Un microprocesador se descompone en dos partes básicas:

Unidad de flujo de datos
- Unidad aritmético-lógica (ALU)
- Registro acumulador (Acc)
- Registros de propósito general

Unidad de control
- Decodificador de instrucciones
- Registro contador de Programa (PC)
- Registro de instrucciones (IR)

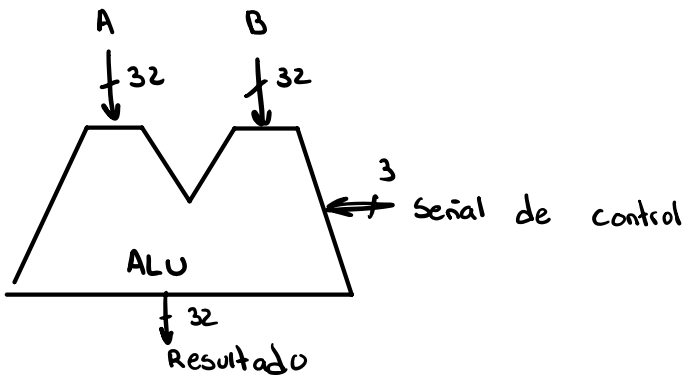
Código que se desea correr en arquitectura Riscv:

```
int main(){
   10188:       fe010113                addi    sp,sp,-32
   1018c:       00812e23                sw      s0,28(sp)
   10190:       02010413                addi    s0,sp,32
  int *r = 0xC7DF;
   10194:       0000c7b7                lui     a5,0xc
   10198:       7df78793                addi    a5,a5,2015 # c7df <exit-0x38b5>
   1019c:       fef42223                sw      a5,-28(s0)
  char a = 'c';
   101a0:       06300793                li      a5,99
   101a4:       fef401a3                sb      a5,-29(s0)
  int b = 33;
   101a8:       02100793                li      a5,33
   101ac:       fef42623                sw      a5,-20(s0)

  for (int i=0; i<2; i++) {
   101b0:       fe042423                sw      zero,-24(s0)
   101b4:       03c0006f                j       101f0 <main+0x68>
    if (i==0){
   101b8:       fe842783                lw      a5,-24(s0)
   101bc:       00079863                bnez    a5,101cc <main+0x44>
      a = 'b';
   101c0:       06200793                li      a5,98
   101c4:       fef401a3                sb      a5,-29(s0)
   101c8:       01c0006f                j       101e4 <main+0x5c>
    }
    else{
      b = b<<1;
   101cc:       fec42783                lw      a5,-20(s0)
   101d0:       00179793                slli    a5,a5,0x1
   101d4:       fef42623                sw      a5,-20(s0)
      b = b&0xF;
   101d8:       fec42783                lw      a5,-20(s0)
   101dc:       00f7f793                andi    a5,a5,15
   101e0:       fef42623                sw      a5,-20(s0)
  for (int i=0; i<2; i++) {
   101e4:       fe842783                lw      a5,-24(s0)
   101e8:       00178793                addi    a5,a5,1
   101ec:       fef42423                sw      a5,-24(s0)
   101f0:       fe842703                lw      a4,-24(s0)
   101f4:       00100793                li      a5,1
   101f8:       fce7d0e3                bge     a5,a4,101b8 <main+0x30>
   101fc:       00000793                li      a5,0
    }

  }
}
   10200:       00078513                mv      a0,a5
   10204:       01c12403                lw      s0,28(sp)
   10208:       02010113                addi    sp,sp,32
   1020c:       00008067                ret
```

Qué se necesita de la ALU?

Para correr este código se necesita que la ALU soporte las operaciones:

- add, suma entre enteros de 32 bits
- sll, shif izquierdo o multiplicación por 2
- and, and lógico
- no igual a cero (neZ), comparación lógica
- mayor o igual (ge), comparación lógica



Qué formatos de funciones se necesitan?

Tipo I (inmediato)

Tipo S (store/load)

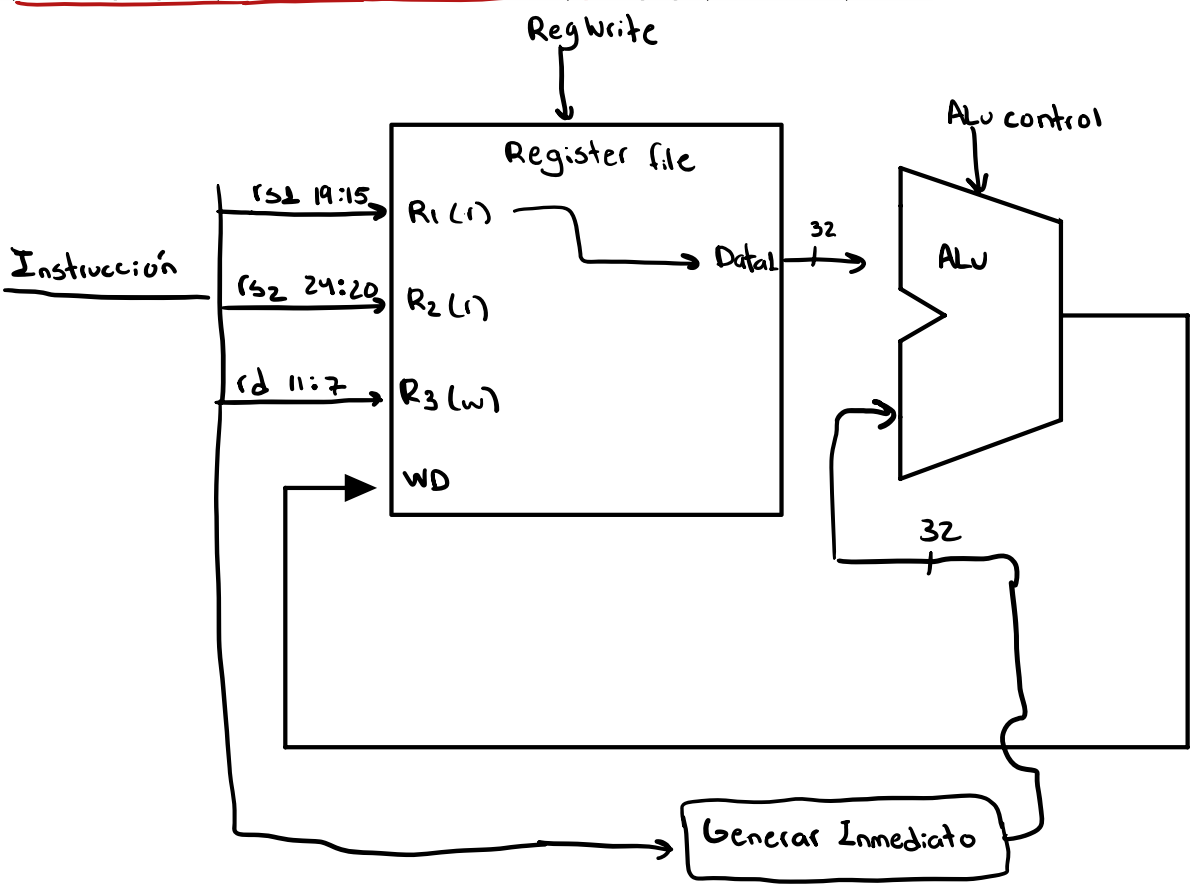
Tipo B (branch condicional)

Tipo U (upper immediate)

Tipo J (Jump)

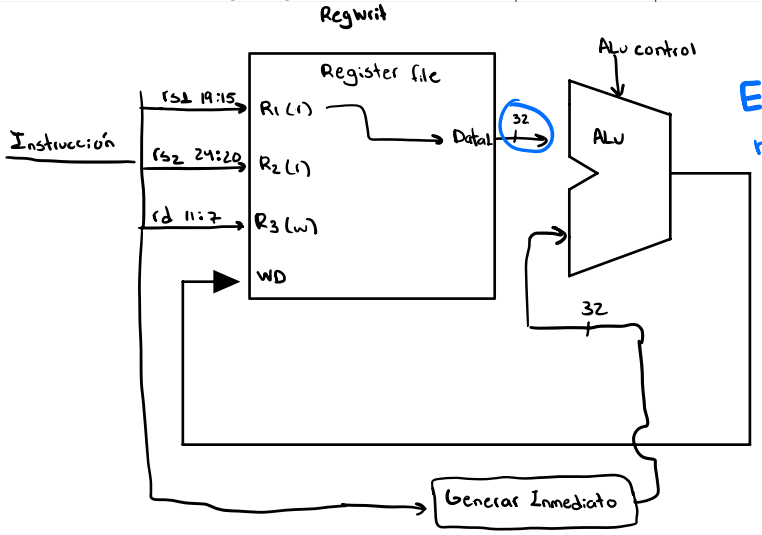
"Data path" para instrucciones de tipo I:

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type



Data path para instrucciones tipo U

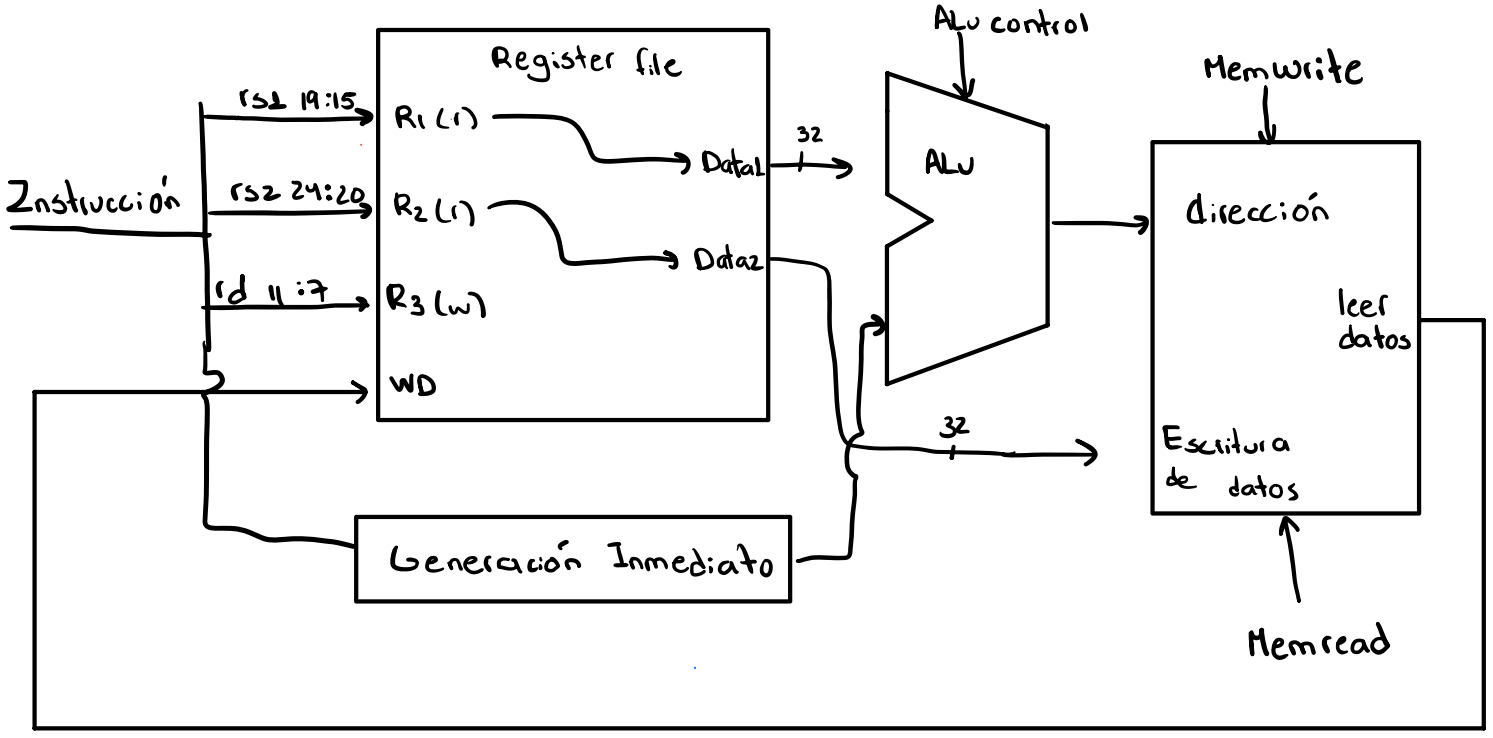
31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type



En operaciones tipo U esto no importa

"Data path" para instrucciones tipo S/L

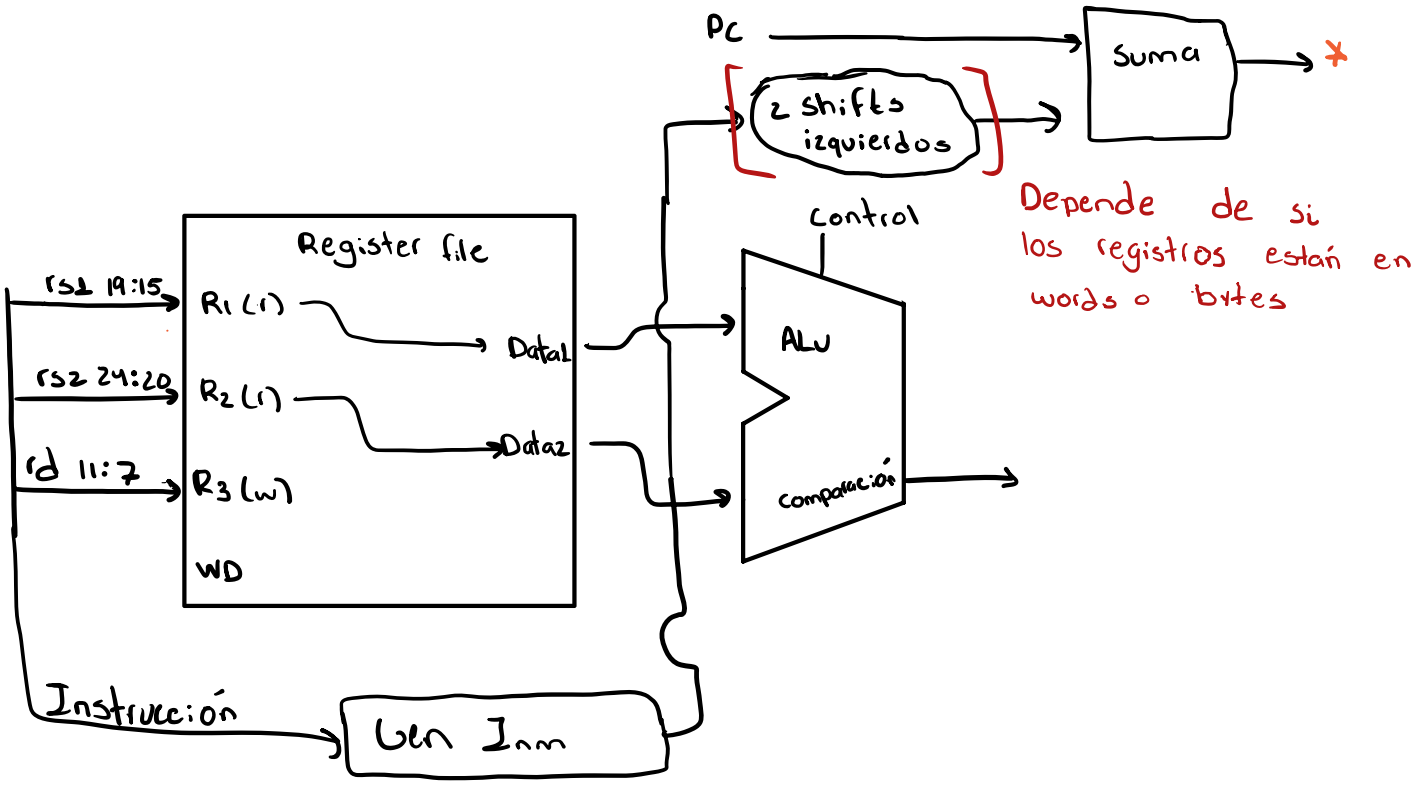
31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type



Si Memread = 1 se lee la dirección y sale leer datos, si Memwrite=1 Entra dirección y los datos a escribir y se hace store

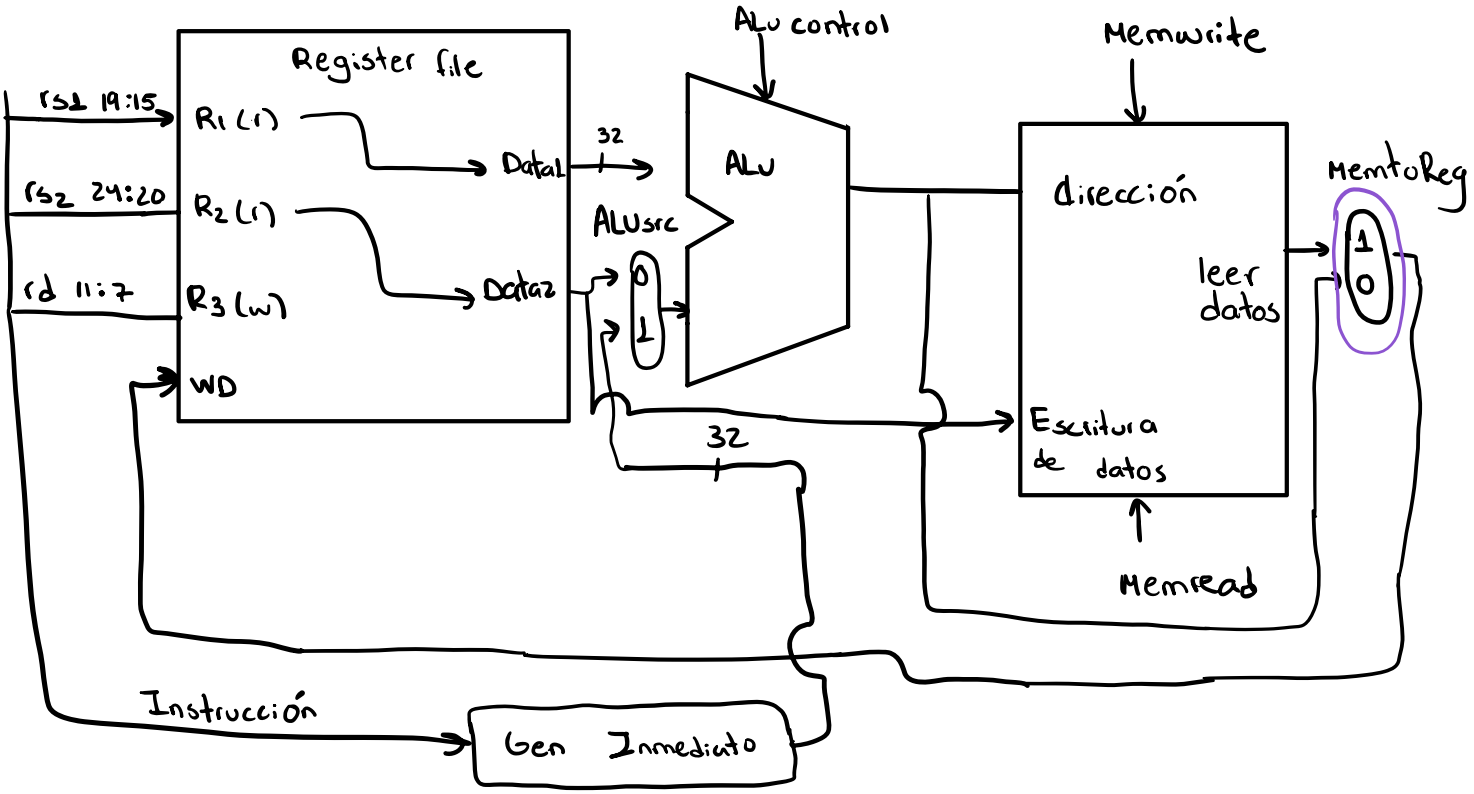
"Data paths" para instrucciones tipo B

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

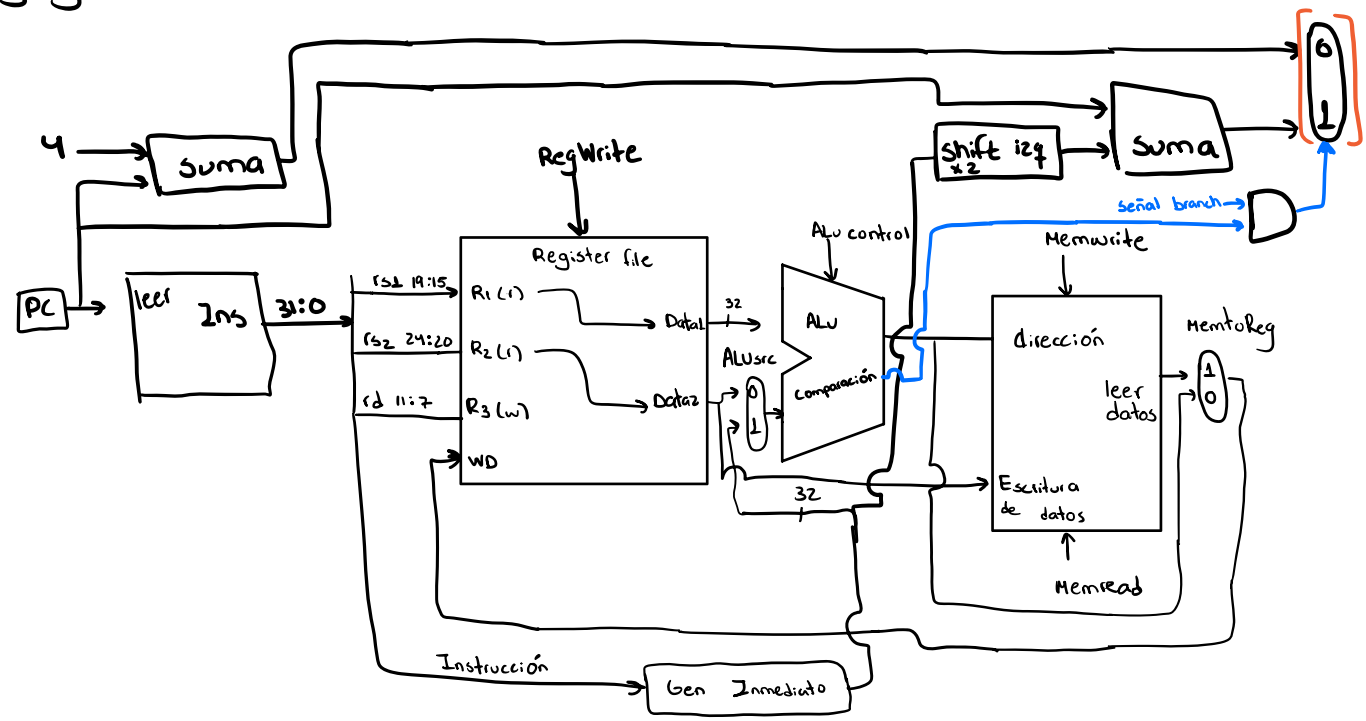


Salida para ser usada en caso de que la condición se cumpla.

Unión de data paths I y S/L



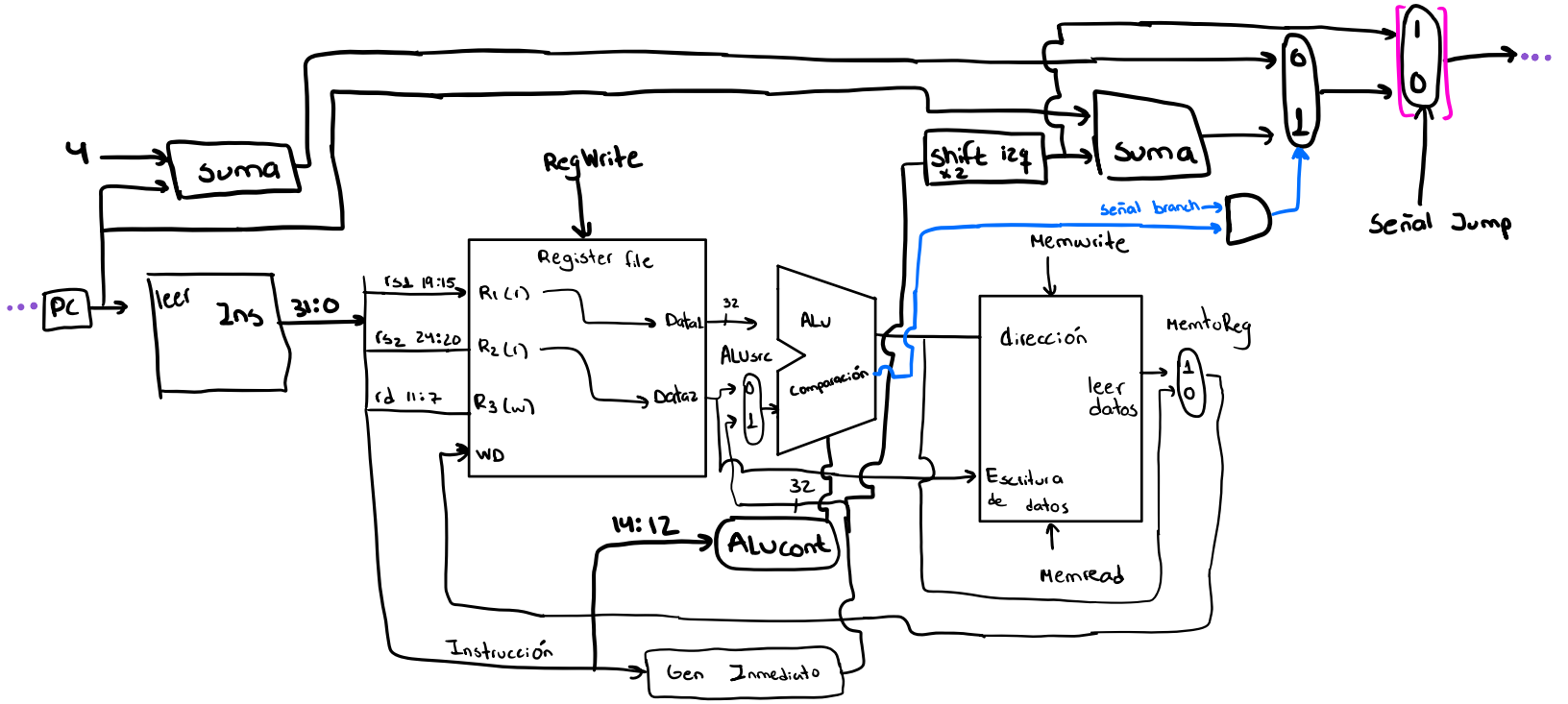
Agregar branch y módulo de instrucciones



Mux que se encarga de escoger entre la siguiente instrucción secuencial o el target branch

Se añade soporte para instrucción Jump:

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type



Este Mux controla si es branch/secuencial o jump

RV32I Base Instruction Set

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
fm | pred	succ	rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

Tabla para unidad de control

OP		0110111	1101111	1100011	0000011	0100011	0010011
outputs	AluSrc	1	1	0	1	1	1
	MemtoReg	0	0	x	1	x	0
	RegWrite	1	0	x	1	0	1
	MemWrite	0	0	x	0	1	0
	MemRead	0	0	x	1	0	0
	Branch	0	0	1	0	0	0
	Jump	0	1	0	0	0	0
	ALUOP	000	xxx	001	010	011	100

Diseño del control de la ALU

ALUOP	funct3	out
000	xxx	101
001	001	011
001	101	100
010	010	000
011	010	000
100	000	000
100	111	010
100	001	001