DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Sistemas Digitales

Microarquitectura

Primer Cuatrimestre 2025

Sistemas Digitales
DC - UBA

Introducción

Clase de hoy

Hoy vamos a ver:

- Definición de **microarquitecturas**.

Clase de hoy

Hoy vamos a ver:

- Definición de **microarquitecturas**.
- Estado de arquitectura, elementos de memoria, datapath y unidad de control.

Clase de hoy

Hoy vamos a ver:

- Definición de **microarquitecturas**.
- Estado de arquitectura, elementos de memoria, datapath y unidad de control.
- Procesador de ciclo simple.

Clase de hoy

Hoy vamos a ver:

- Definición de **microarquitecturas**.
- Estado de arquitectura, elementos de memoria, datapath y unidad de control.
- Procesador de ciclo simple.
- Instrucciones de memoria, registros y saltos condicionales.

Microarquitectura

La microarquitectura se ubica conceptualmente entre la **arquitectura** (aquello que se expone a la persona que programa el sistema) y la lógica combinatoria y secuencial. Implementa el soporte de estado arquitectónico y la lógica de control para actualizar el estado según lo indique la semántica de las instrucciones de la **ISA**.

Estado de la arquitectura

La microarquitectura va encargarse de actualizar el **estado de la arquitectura**, o sea, los **registros de propósito general y el program counter**. Recordemos que nos referimos como estado a los valores almacenados en los elementos de memoria y por ende el estado de la arquitectura se refiere a los elementos de memoria expuestos a la persona que programa el sistema.

Estado de la arquitectura

El procesador puede contener elementos de memoria que constituyen estado por fuera de la arquitectura, registros o banco de memoria utilizados para implementar mecanismos o funciones propios de la arquitectura pero que no son expuestos.

Instrucciones a evaluar

A la hora de justificar las decisiones de diseño e implementación de una microarquitectura para `RISCV`, vamos a enfocarnos en un subconjunto de las instrucciones básicas:

Instrucciones a evaluar

A la hora de justificar las decisiones de diseño e implementación de una microarquitectura para `RISCV`, vamos a enfocarnos en un subconjunto de las instrucciones básicas:

- **Registros:** `add`, `sub`, `and`, `or`, `slt`.

Instrucciones a evaluar

A la hora de justificar las decisiones de diseño e implementación de una microarquitectura para `RISCV`, vamos a enfocarnos en un subconjunto de las instrucciones básicas:

- **Registros:** `add`, `sub`, `and`, `or`, `slt`.
- **Memoria:** `sw`, `lw`.

Instrucciones a evaluar

A la hora de justificar las decisiones de diseño e implementación de una microarquitectura para `RISCV`, vamos a enfocarnos en un subconjunto de las instrucciones básicas:

- **Registros:** `add`, `sub`, `and`, `or`, `slt`.
- **Memoria:** `sw`, `lw`.
- **Salto:** `beq`.

Proceso de diseño - Datapath

Para comenzar con el diseño de un sistema complejo, como es el caso de nuestra microarquitectura, un enfoque posible es comenzar presentando y vinculando a los elementos que realizarán transformaciones con los datos, a esto lo llamaremos **el camino de datos o datapath**.

Proceso de diseño - Unidad de control

Luego decidiremos cómo implementar la unidad que se asegura de coordinar a los elementos del **datapath** para transformar a los datos a partir de la manipulación de sus señales de control, a esto lo llamaremos **unidad de control**.

Sobre los diagramas

En los diagramas se debe observar que:

Sobre los diagramas

En los diagramas se debe observar que:

- Las líneas **gruesas** indican datos de 32 bits.

Sobre los diagramas

En los diagramas se debe observar que:

- Las líneas **gruesas** indican datos de 32 bits.
- Las líneas *delgadas* indican datos de 1 bit.

Sobre los diagramas

En los diagramas se debe observar que:

- Las líneas **gruesas** indican datos de 32 bits.
- Las líneas *delgadas* indican datos de 1 bit.
- Las líneas `intermedias` indican datos de otro tamaño.

Sobre los diagramas

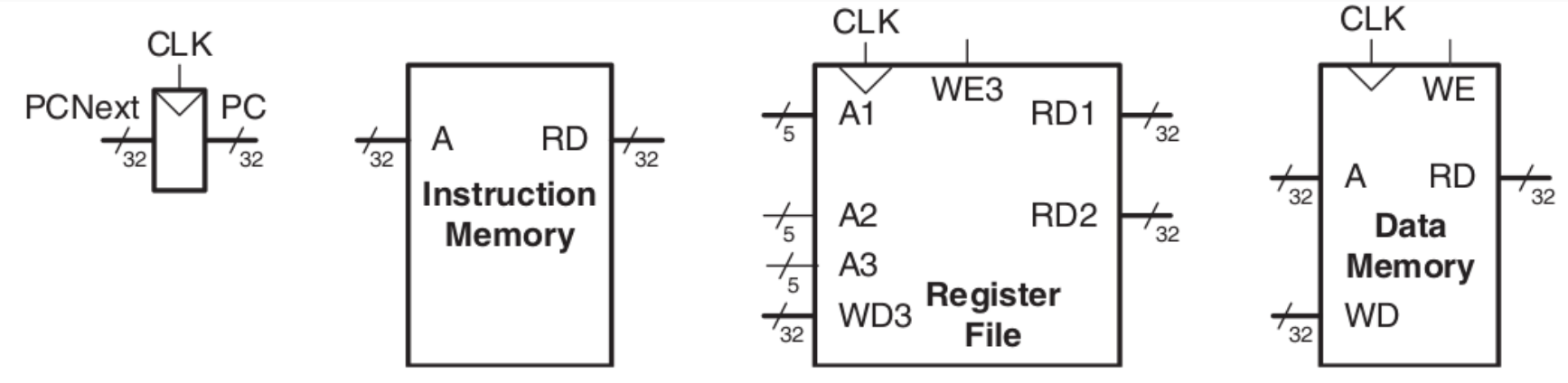
En los diagramas se debe observar que:

- Las líneas **gruesas** indican datos de 32 bits.
- Las líneas *delgadas* indican datos de 1 bit.
- Las líneas `intermedias` indican datos de otro tamaño.
- Las líneas azules indican señales de control.

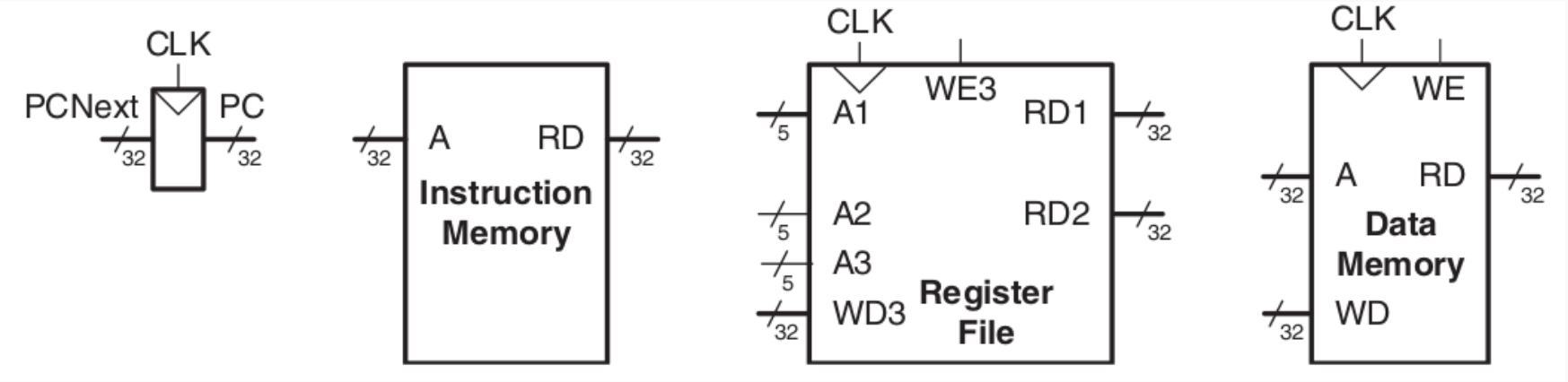
Elementos de memoria

Ahora vamos a presentar y estudiar los elementos de memoria del **datapath**.

Elementos de memoria

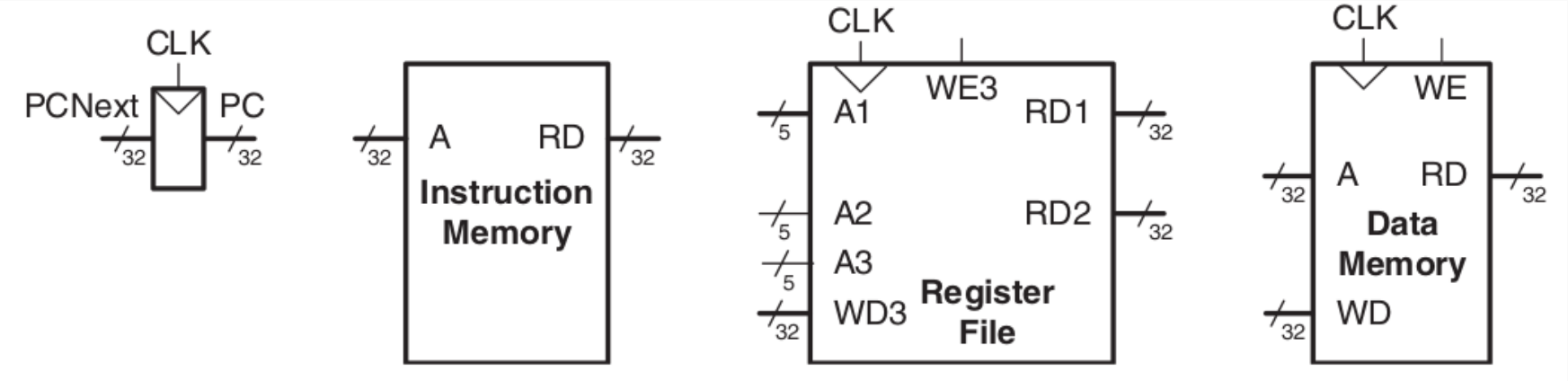


PC



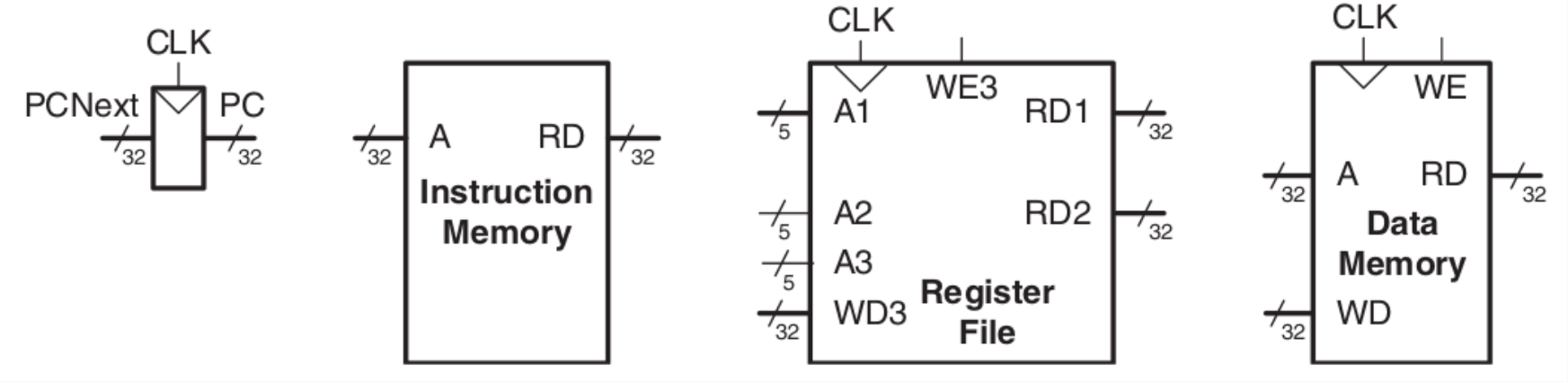
La salida **PC** indica la posición de la instrucción actual, **PCNext** es la entrada que indica la posición de la próxima instrucción.

Memoria de instrucciones



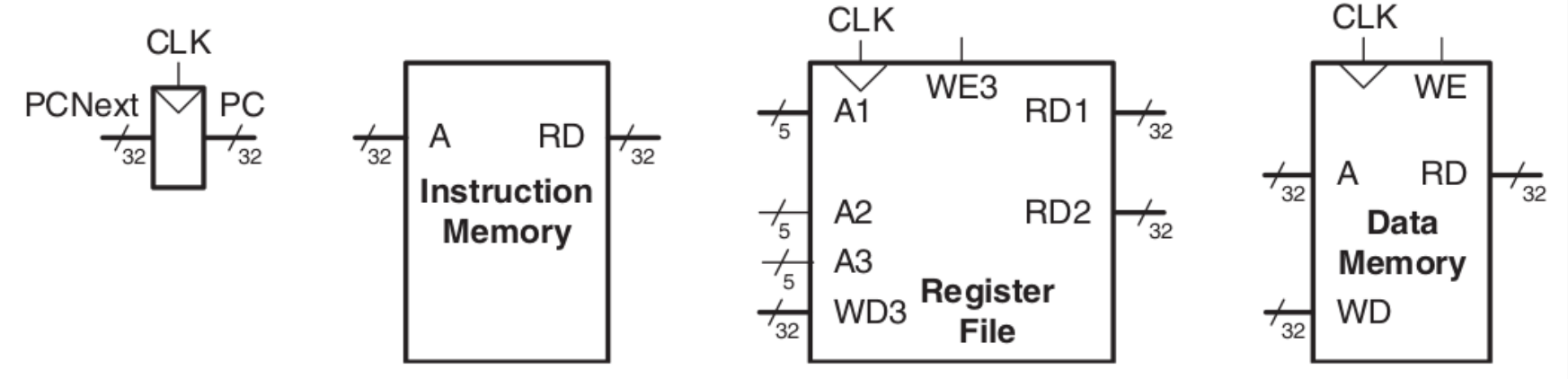
La memoria de instrucciones toma una dirección **A** de 32 bits y vuelca el valor de 32 bits que se encuentra en esa posición por la salida **RD**.

Archivo de registros



El archivo de registros contiene los 32 registros `x0`–`x31` y tiene dos puertos (salidas de datos) de lectura (**RD1** y **RD2**) que vuelcan el valor de los registros en la posiciones indicadas por las entradas **A1** y **A2**. También cuenta con un tercer puerto de escritura (entrada de datos) **WD3** que escribe el dato recibido en la posición indicada por **A3** durante el flanco ascendente de reloj si la señal de control **WE3** se encuentra alta.

Memoria de datos



La memoria de datos toma una dirección **A** de 32 bits y lee el valor de 32 bits que se encuentra en esa posición por la salida **RD** si el valor de la señal de control **WE** se encuentra bajo o escribe el contenido que ingresa por **WD** durante el flanco ascendente del ciclo de clock si se encuentra alto.

Procesador de ciclo simple

Procesador de ciclo simple

Vamos a estudiar una microarquitectura donde **las operaciones se completan durante un único cilo de reloj**, por lo que la duración del ciclo debe ser suficientemente larga como para permitir completar la operación más costosa (las que toma más tiempo). Esto significa que el rendimiento del procesador no será óptimo pero resulta conveniente como ejemplo introductorio a las microarquitecturas.

Programa de ejemplo

Utilizaremos el siguiente programa de ejemplo para justificar la interacción entre el **datapath** y la **unidad de control** e iremos conectando los elementos de memoria y agregando elementos y señales de control a medida que haga falta.

Programa de ejemplo

Address	Instruction	Type	Fields						Machine Language
			$imm_{11:0}$		rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100		01001	010	00110	0000011	FFC4A303
			$imm_{11:5}$	rs2	rs1	f3	$imm_{4:0}$	op	
0x1004	sw x6, 8(x9)	S	0000000	00110	01001	010	01000	0100011	0064A423
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233
			$imm_{12,10:5}$	rs2	rs1	f3	$imm_{4:1,11}$	op	
0x100C	beq x4, x4, L7	B	1111111	00100	00100	000	10101	1100011	FE420AE3

Aquí podemos ver la posición de memoria en la que se encuentran las instrucciones codificadas, sus mnemónicos y la división de los distintas partes de cada palabra de 32 bits según su interpretación para la arquitectura.

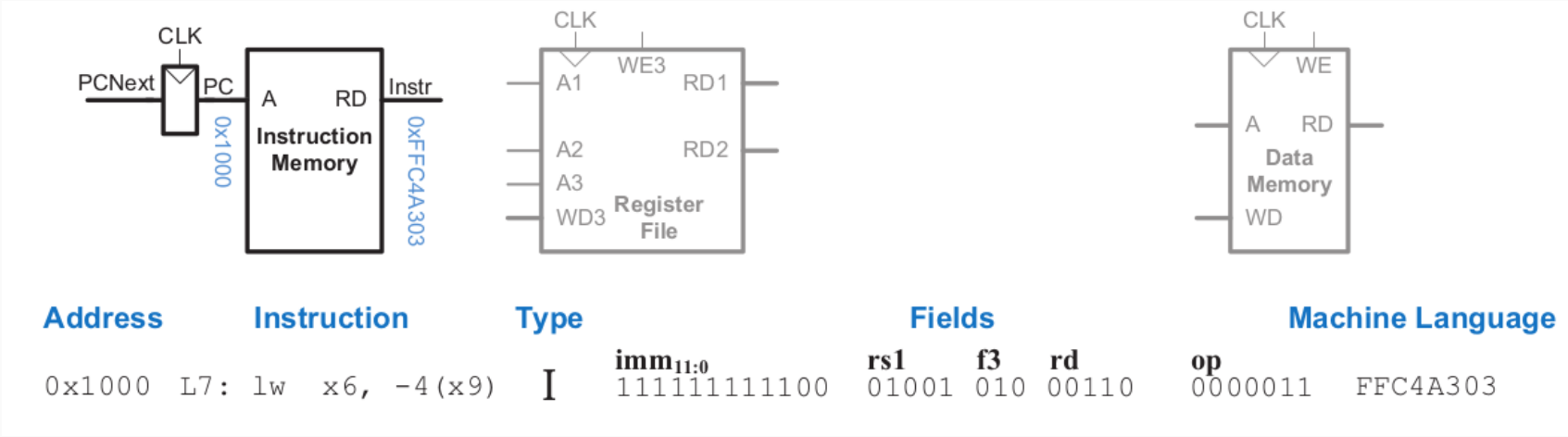
Instrucciones de memoria

Instrucciones de memoria

Address	Instruction	Type	Fields						Machine Language
			$imm_{11:0}$		rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100		01001	010	00110	0000011	FFC4A303
			$imm_{11:5}$	rs2	rs1	f3	$imm_{4:0}$	op	
0x1004	sw x6, 8(x9)	S	0000000	00110	01001	010	01000	0100011	0064A423
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233
			$imm_{12,10:5}$	rs2	rs1	f3	$imm_{4:1,11}$	op	
0x100C	beq x4, x4, L7	B	1111111	00100	00100	000	10101	1100011	FE420AE3

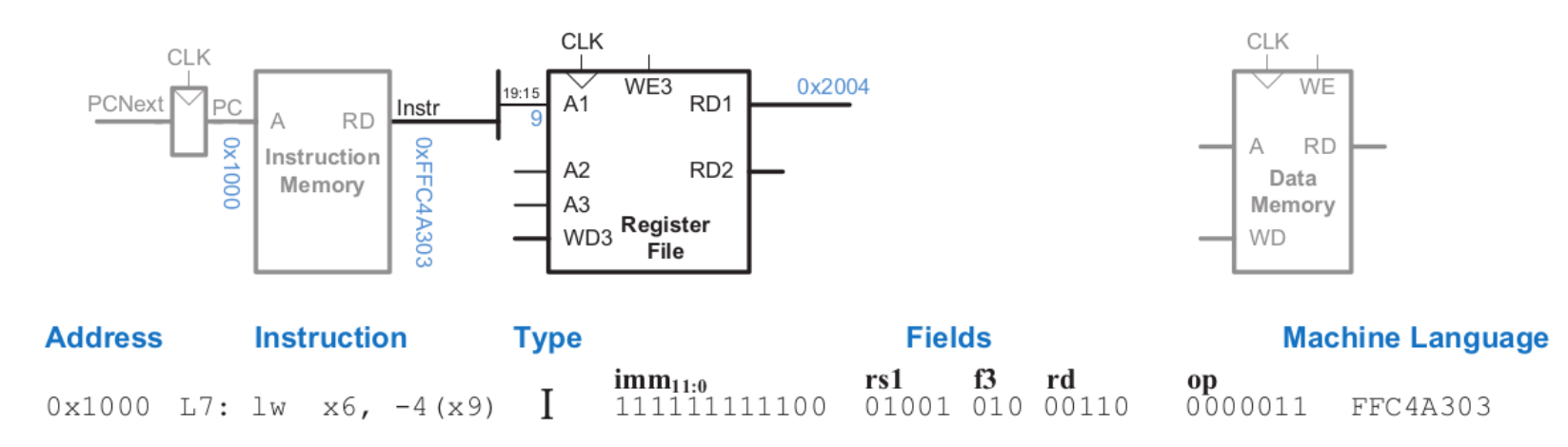
Comenzaremos estudiando los componentes involucrados con la lectura de la instrucción de memoria y la ejecución de la primera instrucción (**fetch** y **lw**).

Fetch



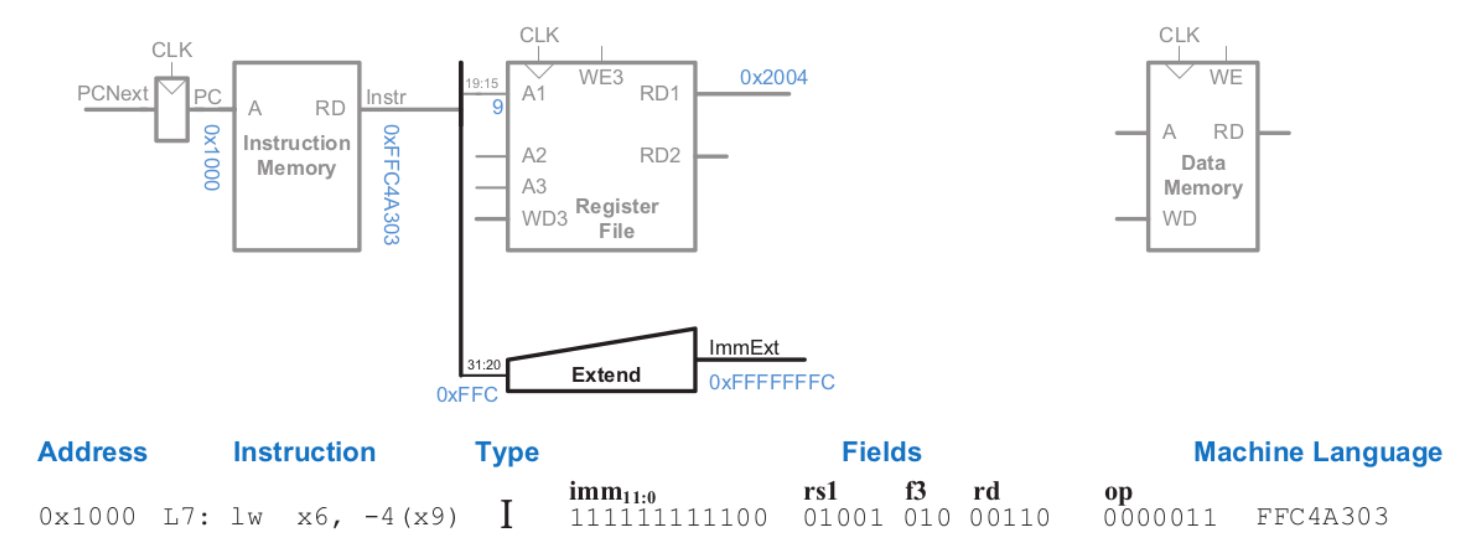
Vemos que la salida de **PC** indica la dirección **A** desde donde leer la instrucción actual de la memoria de instrucciones. La instrucción codificada es la que corresponde a una lectura de memoria (**lw**).

Lectura de memoria



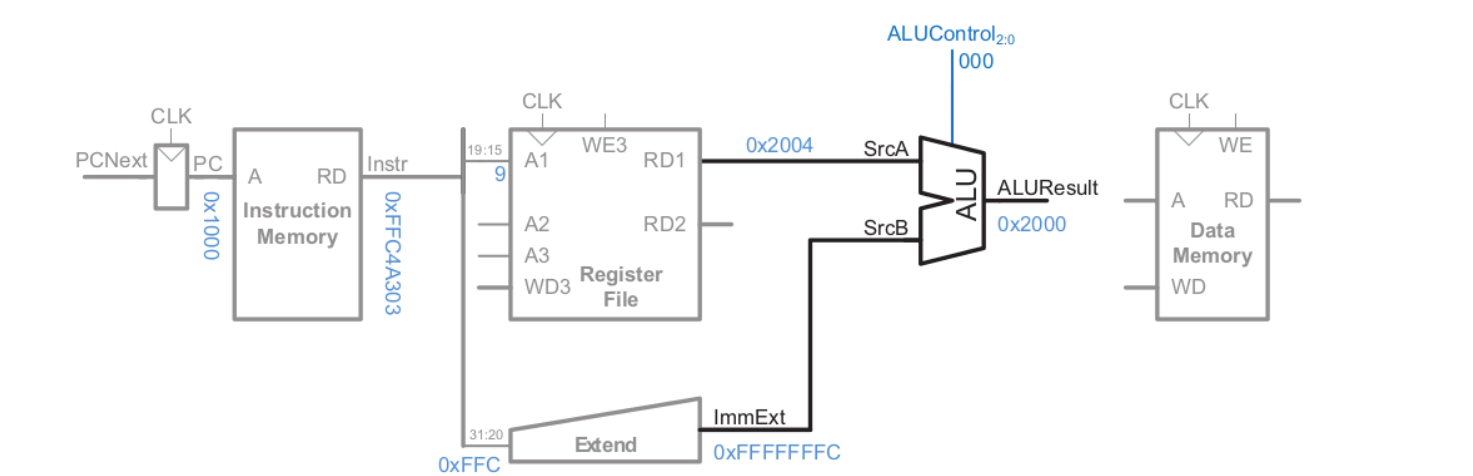
Los bits 19:15 de la instrucción indican el índice de 5 bits del operando fuente (registro) que contiene la base de la dirección a leer de memoria, por este motivo conectamos esta parte de la salida de datos de la memoria de instrucciones **RD** a la entrada de dirección de lectura **A1** del archivo de registros.

Extensión del desplazamiento



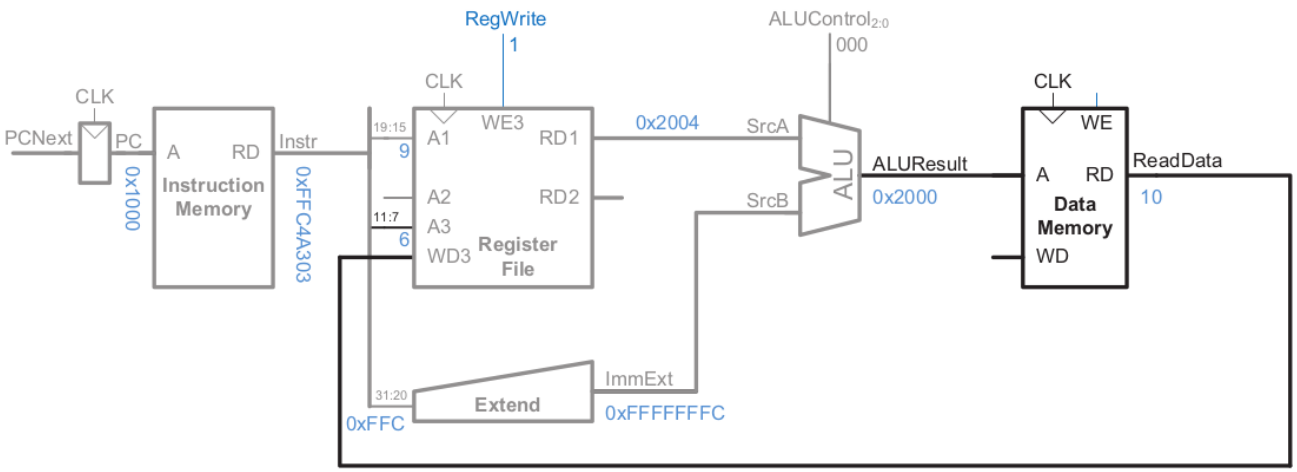
El cómputo de la dirección toma en cuenta también el desplazamiento, que en esta instrucción se codifica como los 12 bits que se encuentran en 31:20, como se trata de un valor en complemento a dos que debemos sumar a la base, es necesario extenderlo con un componente adicional.

Sumando la base



Para calcular la dirección de lectura utilizamos la ALU, ingresando base y desplazamiento como entradas e indicando que la opreación a realizar es una suma.

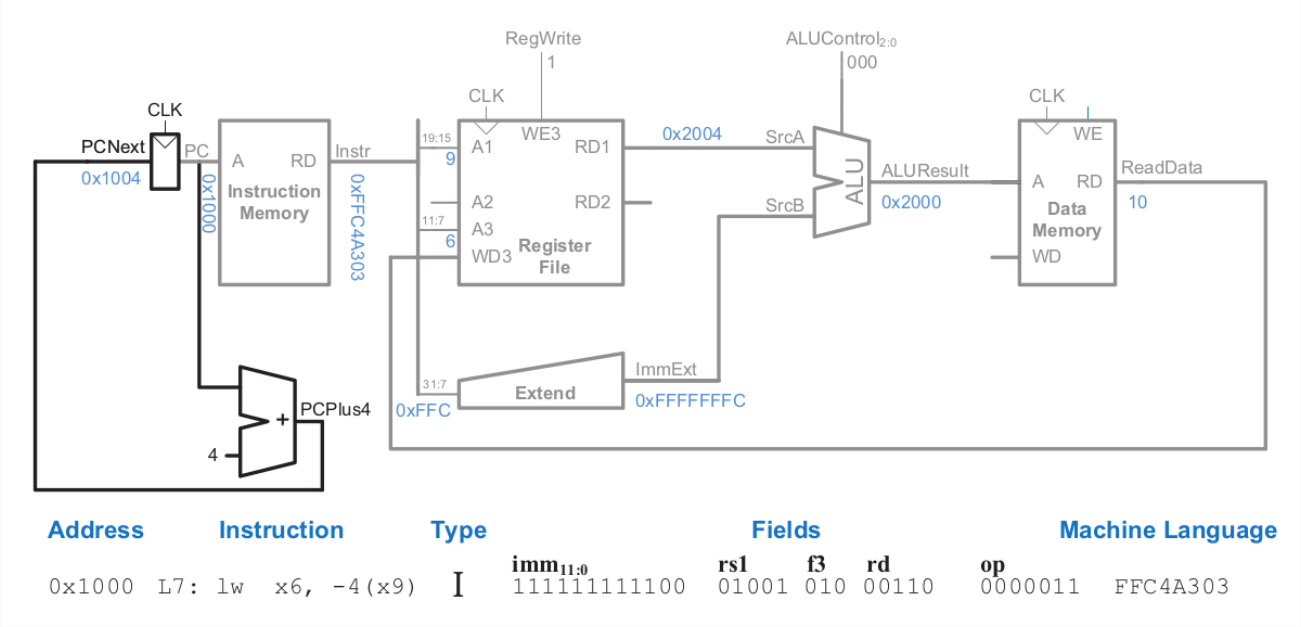
Escritura a registro



Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

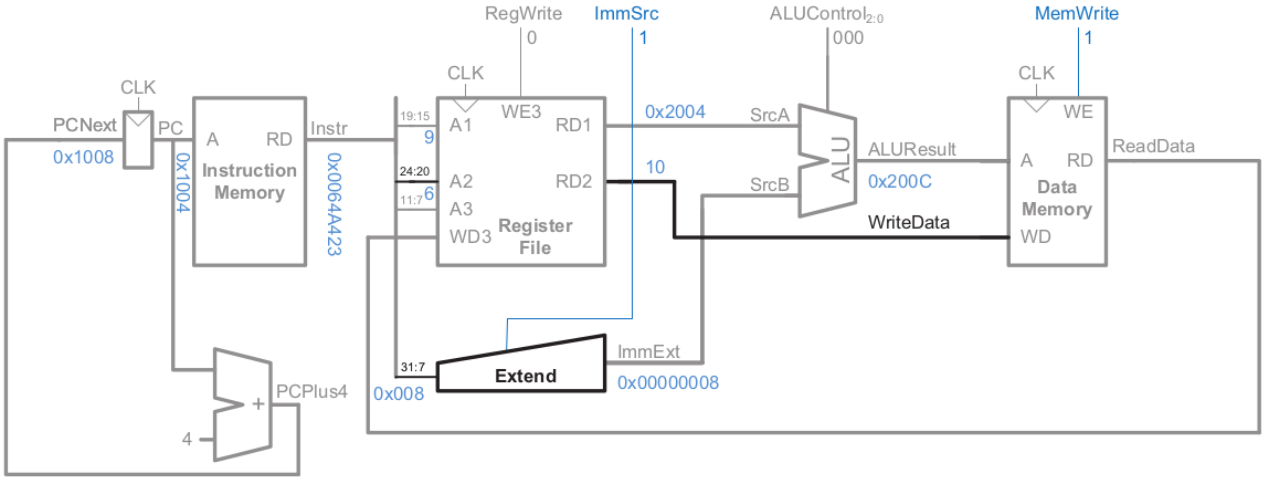
El valor resultante define la dirección **A** desde donde leer la memoria de datos, cuyo resultado **RD** es ingresado en el puerto de escritura **WD3** del archivo de registros a la vez que cargamos los bits 11:7 de la instrucción a la dirección de escritura y habilitamos la señal de control **WE3**.

Actualizando el PC



Mientras se está ejecutando esta instrucción debemos, a la par, calcular la posición desde donde leer la próxima instrucción, para esto utilizamos un sumador que incrementa en 4 el valor acual del PC y lo carga en **PCNext**.

Escritura a registro



Address	Instruction	Type	Fields						Machine Language
			$imm_{11:5}$	rs2	rs1	f3	$imm_{4:0}$	op	
0x1004	sw x6, 8(x9)	S	0000000	00110	01001	010	01000	0100011	0064A423

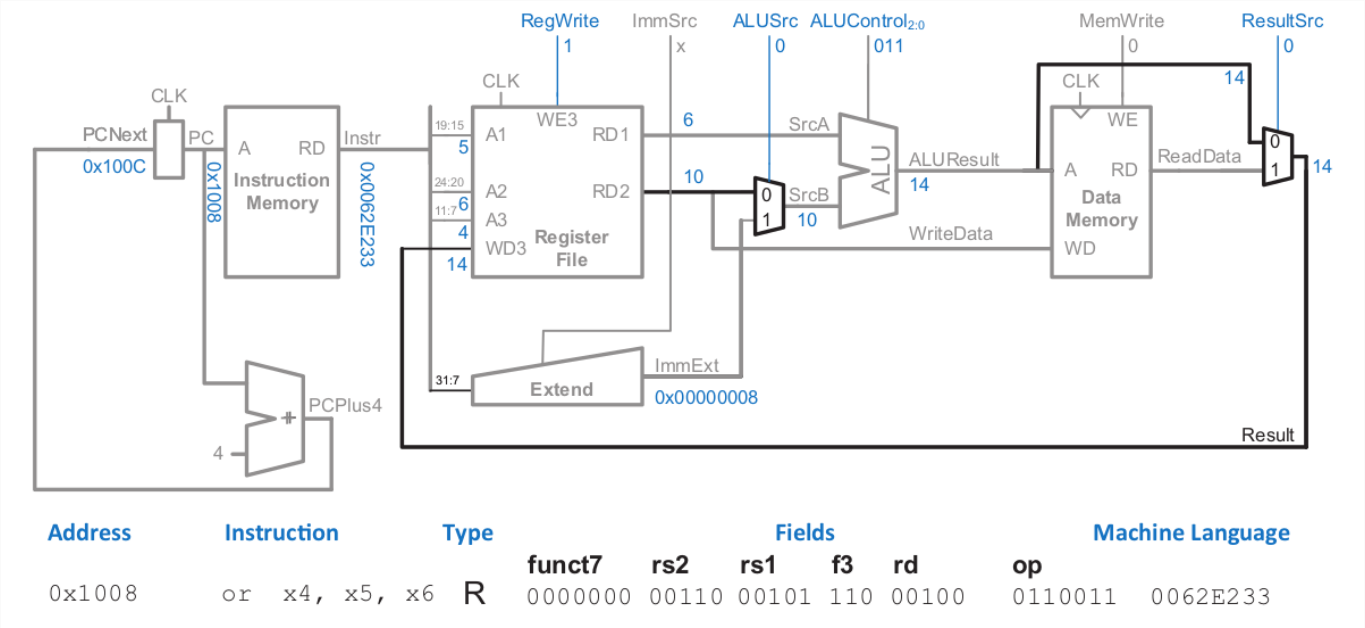
Para una escritura a memoria (**sw**), se utiliza el mismo mecanismo para determinar la dirección con una base y desplazamiento, pero se realiza una segunda lectura desde el banco de registros a través de los bits 24:20 de la instrucción indicando la posición en **A2** y asignando la salida **RD2** al puerto de escritura **WD** de la memoria de datos mientras se habilita **WE**.

Instrucciones con registros

Estructura de las instrucciones R

Las instrucciones con registros van a respetar un mismo esquema, donde tendremos dos registros de fuente, uno de destino y donde vamos a utilizar a la ALU para distintas operaciones según la semántica de cada caso.

Datapath actualizado



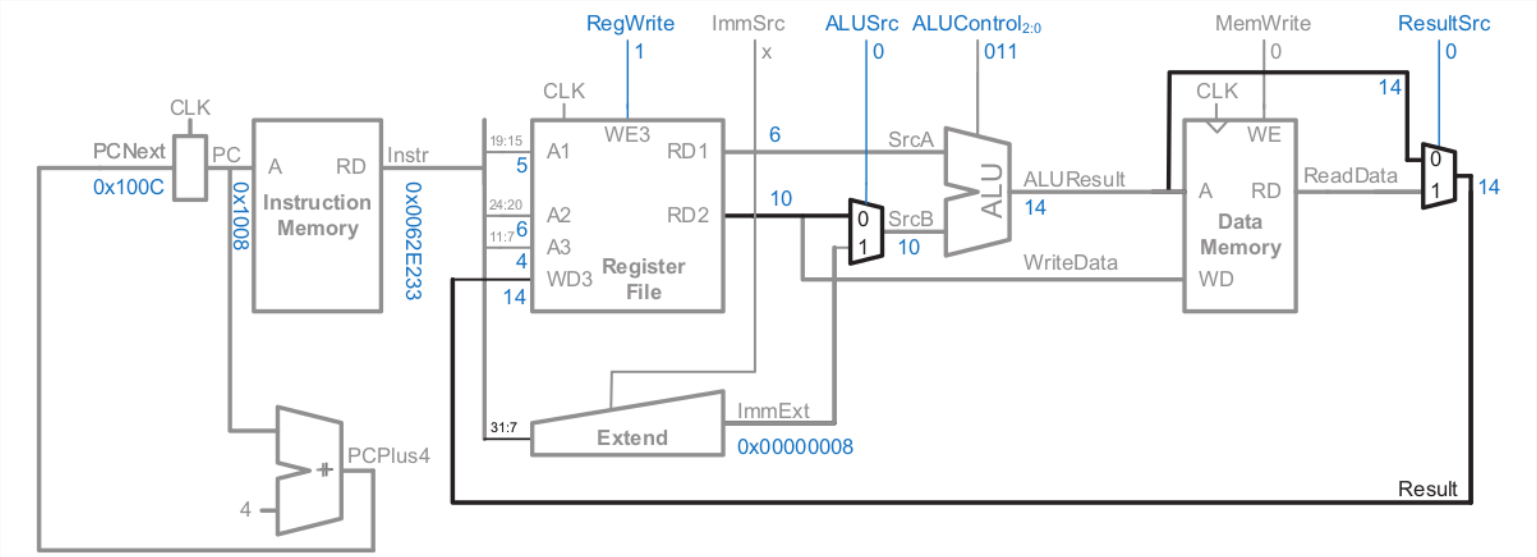
Address	Instruction	Type	Fields					Machine Language
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233

Agregamos dos multiplexores, uno para permitir usar el segundo puerto de lectura del archivo de registros **RD2** como segundo operando de la ALU, y otro para permitir usar la salida de la ALU como dato a escribir en el puerto de escritura del archivo de registros **WD3**. La entrada de operación de la ALU determina la semántica de la instrucción.

Diseño incremental

Podemos notar que el diseño de esta microarquitectura realiza las operaciones necesarias para computar todas la instrucciones presentadas hasta ahora y permite decidir cuáles salidas actualizan el estado del procesador en base a las señales de control de las memorias, el extensor de signo, la ALU y los multiplexores.

Datapath actualizado



Address	Instruction	Type	Fields					Machine Language
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233

Observemos las señales de control: **RegWrite, ImmSrc, ALUSrc, ALUControl, MemWrite, ResultSrc**. El manejo de estas señales será la responsabilidad de la **unidad de control**.

Instrucciones de salto

Salto condicional

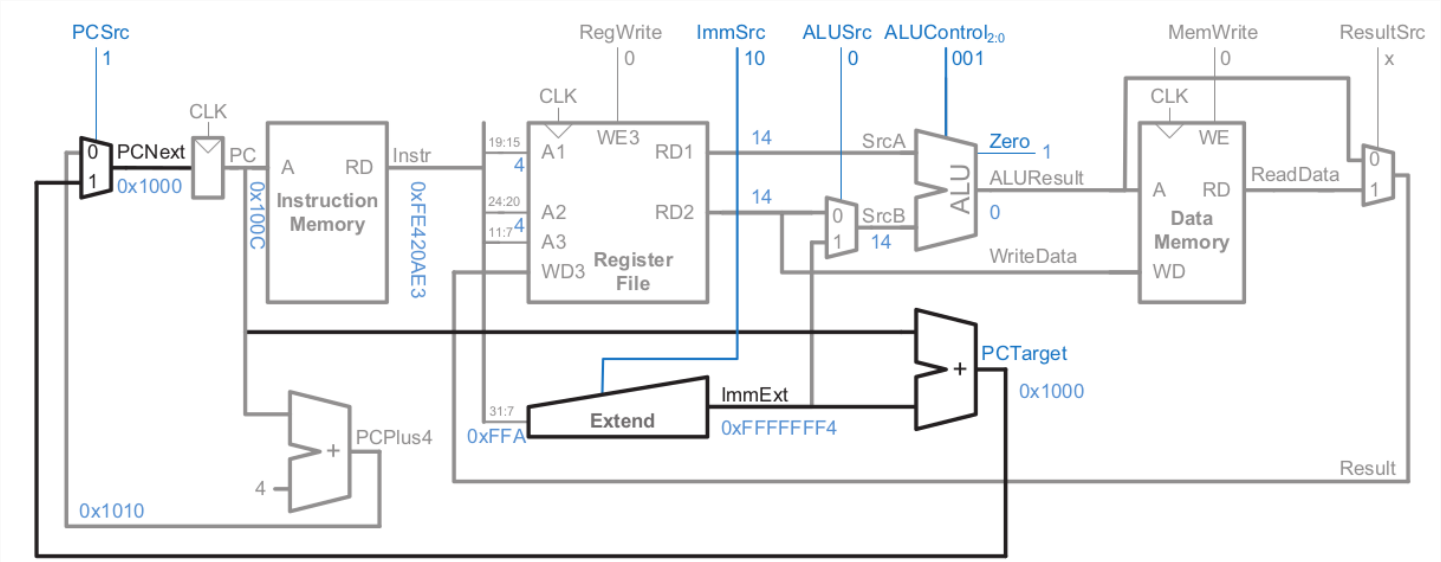
Las instrucciones de salto condicional definen un desplazamiento con respecto al PC de 13 bits codificado en 12 bits, donde el último bit se supone siempre en cero, es por esto que el extensor de signo debe tratar este caso por separado, con lo que su entrada de control pasa a tener dos bits.

Extensión de signo

ImmSrc	ImmExt	Type	Description
00	{{20{*Instr*[31]}}, *Instr*[31:20]}	I	12-bit signed immediate
01	{{20{*Instr*[31]}}, *Instr*[31:25], *Instr*[11:7]}	S	12-bit signed immediate
10	{{20{*Instr*[31]}}, *Instr*[7], *Instr*[30:25], *Instr*[11:8], 1'b0}	B	13-bit signed immediate

En esta tabla vemos cómo debe intepretar y extender las entradas el extensor para cada caso según el tipo de instrucción. Esto se le indica a través de la entrada de control **ImmSrc** de dos bits.

Instrucciones de salto

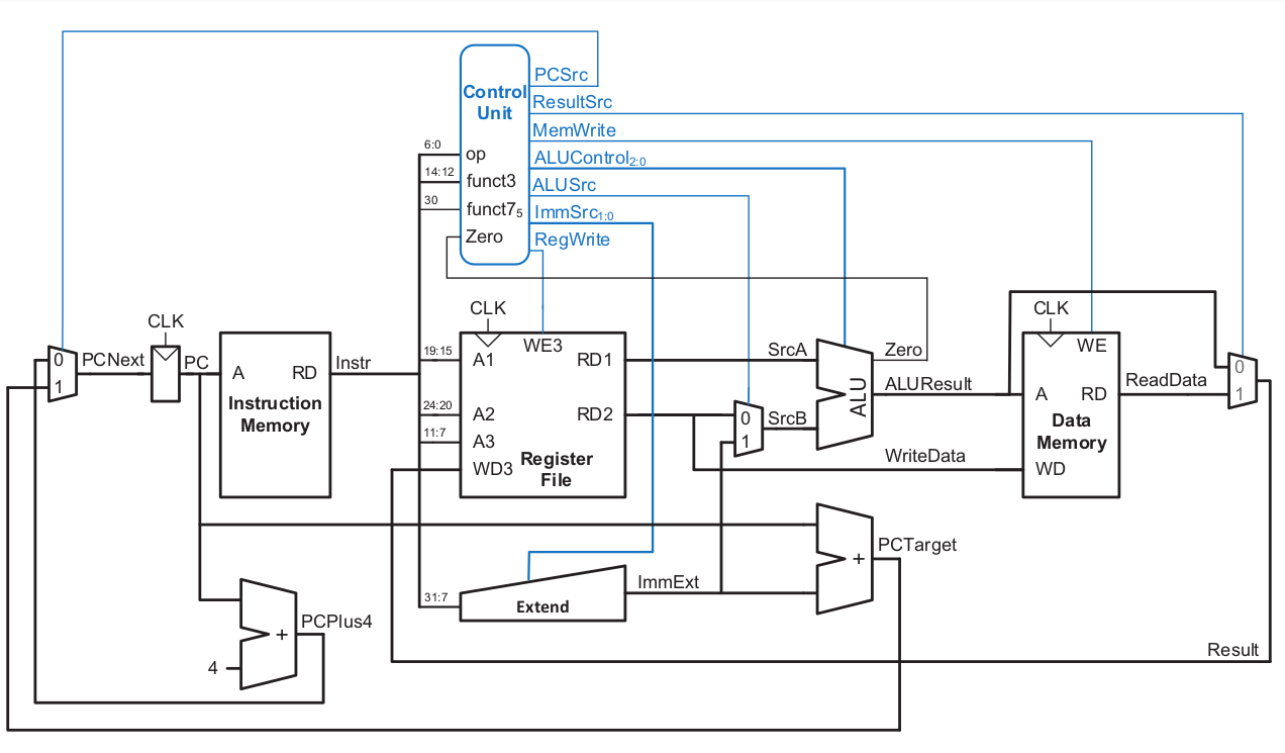


Address	Instruction	Type	Fields						Machine Language
			$imm_{12,10:5}$	rs2	rs1	f3	$imm_{4:1,11}$	op	
0x100C	beq x4, x4, L7	B	1111111	00100	00100	000	10101	1100011	FE420AE3

Agregando el caso necesario al extensor, un multiplexor y un sumador para actualizar la entrada de **PCNext**, podemos implementar soporte para saltos condicionales. El multiplexor selecciona la segunda entrada solamente si se cumple la condición de salto (en este caso si el flag Z está activado).

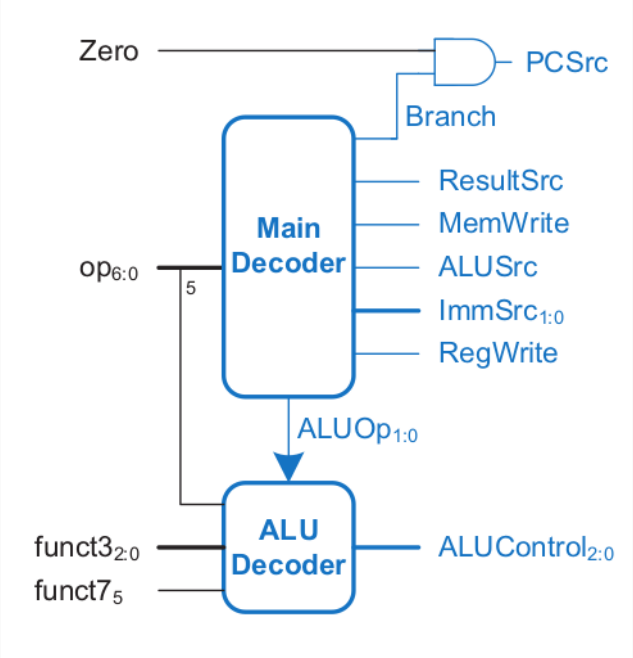
Lógica de control

Lógica de control



Aquí vemos el **datapath** y la **unidad de control** de un procesador de ciclo simple, con sus entradas, salidas y señales de control.

Unidad de control en cascada



La unidad de control se puede desacoplar de forma jerárquica entre el **controlador** y el **decodificador**, donde el decodificador decide qué operación realizar en la ALU, también se agrega una compuerta AND para decidir si se realiza el salto condicional en el caso de **beq**.

Unidad de control

Instruction	Op	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
lw	0000011	1	00	1	0	1	0	00
sw	0100011	0	01	1	1	x	0	00
R-type	0110011	1	xx	0	0	0	0	10
beq	1100011	0	10	0	0	x	1	01

El **controlador** debe implementar esta función en la versión desacoplada, veamos que **ALUOp** indica simplemente si se debe realizar una operación de la ALU o no, el decoder será responsable de decidir cuál operación realizar.

Unidad de control

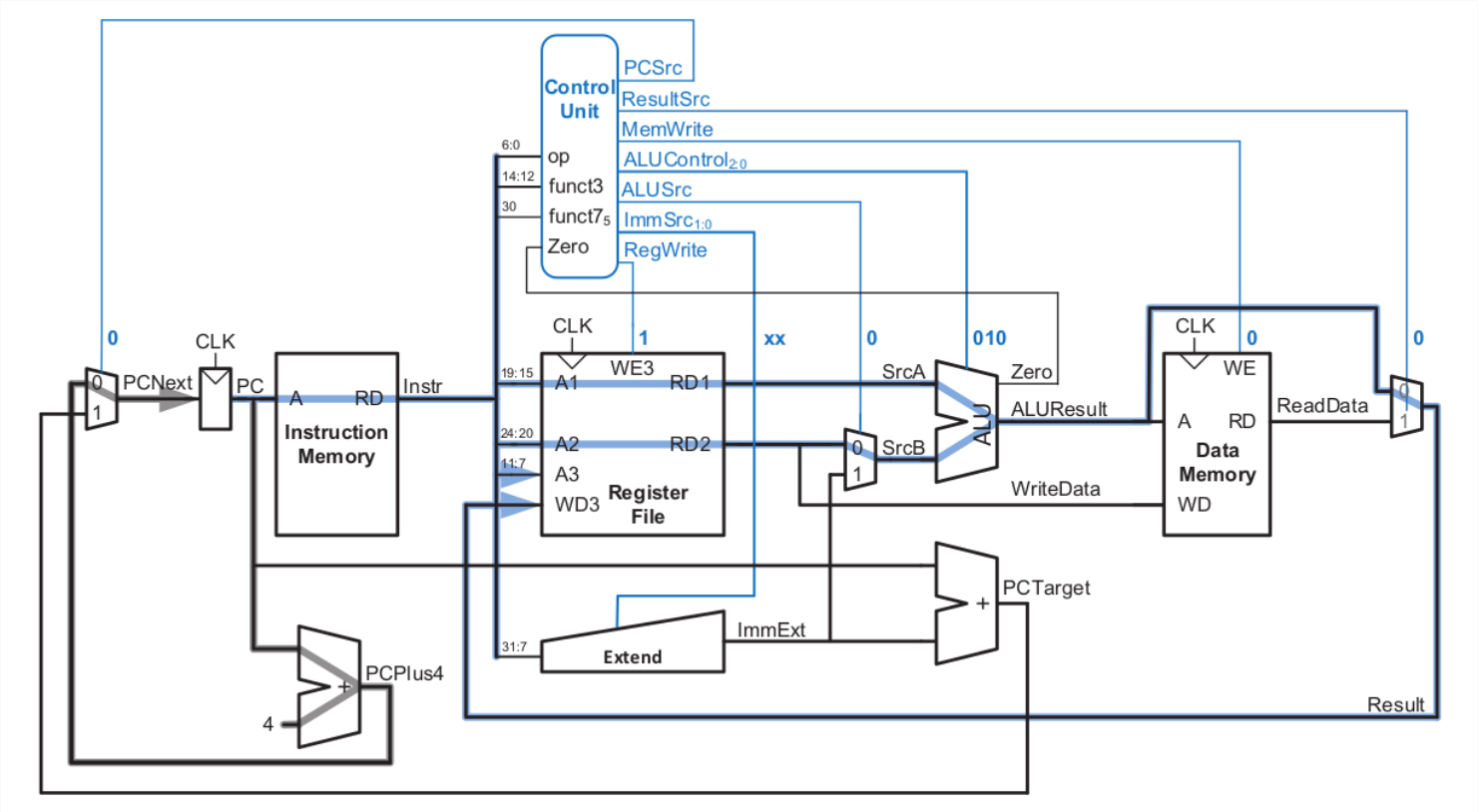
ALUOp	funct3	$\{op_5, funct7_5\}$	ALUControl	Instruction
00	x	x	000 (add)	`lw, sw`
01	x	x	001 (subtract)	`beq`
10	000	00, 01, 10	000 (add)	`add`
	000	11	001 (subtract)	`sub`
	010	x	101 (set less than)	`slt`
	110	x	011 (or)	`or`
	111	x	010 (and)	`and`

El **decodificador** debe implementar esta función en la versión desacoplada.

Implementando ambas unidades

Con las técnicas vistas para los circuitos combinatorios podemos implementar tanto el **controlador** como el **decodificador**.

Ejemplo de ejecución (and)



Add inmediate

Instruction	Opcode	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
lw	0000011	1	00	1	0	1	0	00
sw	0100011	0	01	1	1	x	0	00
R-type	0110011	1	xx	0	0	0	0	10
beq	1100011	0	10	0	0	x	1	01
addi	**0010011**	**1**	**00**	**1**	**0**	**0**	**0**	**10**

Si quisiéramos agregar soporte para una suma con inmediato, sería suficiente agregar la siguiente línea a la función del **controlador**.

Cierre

Repaso

Hoy vimos:

- Definición de **microarquitecturas**.

Repaso

Hoy vimos:

- Definición de **microarquitecturas**.
- Estado de arquitectura, elementos de memoria, datapath y unidad de control.

Repaso

Hoy vimos:

- Definición de **microarquitecturas**.
- Estado de arquitectura, elementos de memoria, datapath y unidad de control.
- Procesador de ciclo simple.

Repaso

Hoy vimos:

- Definición de **microarquitecturas**.
- Estado de arquitectura, elementos de memoria, datapath y unidad de control.
- Procesador de ciclo simple.
- Instrucciones de memoria, registros y saltos condicionales.

Fin