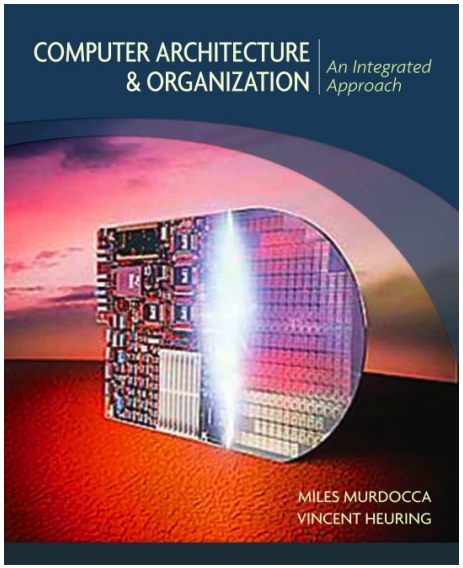
Organización de las Computadoras

Leonardo Giovanini



Microarquitectura
Monociclo

Contenidos

3.1 ISA y Microarquitectura

3.2 El ISA RISC V

3.3 Implementación monociclo

ISA y microarquitectura

Niveles de abstraccion

Una computadora tienen diferentes niveles, desde el usuario a los dispositivos.

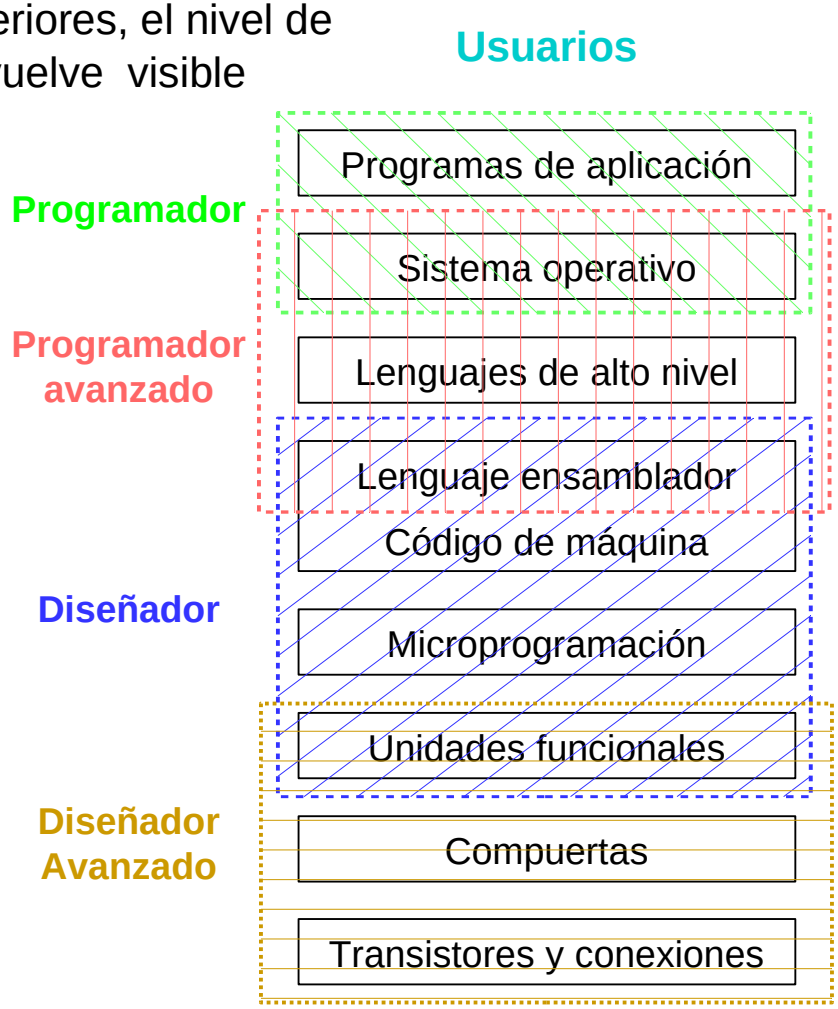
Descendiendo de los niveles superiores a los inferiores, el nivel de abstracción se reduce y la estructura interna se vuelve visible

Los ***lenguajes de alto nivel*** son independientes de la organización y arquitectura de la computadora.

Son utilizados por los programadores para desarrollar **aplicaciones y sistemas operativos**.

El ***Ensamblador*** es un lenguaje de bajo nivel que depende del procesador empleado y su uso requiere de un conocimiento profundo de la **organización y arquitectura de la computadora**.

Es utilizado por los programadores avanzados para desarrollar **software básico** (drivers y compiladores) o **aplicaciones embebidas**.



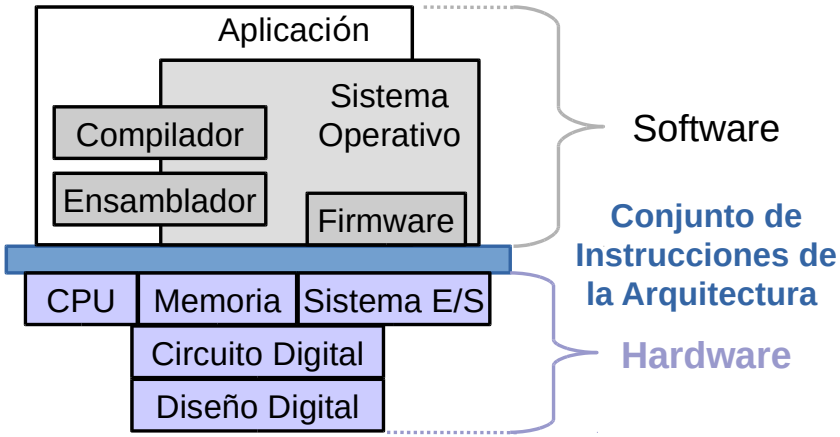
ISA y microarquitectura

Conjunto de instrucciones de la arquitectura

El **conjunto de instrucciones de la arquitectura** (ISA) es un ***modelo abstracto*** que define toda la información necesaria para ***programarla.***

Es la **interfaz entre el hardware y el software**.

Existen dos tipos:



- **ISA comerciales** desarrollados por companias comerciales que producen procesadores para aplicaciones especificas; y
- **ISA abiertos** (*open-ISA*) desarrollados por companias comerciales e instituciones académicas que producen procesadores de uso general, investigación y docencia.

Los ISA comerciales tienen los siguientes problemas cuando se los utiliza en la academia:

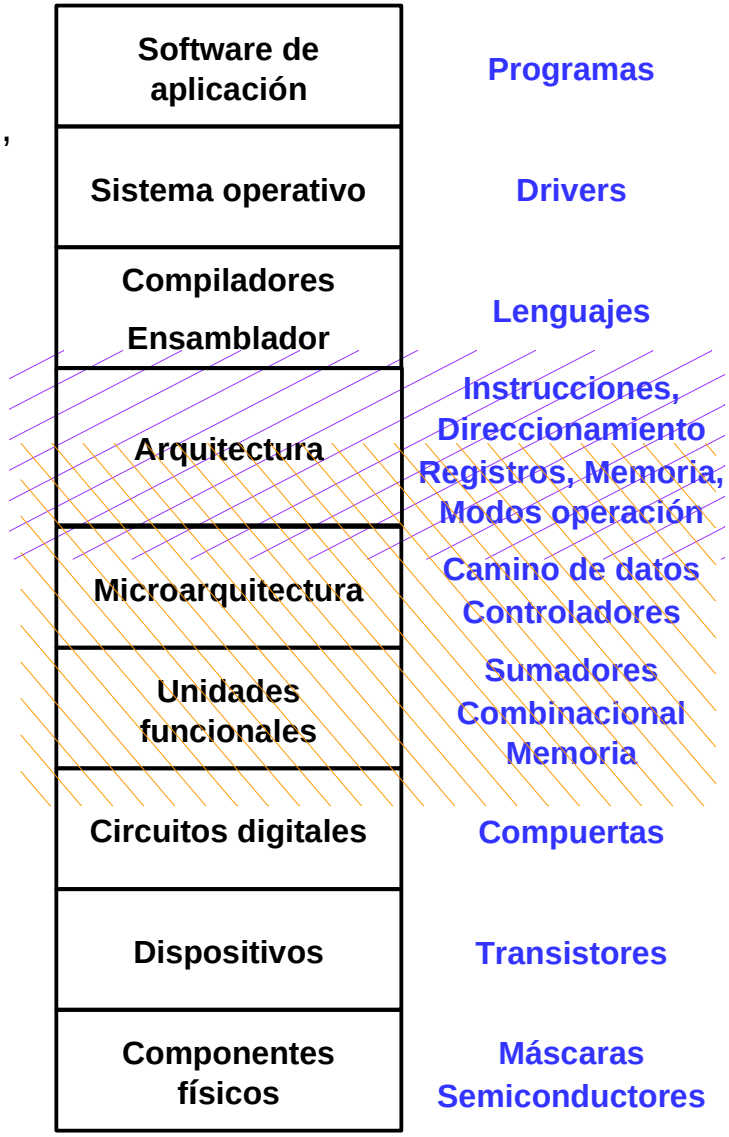
- Se enfocan en **aplicaciones muy especificos** - resuelven eficientemente una familia reducida de problemas. Por ejemplo la arquitectura ARM se enfoca en aplicaciones móviles y embebidas, mientras que la X86 de Intel se enfoca en computadoras personales y servidores;
- Son diseñados para **un conjunto especifico de instrucciones**, por lo tanto es **difícil agregarle** nuevas instrucciones y funcionalidades para mejorar su desempeño y prestaciones;
- Son **complejos de implementar en hardware** hasta el nivel de compatibilidad con el software disponible. La complejidad se debe a **malas decisiones de diseño** o al menos **desactualizadas**.

ISA y microarquitectura

Conjunto de instrucciones de la arquitectura

El ISA define

- Los **modos de operación** soportados por la CPU (usuario, supervisor, maquina virtual);
- Los **tipos de datos** soportados (bit, nibble, byte, palabra) y su formato (entero, punto flotante);
- El **conjunto de instrucciones** que puede realizar la CPU (movimientos de datos, aritmeticas, lógicas, control de flujo);
- Los **operandos** utilizados por las instrucciones y los **modos de direccionamiento** para acceder a dichos operandos;
- La **organización de la memoria** y el **modelo de entrada/salida** utilizado para gestionar los periféricos; y
- El **modelo de gestión de los eventos** ajenos al programa y la ubicación de la información relacionada.



ISA y microarquitectura

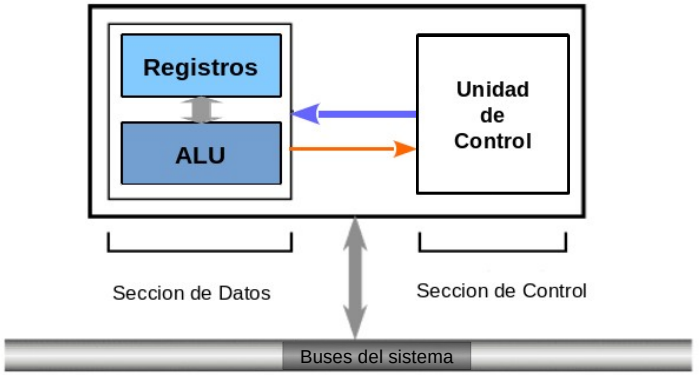
Microarquitectura del procesador

La microarquitectura **describe cómo se implementa el comportamiento descrito por el ISA** a través de un sistema digital. La microarquitectura define cómo **cada una de las señales digitales se enruta y manipula para lograr el resultado deseado**.

La microarquitectura generalmente se **representa como diagramas de bloques** que describen las interconexiones de los diversos elementos de la microarquitectura.

Estos diagramas generalmente separan la **sección de datos** y la **sección de control**.

Las unidades de ejecución (***unidades lógicas aritméticas, unidades de puntos flotantes, unidades de carga / almacenamiento y predicción de salto***, entre otras) son esenciales para la definicion de la microarquitectura, ya que realizan las operaciones del procesador.



La **elección del número de unidades de ejecución**, su **latencia** y **rendimiento** es una tarea central en el diseño de microarquitectura.

Las decisiones de diseño a nivel del sistema, como la **inclusión o no de periféricos**, pueden considerarse parte del proceso de diseño de la microarquitectura. Esto incluye decisiones sobre el nivel de **rendimiento y la conectividad** de estos periféricos.

Las decisiones de **diseño de la microarquitectura** afectan directamente al **desempeño general del sistema**, prestando atención a temas tales como el área del chip/ costo, el consumo de energía, la complejidad lógica, la facilidad de conectividad, la capacidad de fabricación, la depuración y la capacidad de testeo.

ISA y microarquitectura

Microarquitectura del procesador – Ciclo de instrucciones

El **ciclo de instrucción** es el proceso mediante el cual una computadora recupera una instrucción de programa de su memoria, determina qué acciones describe la instrucción y luego las realiza.

En las CPU más simples el ciclo de instrucciones se **ejecuta secuencialmente**, mientras que en las CPU modernas los ciclos de instrucciones se **ejecutan de forma concurrente**.

Esto es posible porque el ciclo se divide en **cinco operaciones básicas**:

Leer instrucción (***Fetch***)	Donde está la instrucción?	Dirección de memoria de la instrucción
Decodificar instrucción (***Decode***)	Cual es la instrucción a ejecutar?	Instrucción
Leer datos (***Operand Fetch***)	Donde están los operandos?	Dirección de memoria / Registro
Ejecutar la instrucción (**Execute**)	Cual es la función que se debe ejecutar?	Operación
Escritura de resultados (**Write Back**)	Donde escribo los resultados?	Dirección de memoria / Registro

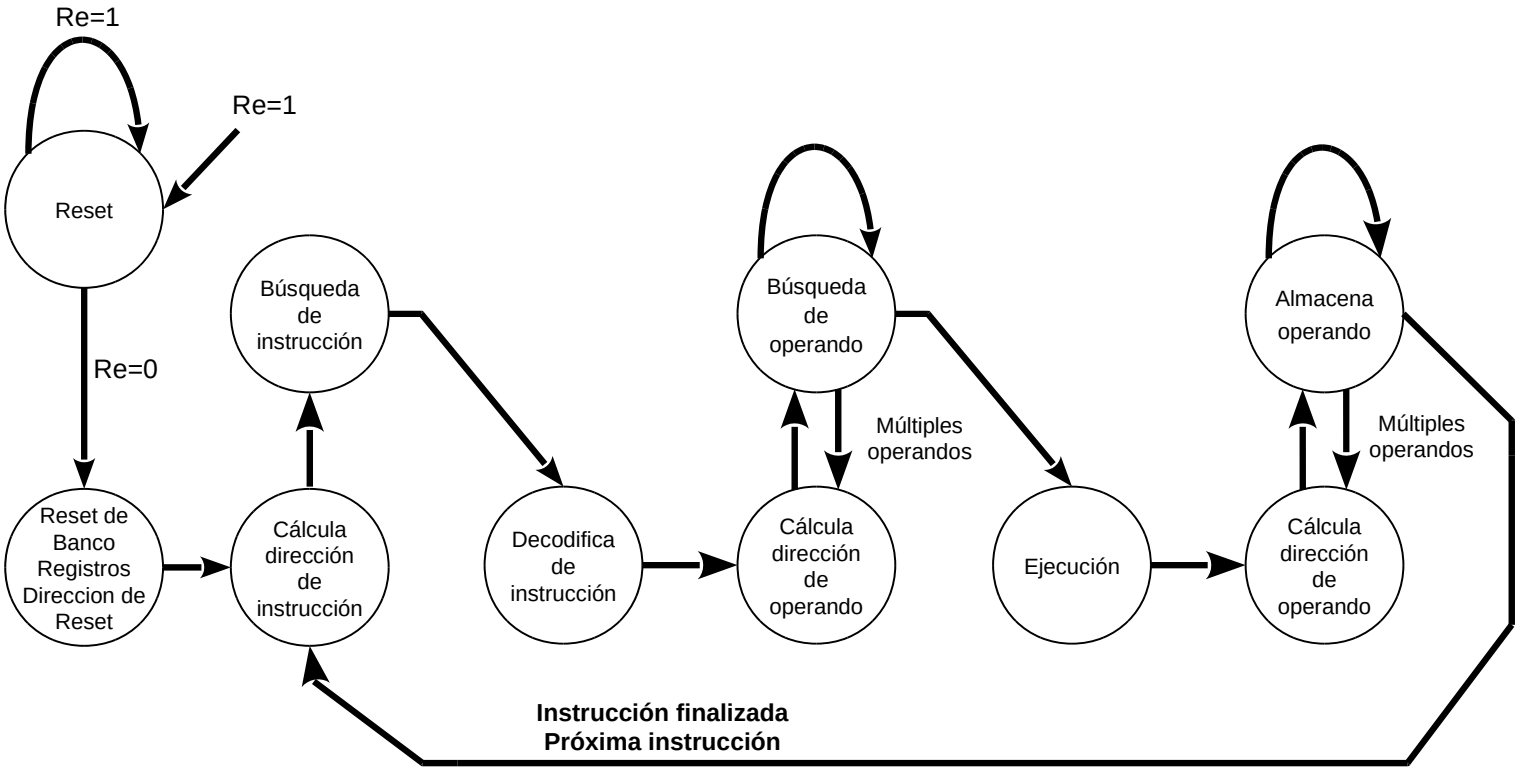
ISA y microarquitectura

Microarquitectura del procesador – Ciclo de instrucciones

Cuando se inicia la ejecucion de un programa, la computadora primer pasa por un proceso de inicialización (Reset) durante el cual se ***fuerza a 0*** el contenido del banco de registros y, los registros de estados y control y el contador de programa (***direccion de reset***) ***asumen los valor predeterminados*** por el diseñador.el programa.

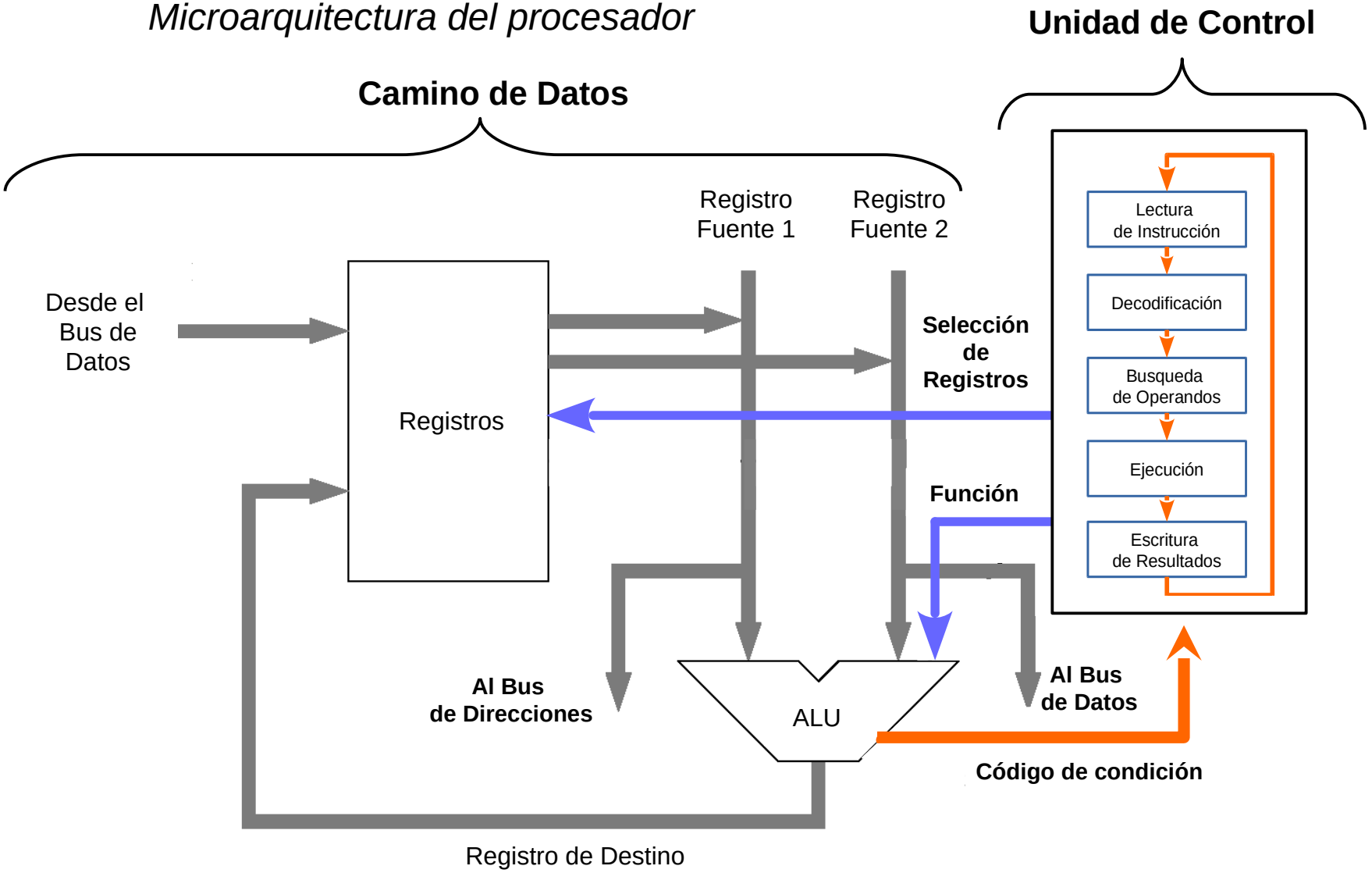
El estado de reset puede ser forzado en cualquier momento llevando a 1 la entrada de reset.

Luego, el procesador inicia el ciclo de instrucción, el cual se repite hasta que finaliza la ejecución del programa.



ISA y microarquitectura

Microarquitectura del procesador



La elección del conjunto de instrucciones de la arquitectura afecta la complejidad de la implementación de la computadora.

El ISA RISC-V

El ISA RISC-V

El ISA RISC-V se define como un ISA base de operaciones enteras que debe estar presente en cualquier implementación, más las extensiones opcionales.

El ISA base es similar al de los primeros procesadores RISC, pero no incluye **branch-delay-slots** y es compatible con **codificaciones de instrucciones de longitud variable**.

El ISA base (RV32I) está restringido a un conjunto mínimo de instrucciones suficientes para implementar compiladores, ensambladores, enlazadores y sistemas operativos, con operaciones adicionales de nivel de supervisor.

Por lo tanto proporciona un "*esqueleto*" de herramientas de software alrededor del cual se pueden construir un **ISA personalizado**.

Nombre	Descripción	Versión
RV32I	Conjunto de Instrucciones Base, 32 bits	2.0
RV32E	Conjunto de Instrucciones Base (Embebido), 32 bits	1.9
M	Extensión para multiplicaciones y divisiones enteras	2.0
A	Extensión para instrucciones atómicas	2.0
F	Extensión para operaciones en simple precisión	2.0
D	Extensión para operaciones en doble precisión	2.0
Q	Extensión para operaciones en cuadruple precisión	2.0
C	Extensión para instrucciones comprimido	2.0
B	Extensión para manipulación de bits	0.4
J	Extensión para lenguajes traducidos dinámicamente	0.0
T	Extensión para memoria transaccional	0.0
P	Extensión para instrucciones SIMD empauetadas	0.1
V	Extensión para operaciones vectoriales	0.2
N	Extensión para interrupciones a nivel usuario	1.1

El ISA RISC-V

Modos de operacion, Datos, Registros

- El RV32I soporta tres modos de operación: usuario, supervisor y máquina;

  El **modo máquina** (modo M) tiene los privilegios más altos y es el único nivel obligatorio para una plataforma de hardware RISC-V. El código ejecutado en modo M tiene acceso de bajo nivel a la implementación de la máquina.

  El **modo de usuario** (modo U), el **modo de hipersupervisor** (modo HS) el **modo de supervisor** (modo VS) están destinados a la aplicación convencional y al uso del sistema operativo, respectivamente.

- Los **tipos de datos** soportados bit, nibble, byte, palabra y doble palabra en formato entero.

- Utiliza una arquitectura de 32 registros de propositos generales de 32 bits con instrucciones de tres operandos (arquitectura *Registro-Registro*) y un esquema Load y Store para mover datos entre memoria y registros.

  El contador de programa (PC) es un ***registro de uso especifico*** que no puede ser accedido por el programador.

  Para aplicaciones embebidos con recursos limitados, se ha definido el subconjunto RV32E, que solo tiene 16 registros (x0 - x15).

31 … 0	
x0 / zero	Alambrado a cero
x1 / ra	Dirección de retorno
x2 / sp	Stack pointer
x3 / gp	Global pointer
x4 / tp	Thread pointer
x5 / t0	Temporal
x6 / t1	Temporal
x7 / t2	Temporal
x8 / s0 / fp	Saved register, frame pointer
x9 / s1	Saved register
x10 / a0	Argumento de función, valor de retorno
x11 / a1	Argumento de función, valor de retorno
x12 / a2	Argumento de función
x13 / a3	Argumento de función
x14 / a4	Argumento de función
x15 / a5	Argumento de función
x16 / a6	Argumento de función
x17 / a7	Argumento de función
x18 / s2	Saved register
x19 / s3	Saved register
x20 / s4	Saved register
x21 / s5	Saved register
x22 / s6	Saved register
x23 / s7	Saved register
x24 / s8	Saved register
x25 / s9	Saved register
x26 / s10	Saved register
x27 / s11	Saved register
x28 / t3	Temporal
x29 / t4	Temporal
x30 / t5	Temporal
x31 / t6	Temporal

32

31 … 0
pc

Registros de RV32I y sus nombres para la Interfaz Binaria de Aplicaciones

El ISA RISC-V

Modelo de memoria

- La **memoria** esta organizada de la siguiente manera;

  El **área reservada** de memoria almacena los ***registros de estados*** y ***configuración*** para los diferentes ***modos de operación*** del procesador y gestión de eventos (interrupciones y excepciones).

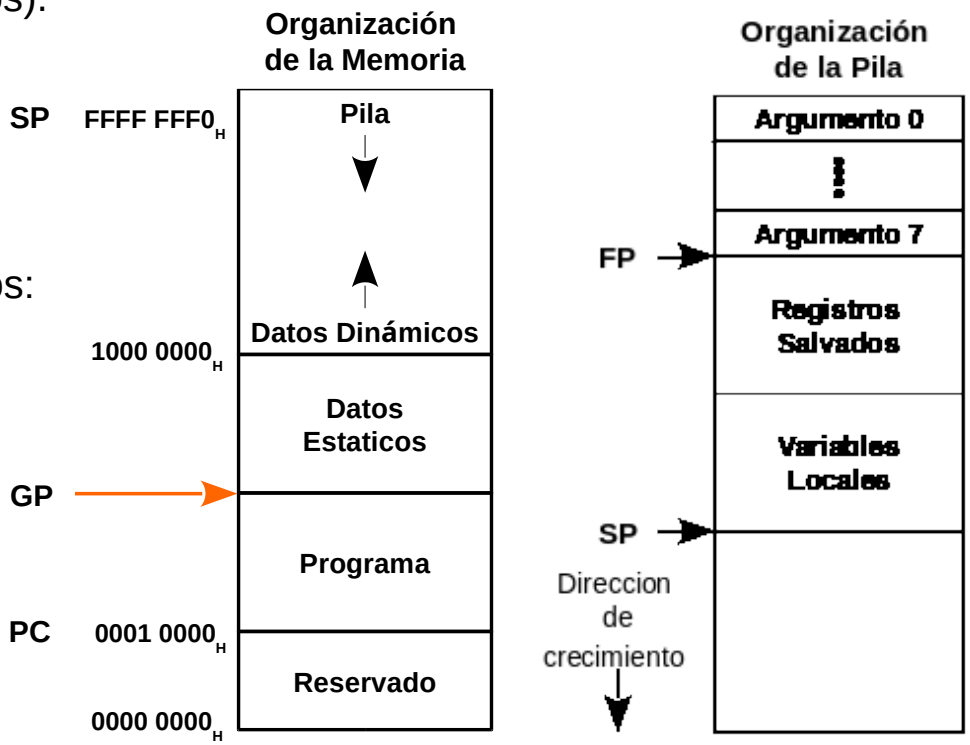
  Los periféricos son mapeados en la memoria del procesador y son gestionados a través de instrucciones load/store.

  El ISA base admite la ejecución de ***múltiples procesos*** simultáneos dentro de una dirección de usuario (areas de programa y datos estáticos).

  Cada hilo de hardware, o ***hart***, tiene su propio estado de registro de usuario y contador de programa, y se ejecuta de manera independiente.

  La pila es manejada a traves de dos punteros:

  - Un **puntero de frame** – **FP** – (x8) que apunta al primer registro de argumento guardado, y
  - Un **puntero de pila** – **SP** – (x2) que apunta a la parte superior de la pila donde se guardan los registros guardados y las variables locales residente en la memoria.



El ISA RISC-V

Modelo de memoria

Los ***registros de control y estado*** (CSR) identifican características del procesador y ayudan a medir su rendimiento.

Los CSRs que proveen ***información sobre el procesador***

- *misa* provee el ancho de la dirección del procesador e identifica cuáles extensiones de instrucciones estan incluidas en la implementación;
- *mvendorid* provee la identificacion JEDEC del proveedor del procesador;
- *marchid* provee la microarquitectura base. Combinar mvendorid con marchid identifica de manera única la microarquitectura implementada;
- *mimpid* provee la versión de la implementación de la microarquitectura base en marchid;
- *mhartid* provee el identificador entero del hart que se encuentra en ejecución.

Los CSRs que proveen ***información sobre el desempeño*** del procesador

- *mtime* contador de tiempo real de 64 bits que provee el tiempo de máquina;
- *mtimecmp* genera una interrupción cuando *mtime* iguala o excede su valor;
- *mcounteren* controla la disponibilidad de los registros monitores de rendimiento de hardware en el nivel supervisor;
- *scounteren* controla la disponibilidad de los registros monitores de rendimiento de hardware en el nivel usuario;
- Registros monitores (32 registros) de rendimiento (*mcycle*, *minstret*, *mhpmcounter3*, *… , mhpmcounter31*) que cuentan ciclos de reloj, instrucciones ejecutadas, y luego hasta 29 eventos seleccionados por el programador usando los registros *mhpmevent3, … , mhpmevent31*.

El ISA RISC-V

Gestión de eventos

RISC-V clasifica las excepciones en dos categorías:

Excepciones son errores que ocurren ***durante*** la ejecución de una instrucción. Las instrucciones **previas** a la excepción **son ejecutadas completamente**, y **ninguna** de las instrucciones **siguientes** aparentan haber **iniciado su ejecución**.

Los tipos de excepciones síncronas pueden ocurrir durante la ejecución en modo M:

- **Falla de acceso**, ocurre cuando una dirección física no soporta el tipo de acceso requerido;
- **Direccion desalineada**, ocurre cuando la dirección efectiva no es divisible por el tamaño de acceso;
- **Breakpoint** ocurre al ejecutar una instrucción **ebreak**, o una dirección o dato coincide con un **debug trigger**;
- **Llamadas al entorno**, ocurren al ejecutar una instrucción **ecall**; y
- **Instrucción ilegal** es el resultado de decodificar un opcode inválido.

Interrupciones son eventos **externos asíncronos** al flujo de instrucciones. Hay tres fuentes estándar de interrupciones:

- **Interrupciones de software:** son disparadas al escribir a un registro mapeado en memoria y son generalmente usadas por un hart para interrumpir a otro (*interrupción interprocesador*).
- **Interrupciones de temporizador:** son disparadas cuando el comparador de tiempo de un hart, el registro en memoria *mtimecmp*, iguala o excede al contador de tiempo real *mtime*.
- **Interrupciones externas:** son disparadas por un controlador de interrupciones a nivel de plataforma, al cual los dispositivos externos están conectados. **Diferentes plataformas de hardware** tienen **diferentes mapas de memoria** y demandan diferentes **características de sus controladores**, los mecanismos para detectar y gestionar estas interrupciones **difieren de plataforma en plataforma**.

El ISA RISC-V

Modos de direccionamiento

- Las instrucciones dispones de los siguientes **modos de direccionamiento** para acceder a los operandos;
    - Direccionamiento directo a registro.
    - Direccionamiento indirecto a registro con desplazamiento.
    - Direccionamiento relativo a PC con desplazamiento.
    - Direccionamiento inmediato.
    - Direccionamiento pseudodirecto.
    - Direccionamiento implícito.

Los modos de direccionamiento del RV32I no discriminan a ningún tipo de dato.

RISC-V puede imitar algunos modos de direccionamiento del x86-32. Por ejemplo, dejando el valor inmediato en cero, es igual al modo registro-indirecto.

A diferencia del x86-32, RISC-V no tiene instrucciones de stack específicas. Utilizando un registro como stack pointer, el modo de direccionamiento estándar obtiene la mayoría de los beneficios de las instrucciones push y pop sin agregarle complejidad al ISA.

Mientras que ARM-32 y MIPS-32 requieren que los datos estén alineados en memoria, RISC-V no lo exige. Accesos desalineados a veces son requeridos cuando se migra códigos antiguos. Permitir que los loads y stores pudieran acceder a memoria desalineada simplificaba el diseño general.

El ISA RISC-V

El conjunto de instrucciones

Las instrucciones disponibles en el ISA RISC-V son

- Instrucciones de **transferencia de datos;**

lui	load upper immediate	[rt] = {Imm, 16'b0}
lb	load byte	[rt] = SignExt ([Address]$_{7:0}$)
lh	load halfword	[rt] = SignExt ([Address]$_{15:0}$)
lw	load word	[rt] = [Address]
lbu	load byte unsigned	[rt] = ZeroExt ([Address]$_{7:0}$)
lhu	load halfword unsigned	[rt] = ZeroExt ([Address]$_{15:0}$)
sb	store byte	[Address]$_{7:0}$ = [rt]$_{7:0}$
sh	store halfword	[Address]$_{15:0}$ = [rt]$_{15:0}$
sw	store word	[Address] = [rt]

- Instrucciones de **control de programa;**

jalr	jump and link register	$ra = PC + 4, PC = [rs]
jal	jump and link	$ra = PC+4, PC = JTA
beq	branch if equal	if ([rs]==[rt]) PC = BTA
bne	branch if not equal	if ([rs]!=[rt]) PC = BTA
blez	branch if less than or equal to zero	if ([rs] ≤ 0) PC = BTA
bgtz	branch if greater than zero	if ([rs] > 0) PC = BTA

El ISA RISC-V

El conjunto de instrucciones

- Instrucciones **aritméticas para enteros, lógicas, rotación y desplazamiento**;

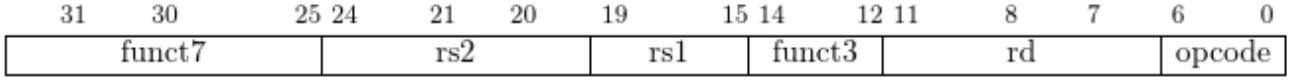
add	add	[rd] = [rs] + [rt]
addu	add unsigned	[rd] = [rs] + [rt]
sub	subtract	[rd] = [rs] − [rt]
subu	subtract unsigned	[rd] = [rs] − [rt]
and	and	[rd] = [rs] & [rt]
or	or	[rd] = [rs] | [rt]
xor	xor	[rd] = [rs] ^ [rt]
nor	nor	[rd] = ~([rs] | [rt])
slt	set less than	[rs] < [rt] ? [rd] = 1 : [rd] = 0
sltu	set less than unsigned	[rs] < [rt] ? [rd] = 1 : [rd] = 0
addi	add immediate	[rt] = [rs] + SignImm
addiu	add immediate unsigned	[rt] = [rs] + SignImm
slti	set less than immediate	[rs] < SignImm ? [rt] = 1 : [rt] = 0
sltiu	set less than immediate unsigned	[rs] < SignImm ? [rt] = 1 : [rt] = 0
andi	and immediate	[rt] = [rs] & ZeroImm
ori	or immediate	[rt] = [rs] | ZeroImm
xori	xor immediate	[rt] = [rs] ^ ZeroImm
sll	shift left logical	[rd] = [rt] << shamt
srl	shift right logical	[rd] = [rt] >> shamt
sra	shift right arithmetic	[rd] = [rt] >>> shamt
sllv	shift left logical variable	[rd] = [rt] << $[rs]_{4:0}$ assembly: sllv rd, rt, rs
srlv	shift right logical variable	[rd] = [rt] >> $[rs]_{4:0}$ assembly: srlv rd, rt, rs
srav	shift right arithmetic variable	[rd] = [rt] >>> $[rs]_{4:0}$ assembly: srav rd, rt, rs

El Conjunto de Instrucciones - Codificación

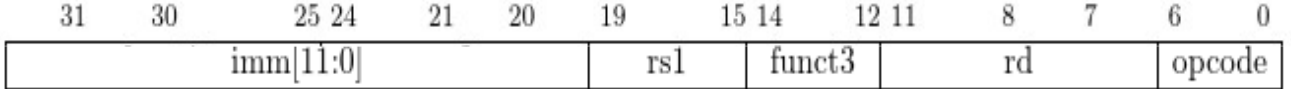
Las instrucciones se codifican utilizando el código de operación (*opcode*), las fuentes de datos (*rs1*, *rs2* e *imm*), el destino del resultado (*rd*) y la función a realizar (*funct3* y *funct7*).

Las instrucciones se codifican en seis tipos de de acuerdo con los ***operandos que utiliza*** y el ***tipo de peración que realiza***.

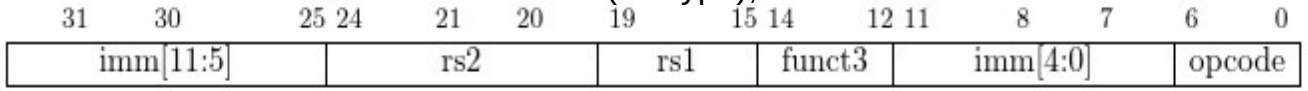
- Instrucciones que involucran **sólo registros** (R - type);



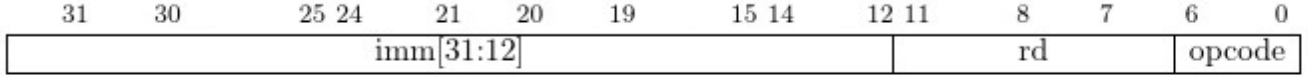
- Instrucciones que involucran **operaciones con datos inmmediatos y registros** (I - type);



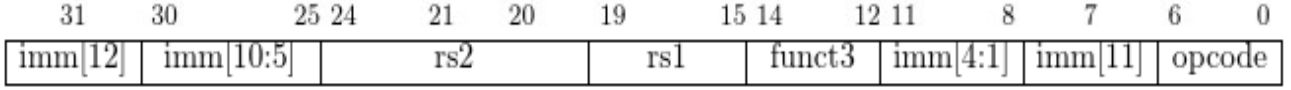
- Instrucciones que involucran **almacenamiento de datos** (S - type);



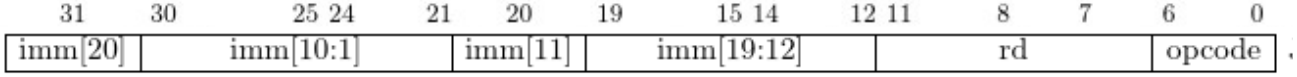
- Instrucciones con **desplazamientos en la parte superior del dato** (U - type);



- Instrucciones de **salto condicional** (B - type);



- Instrucciones de **salto incondicional** (J - type);



El ISA RISC-V

El Conjunto de Instrucciones - Codificación

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Implementación monociclo

Implementación monociclo

Arquitectura: repertorio de instrucciones, registros, modelo de excepciones, mapa de direcciones físicas y otras características comunes.

Implementación: forma en que los procesadores implementan la arquitectura a través de circuitos digitales.
La arquitectura RISC-V provee de un esqueleto basico y modulos de expansion para desarrollar procesadores para aplicaciones especificas, desacoplando la arquitectura de las implementaciones hardware específicas.

Da libertad a los diseñadores de microprocesadores para crear sus propios diseños dentro del marco de definición de la arquitectura.

Las implementaciones de las arquitectura RISC-V deben cumplir una serie de requisitos:

- Todas las instrucciones del ISA RV32I deben estar implementadas;
- El modo máquina es obligatorio, mientras que los modos supervisor y usuario son opcionales;
- La metodología de atención y gestión de interrupciones es definida por el diseñador; y
- Los implementación de las extensiones es opcional.

Implementación monociclo

Arquitectura a implementar

La arquitectura a implementar en el procesador esta definida por el ISA RV32I con las siguientes restricciones:

- El ***único modo de operación*** soportado por el procesador será el ***modo máquina***;
- Se ***omitiran*** las intrucciones de operación de los registros de control y estados;
- El procesador utilizará sólo ***32 registros de propositos generales*** con una arquitectura Load/Store, los modos de direccionamiento son los soportados por el RV32I;
- Para gestionar los eventos internos se utilizaran las excepciones definidas en el RV32I sin incluir las exepciones de ***Llamadas al entorno*** y ***Direccion desalineada***. Las direcciones de las rutinas de atención serán ***predeterminadas***.

Se utilizará una ***microarquitectura monociclo*** para implementar el procesador, empleando una ***configuración Hardvard*** para la computadora.

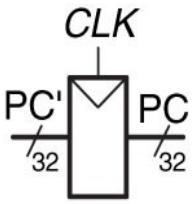
Implementación monociclo

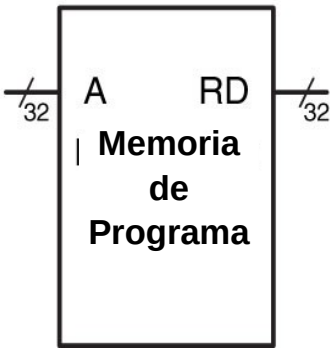
Componentes

Los componentes que se utilizarán para implementar la microarquitectura propuestas son:

- **Contador de programa**: es un registro de 32 bits que guarda la dirección de la instrucción que se está ejecutando.

  Sus entradas son la dirección de la ***próxima instrucción a ejecutar*** (PC`) y el reloj del sistema (CLK), que indica cuando actualizar la salida; mientras que la salida es la dirección de la **instrucción que se ejecuta** (PC).



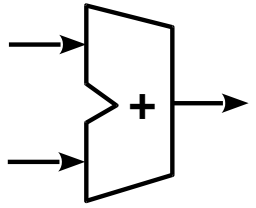


- **Memoria de programa**: es la región de la memoria del sistema donde se aloja el programa que se está ejecutando.

  Sus entradas son la dirección de memoria accedida (A) y la salida es la información (RD) que se halla en dicha dirección, la cual puede ser una instrucción o un dato. Supondremos que todas las señales son de 32 bits de ancho.

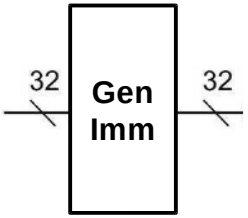
  En esta implementacion suponemos que la memoria de programa es de solo lectura (ROM), como en aplicaciones embebidas, sin embargo en computadoras personales esta memoria es de lectura/escritura (RAM) y el programa es cargado (load) por el sistema operativo

- **Sumador**: es un circuito combinacional que calcula la suma de los dos operandos de entrada, cada uno de los cuales tiene una longitud de 32 bits. Estas unidades se utilizan para calcular direcciones efectivas de saltos y datos.



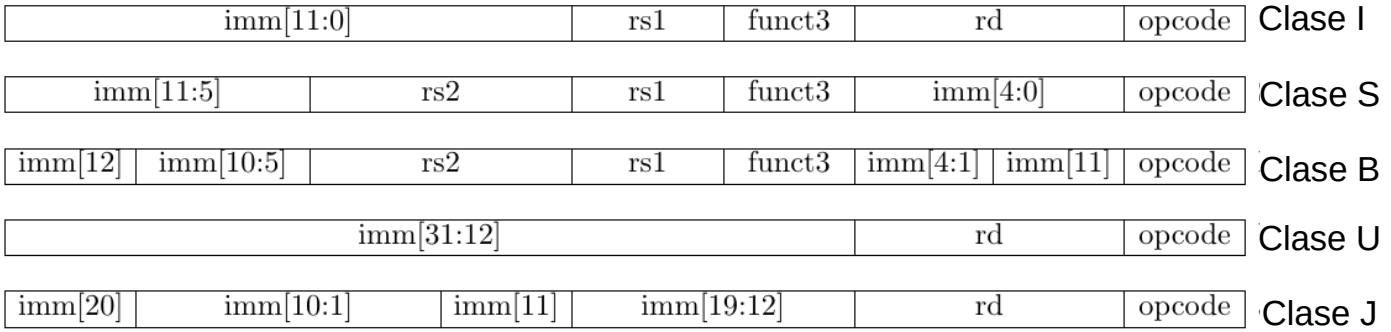
Implementación monociclo

Componentes

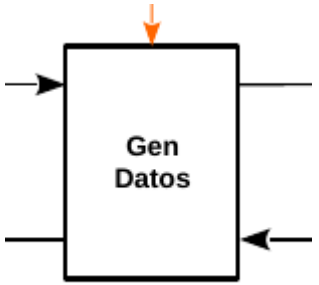


- **Unidad de generacion de datos:** es un circuito combinacional que genera el campo de dato inmediato, de 12 ó 20 bits, para las instrucciones aritméticas, lógicas, carga, almacenamiento, y saltos. Reorganiza la información del campo inmediato, de acuerdo con el código de operación, para ser utilizada por el procesador para ejecutar la instrucción correspondiente.

  Estas tareas involucran reordenar los bits, completar con 0 y desplazar el dato resultante de acuerdo al tipo de operación.

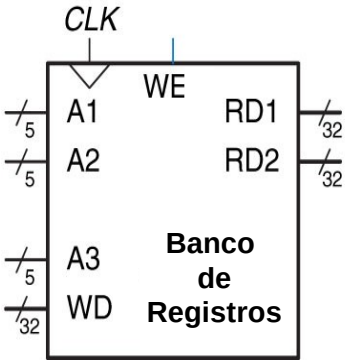


- **Unidad de generacion de datos:** es un circuito combinacional que organiza los datos para su carga y almacenamiento en memoria. Reorganiza los datos recibidos desde los registros (almacenamiento) y memoria (carga) para obtener el formato (8 bits, 16 bits ó 32 bits) y la posición (parte superior o inferior de la palabra) correspondiente a cada instrucción. El resto de las posiciones las completa con 0.



Implementación monociclo

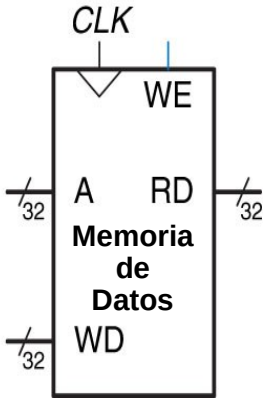
Componentes



- **Banco de registros**: El banco de registro contiene todos los registros y tiene dos puertos de lectura y uno de escritura. El banco genera el contenido de los registros correspondientes a la entradas de lectura (A1, A2) en las salidas (RD1, RD2). Una escritura debe indicarse explícitamente mediante la señal de control de escritura.

  Como las escrituras se activan por flancos, se puede leer y escribir el mismo registro dentro de un ciclo de reloj: la lectura obtendrá el valor escrito en el ciclo anterior, mientras que el valor escrito estará disponible para una lectura en el ciclo posterior.

  Las entradas que seleccionan los registros (A1, A2 y A3) tienen 5 bits de ancho, mientras que las líneas que llevan los datos (RD1, RD2 y WD3) tienen 32 bits de ancho.

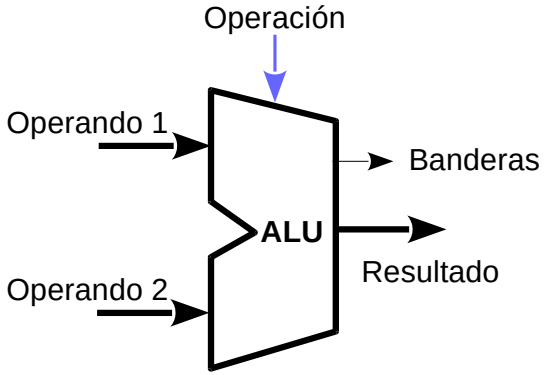


- **Memoria de datos**: es la región de la memoria del sistema donde se alojan los datos utilizados por el programa (datos dinamicos y estaticos, pila).

  Sus entradas son la dirección de memoria accedida (A), los datos que se escriben (WD), la señal de escritura (WE) y el reloj del sistema (CLK), la salida es la información (RD) que se halla en dicha dirección. Las señales RD, WD y A tienen 32 bits de ancho.

- **Unidad Aritmético Lógica**: es un circuito digital que calcula operaciones aritméticas (suma, resta, multiplicación, etc.), lógicas (AND, OR, NOT, XOR) y comparaciones para los saltos condicionales entre los operandos. .

  Sus entradas son los operandos (OP1, OP2) y la operación a realizar (ALU_Op) y las salida son el resultado (ALU_R) y las banderas de condición (F). Las señales OP1, OP2 y ALU_R tienen de 32 bits de ancho, la señal ALU_OP y F tienen 3 bits de ancho.



Implementación monociclo

Componentes

- **Unidad de Control**: es la circuitería que controla el flujo de datos a través del procesador, generando las señales necesarias para operar el camino de datos. Es una **máquina de estados finitos** cuyas tareas son leer, decodificar, ejecutar la instruccion y almacenar los resultados.

  Esta constituida por

  Un **registro de instrucción** encargado de almacenar la instrucción en ejecución;

  Un **contador de programas** (PC) que contiene la dirección de memoria de la siguiente instrucción a ejecutar;

  Un **decodificador** que interpreta la instrucción para su posterior proceso, extrayendo el código de operación de la instrucción en curso;

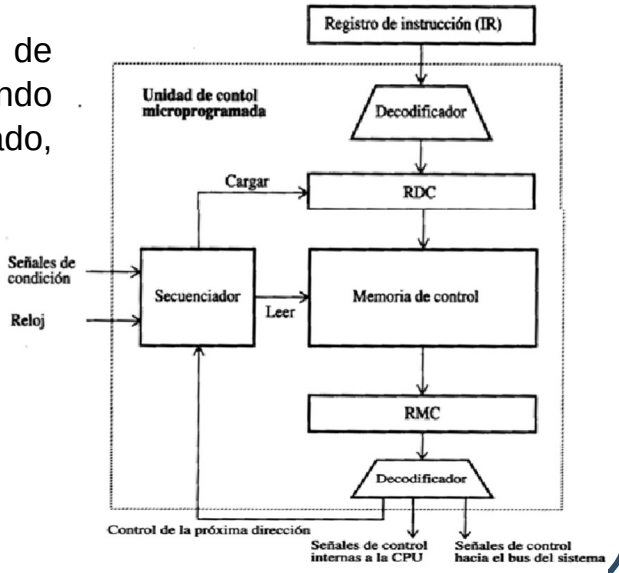
  Un **secuenciador** que genera las microinstrucciones necesarias para ejecutar la instrucción; y

  Un **reloj** que proporciona sincronización entre los componentes.

  Existen dos tipos de unidades de control

  - **Cableadas**, implementan el algoritmo a través de una máquina de estados finitos. Están basadas en una ***arquitectura fija***, requiriendo cambios en el cableado si el conjunto de instrucciones es modificado, pero su velocidad de ejecución es elevada.

  - **Microprogramadas**, implementan el algoritmo como una secuencia de microinstrucciones (microprograma) almacenados en una memoria del control especial. El algoritmo es especificado por la descripción de un diagrama de flujo. La ventaja es la simplicidad de la estructura resultante y su flexibiliad, pero su velocidad de ejecución es baja.

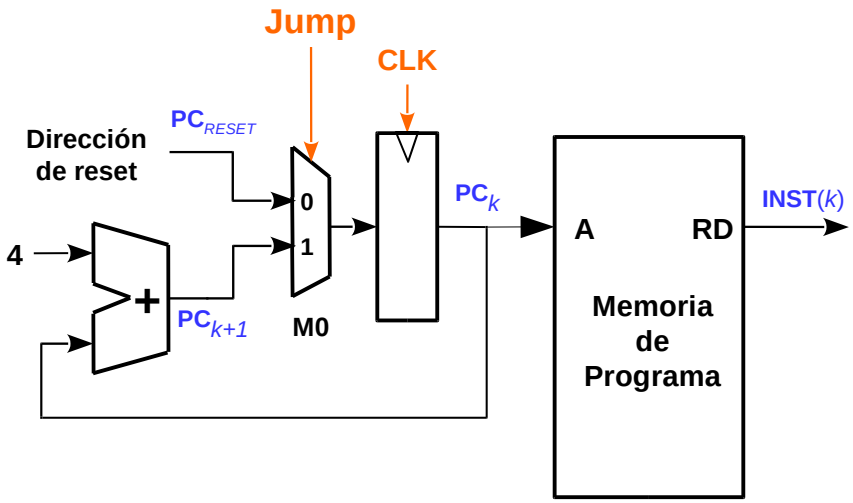


Implementación monociclo

Microarquitectura

El primer paso consiste en implementar el bloque para ***leer una instrucción desde la memoria de programa*** (**Fetch**).

Los estados son el contenido de la memoria de programa (*Instrucción*) y el contador de programas (**PC**$_k$). La memoria de programa sólo necesita proporcionar acceso de lectura porque el datapath no escribe instrucciones (Se escribe la memoria de instrucciones cuando cargamos el programa, esto no es difícil de agregar, y lo ignoramos por simplicidad). Como la memoria de programa solo lee, la tratamos como lógica combinacional: la salida **RD** en refleja el contenido de la ubicación especificada por la entrada de dirección **A** (INST(*k*)) y no es necesaria una señal de lectura.



El contador del programa (**Pc**$_k$) es un registro de 32 bits que se escribe al final de cada ciclo de reloj (**Jump=1**) y no necesita una señal de escritura.

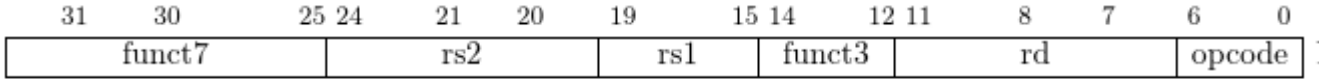
Prepara la ejecución de la siguiente instrucción, incrementando el contador del programa para que apunte a la siguiente instrucción (**PC**$_{k+1}$).

Inicializa el contador del programa a un valor preestablecido (**PC**$_{RESET}$) durante la inicialización del procesador (**Jump=0**).

Implementación monociclo

Microarquitectura – Instrucciones clase R

A continuación se procede a diseñar el datapath para implementar ***instrucciones que involucren solo registros***. Estas instrucciones ejecutan operaciones aritméticas, lógicas y desplazamientos (ADD, SUB, AND, OR, XOR, SLL, SRL y SRA). Estas instrucciones pertenecen a la ***clase R*** y tienen el siguiente formato

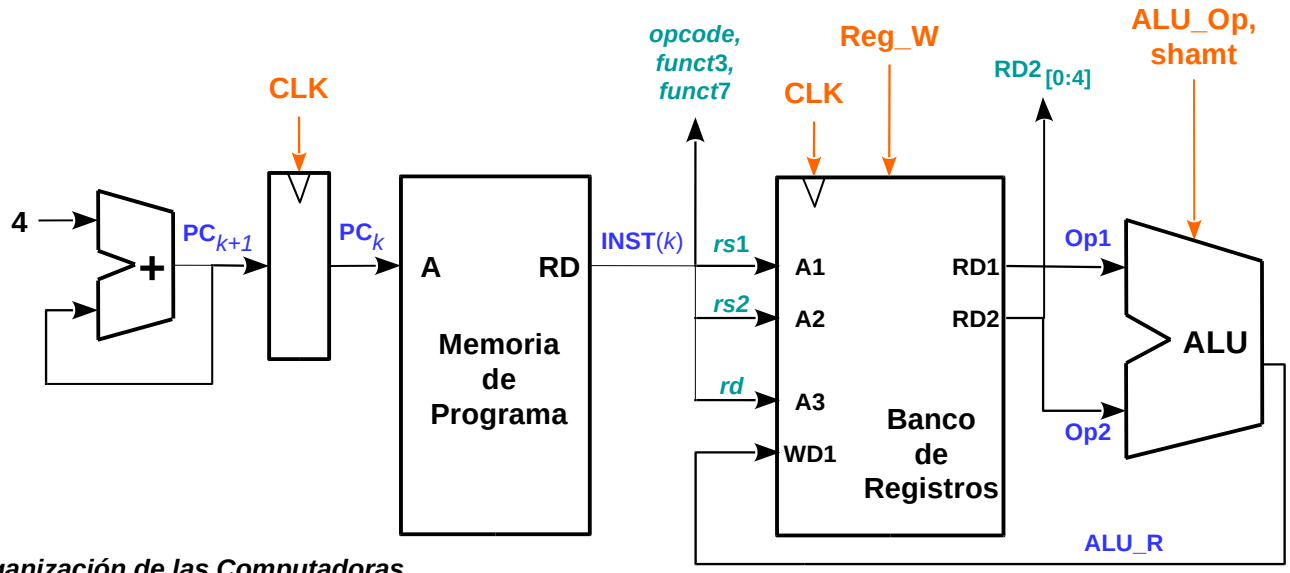


Los campos que indican los registros (**A1**=*rs*1, **A2**=*rs*2 y **A3**=*rd*) tienen 5 bits, mientras que las líneas de datos (**RD1**, **RD2** y **WD1**) tienen 32 bits. La operación a realizar por la ALU (*funct*3 y *funct*7) se controla con la señal **ALU_Op** de 4 bits.

La señal **Reg_W** indica que se escriba el dato en el registro indicado por **A3**. En el flanco de **CLK**. La operación a realizar por la ALU es controlada por la señal **ALU_Op** , la cual es generada por la unidad de control a partir de *opcode* (0110011), *funct3* y *funct7*.

Reg_W	Operación
0	Niguna
1	Escritura

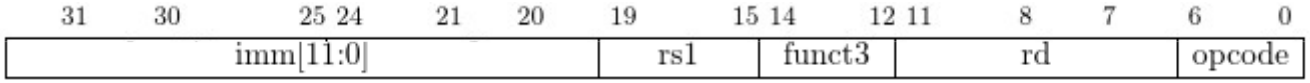
ALU_Op	Operación
0000	ALU_R = 0
0001	ADD
0010	SUB
0011	AND
0100	OR
0101	XOR
0110	SLL
0111	SRL
1000	SRA
1111	SUBS



Implementación monociclo

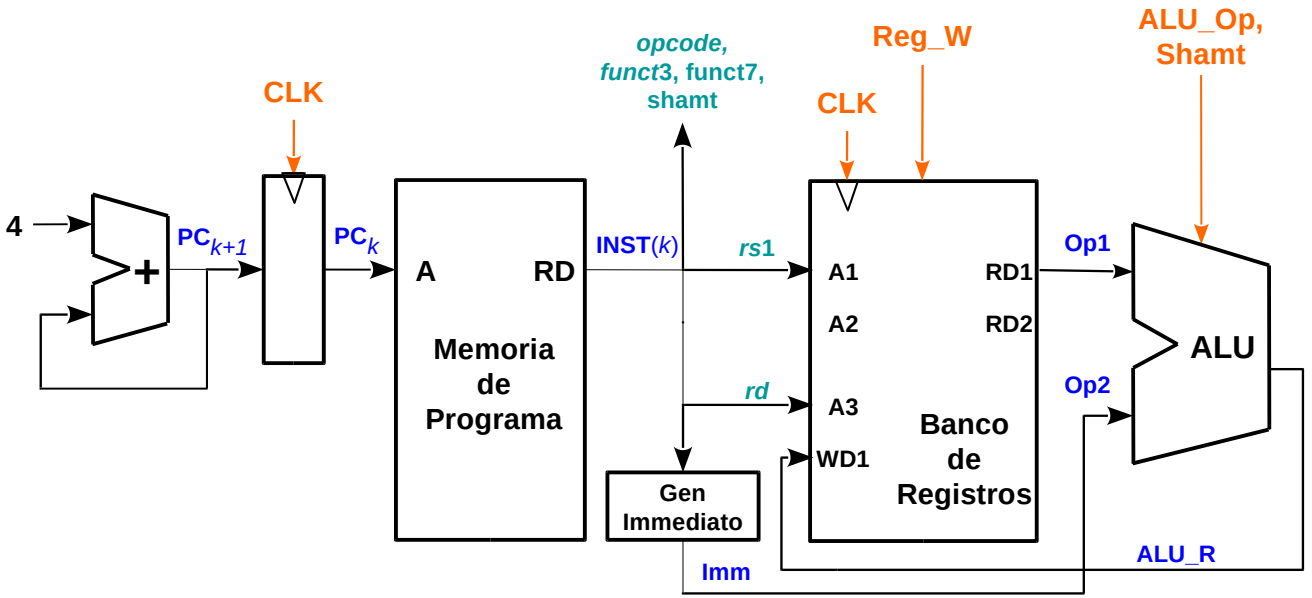
Microarquitectura – Instrucciones clase I

A continuación se procede a diseñar el datapath para implementar ***instrucciones que involucran datos inmediatos***. Estas instrucciones ejecutan operaciones aritméticas-lógicas, desplazamientos y carga de datos, pertenecen a la ***clase I*** y tienen el siguiente formato



Los campos que indican los registros (**A1**=*rs*1 y **A3**=*rd*) tienen de 5 bits y el datos inmediato 12 bits (*imm*[0:11]). La operación a realizar (*funct*3).

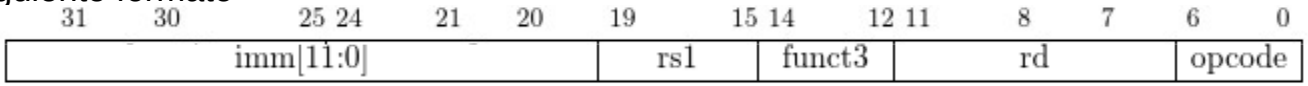
El siguiente circuito implementa las operaciones aritméticas-lógicas y desplazamientos (ADD, AND, OR, XOR, SLL, SRL y SRA) . La señal **Reg_W** indica la escritura del dato en el registro indicado por **A3**. La operación a realizar por la ALU es controlada por **ALU_Op**, la cual es generada por la unidad de control a partir de *opcode* (0110011) y *funct3*.



Implementación monociclo

Microarquitectura – Instrucciones clase I

El siguiente circuito implementa las operaciones de ***carga de datos desde memoria*** (LB, LH, LW, LBU y LHU) y tienen el siguiente formato



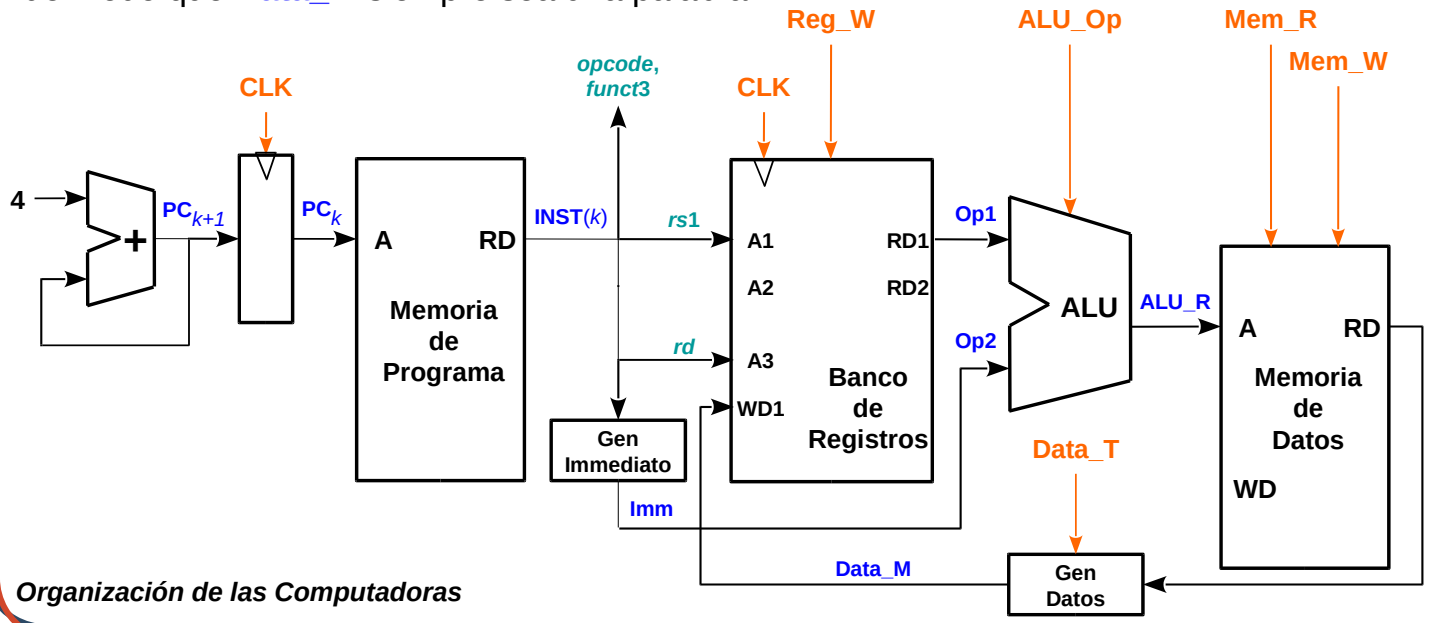
En estas instrucciones *rs*1 es utilizado como puntero, que sumado a *imm*, genera la direccion efectiva del dato buscado. El tipo de carga realizada queda definida por el valor de *funct*3.

La señal **Reg_W=1** indica la escritura del dato en **WD1** en el registro **A3** y la operación a realizar por la ALU es una suma, la cual es generada por la unidad de control a partir de *opcode* (0000011) y *funct3*. La señal **Mem_R** indica la lectura de la posicion **ALU_R**.

La señal **Data_T** indica el tipo de dato que se cargará (byte, media-palabra o palabra) y su ubicación en la palabra (parte alta o baja de la palabra) para que **Gen Datos** lo ajuste de modo que **Data_M** siempre sea una palabra.

Mem_R	Operación
0	Niguna
1	Lectura

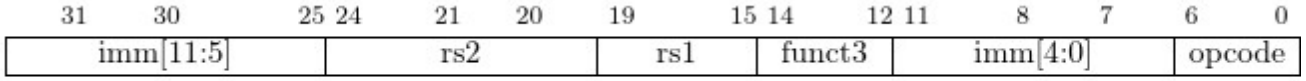
Data_T	Operación
000	
010	LB
001	LH
000	LW
011	LBU
111	LHU



Implementación monociclo

Microarquitectura – Instrucciones clase S

A continuación se procede a diseñar el datapath para implementar instrucciones de ***almacenamiento de datos en memoria*** (SB, SH y SW). Estas instrucciones pertenecen a la ***clase S*** y tienen el siguiente formato



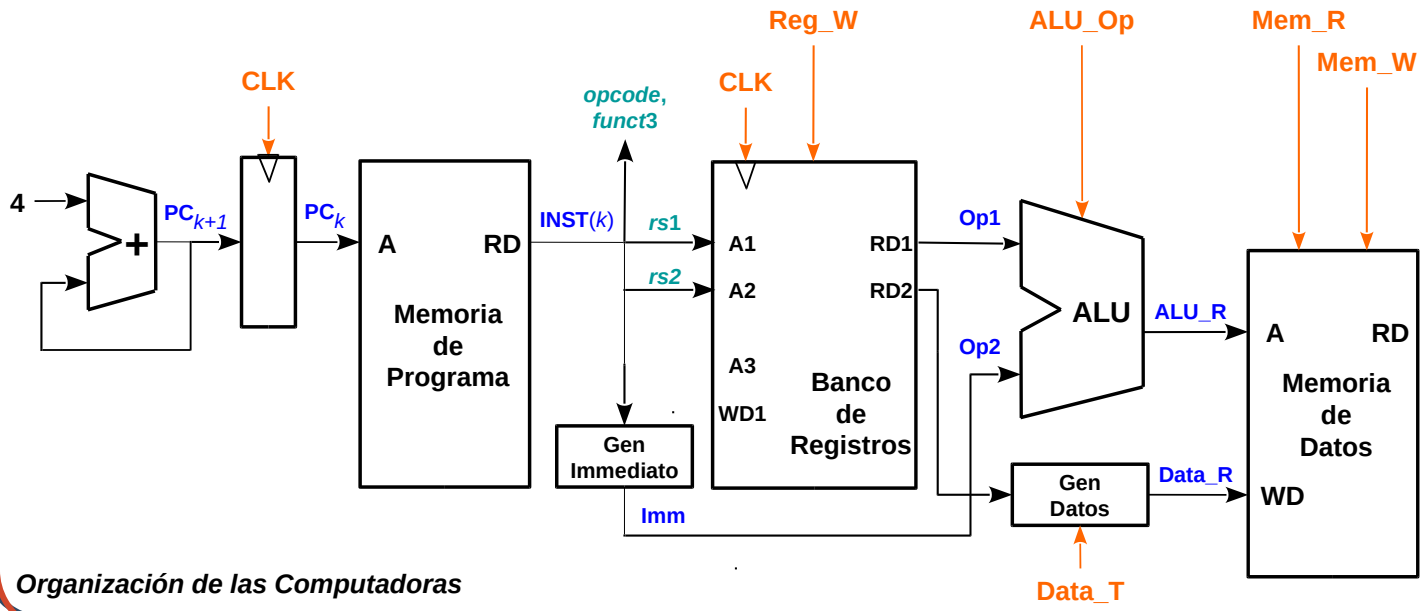
Los campos que indican los registros (A1=*rs*1 y A2=*rs*2) tienen un ancho de 5 bits e incluye el datos inmediato (*imm*[0:11]). *rs*1 define el puntero del dato e *imm*[0:11] el desplazamiento, mientras *rs*2 define el origen del dato. El tipo de carga realizada queda definida por el valor de *funct*3

La señal **Reg_W=0** indica la lectura de los registro *rs*1 y *rs*2. La operación a realizar por la ALU es una suma la cual es generada por la unidad de control a partir de *opcode* (0100011) y *funct3*. La señal **Mem_W** indica la escritura del dato **Data_R** en la posicion de memoria **ALU_R**.

La señal **Data_T** indica el tipo de dato **Data_M** que se almacenará.

Mem_W	Operación
0	Niguna
1	Escritura

Data_T	Operación
000	
101	SB
110	SH
100	SW



Implementación monociclo

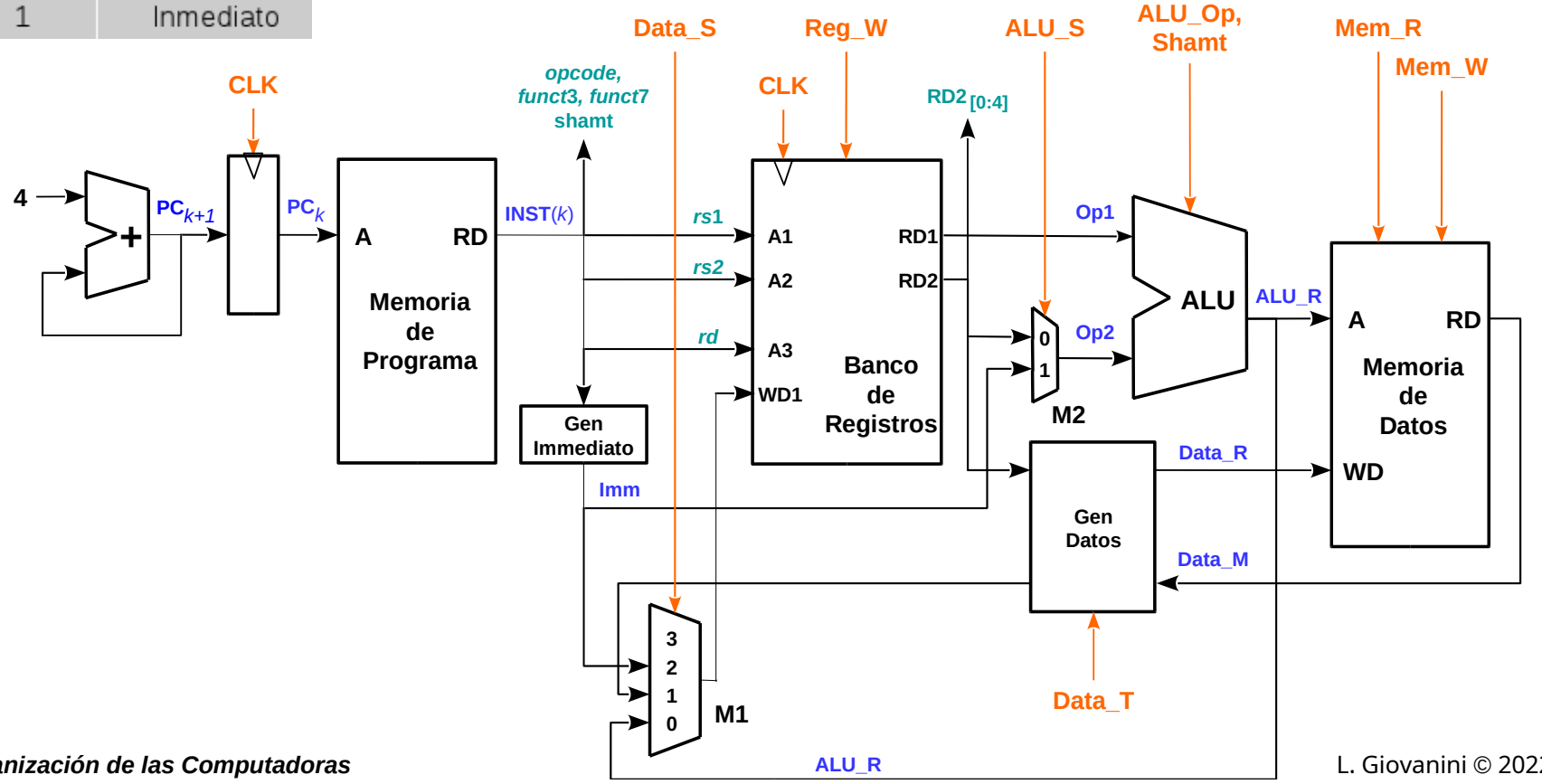
Microarquitectura – El datapath de los datos

Data_S	Operando
00	ALU
01	Memoria
10	Inmediato
11	---

ALU_S	Operando
0	Registro
1	Inmediato

A continuación se integran los circuitos resultantes a partir de incorporar dos multiplexores para enrutar las señales de datos correspondientes.

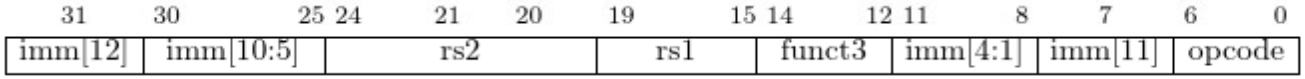
- El multiplexor M1 es manejado por la señal **Data_S** y enruta las fuentes de datos de entrada al banco de registros;
- El multiplexor M2 es manejado por la señal **ALU_S** y enruta las fuentes de datos del operando 2 de la ALU.



Implementación monociclo

Microarquitectura – Instrucciones clase B

A continuación se procede a diseñar el datapath para ejecutar **saltos condicionales relativos**. Estas instrucciones pertenecen a la ***clase B*** y tienen el siguiente formato



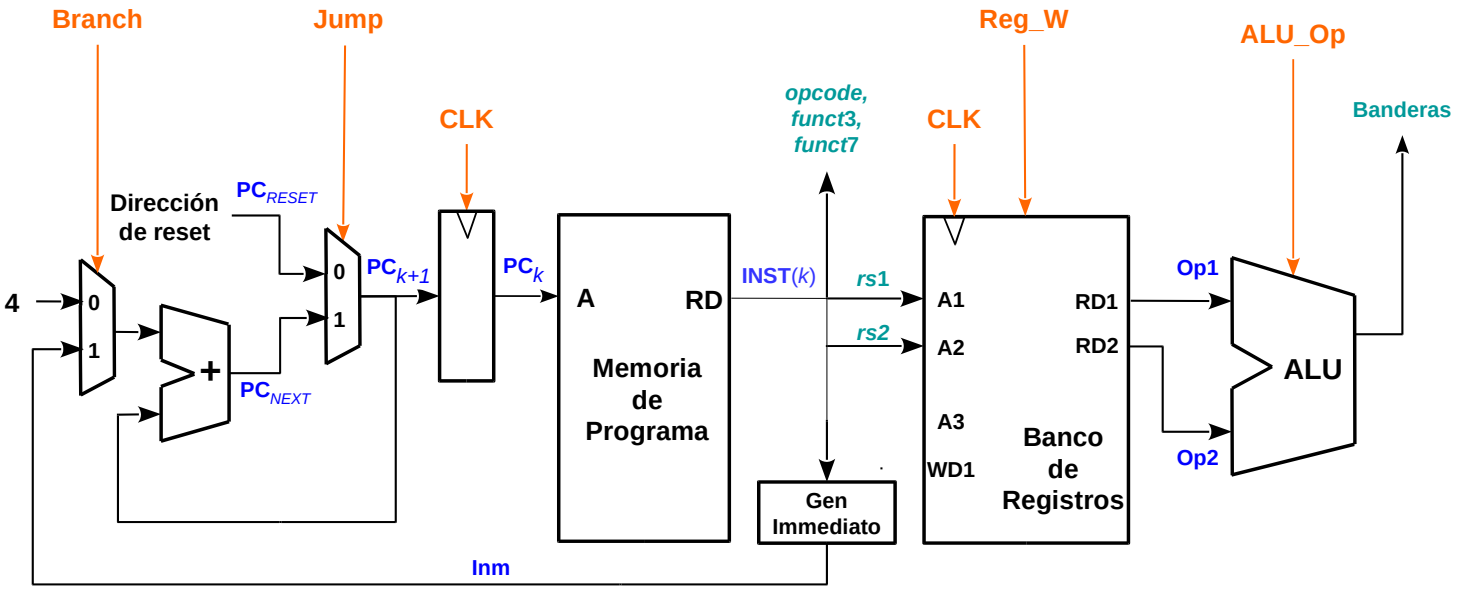
Los campos que indican los registros (**A1**=*rs*1 y **A2**=*rs*2) tienen un ancho de 5 bits e incluye el datos inmediato (*imm*[0:11]). *rs*1 y rs2 son los registros a comparar y *imm*, de 12 bits de ancho, el desplazamiento con respecto al valor actual del contador de programa (PC_k), con un rango de ± 4 KB.

El siguiente circuito implementa las instrucciones de salto condicional (BEQ, BGE, BLT y BNE). Se comparan los registro indicados **A1** y **A2**, y se suma **Imm** al contador de programa si se cumple la condición (**Branch=1**).

Branch	Operación
0	Normal
1	Salto

La ALU realiza una comparacion (**ALU_Op**=0001 ó **ALU_Op**=1111) según sea con o sin signo a partir de *opcode* (1100111) y *funct3*.

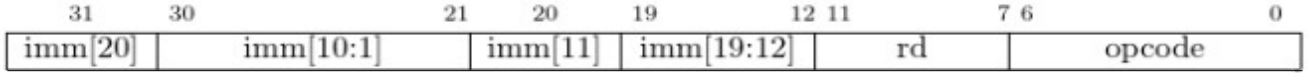
Esta operación genera la señal **Banderas** que es utilizada por la **unidad de control** para decidir.



Implementación monociclo

Microarquitectura – Instrucciones clase J

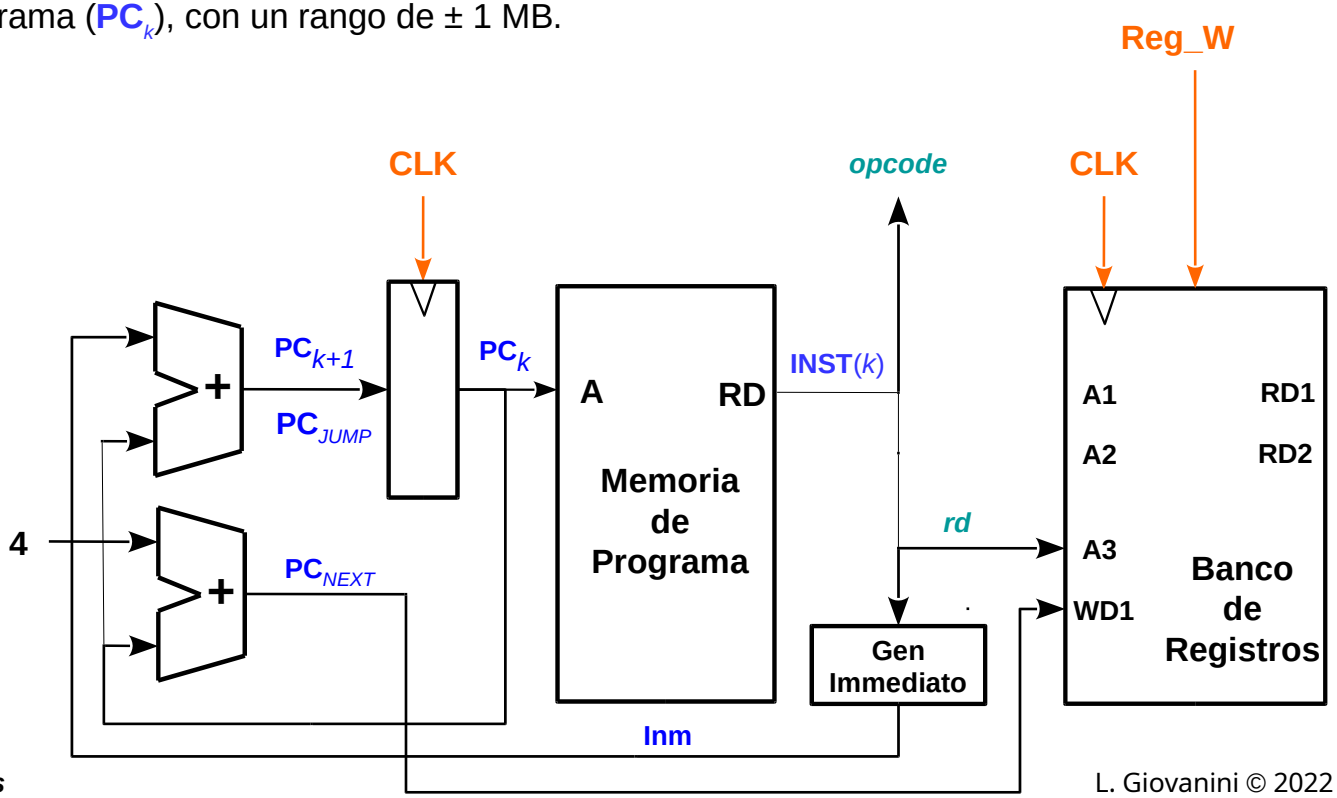
A continuación se procede a diseñar el datapath para implementar instrucciones para ejecutar ***saltos incondicionales relativos al contador de programa***. Esta instrucción pertenecen a la ***clase J*** y tienen el siguiente formato



El campo que indica el registro destino (A3=*rd*) tiene un ancho de 5 bits y el dato inmediato (*imm*[1:20]) tiene 20 bits. *rd* es el registro donde se almacena la dirección de retorno y *imm* el desplazamiento con respecto al valor actual del contador de programa (**PC**$_k$), con un rango de ± 1 MB.

El siguiente circuito implementa la instrucción JAL. La señal **Reg_W=1** indica que la próxima instrucción a ejecutar (**PC**$_{NEXT}$) se almacene en el registro **A3**, y se suma **Imm** al contador de programa.

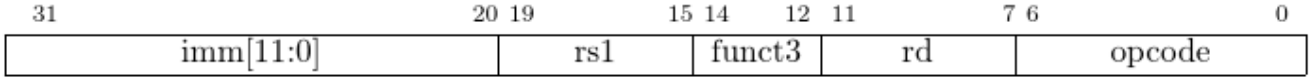
Los saltas incodicionales sin retorno (pseudo operación J) se codifican como una instrucción JAL con *rd* = *x*0.



Implementación monociclo

Microarquitectura – Instrucciones clase J

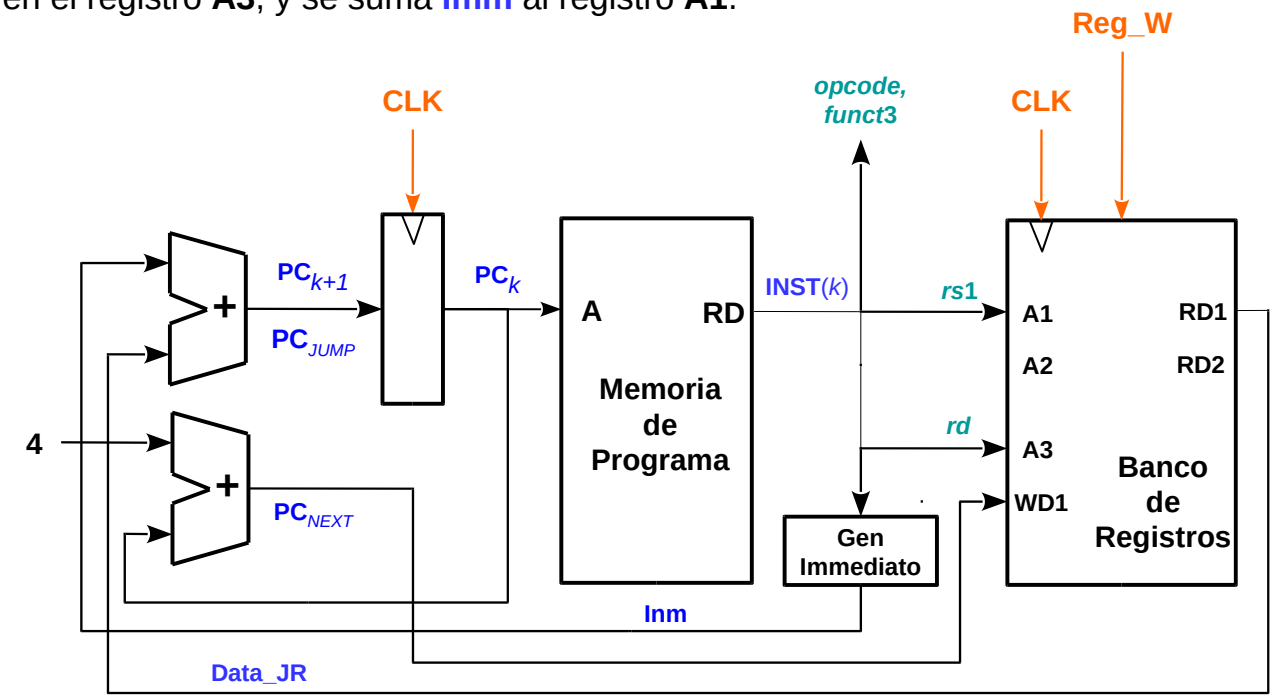
A continuación se procede a diseñar el datapath para implementar instrucciones de ***saltos incondicionales relativos a registro***. Esta instrucción pertenece a la ***clase J*** y tienen el siguiente formato



Los campos que indican los registros fuente (A1=*rs*1) y destino (A3=*rd*) tienen 5 bits cada uno de ellos (*funct*3=000). *rs*1 define el puntero con el cual se generará la dirección de destino y *rd* donde se guardará la dirección de retorno. El campo inmediato *imm*, de 12 bits de ancho, provee el desplazamiento relativo a *rs*1 en un rango de ± 4 KB.

El siguiente circuito implementa la instrucción JALR. La señal **Reg_W=1** indica que la proxima instrucción a ejecutar (PC_{NEXT}) se almacene en el registro **A3**, y se suma **Imm** al registro **A1**.

La instrucción JALR se definió para permitir que una secuencia de dos instrucciones salte en cualquier lugar en un rango de direcciones absolutas de 32 bits. Una instrucción LUI puede cargar primero rs1 con los 20 bits superiores de una dirección de destino, luego JALR puede agregar los bits inferiores.



Implementación monociclo

Microarquitectura – El datapath de saltos

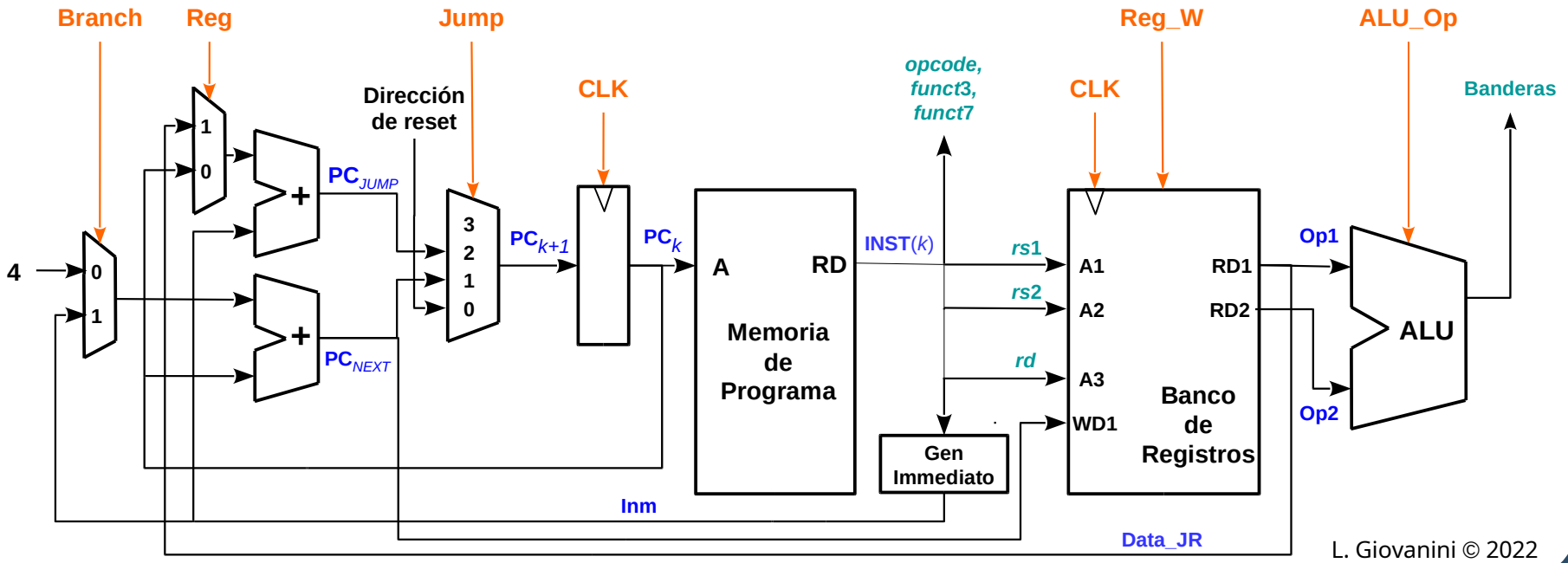
A continuación se integran los circuitos resultantes para implementar los diferentes tipos de saltos a partir de incorporar dos multiplexores para enrutar las señales de datos correspondientes.

- El multiplexor **M3** es manejado por la señal **Jump** y enruta las direcciones de los saltos y las direcciones de ejecucion normal y branchs;
- El multiplexor **M4** es manejado por la señal **Branch** y enruta los desplazamientos (4 o Imm) que seran utilizados para calcular la direccion relativa al contador de programa.
- El multiplexor **M5** es manejado por la señal **Reg** y enruta los registros base que seran utilizados para calcular la direccion de salto (**PC** o *rs*1).

Jump	Operación
00	Normal
01	Branch
10	Salto
11	Interrupción

Branch	Operación
0	Normal
1	Salto

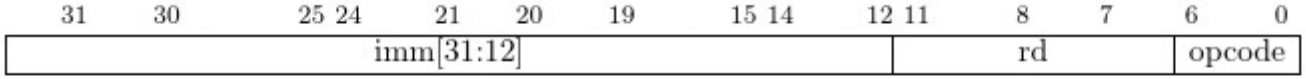
Reg	Puntero
0	PC
1	*rs*1



Implementación monociclo

Microarquitectura – Instrucciones clase U

A continuación se procede a diseñar el datapath para implementar instrucciones para ***operar con datos en la parte superior de los registros***. Estas instrucciones pertenecen a la ***clase U*** y tienen el siguiente formato

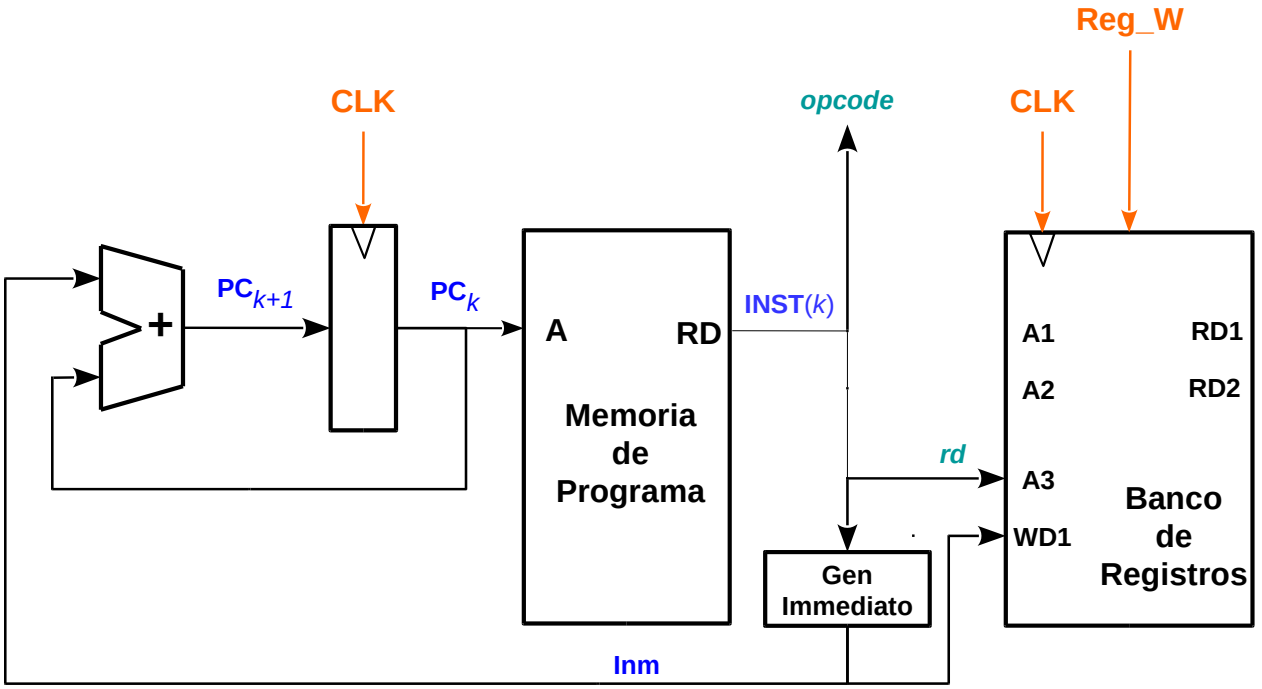


El campo que indica el registro (A3=*rd*) tiene 5 bits de ancho y el campo inmediato (*imm*[12:31]) de 19 bits de ancho. El campo *rd* define el destino del dato e *imm*[12:31] el dato inmediato con el cual trabajar. *imm* es realineado para que su bit mas significativo coincida con el de *rd*.

El siguiente circuito implementa las instrucciones LUI y AUIPC.

La señal **Reg_W=1** indica que **Imm** se almacena en la parte superior del registro **A3** (instrucción LUI). La instrucción AUIPC suma **Imm** al contador de programa y equivale a un salto incondicional.

El realineamiento se produce en el bloque de generacion de dato inmediato.



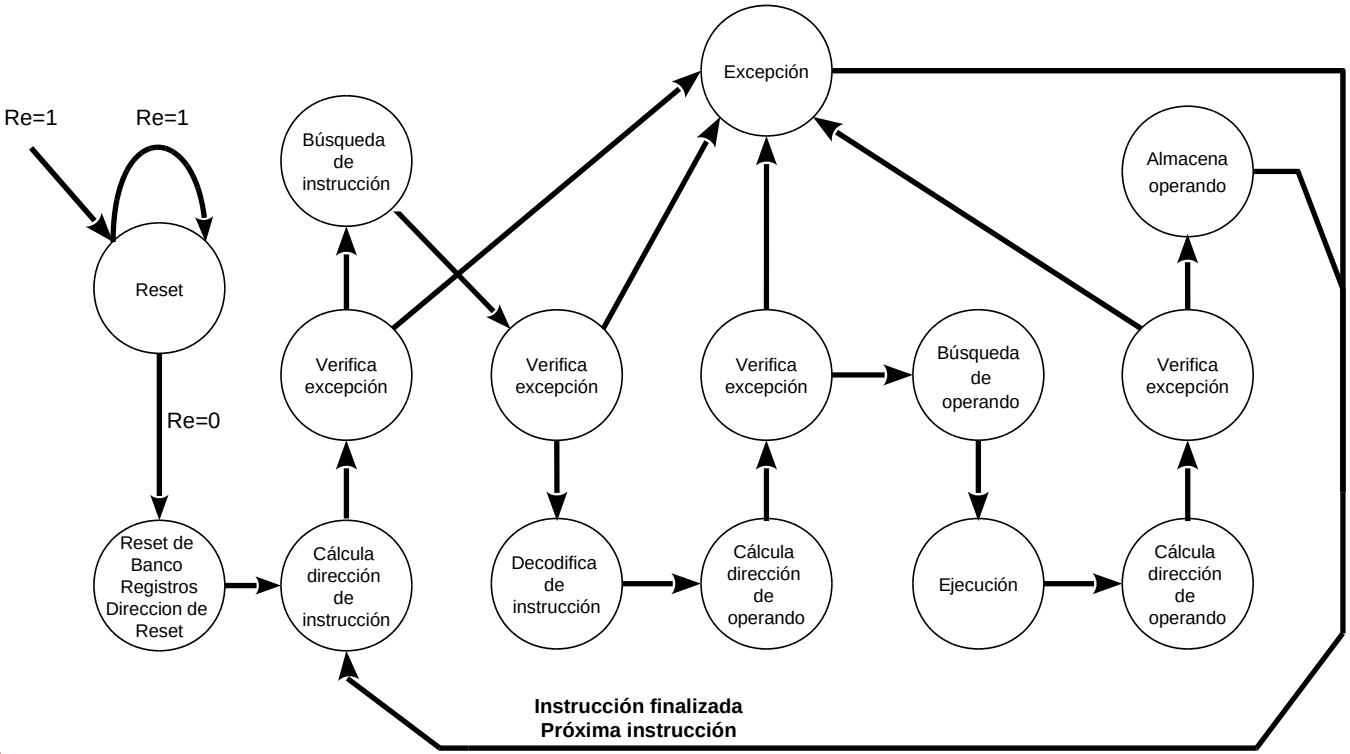
Implementación monociclo

Microarquitectura – Excepciones

Una ***excepción*** es la indicación de un problema que ocurre durante la ejecución de un programa debido a que algún dato, dirección o instrucción no se apega al funcionamiento del procesador. Es decir:

Una excepción es un evento no planificado desde dentro del procesador que altera la ejecución de un programa,

Las excepciones pueden producirse cuando hay un overflow, la insrtucción es desconocida ó la dirección no está alineada con la memoria, entre otras. Cuando una de estas condiciones se produce, el ***control es transferido a un programa encargado de resolver el problema*** (manejador de excepciones). Después de atender la excepción, el control es devuelto al programa que se estaba ejecutando cuando ocurrió la excepción: este programa continúa como si nada hubiese ocurrido.



Las causas de excepción son monitoreadas durante el ciclo de ejecución por la unidad de control, que se encargada de generar y gestionar las excepciones correspondientes.

Es importante que los manejadores de excepciones deben mantener el estado del programa que fue interrumpido de manera que su ejecución pueda luego continuar.

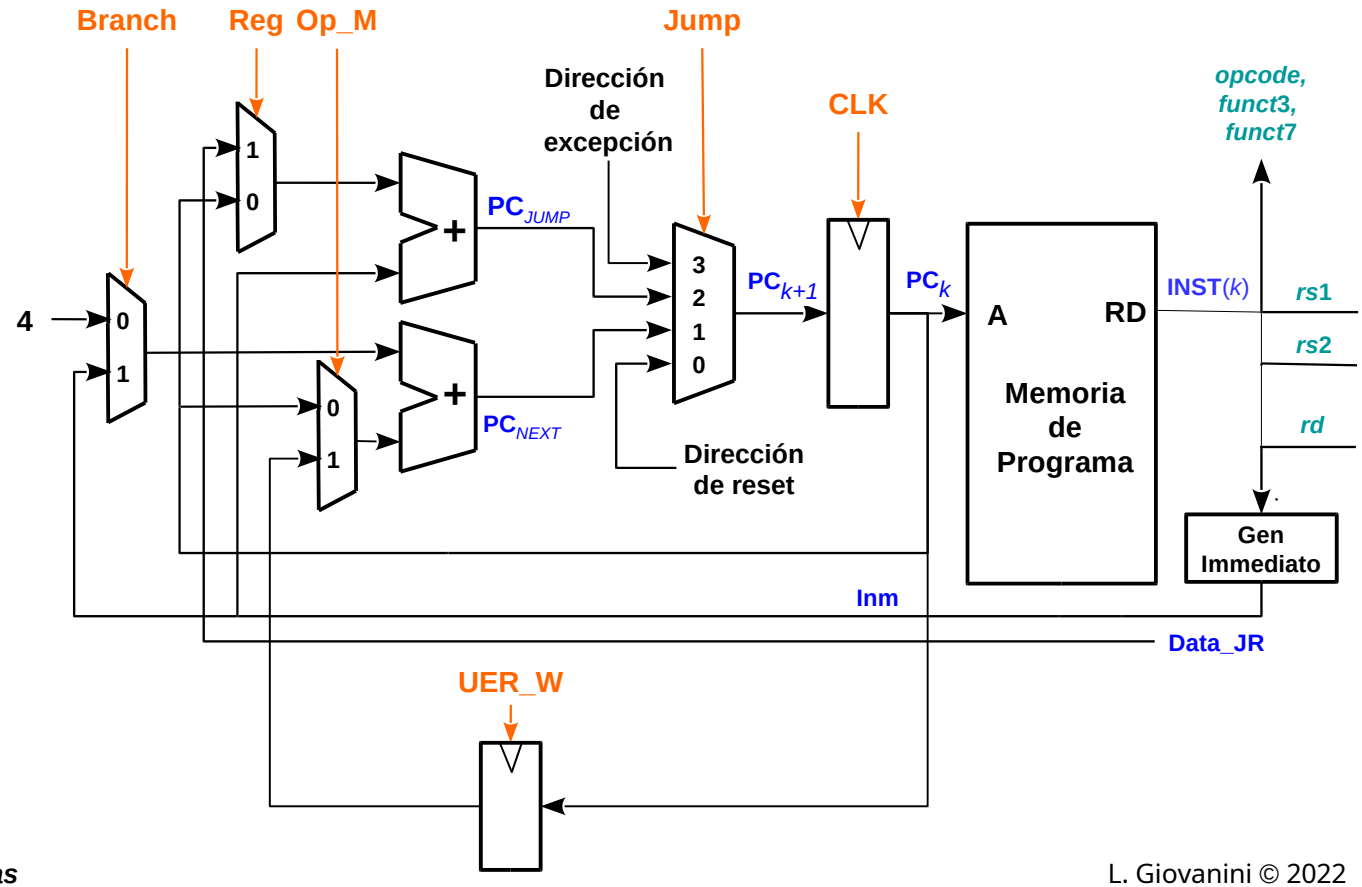
Implementación monociclo

Microarquitectura – Excepciones

A continuación se procede a diseñar el datapath para implementar ***el llamado a excepciones***. La excepción es generada de manera interna por la unidad de control cuando detecta un código de operación no válido. En estas circunstancias, la unidad de control almacena la dirección de la instrucción en curso (PC_k) en el registro interno a través de un flanco de ascenso en la señal **UER_W**, y cambia la dirección de ejecución a la dirección de excepcion partir de asignar **Jump = 11**.

Una vez finalizada la rutina de atención de la excepción, el procesador retoma la ejecución normal del programa a partir de asigna **Jump=01, Op_M = 1** y **Branch = 0**, con lo cual se reasume la ejecución normal de programa.

Op_M	Operación
0	Normal
1	Excepción

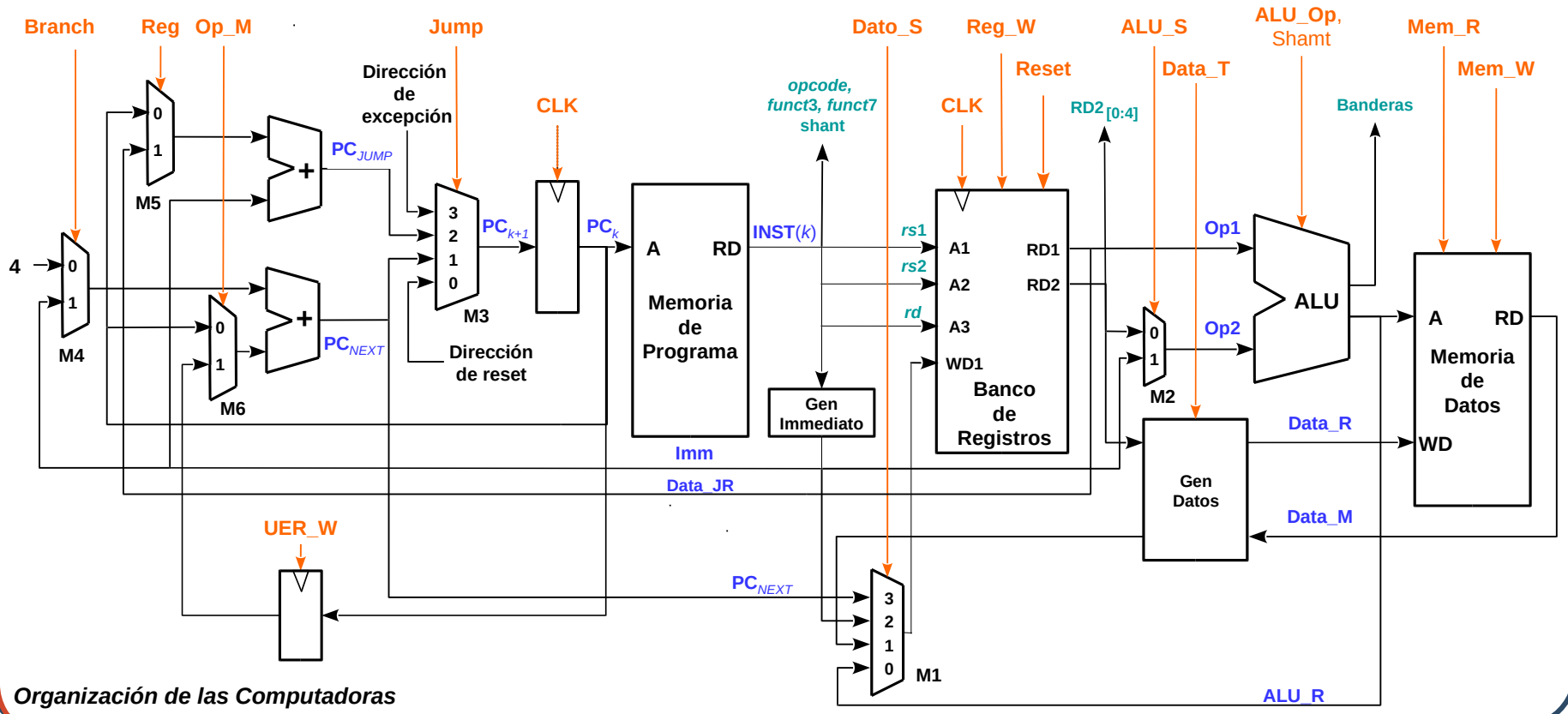


Implementación monociclo

Microarquitectura – El datapath

Finalmente se integran todos los circuitos (datos, operaciones y saltos) para implementar todas las instrucciones del ISA a partir de modificar los multiplexores para enrutar la ruta de datos (M1) y la informacion de salto (M4).

- El multiplexor **M1** es manejado por la señal **Data_S** y se le agrega una entrada (3) para enrutar el PC al banco de registros para los saltos con linkeo.
- El multiplexor **M4** es manejado por la señal **Branch** y se lo modifica para cuando haya una instrucción AUIPC utilice el desplazamiento **Imm** para calcular la direccion final (**Branch=1**).

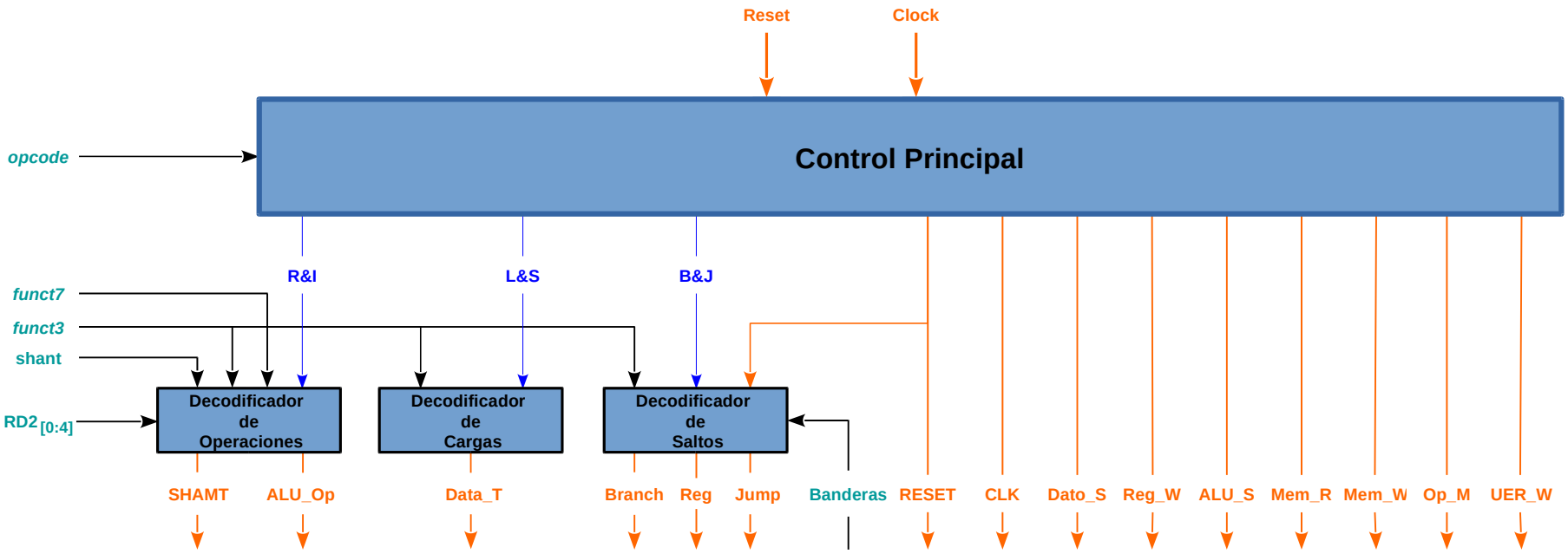


Implementación monociclo

Microarquitectura – La Unidad de Control

La ***unidad de control*** está compuesta por:

- Una unidad ***control principal*** que es la encargada de generar las señales de los bloques que gestionan el flujo de información (señales y datos) a través del datapath;
- Un ***decodificador de salto*** que genera las señales de los bloques que implementan la tarea de fectch (saltos, interrupciones, excepciones y reset);
- Un ***decodificador de carga*** que genera las señales de los bloques que implementan las tareas de carga y almacenamiento de datos en memoria; y
- Un ***decodificador de operaciones*** que genera las señales para operar la ALU.



Implementación monociclo

Microarquitectura – La Unidad de Control

El ***control principal*** es el bloque encargado de generar las señales de los bloques que gestionan el flujo de información a través del datapath (**Dato_S, Reg_W, ALU_S, Mem_R** y **Mem_W**) y las señales de control de los otros bloque (**R&I, L&S, B&J** y **EXC**).

Estas señales son las encargadas de habilitar los bloques correspondientes a cada tipo de instrucción.

Opcode		R&I	L&S	B&J	EXC	Data_S	Reg_W	ALU_S	Mem_W	Mem_R
Tipo R	0110011	001	00	000	0	00	1	0	0	0
Tipo I	0010011	010	00	000	0	00	1	1	0	0
Load	0000011	110	10	000	0	01	1	1	0	1
Store	0100011	110	01	000	0	01	0	1	1	0
Branch	1100011	101	00	001	0	00	0	0	0	0
JALR	1100111	000	00	101	0	11	1	0	0	0
JAL	1101111	000	00	100	0	11	1	0	0	0
AUIPC	0010111	000	00	010	0	00	0	0	0	0
LUI	0110111	000	00	000	0	10	1	0	0	0
	Cualquier otro	000	00	000	1	00	0	0	0	0

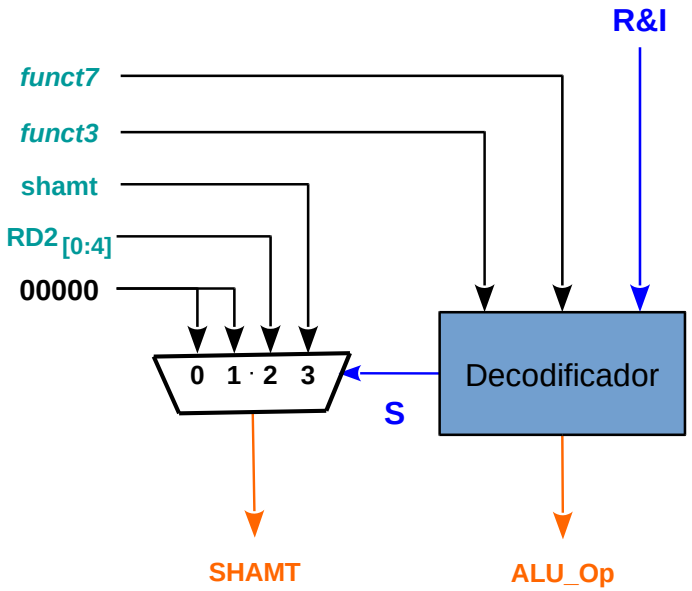
Implementación monociclo

Microarquitectura – La Unidad de Control

El ***decodificador de operaciones*** es el bloque de la unidad de control que determina las operaciones realizadas por la ALU a partir de las señales R&I, funct3 y funct7.

La señal R&I es generada por el control principal indicando cuando una operación aritmética, lógica o desplazamiento es necesaria.

Las señales funct3 y funct7 indican el tipo de operación que se realizará.



funct7	funct3	R&I	ALU_Op	S	SHAMT	
*******	***	000	0000	00	00000	
0000000	000	001 / 010	0001	00	00000	ADD / ADDI
0100000	000	001 / 010	0010	00	00000	SUB
0000000	001	001 / 010	0110	10 / 11	RD2[0:4] / Shamt	SLL / SLLI
0000000	100	001 / 010	0101	00	00000	XOR / XORI
0000000	101	001 / 010	0111	10 / 11	RD2[0:4] / Shamt	SRL / SRLI
0100000	101	001 / 010	1000	10 / 11	RD2[0:4] / Shamt	SRA / SRAI
0000000	110	001 / 010	0100	00	00000	OR / ORI
0000000	111	001 / 010	0011	00	00000	AND / ANDI
*******	000	110	0001	00	00000	LB / SB
*******	001	110	0001	00	00000	LH / SH
*******	010	110	0001	00	00000	LW / SW
*******	100	110	0001	00	00000	LBU
*******	101	110	0001	00	00000	LHU
*******	000	101	1111	00	00000	BEQ
*******	001	101	1111	00	00000	BNE
*******	100	101	1111	00	00000	BLT
*******	101	101	1111	00	00000	BGE
*******	110	101	0010	00	00000	BLTU
*******	111	101	0010	00	00000	BGEU

Las operación a realizar se indica a través de ALU_Op y la cantidad de desplazamiento se indica a través de SHAMT, la cual es determinada por shamt ó RD2[0:4].

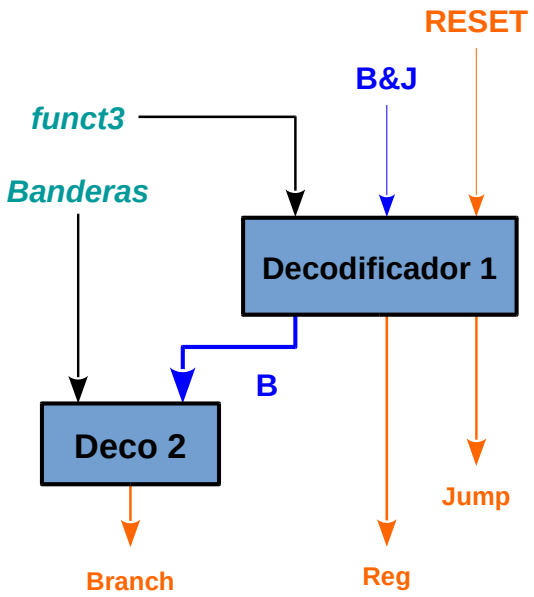
Implementación monociclo

Microarquitectura – La Unidad de Control

El ***decodificador de saltos*** es el bloque de la unidad de control que determina el tipo de salto que realizará el procesador a partir de las señales B&J, funct3 y RESET.

La señal B&J es generada por el control principal indicando el tipo de salto que se implementa.

Las señal funct3 indica el tipo de salto que se realizará y la señal RESET indica cuando se inicializa el procesador



funct3	RESET	B&J	EXC	B	Reg	Jump	
000	0	001	0	1	0	01	BEQ
001	0	001	0	1	0	01	BNE
100	0	001	0	1	0	01	BLT
101	0	001	0	1	0	01	BGE
110	0	001	0	1	0	01	BLTU
111	0	001	0	1	0	01	BGEU
***	0	100	0	0	0	10	JAL
000	0	101	0	0	1	10	JALR
***	0	010	0	0	0	10	AUIPC
***	0	000	0	0	0	01	Normal
***	1	***	0	0	0	00	Reset
***	0	***	1	0	0	11	Excepción

El tipo de salto a ejecutar se indica a través de ALU_Op, Reg y Branch. El **decodificador 2** genera la señal Branch a partir de una señal interna B y las Banderas, que determinan si se cumple o no la condición requerida por la instrucción.

Implementación monociclo

Microarquitectura – La Unidad de Control

El ***decodificador de datos*** es el bloque de la unidad de control que organiza y estructura los datos que se transfieren entre los registros y memoria a partir de las señales L&S,y funct3.

La señal L&S es generada por el control principal indicando el tipo de transferencia de datos que se realizará (memoria-registro ó registro-memoria).

Las señal funct3 indica el tamaño del dato (byte, half-word ó word) y el lugar donde se ubicará el dato dentro de la palabra (parte superior o inferior), esto último sólo es valido cuando se carga un byte.

funct3	L&S	Data_T	
***	00	000	
000	10	010	LB
001	10	001	LH
010	10	000	LW
100	10	011	LBU
101	10	111	LHU
000	01	101	SB
001	01	110	SH
010	01	100	SW

Las instrucciones SW, SH y SB almacenan valores de 32 bits, 16 bits y 8 bits desde los bits bajos del registro rs2 a la memoria.

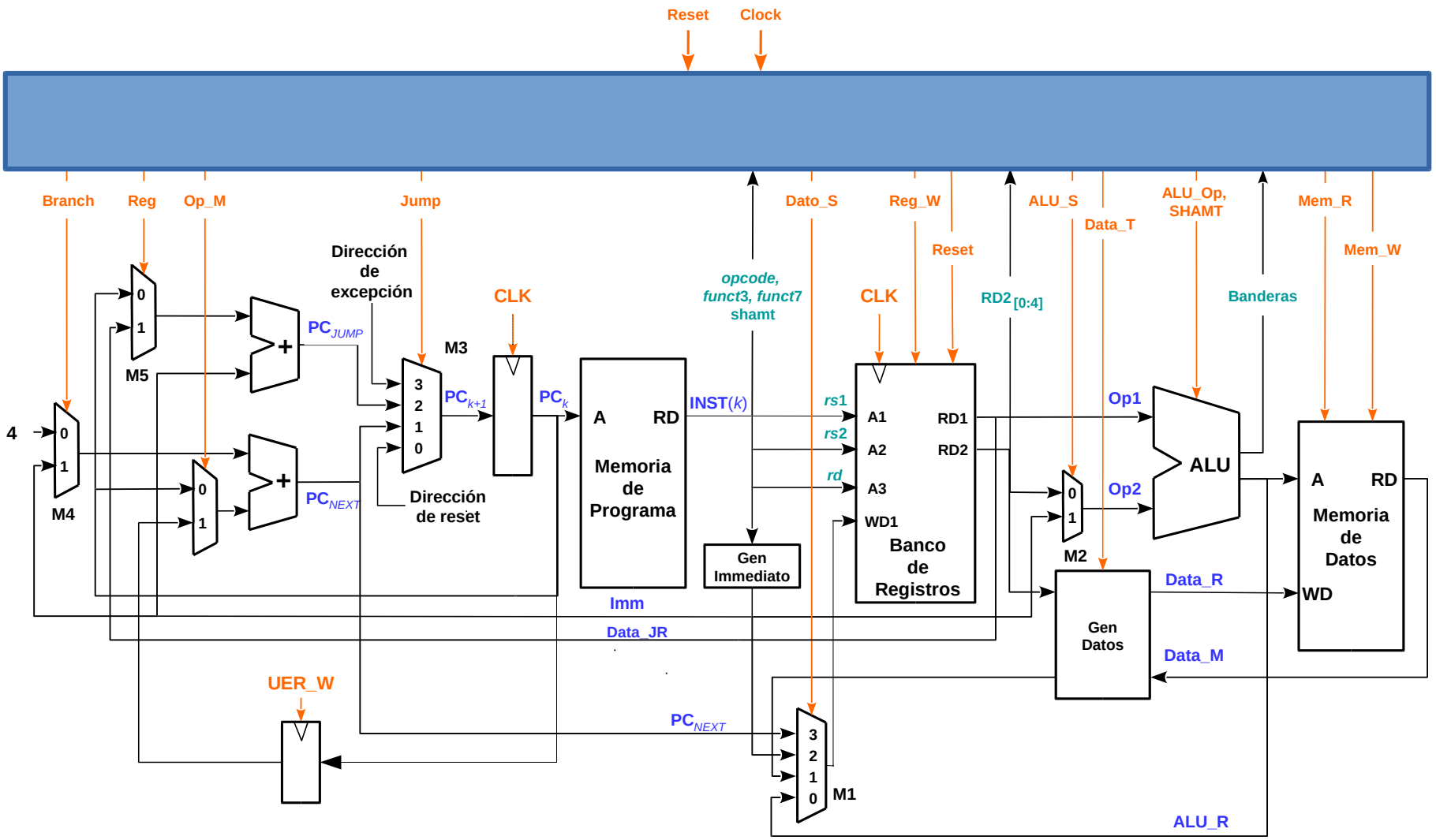
La instrucción LW carga un valor de 32 bits de la memoria en *rd*.

LH carga un valor de 16 bits de la memoria en *rd*, extendiendo el dato leido a 32 bits a partir de agregar ceros en la parte superior de la palabra antes de almacenarlo. LHU carga un valor de 16 bits desde la memoria en rd, extendiendo el dato leido a 32 bits a partir de agregar ceros en la parte inferior de la palabra antes de almacenarlo.

LB y LBU se definen de manera análoga para valores de 8 bits.

Implementación monociclo

Microarquitectura

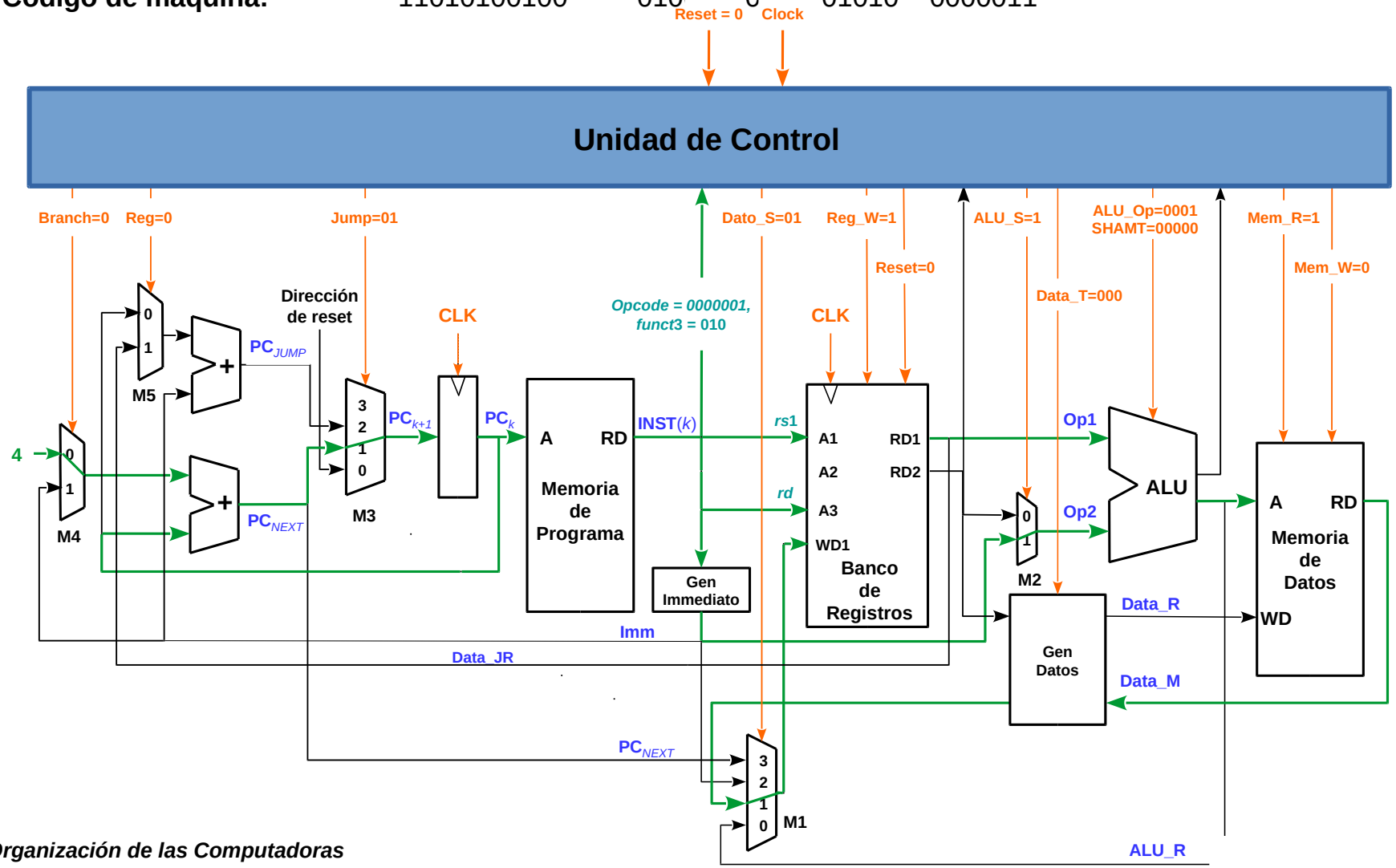


Señales de control y flujo de información cuando ***se carga en el registro*** 10 ***el contenido de la dirección memoria*** [x8+256H]

Código assembler: lw x10, 256(x8)

31 — 20	19 — 15	14 — 12	11 — 7	6 — 0
offset[11:0]	rs1	010	rd	0000011

Código de máquina: 11010100100 01000 010 01010 0000011

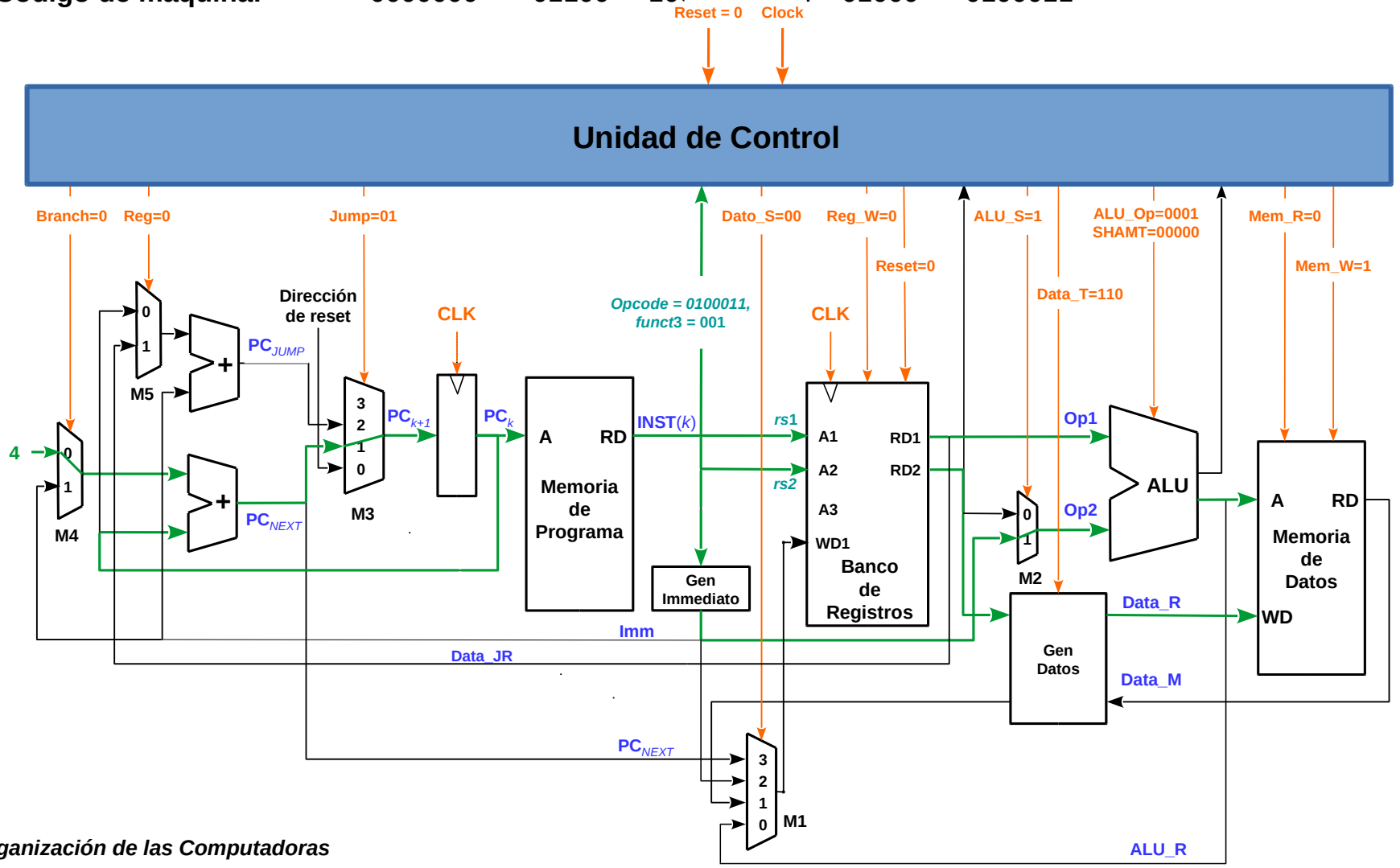


Señales de control y flujo de información cuando se ***guarda una media palabra del registro*** 18 ***en la posición de memoria*** [x12+100H]

Código assembler: sw x18, 100 (x12)

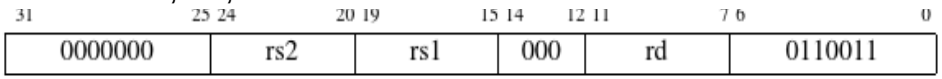
31 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0
offset[11:5]	rs2	rs1	010	offset[4:0]	0100011

Código de máquina: 0000000 01100 10010 001 01000 0100011



Señales de control y flujo de información cuando se ***calcula la suma de los registros*** 5 ***y*** 20 ***y se guarda el resultado en el registro*** 28

Código assembler: add x28, x5, x20



Código de máquina: 0000000 10100 00101 000 11100 0110011

Reset = 0 Clock

Unidad de Control

Branch=0 Reg=0 Jump=01 Dato_S=00 Reg_W=1 ALU_S=0 ALU_Op=0001 SHAMT=00000 Mem_R=0 Mem_W=0

Reset=0 Data_T=000

Dirección de reset

CLK CLK

opcode = 0110011
funct3 = 000
funct7=000000

M5 M4 M3 M2 M1

PC_{JUMP} PC_{NEXT} PC_{k+1} PC_k PC_{NEXT}

A RD Memoria de Programa

INST(*k*)

Gen Immediato

rs1 rs2 rd

A1 A2 A3 WD1 RD1 RD2 Banco de Registros

Op1 Op2 ALU

A RD Memoria de Datos WD

Data_R Data_M

Gen Datos

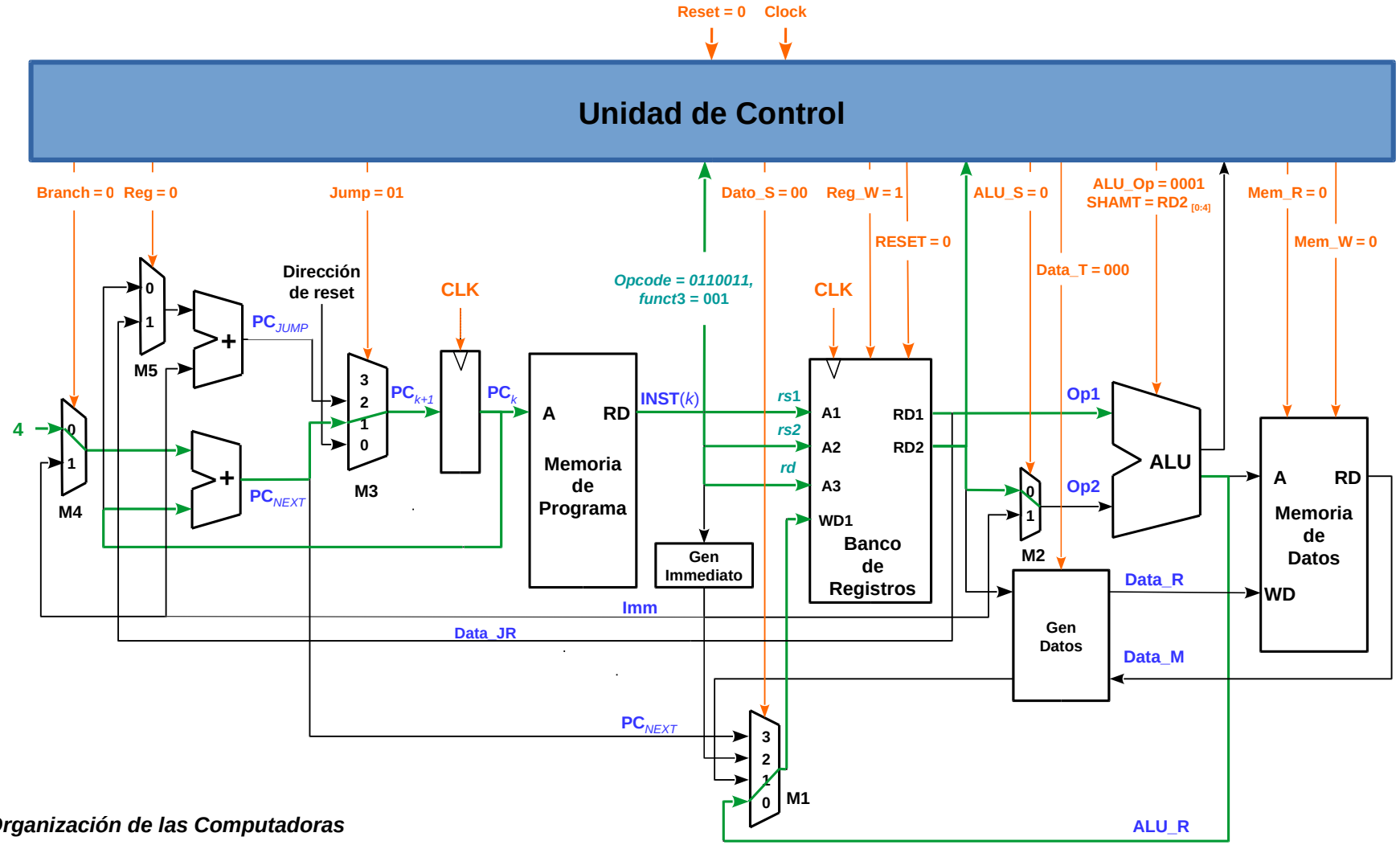
Imm Data_JR ALU_R

Señales de control y flujo de información cuando se ***desplaza a la izquierda el registro*** 15 ***y se guarda el resultado en el registro*** 12. La cantidad de posiciones está en el registro 4

Código assembler: sll x12, x15, x04

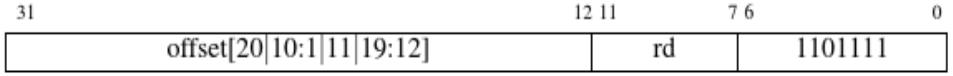
31 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0
0000000	rs2	rs1	001	rd	0110011

Código de máquina: 0000000 00100 01111 001 01100 0110011

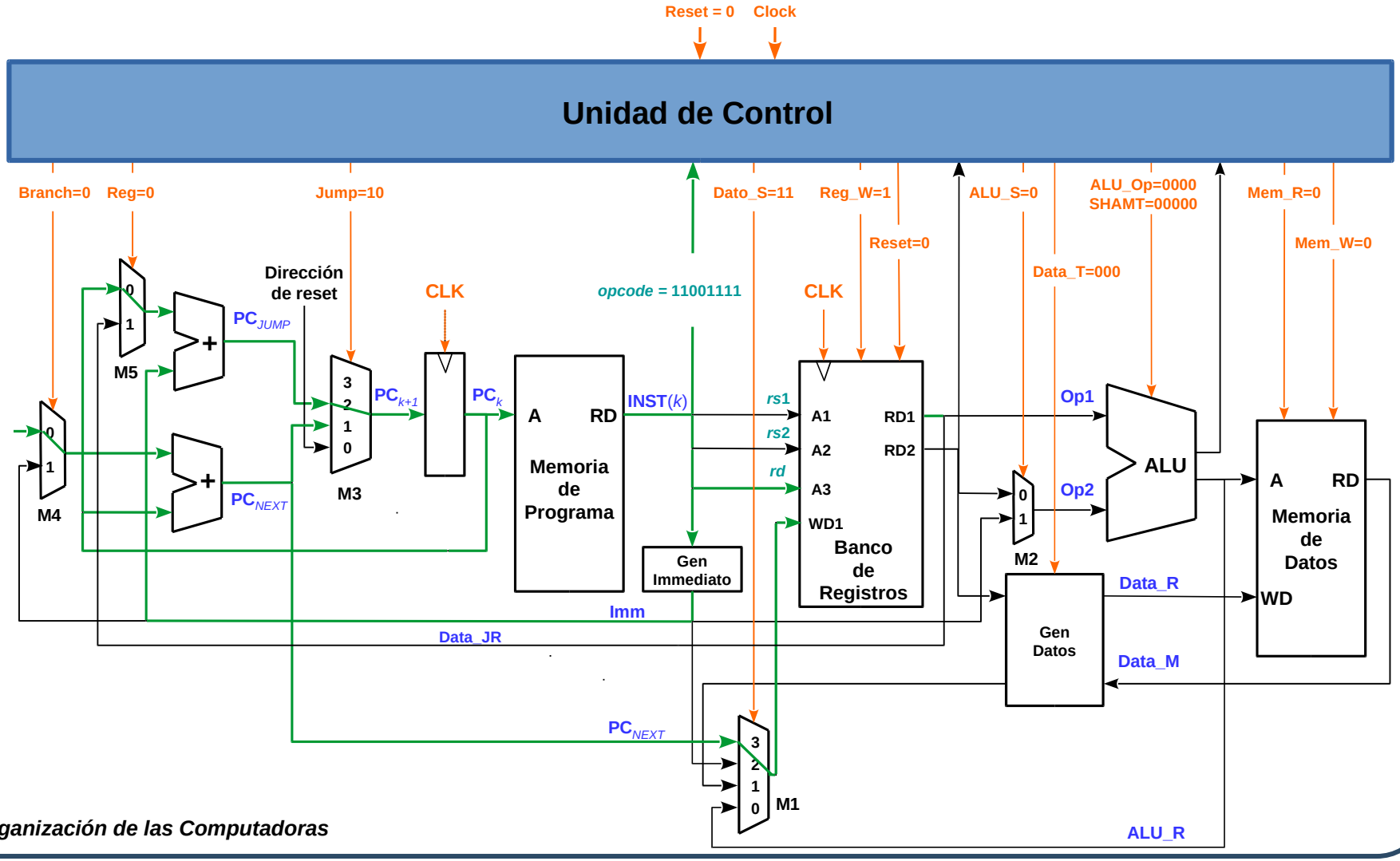


Señales de control y flujo de información cuando se ***realiza un salto relativo al PC desplazando*** 456 ***posiciones hacia adelante y guardando la direeccion de retorno en el registro*** 8

Código assembler: jal x8, 000E4H



Código de máquina: 00000000000011100100 01000 1101111



Implementación monociclo

Microarquitectura – Registros de estados y control (CSR)

El ISA estándar de RISC-V reserva un espacio de codificación de 12 bits para hasta 4096 registros de control y estados (CSR) del procesador.

Por convención, los 4 bits superiores se utilizan para codificar la accesibilidad de lectura y escritura de los CSR según el nivel de privilegio:

Los dos bits superiores (csr[11:10]) indican si el registro es de lectura/escritura (00, 01 o 10) o de solo lectura (11).

Los dos bits siguientes (csr[9:8]) codifican el nivel de privilegio más bajo que puede acceder a la CSR.

CSR Address [11:10]	CSR Address [9:8]	CSR Address [7:6]	Hex	Use and Accessibility
User CSRs				
00	00	XX	0x000–0x0FF	Standard read/write
01	00	XX	0x400–0x4FF	Standard read/write
10	00	XX	0x800–0x8FF	Non-standard read/write
11	00	00–10	0xC00–0xCBF	Standard read-only
11	00	11	0xCC0–0xCFF	Non-standard read-only
Supervisor CSRs				
00	01	XX	0x100–0x1FF	Standard read/write
01	01	00–10	0x500–0x5BF	Standard read/write
01	01	11	0x5C0–0x5FF	Non-standard read/write
10	01	00–10	0x900–0x9BF	Standard read/write
10	01	11	0x9C0–0x9FF	Non-standard read/write
11	01	00–10	0xD00–0xDBF	Standard read-only
11	01	11	0xDC0–0xDFF	Non-standard read-only
Reserved CSRs				
XX	10	XX	Reserved	
Machine CSRs				
00	11	XX	0x300–0x3FF	Standard read/write
01	11	00–10	0x700–0x79F	Standard read/write
01	11	10	0x7A0–0x7AF	Standard read/write debug CSRs
01	11	10	0x7B0–0x7BF	Debug-mode-only CSRs
01	11	11	0x7C0–0x7FF	Non-standard read/write
10	11	00–10	0xB00–0xBBF	Standard read/write
10	11	11	0xBC0–0xBFF	Non-standard read/write
11	11	00–10	0xF00–0xFBF	Standard read-only
11	11	11	0xFC0–0xFFF	Non-standard read-only

Implementación monociclo

Microarquitectura – CSRs modo máquina

Number	Privilege	Name	Description
Machine Information Registers			
0xF11	MRO	mvendorid	Vendor ID.
0xF12	MRO	marchid	Architecture ID.
0xF13	MRO	mimpid	Implementation ID.
0xF14	MRO	mhartid	Hardware thread ID.
Machine Trap Setup			
0x300	MRW	mstatus	Machine status register.
0x301	MRW	misa	ISA and extensions
0x302	MRW	medeleg	Machine exception delegation register.
0x303	MRW	mideleg	Machine interrupt delegation register.
0x304	MRW	mie	Machine interrupt-enable register.
0x305	MRW	mtvec	Machine trap-handler base address.
0x306	MRW	mcounteren	Machine counter enable.
Machine Trap Handling			
0x340	MRW	mscratch	Scratch register for machine trap handlers.
0x341	MRW	mepc	Machine exception program counter.
0x342	MRW	mcause	Machine trap cause.
0x343	MRW	mtval	Machine bad address or instruction.
0x344	MRW	mip	Machine interrupt pending.
Machine Protection and Translation			
0x3A0	MRW	pmpcfg0	Physical memory protection configuration.
0x3A1	MRW	pmpcfg1	Physical memory protection configuration, RV32 only.
0x3A2	MRW	pmpcfg2	Physical memory protection configuration.
0x3A3	MRW	pmpcfg3	Physical memory protection configuration, RV32 only.
0x3B0	MRW	pmpaddr0	Physical memory protection address register.
0x3B1	MRW	pmpaddr1	Physical memory protection address register.
		⋮	
0x3BF	MRW	pmpaddr15	Physical memory protection address register.

Implementación monociclo

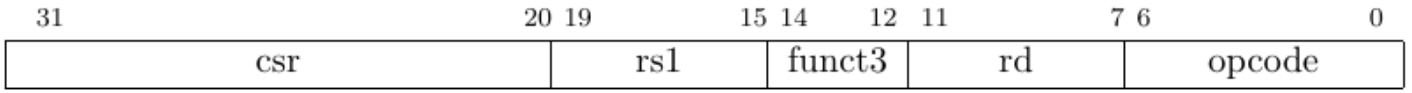
Microarquitectura – CSRs modo usuario

Number	Privilege	Name	Description
		Machine Counter/Timers	
0xB00	MRW	mcycle	Machine cycle counter.
0xB02	MRW	minstret	Machine instructions-retired counter.
0xB03	MRW	mhpmcounter3	Machine performance-monitoring counter.
0xB04	MRW	mhpmcounter4	Machine performance-monitoring counter.
		⋮	
0xB1F	MRW	mhpmcounter31	Machine performance-monitoring counter.
0xB80	MRW	mcycleh	Upper 32 bits of mcycle, RV32I only.
0xB82	MRW	minstreth	Upper 32 bits of minstret, RV32I only.
0xB83	MRW	mhpmcounter3h	Upper 32 bits of mhpmcounter3, RV32I only.
0xB84	MRW	mhpmcounter4h	Upper 32 bits of mhpmcounter4, RV32I only.
		⋮	
0xB9F	MRW	mhpmcounter31h	Upper 32 bits of mhpmcounter31, RV32I only.
		Machine Counter Setup	
0x323	MRW	mhpmevent3	Machine performance-monitoring event selector.
0x324	MRW	mhpmevent4	Machine performance-monitoring event selector.
		⋮	
0x33F	MRW	mhpmevent31	Machine performance-monitoring event selector.
		Debug/Trace Registers (shared with Debug Mode)	
0x7A0	MRW	tselect	Debug/Trace trigger register select.
0x7A1	MRW	tdata1	First Debug/Trace trigger data register.
0x7A2	MRW	tdata2	Second Debug/Trace trigger data register.
0x7A3	MRW	tdata3	Third Debug/Trace trigger data register.
		Debug Mode Registers	
0x7B0	DRW	dcsr	Debug control and status register.
0x7B1	DRW	dpc	Debug PC.
0x7B2	DRW	dscratch	Debug scratch register.

Implementación monociclo

Microarquitectura – Registros de estados y control (CSR)

Todas las instrucciones CSR leen, modifican y escriben atómicamente un solo CSR, cuyo especificador CSR.



12	5	3	5	7
source/dest	source	CSRRW	dest	SYSTEM
source/dest	source	CSRRS	dest	SYSTEM
source/dest	source	CSRRC	dest	SYSTEM
source/dest	uimm[4:0]	CSRRWI	dest	SYSTEM
source/dest	uimm[4:0]	CSRRSI	dest	SYSTEM
source/dest	uimm[4:0]	CSRRCI	dest	SYSTEM

CSRRW (Atomic Read/Write) intercambia atómicamente los valores del CSR y el registro. CSRRW lee el valor del CSR, lo extiende con ceros a 32 bits y luego lo escribe en rd. El valor almacenado en rs1 se escribe en el CSR.

CSRRS (Atomic Read and Set Bits) lee el valor del CSR, lo extiende con ceros a 32 bits y lo escribe en rd. El dato almacenado en rs1 se trata como una máscara de bits que especifica con 1 las posiciones de bits que se establecerán en el CSR.

CSRRC (Atomic Read and Clear Bits) lee el valor del CSR, lo extiende con ceros a 32 bits y lo escribe en rd. El dato almacenado en rs1 se trata como una máscara de bits que especifica con 1 las posiciones de bits que se borrarán en el CSR.

Implementación monociclo

Microarquitectura – Instrucciones para los CSRs

A continuación se procede a diseñar el datapath para implementar ***instrucciones que involucran operaciones con los CSR***. Estas instrucciones ejecutan operaciones aritméticas-lógicas, desplazamientos y carga de datos, pertenecen a la ***clase I*** y tienen el siguiente formato

csr	rs1	001	rd	1110011	CSRRW
csr	rs1	010	rd	1110011	CSRRS
csr	rs1	011	rd	1110011	CSRRC
csr	uimm	101	rd	1110011	CSRRWI
csr	uimm	110	rd	1110011	CSRRSI
csr	uimm	111	rd	1110011	CSRRCI

Los campos que indican los registros (**A1**=*rs*1 y **A3**=*rd*) tienen de 5 bits y el datos inmediato 5 bits (*uimm*[15:19]). La operación a realizar (*funct*3).

Estas instrucciones operan con datos de 8 bits, de manera similar que las instrucciones LB y SB, por lo que se hace necesario utilizar el bloque de Generacion de Datos para leer y escribir en memoria.

Se modifica el bloque de Generacion de Datos Inmediatos para poder generar la dirección **csr**.

Se modifica el multiplexor **M1** para poder enrutar los datos de mascara de las instrucciones CSRRS y CSRRC

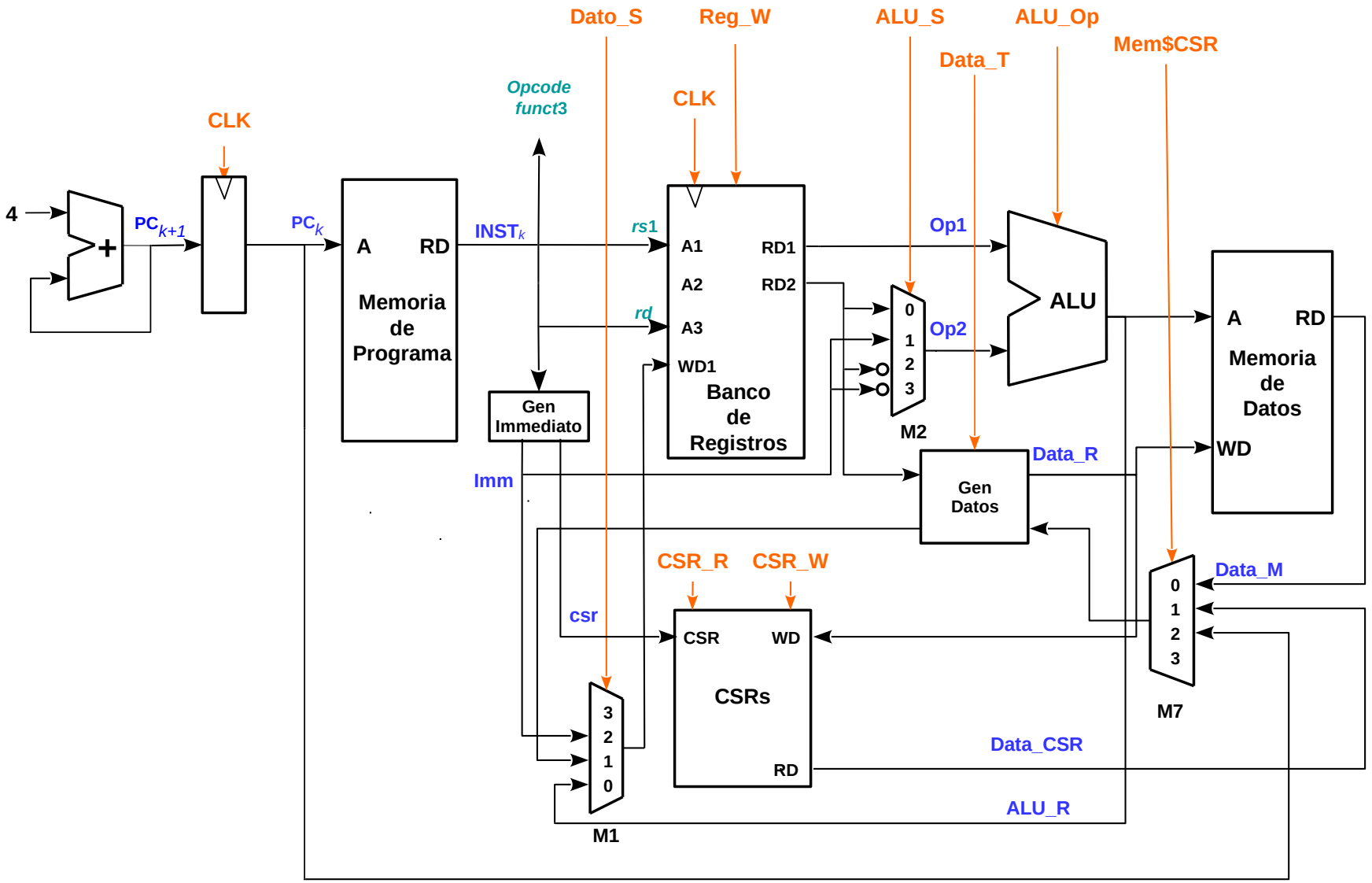
Se agrega un multiplexor **M7** a la entrada del bloque Generacion de Datos para elegir la fuente de datos: la memoria de datos o los CSR.

La señal **CSR_W** indica la escritura del dato en el CSR indicado por **csr**.

La operación a realizar por la ALU es controlada por **ALU_Op** y los operandos utilizados por **ALU_S**, lad cuales son generadas por la unidad de control a partir de *opcode* (1110011) y *funct3*.

Implementación monociclo

Microarquitectura – Instrucciones para los CSRs



Implementación monociclo

Microarquitectura con CSRs