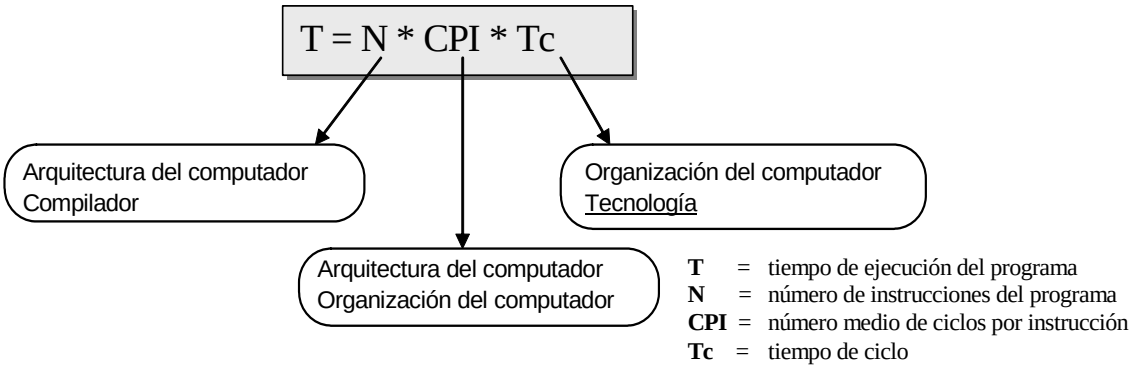
Rendimiento de los procesadores

- Nº de instrucciones máquina depende de:
  - Arquitectura
  - Compilador



Rendimiento de los procesadores

- **Tiempo de ejecución de un programa**



- **CPI =** Ciclos medios por instrucción
  - Una instrucción necesita varios ciclos de reloj para su ejecución
  - Además, diferentes instrucciones tardan diferentes cantidades de tiempo
  - **CPI**= Es una suma ponderada del número de ciclos que tarda por separado cada tipo de instrucción

fc²

Rendimiento de los procesadores

- **Unidades para expresar el rendimiento de un procesador**
  - ¿Cuántas instrucciones ejecuta un computador en un segundo?
  - MIPS: (instrucciones por programa) / (tiempo de ejecución x 10^6)
  - MIPS: (frecuencia de reloj) / (CPI x 10^6)

- No sirve para comparar dos computadores de diferente familia arquitectónica.

fc²