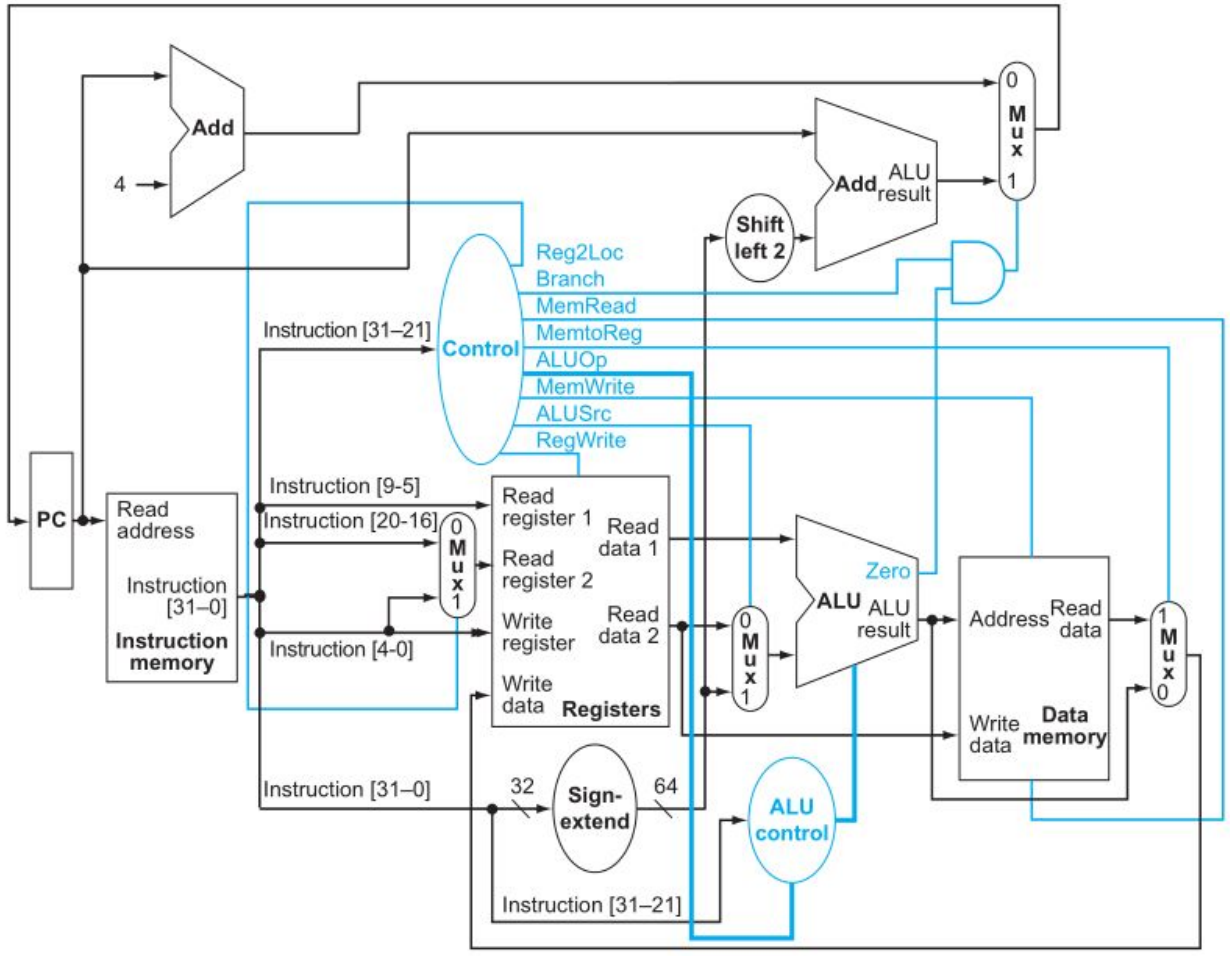
PRÁCTICO 9 - Implementación de la ISA

Gran parte de los ejercicios están basados en la Figura 4.17 “The simple datapath with the control unit” del libro COaD-LEGv8 que se copia abajo.



Instrucciones implementadas:

Instruction	ALUOp	Instruction operation	Opcode field	Desired ALU action	ALU control input
LDUR	00	load register	XXXXXXXXXXX	add	0010
STUR	00	store register	XXXXXXXXXXX	add	0010
CBZ	01	compare and branch on zero	XXXXXXXXXXX	pass input b	0111
R-type	10	ADD	10001011000	add	0010
R-type	10	SUB	11001011000	subtract	0110
R-type	10	AND	10001010000	AND	0000
R-type	10	ORR	10101010000	OR	0001

Ejercicio 1:
Marque con una barra invertida e indique el número de bits que representa cada una de las líneas del *data path*.

Ejercicio 2:
Agregando una compuerta en diagrama del *data path & control*, cambie la implementación de la instrucción `CBZ` a `CBNZ`.

Ejercicio 3:
Agregar a la implementación de la ISA la instrucción de **salto incondicional** `B`, a partir de una nueva señal que sale de `Control`, denominada `UncondBranch`.

Ejercicio 4:
Complete la tabla con el estado de las señales (sin mirar el libro). Indique con `X` las condiciones no-importa.

Instr	**Reg2Loc**	**ALUSrc**	**MemtoReg**	**RegWrite**	**MemRead**	**MemWrite**	**Branch**	**AluOp1**	**AluOp0**
R-type									
LDUR									
STUR									
CBZ									

Ejercicio 5:
Al implementar los circuitos `Control`, `ALU control` se utilizaron muchas **condiciones no-importa** para simplificar la lógica. Esto produce **efectos laterales**. Cuando `Instruction[31:21]` es `0x5B8-0x5BF`, obtenemos del `Control`

Reg2Loc	**ALUSrc**	**MemtoReg**	**RegWrite**	**MemRead**	**MemWrite**	**Branch**	**AluOp1**	**AluOp0**
1	0	0	0	0	0	1	1	1

Mientras que de `ALU control` tiene una implementación utilizando condiciones no importa que produce:

AluOp1	**AluOp0**	**I[31:21]**	**Operation**
1	1	0x5B8	0110
⋮	⋮	⋮	⋮
1	1	0x5BF	0110

Este es un típico caso de **instrucción no documentada** con un comportamiento no del todo claro. Indicar que hace esta instrucción, asignarle un mnemónico y describir la operación, a fin de completar la fila correspondiente a la nueva instrucción en la *green sheet*.

Ejercicio 6:
Suponiendo que los diferentes bloques dentro del procesador tienen las siguientes **latencias**:

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps

- **6.1)** ¿Cuál es la latencia si lo único que tuviera que hacer el procesador es *fetch* de instrucciones consecutivas?
- **6.2)** Hay un modo alternativo de generar la señal `Reg2Loc` fácilmente de la instrucción sin tener que esperar la latencia de `Control`. Explique cómo sería. Ignore la instrucción `STXR`.
- **6.3)** ¿Cuál es la latencia si solo hubiera instrucciones R?
- **6.4)** ¿Cuál es la latencia para `LDUR`?
- **6.5)** ¿Cuál es la latencia para `STUR`?
- **6.6)** ¿Cuál es la latencia para `CBZ`?
- **6.7)** ¿Cuál es el mínimo periodo de reloj para esta implementación de la ISA? Indicar también la frecuencia de reloj correspondiente ($f = 1/t$).

Ejercicio 7:
Para la tabla de latencias del Ejercicio 6, indicar el porcentaje de aumento de la velocidad de procesamiento si quitamos de las instrucciones de carga y almacenamiento la posibilidad de desplazamiento por el operando inmediato, es decir todas las instrucciones de carga son de la forma `LDUR X0, [X1]`.

Ejercicio 8:
Supongamos que podemos construir una CPU donde el ciclo de reloj es distinto para cada instrucción. Cuál sería la aceleración de esta nueva CPU sobre la anterior (ejercicio 6) si el mix de instrucciones de un programa es el siguiente:

R-type/I-Type	LDUR	STUR	CBZ	B
52%	25%	10%	11%	2%

* Como no tenemos implementado `B`, sumar el porcentaje a `CBZ`.

Ejercicio 9 (opcional):
Mostrar cómo se implementan los módulos:

1. `Sign-extend`: 32 bits de entrada, 64 bits de salida.
2. `Shift left 2`: 64 bits de entrada, 64 bits de salida.
3. Obtenga una implementación de la **composición** (`Sign-extend; Shift left 2`) en un único circuito.

Ejercicio 10 (opcional):
Muestre el circuito interno de la ALU que, a partir de los 64 bits de salida, produce la salida `Zero` (`Z`).