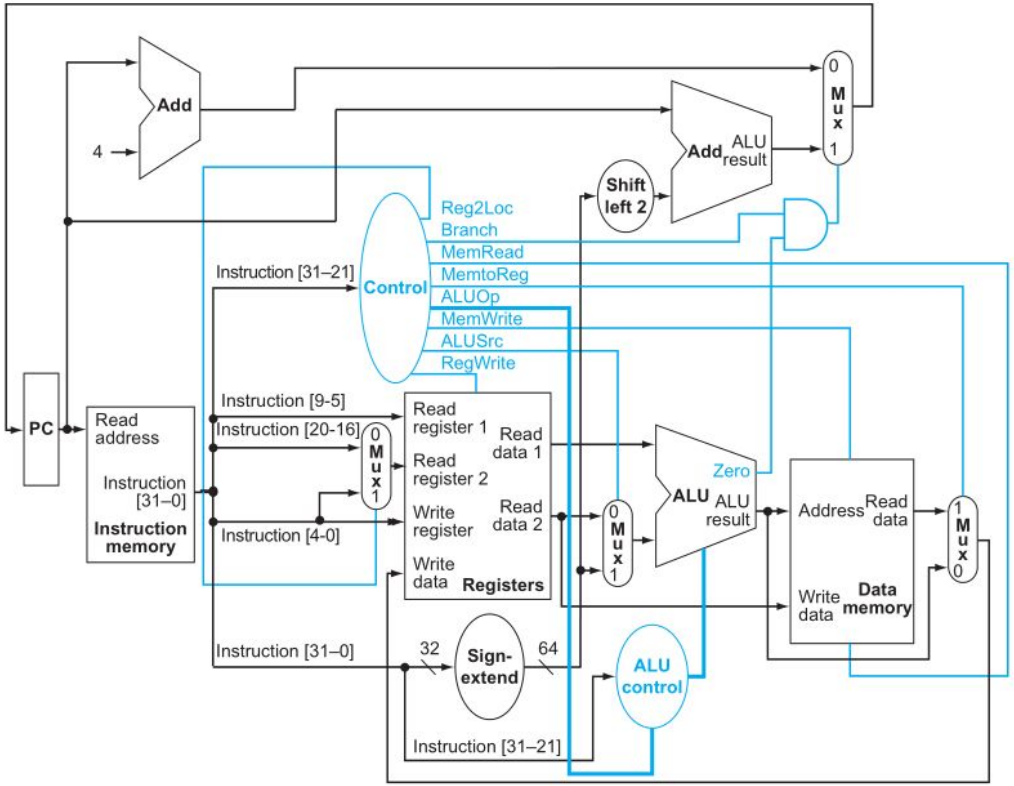
Implementación de la ISA

Parte 2

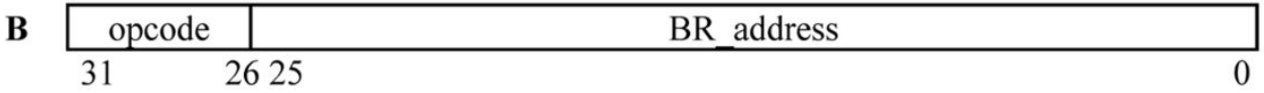
OdC 2020

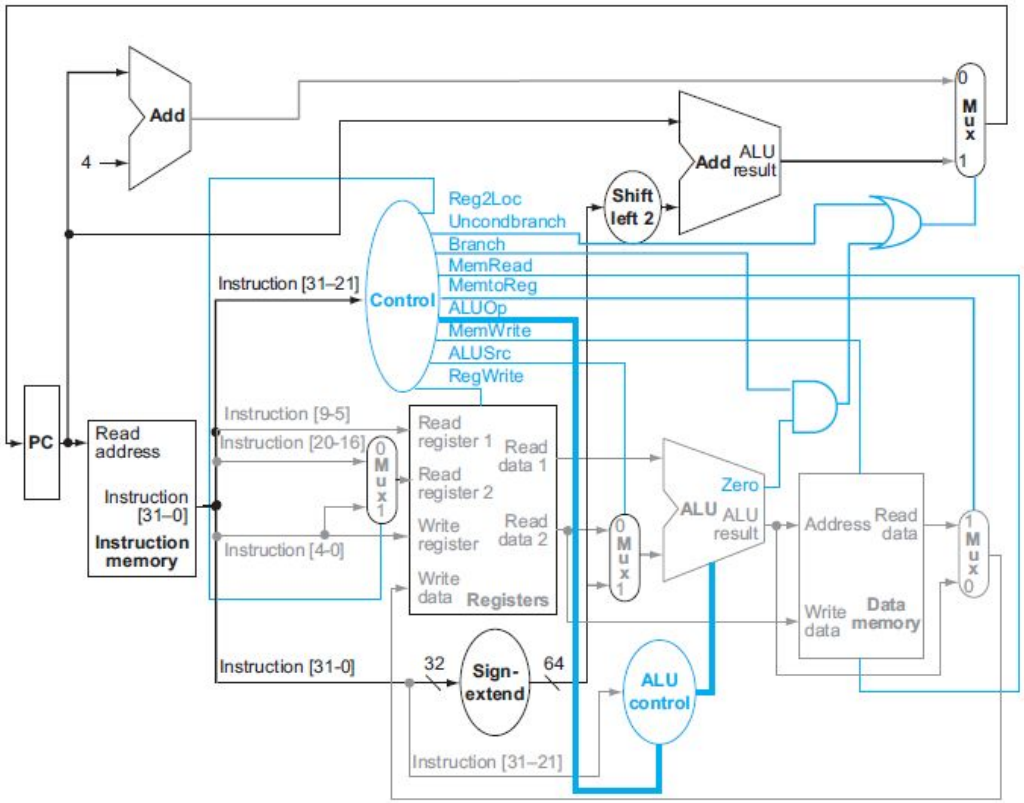
Ejercicio 7

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 7





- Sign-extend: debe reconocer el opcode de la instrucción B, tomar el campo de inmediato (los 25 bits menos significativos) y extender el signo (replicar 36 veces el bit 25).
- Control: debe decodificar el opcode de B y generar la señal de control “Uncondbranch”, que tomará valor ‘1’ en caso de ejecutar la instrucción B y ‘0’ en cualquier otro caso.

Reg2Loc	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp	Uncondbranch
X	X	X	0	X	0	0	XX	1

Ejercicio 8

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	SI2
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps	25 ps

8.1) ¿Cuál es la latencia si lo único que tuviera que hacer el procesador es fetch de instrucciones consecutivas?

8.2) 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.

8.3) ¿Cuál es la latencia si solo hubiera instrucciones de tipo R?

8.4) ¿Cuál es la latencia para LDUR?

8.5) ¿Cuál es la latencia para STUR?

8.6) ¿Cuál es la latencia para CBZ?

8.7) ¿Cuál es el mínimo periodo de reloj para esta implementación de la ISA? Indicar también la frecuencia de reloj correspondiente (f = 1/t).

Ejercicio 8.1

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control	SI2
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps	25 ps

¿Cuál es la latencia si lo único que tuviera que hacer el procesador es fetch de instrucciones consecutivas?

Latencia de fetch = RegRead + Imem = 30+250 = **280ps**

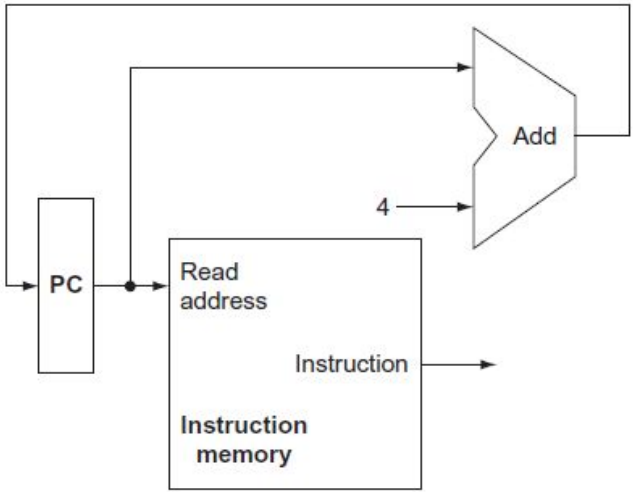
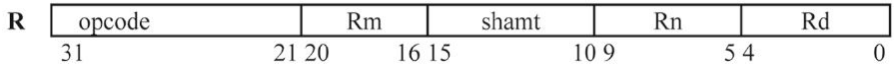


FIGURE 4.6: A portion of the datapath used for fetching instructions and incrementing the program counter.

★ Register read: tiempo posterior al flanco ascendente de reloj, necesario para que el nuevo valor de registro aparezca en la salida. Este valor se aplica sólo al PC.

★ Register setup: tiempo que la entrada de datos de un registro debe permanecer estable antes del flanco ascendente de reloj . Este valor se aplica tanto al PC como al Register File.



Ejercicio 8.3

¿Cuál es la latencia si solo hubiera instrucciones de tipo R?

Latencia de instrucción: tiempo de ejecución inherente a una instrucción.

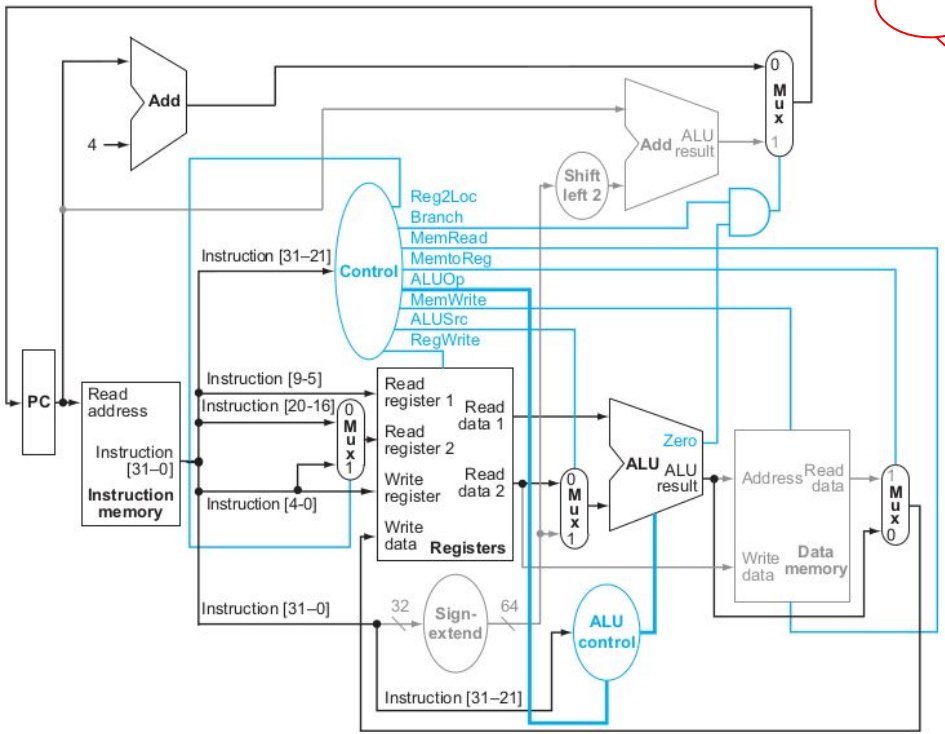
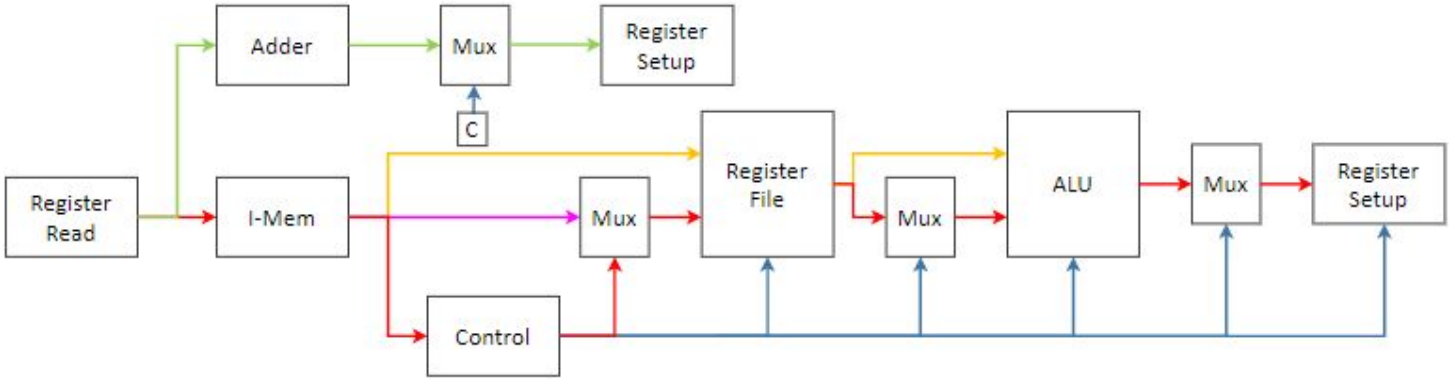


FIGURE 4.19: The datapath in operation for an R-type instruction, such as ADD X1,X2,X3. The control lines, datapath units, and connections that are active are highlighted.

Ejercicio 8.3

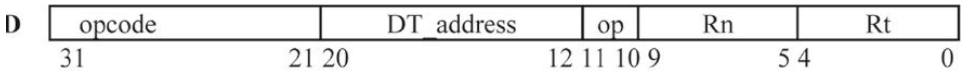
I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control	SI2
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps	25 ps



- Camino verde: Register Read + I-Mem + Control + Mux + Register Setup
- Camino rojo: Register Read + I-Mem + Control + Mux + Register File + Mux + ALU + Mux + Register Setup
- Caminos alternativos: se evitan el Control y/o los Mux, por lo que el tiempo total será menor al del camino rojo.

Camino Crítico (rojo) = 30 + 250 + 50 + 25 + 150 + 25 + 200 + 25 + 20 = 775ps

Latencia de una instrucción tipo R = 775ps



Ejercicio 8.4

¿Cuál es la latencia para LDUR?

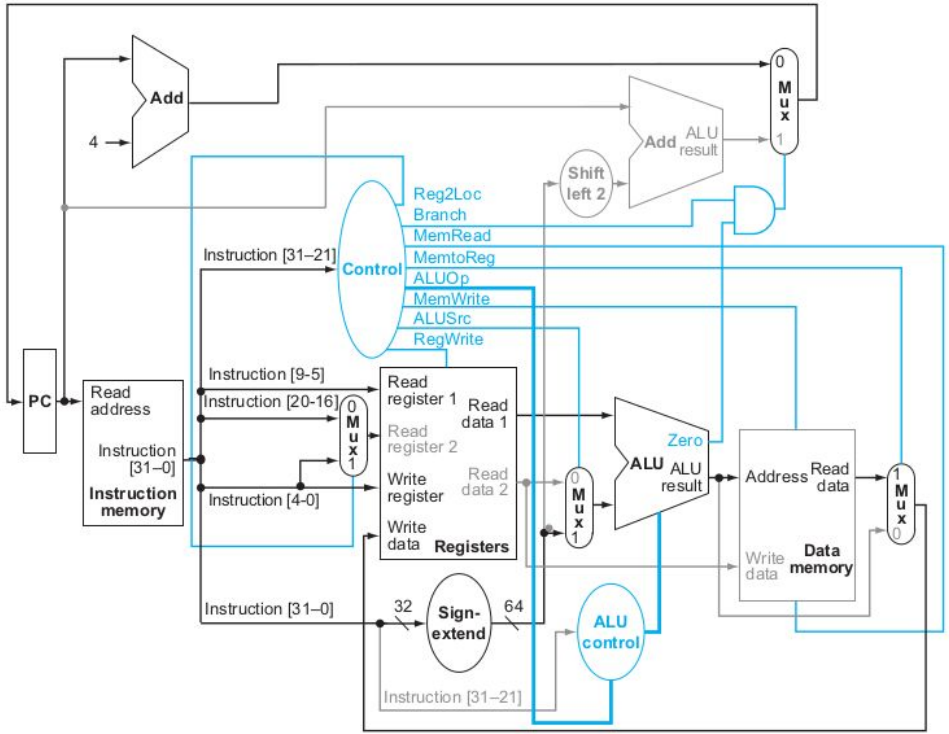
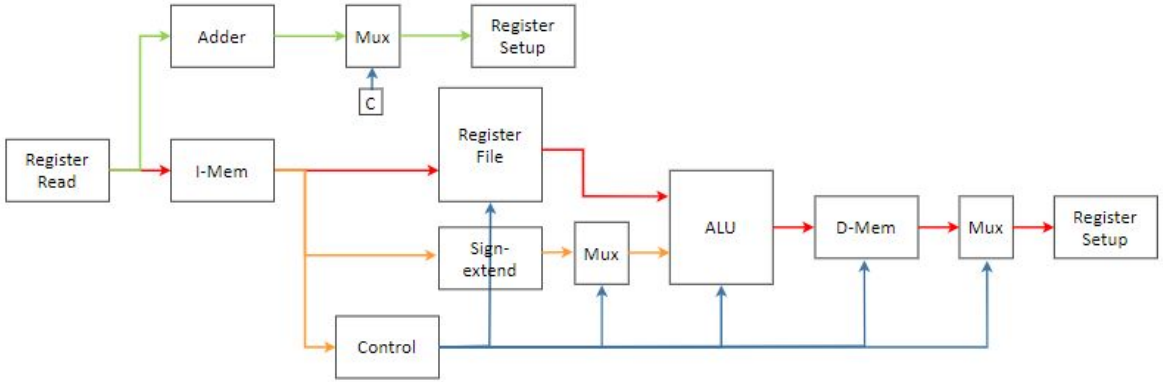


FIGURE 4.20 The datapath in operation for a load instruction. The control lines, datapath units, and connections that are active are highlighted.

Ejercicio 8.4

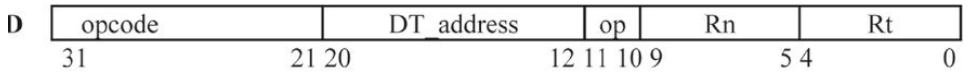
I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control	SI2
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps	25 ps



- Camino verde: Register Read + I-Mem + Control + Mux + Register Setup
- Camino rojo: Register Read + I-Mem + Register File + ALU + D-Mem + Mux + Register Setup
- Caminos alternativos: el tiempo de Sign-extend + Mux es menor al tiempo de acceso al Register File.

Camino Crítico (rojo) = 30 + 250 + 150 + 200 + 250 + 25 + 20 = 955ps

Latencia de la instrucción LDUR = 955ps

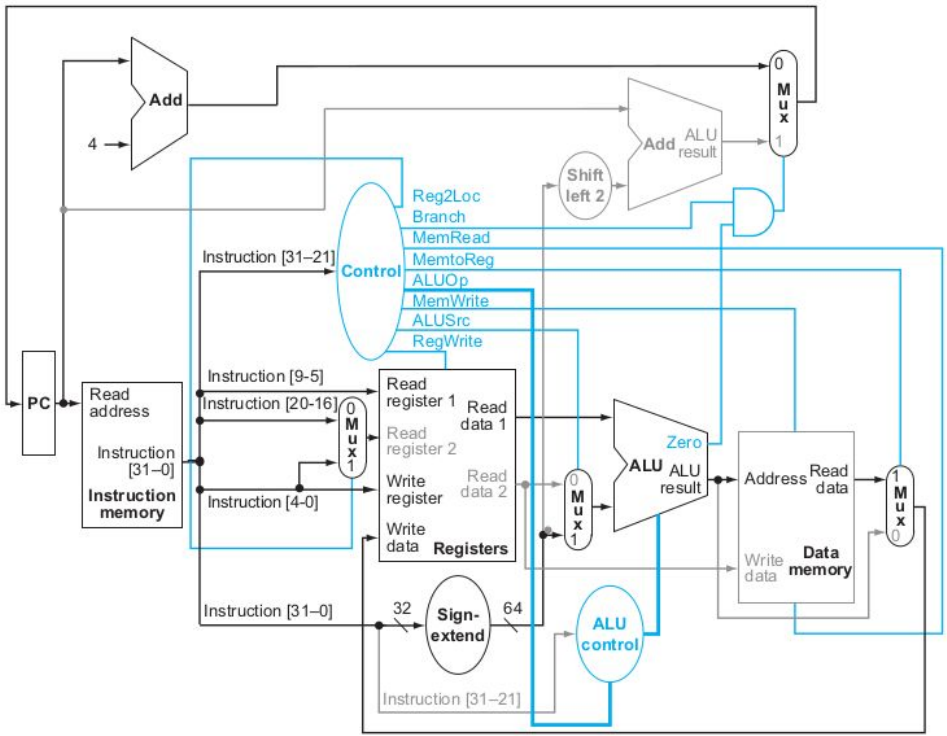


Ejercicio 8.5

¿Cuál es la latencia para STUR?

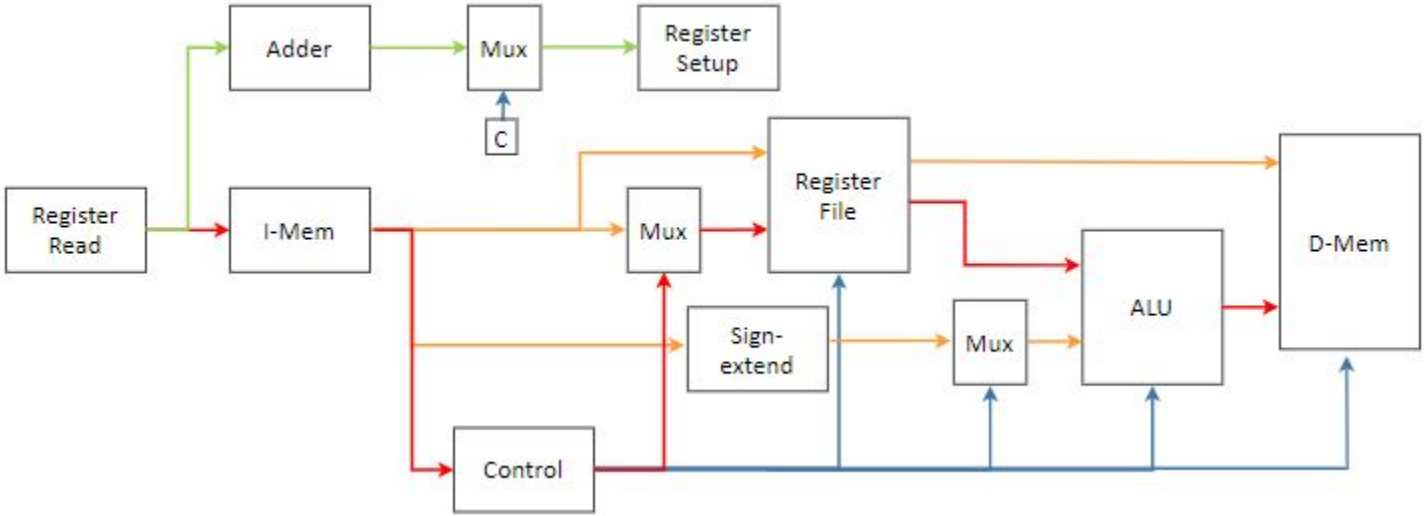
La ejecución de STUR es similar a la de LDUR, pero:

- en lugar de leer, se escribe D-Mem,
- el registro 2 contiene el dato a guardar en memoria,
- no se escribe ningún registro.



Ejercicio 8.5

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control	SI2
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps	25 ps



Camino Crítico (rojo) = 30 + 250 + 50 + 25 + 150 + 200 + 250 = 955ps

Latencia de la instrucción STUR = 955ps

Ejercicio 8.6

¿Cuál es la latencia para CBZ?

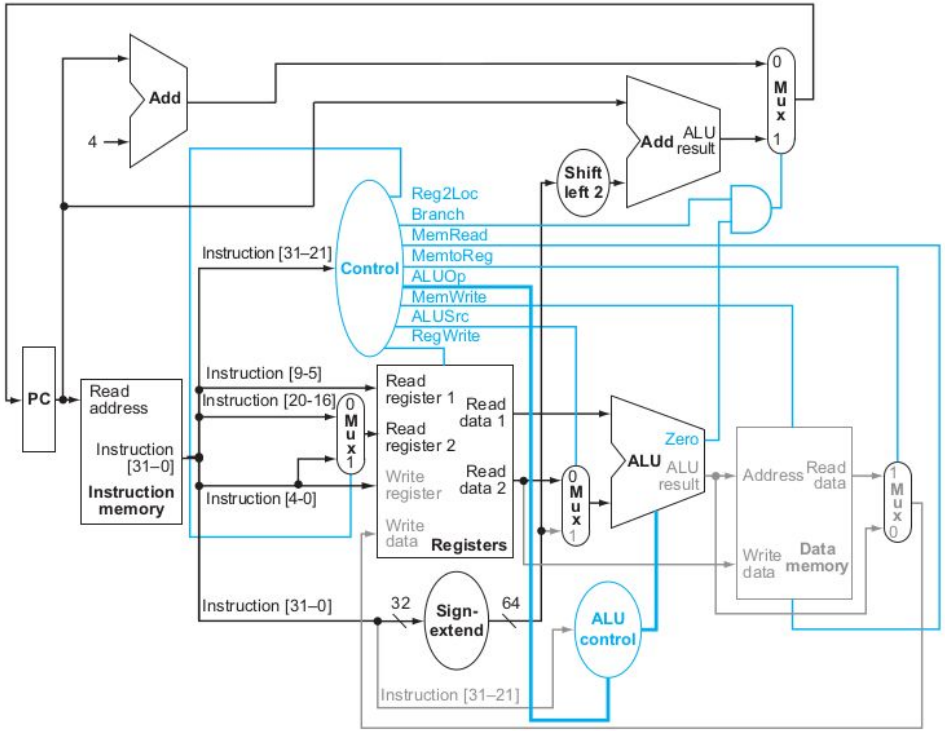
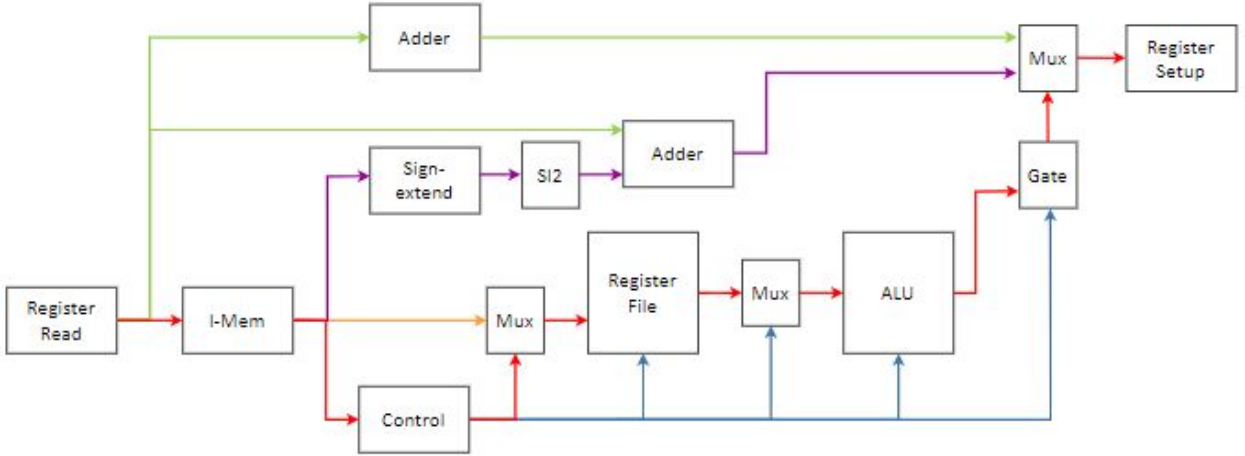


FIGURE 4.21 The datapath in operation for a compare-and-branch-on-zero instruction. After using the register file and ALU to perform the compare, the Zero output is used to select the next program counter from between the two candidates.

Ejercicio 8.6

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control	SI2
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps	25 ps



- Camino lila: Register Read + I-Mem + Sign-extend + SI2 + Adder + Mux + Register Setup
- Camino rojo: Register Read + I-Mem + Control + Mux + Register File + Mux + ALU + Gate + Mux + Register Setup

Camino Crítico (rojo) = 30 + 250 + 50 + 25 + 150 + 25 + 200 + 5 + 25 + 20 = 780ps

Latencia de la instrucción CBZ = 780ps

Ejercicio 8.7

¿Cuál es el mínimo periodo de reloj para esta implementación de la ISA? Indicar también la frecuencia de reloj correspondiente (f = 1/t).

★ Para que todas las instrucciones puedan ejecutarse completamente el periodo de reloj debe ser mayor o iguala la latencia de instrucción más grande. En este caso, corresponde a la latencia de LDUR y STUR.

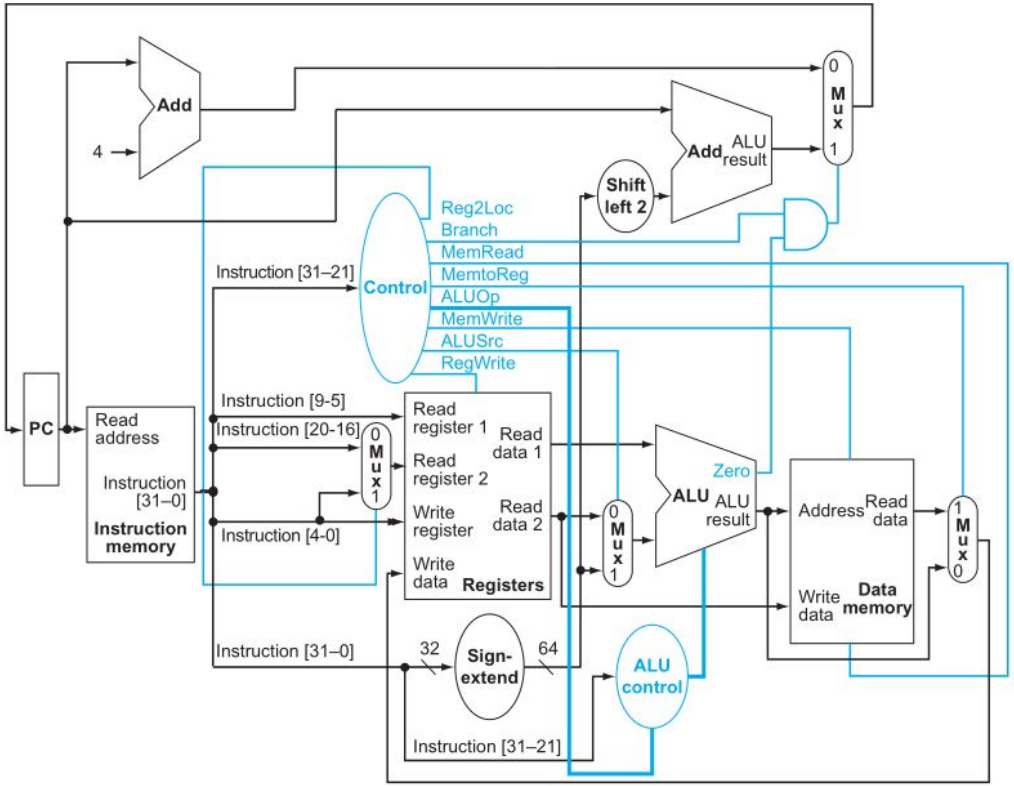
Mínimo periodo de reloj para esta implementación de la ISA (t) = 955ps = 955e-12 s

★ Se calcula la frecuencia como la inversa del periodo.

Máxima frecuencia de reloj (f) = 1/t ≈ 1.047 GHz

Ejercicio 8.2

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.



Ejercicio 8.2

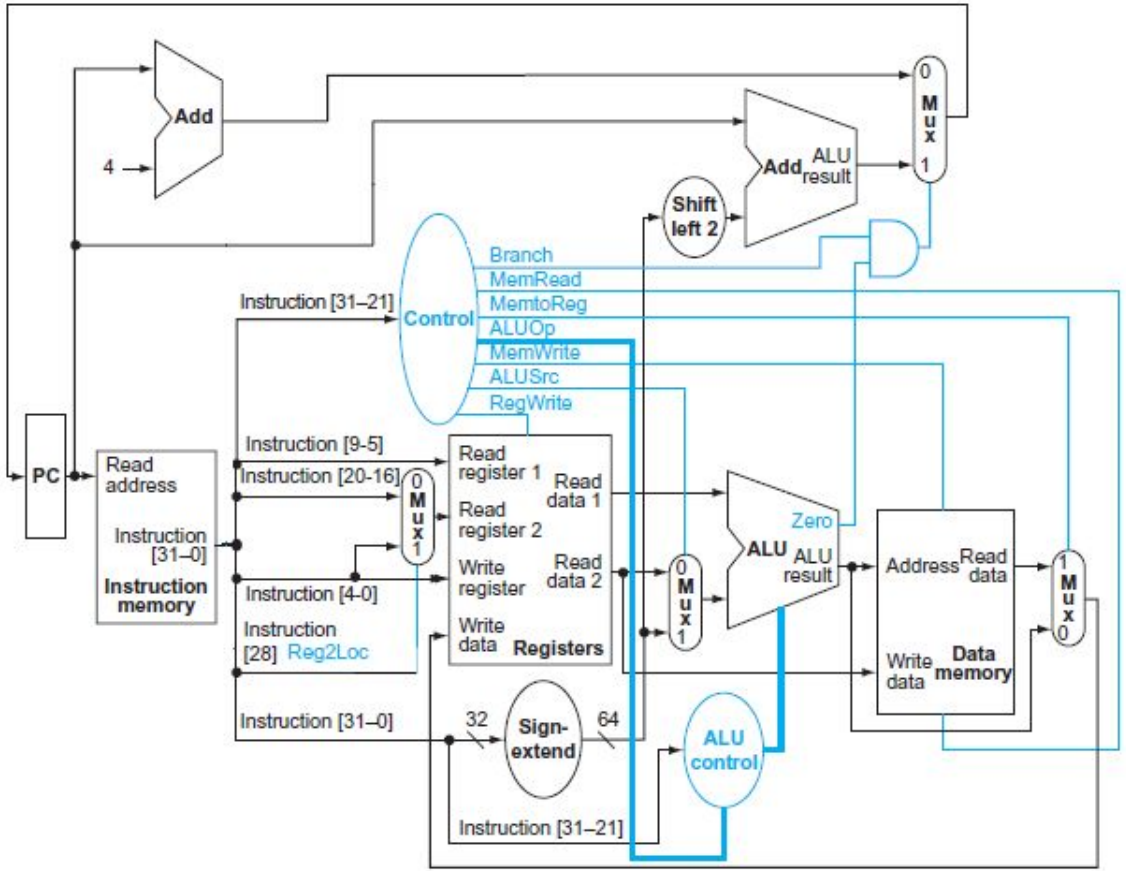
Hay un modo alternativo de generar la señal Reg2Loc fácilmente de la instrucción *sin tener que esperar la latencia de Control*. Explique cómo sería. Ignore la instrucción STXR.

El bit 28 de las instrucciones contiene el valor correcto para la línea de control Reg2Loc

Input or output	Signal name	R-format	LDUR	STUR	CBZ
Inputs	I[31]	1	1	1	1
	I[30]	X	1	1	0
	I[29]	X	1	1	1
	I[28]	0	1	1	1
	I[27]	1	1	1	0
	I[26]	0	0	0	1
	I[25]	1	0	0	0
	I[24]	X	0	0	0
	I[23]	0	0	0	X
	I[22]	0	1	0	X
	I[21]	0	0	0	X
Outputs	Reg2Loc	0	X	1	1
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Figure 4.22: Defines the logic in the control unit as one large truth table that combines all the outputs and that uses the opcode bits as inputs. It completely specifies the control function, and we can implement it directly in gates in an automated fashion.

Ejercicio 8.2



Ejercicio 9

Para la tabla de latencias del Ejercicio 8, 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].

- ★ Si modificamos LDUR y STUR para que no haya desplazamiento, ninguna de ellas usaría la ALU. Si quitamos la ALU del camino crítico de LDUR tenemos:

Camino Crítico = Reg.Read + I-Mem + Register File + D-Mem + Mux + Reg.Setup
= 30 + 250 + 50 + 150 + 250 + 25 + 20 = **755ps**

- ★ Con esta modificación, la instrucción con mayor latencia será CBZ, con **780ps**.
- ★ Calculamos el porcentaje de aumento de la velocidad como: (periodo original / nuevo periodo) * 100% = (955ps / 780ps)*100% = **122%**

Ejercicio 10

Supongamos que podemos construir una CPU con un ciclo de reloj variable, que se puede adaptar a la latencia de cada instrucción. Cuál sería la aceleración de esta nueva CPU sobre la anterior (Ejercicio 8) 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.

★ Tiempo promedio por instrucción = 0.52*775ps + 0.25*955ps + 0.1*955ps + (0.11+0.02)*780ps = **838.65ps**

★ Aceleración de la nueva CPU = periodo original / nuevo periodo
= 955ps / 838.65ps ≈ **1.139**

Ejercicio 11

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] está en el rango 0x5B8-0x5BF, obtenemos del Control

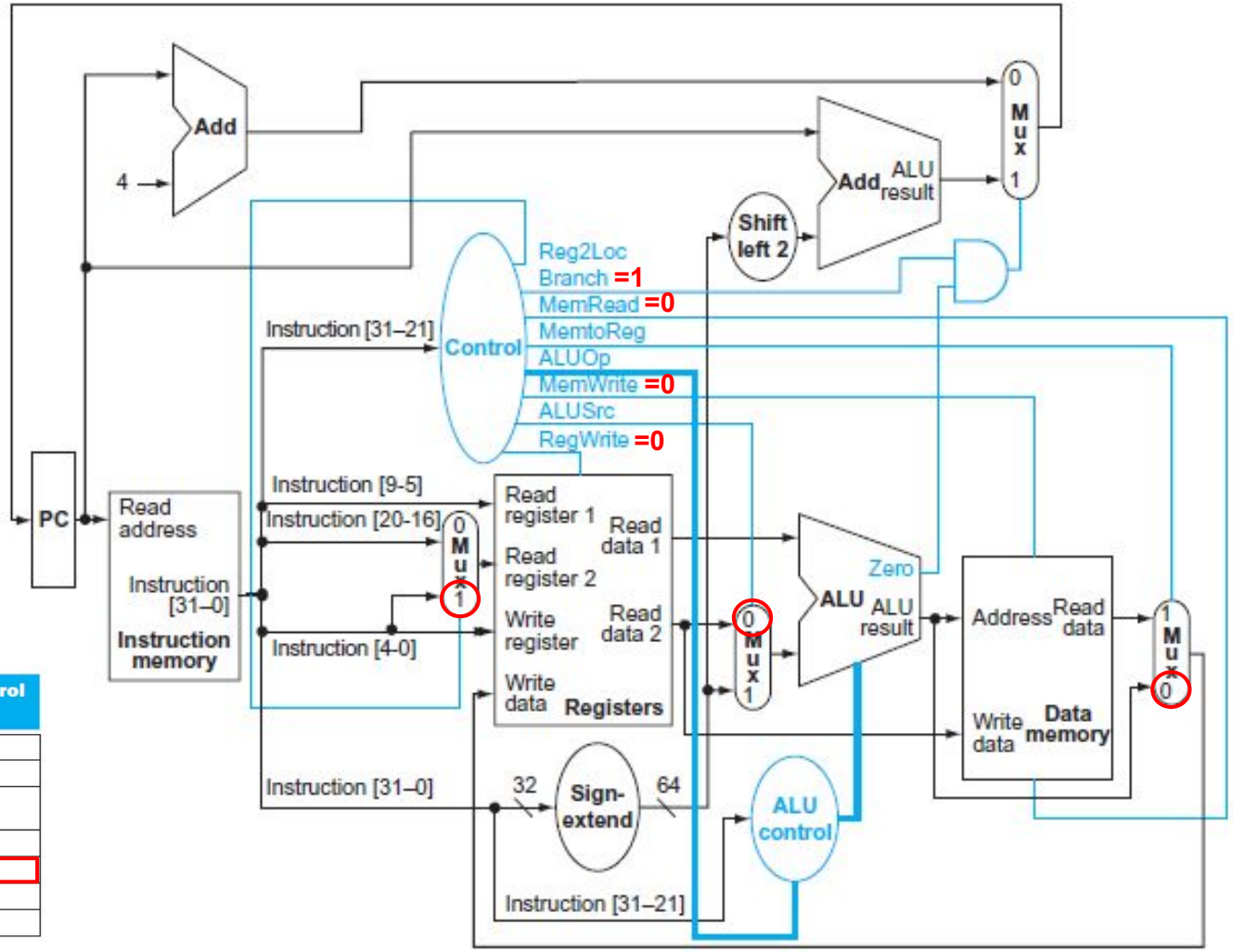
Reg2Loc	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	AluOp1	AluOp0
1	0	0	0	0	0	1	1	1

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 card.

Ejercicio 11



Instruction	ALUOp	Desired ALU action	ALU control input
LDUR	00	add	0010
STUR	00	add	0010
CBZ	01	pass input b	0111
R-type	10	add	0010
R-type	10	subtract	0110
R-type	10	AND	0000
R-type	10	OR	0001

Ejercicio 11

- 0x5B8-0x5BF es un opcode no usado en LEGv8.
- No se escriben registros ni la memoria de datos.
- Se leen 2 registros, ambos entran a la ALU y se hace la resta entre ellos.
- Si la resta da cero (si R1 = R2), como Branch = 1, salta a lo que haya en Sign-extend con Sl2 + PC.
- Como Reg2Loc = 1, los bits 9-5 se utilizan para direccionar el segundo registro.Si asumimos que en el módulo Sign-extend esta instrucción se interpreta como un formato CB, esos bits también se utilizan como parte del campo “COND_BR_address” para calcular el desplazamiento del branch.

★Operación realizada: compara y salta si el registro 1 es igual al registro 2.

★Mnemónico propuesto: **CBEQ**

Ejercicio 12 (opcional)

Mostrar cómo se implementan los módulos:

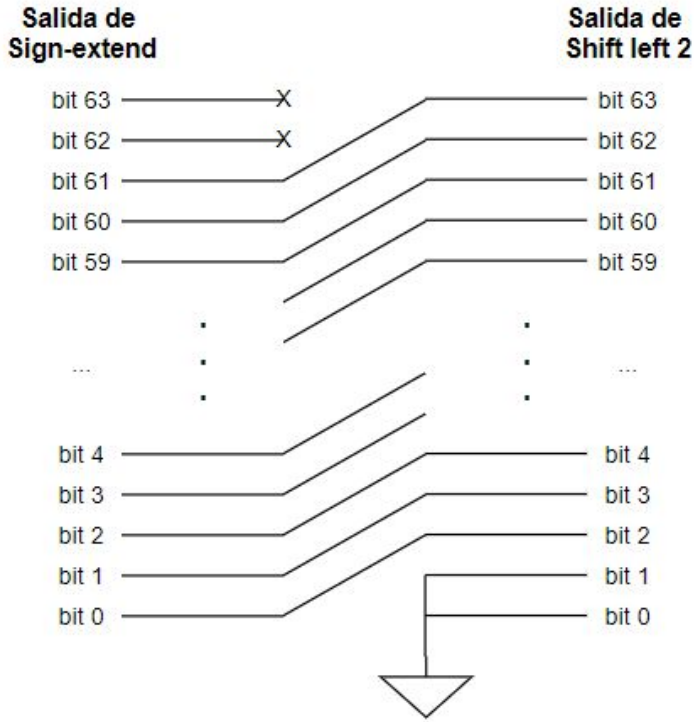
12.1) Shift-left-2: 64 bits de entrada, 64 bits de salida.

12.2) Sign-extend: 32 bits de entrada, 64 bits de salida.

12.3) Obtenga una implementación de la composición (Sign-extend; Shift-left-2) en un único circuito.

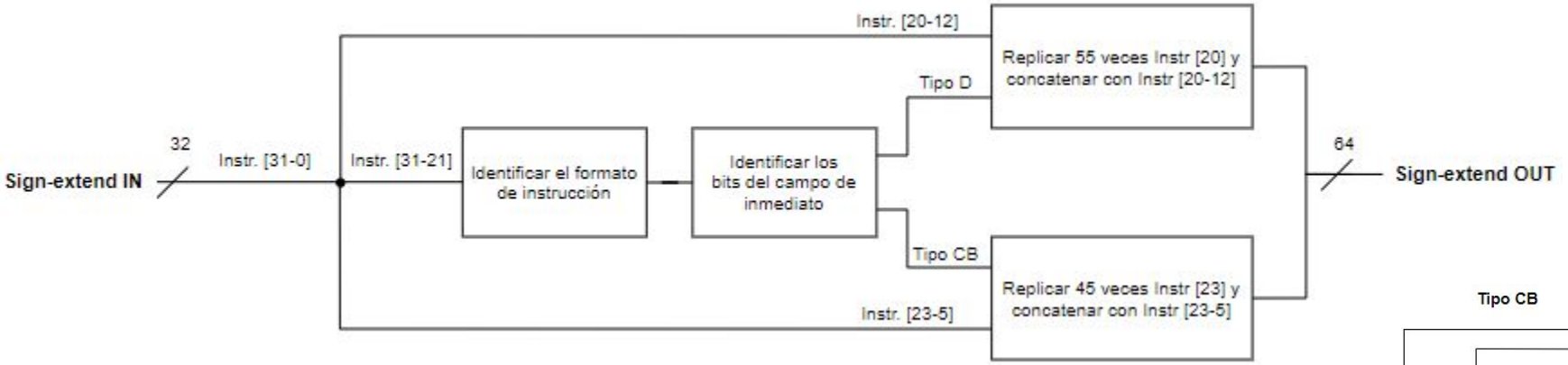
Ejercicio 12.1

Shift-left-2: 64 bits de entrada, 64 bits de salida.

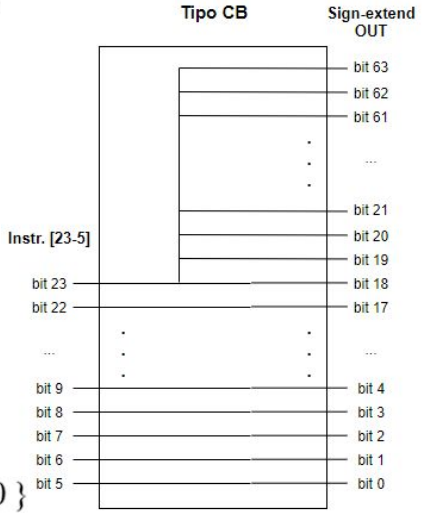


Ejercicio 12.2

Sign-extend: 32 bits de entrada, 64 bits de salida.




```
module signext (input logic [31:0] a,
                output logic [63:0] y);
always_comb
   casez(a[31:21])
      // LDUR:
      11'b11111000010: y = {{55{a[20]}}, a[20:12]};
      // STUR:
      11'b11111000000: y = {{55{a[20]}}, a[20:12]};
      // CBZ:
      11'b10110100???: y = {{45{a[23]}}, a[23:5]};
      // undefined:
      default: y = '0;
   endcase
endmodule
```



(4) CondBranchAddr = { 43{COND_BR_address [18]}, COND_BR_address, 2'b0 }

Ejercicio 13 (opcional)

Muestre el circuito interno de la ALU que, a partir de los 64 bits de salida, produce la salida Zero.