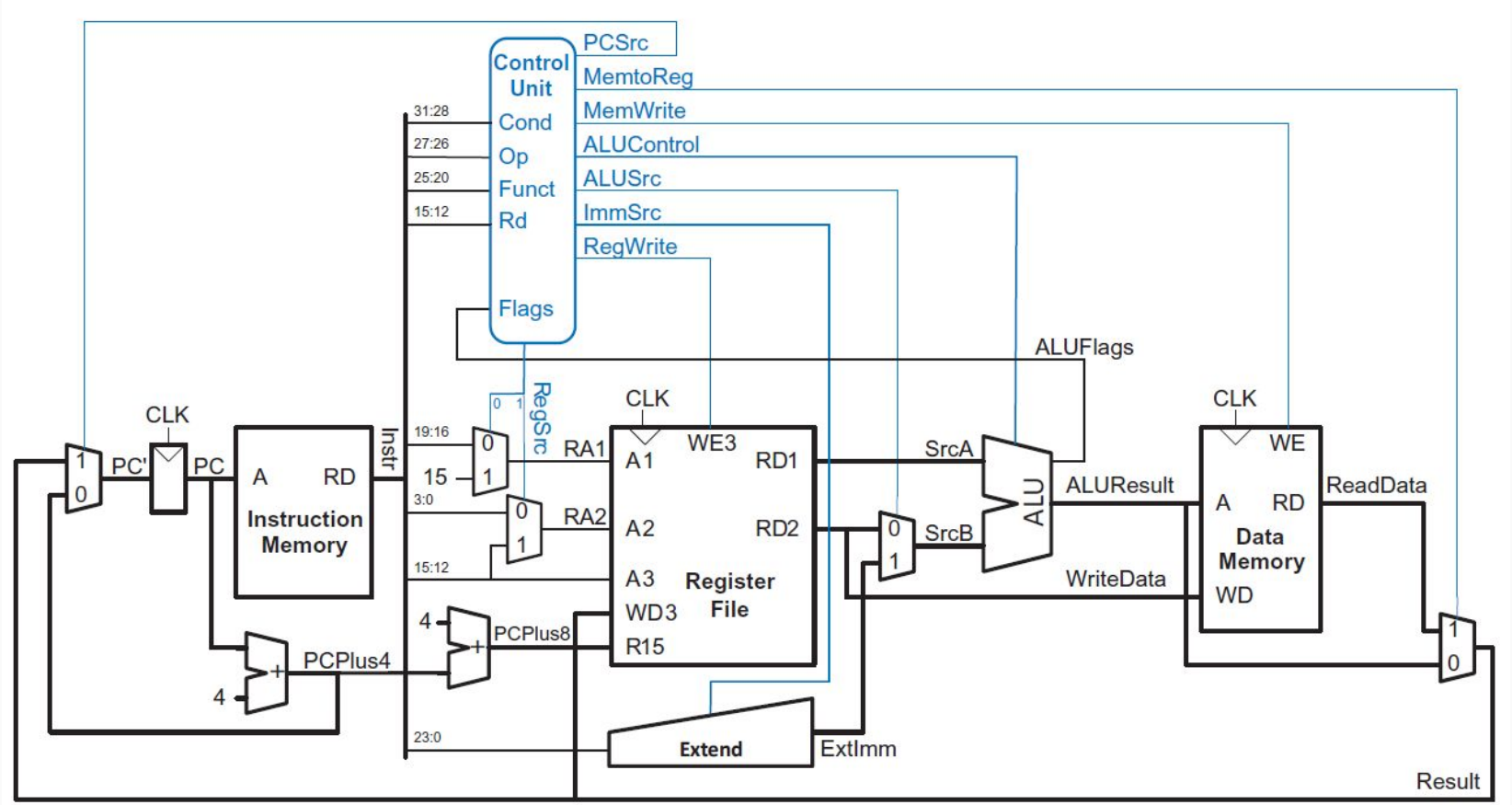
Implementación de la ISA + Ensamblado y desensamblado de LEGv8

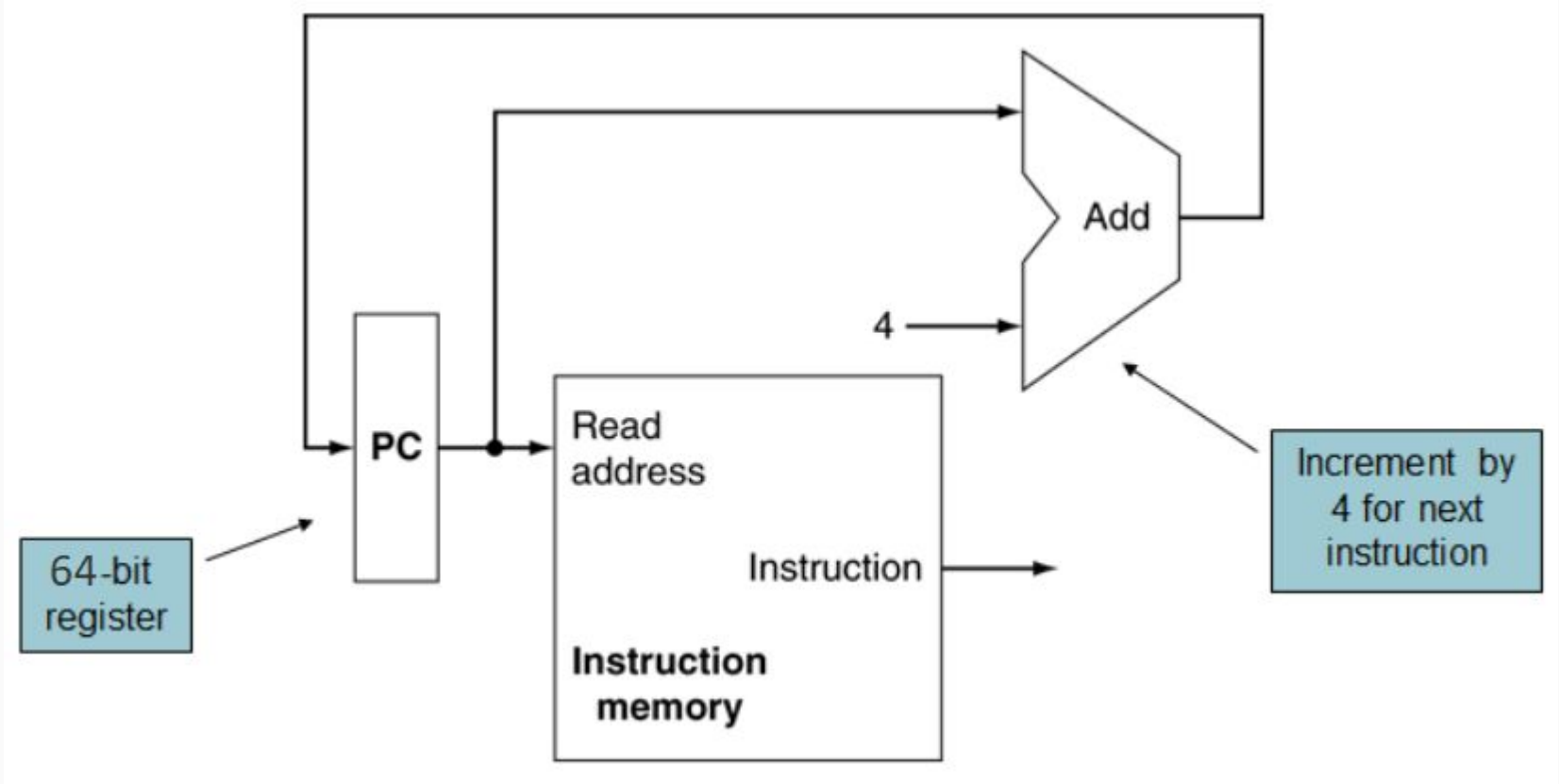
Arquitectura de Computadoras II - 2023

Dr. Ing. Agustín Laprovitta (alaprovitta@ucc.edu.ar)
Ing. Delfina Velez Ibarra (0703883@ucc.edu.ar)

ARMv4: Complete single-cycle processor

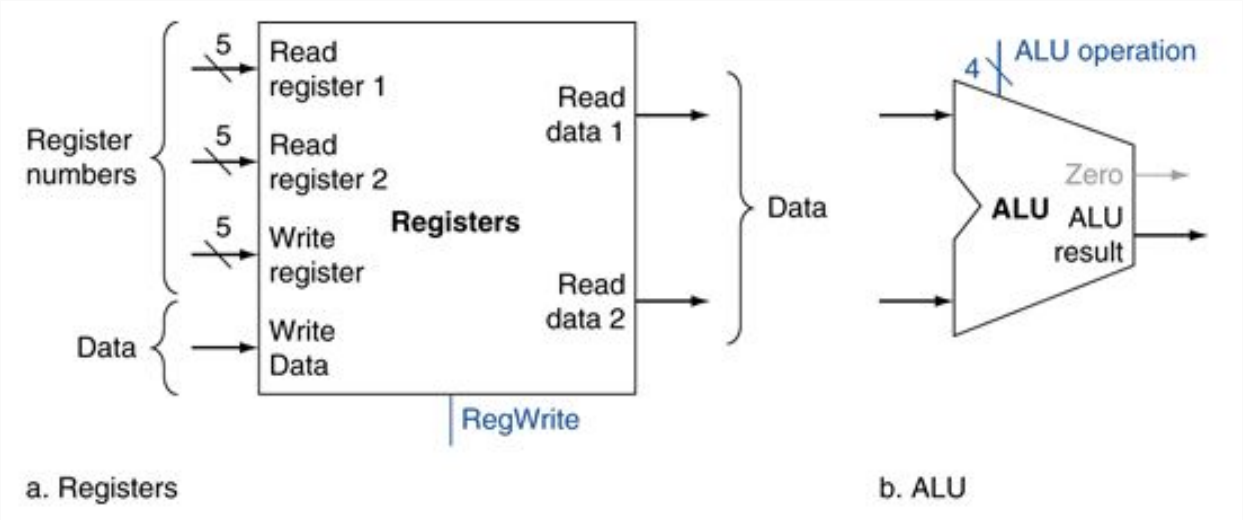


LEGv8: Instruction Fetch



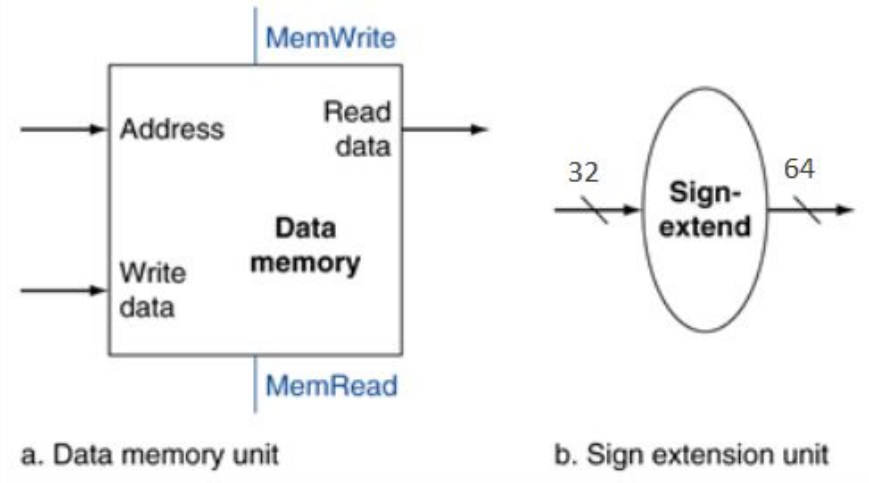
LEGv8: R-Format Instructions

- Read two register operands
- Perform arithmetic/logical operation
- Write register result

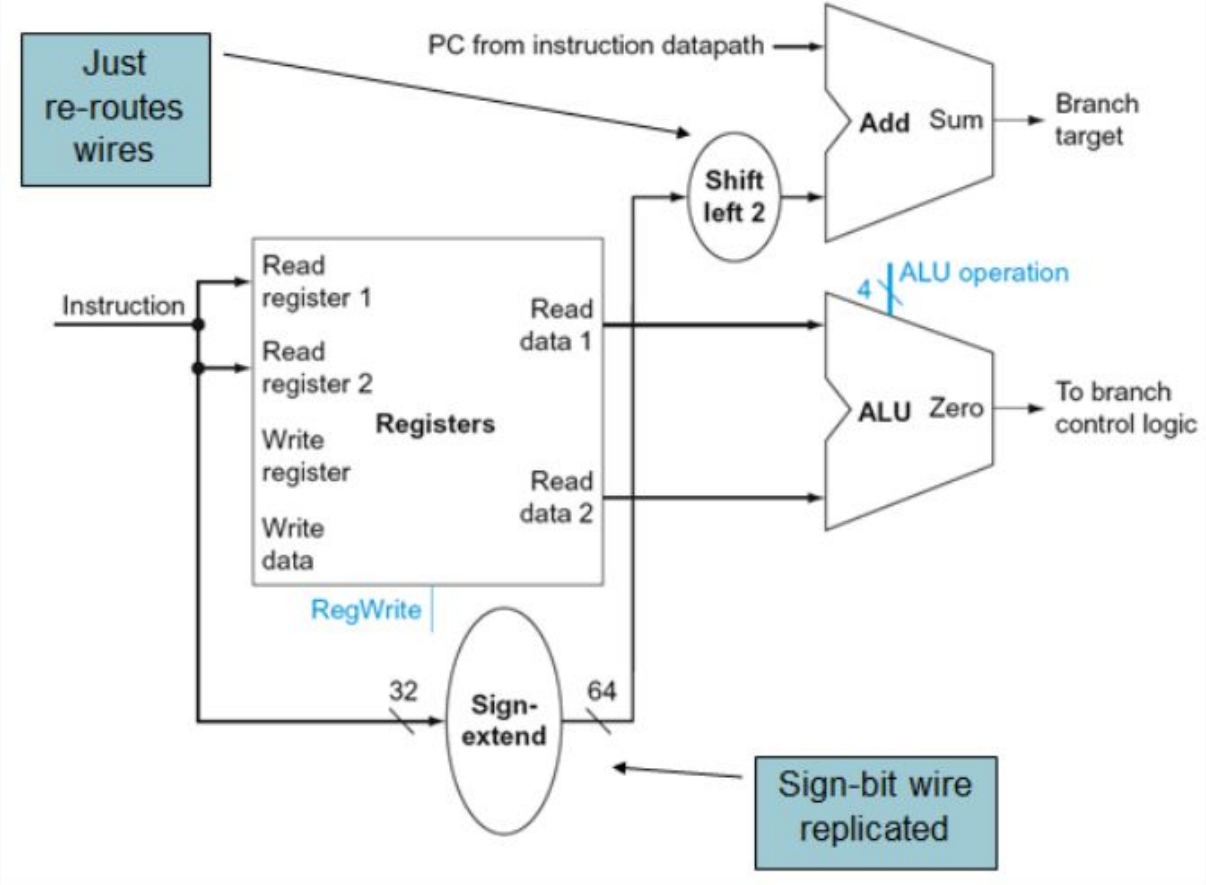


LEGv8: Load/Store Instructions

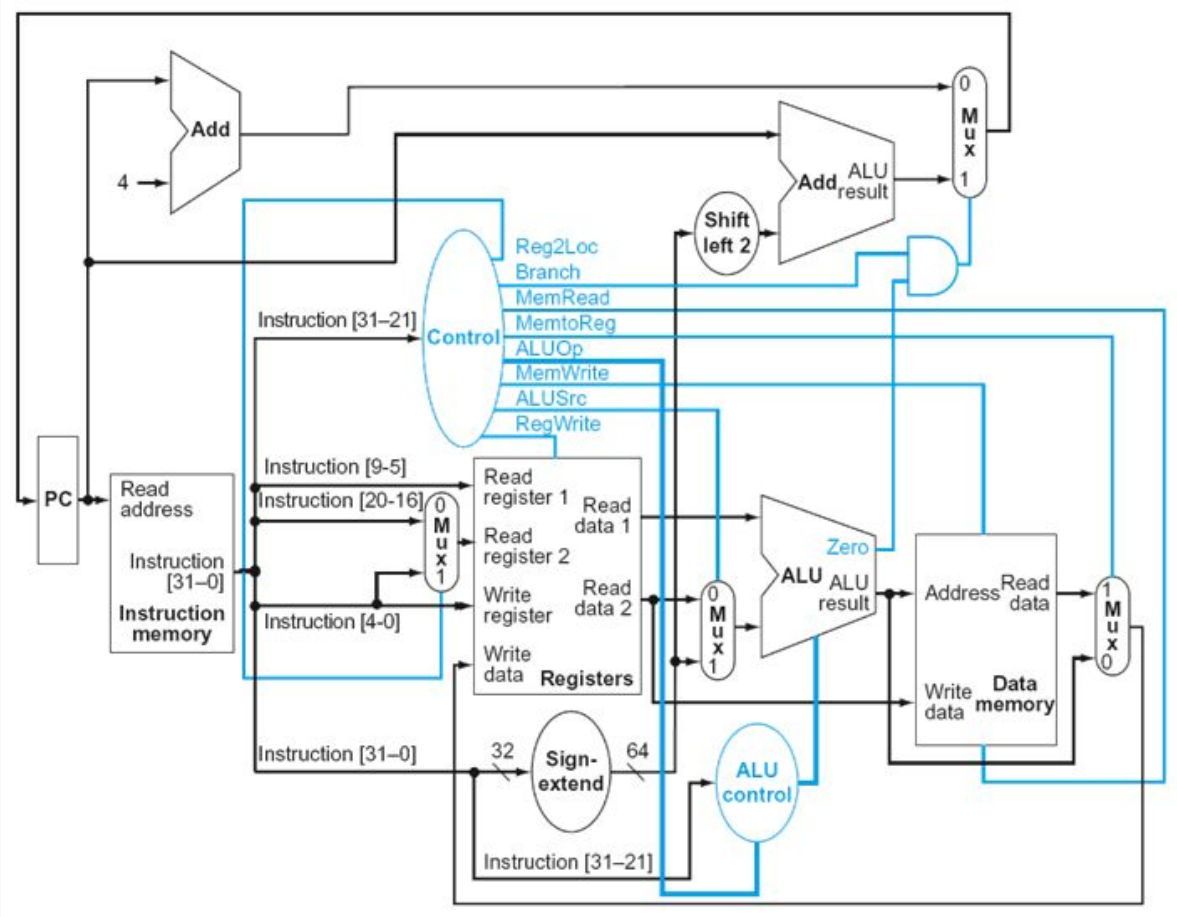
- Read register operands
- Calculate address using 9-bit offset (use ALU)
- Load: Read memory and update register
- Store: Write register value to memory



LEGv8: Branch Instructions



LEGv8: The simple datapath with the control unit



- La entrada a la unidad de control son los 11 bits de opcode de la instrucción.
- Las salidas de la unidad de control son similares, pero la operación de la ALU se determina en una unidad separada (ALU control).
- La única instrucción de salto implementada es CBZ.
- El bloque Sign-extend recibe la instrucción completa.
- La entrada a Read register 1 no está multiplexada.
- Posee 32 registros de 64 bits.

LEGv8: ALU Control

- ALU used for
  - Load/Store: F = add
  - Branch: F = subtract
  - R-type: F depends on opcode

opcode	ALUOp	Operation	Opcode field	ALU function	ALU control
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	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		ORR	100101	OR	0001

LEGv8: Setting of the control lines

Instruction	Reg2Loc	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-format	0	0	0	1	0	0	0	1	0
LDUR	X	1	1	1	1	0	0	0	0
STUR	1	1	X	0	0	1	0	0	0
CBZ	1	0	X	0	0	0	1	0	1

Arithmetic Operations

Three operands: two sources and one destination

ADD a, b, c // a gets b + c

- C code:

```
f = (g + h) - (i + j); // g->X20, h->X21, i->X22,
                       // j->X23, f->X24
```

- Compiled LEGv8 code:

```
???
```

Arithmetic Operations

Three operands: two sources and one destination
ADD a, b, c // a gets b + c

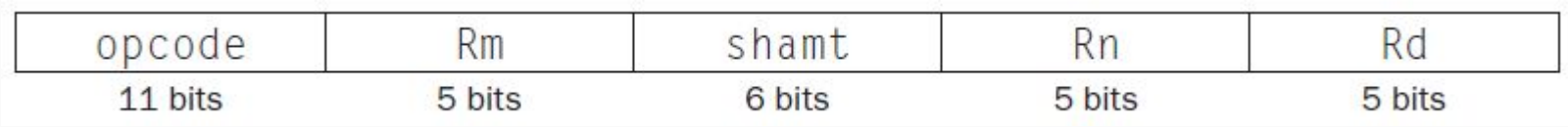
- C code:

```
f = (g + h) - (i + j); // g->X20, h->X21, i->X22, j->X23
```

- Compiled LEGv8 code:

```
ADD X0, X20, X21      // X0 = g + h
ADD X1, X22, X23  // X1 = i + j
SUB X24, X0, X1       // f = X0 - X1
```

LEGv8 R-format Instructions



opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits

- Instruction fields
  - opcode: operation code
  - Rm: the second register source operand
  - shamt: shift amount (00000 for now)
  - Rn: the first register source operand
  - Rd: the register destination

R-format Example

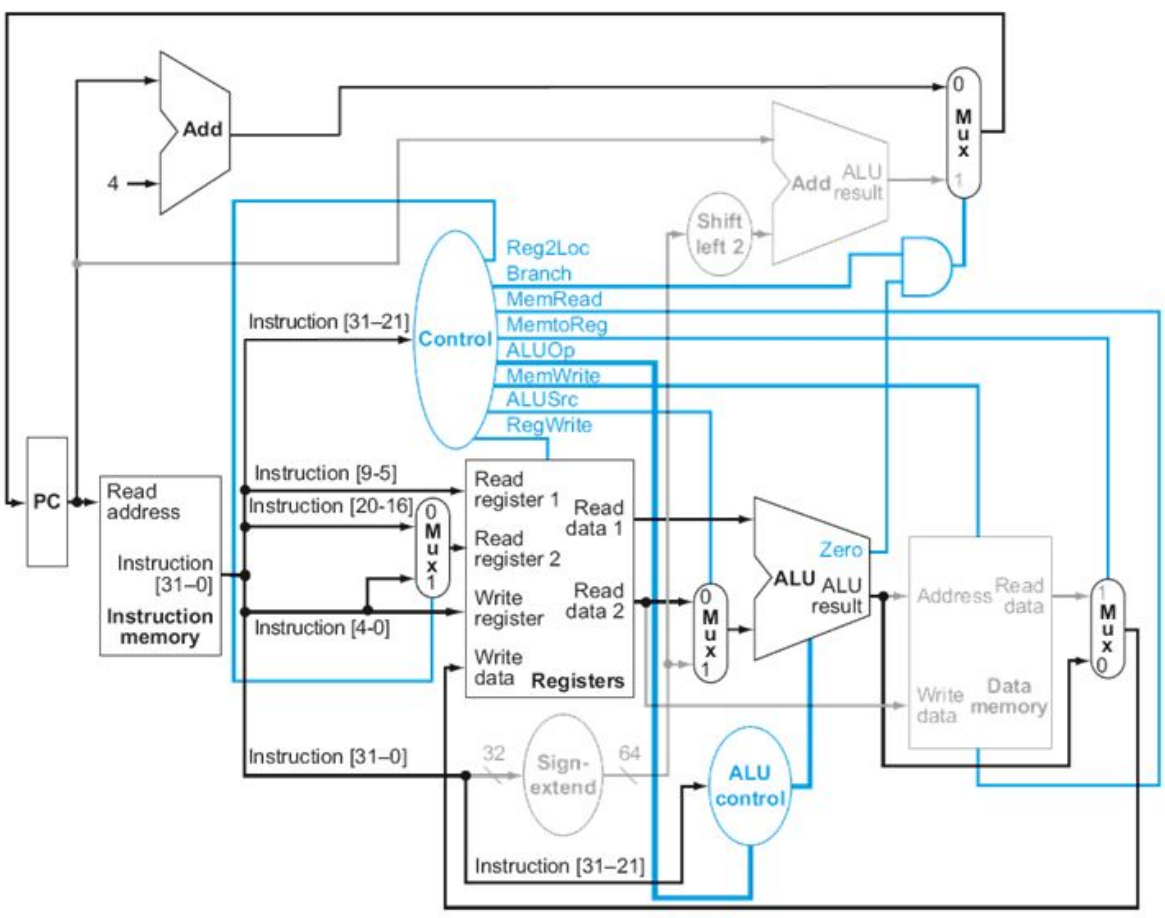
ADD X9,X20,X21

1112_{ten}	21_{ten}	0_{ten}	20_{ten}	9_{ten}
10001011000_{two}	10101_{two}	000000_{two}	10100_{two}	01001_{two}

$$1000\ 1011\ 0001\ 0101\ 0000\ 0010\ 1000\ 1001_{two} =$$

$$8B150289_{16}$$

R-Type Instruction



Memory Operands

Load values from memory into registers
Store result from register to memory
Memory is byte addressed

- C code: (elementos de A = DobleWord)
  `A[12] = h + A[8]; // h->X21, base address of A->X22`

- Compiled LEGv8 code:
  `???`

Memory Operands

Load values from memory into registers
Store result from register to memory
Memory is byte addressed

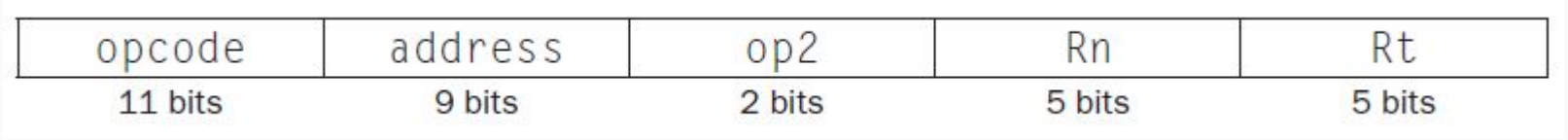
- C code:

```
A[12] = h + A[8]; // h->X21, base address of A->X22
```

- Compiled LEGv8 code:

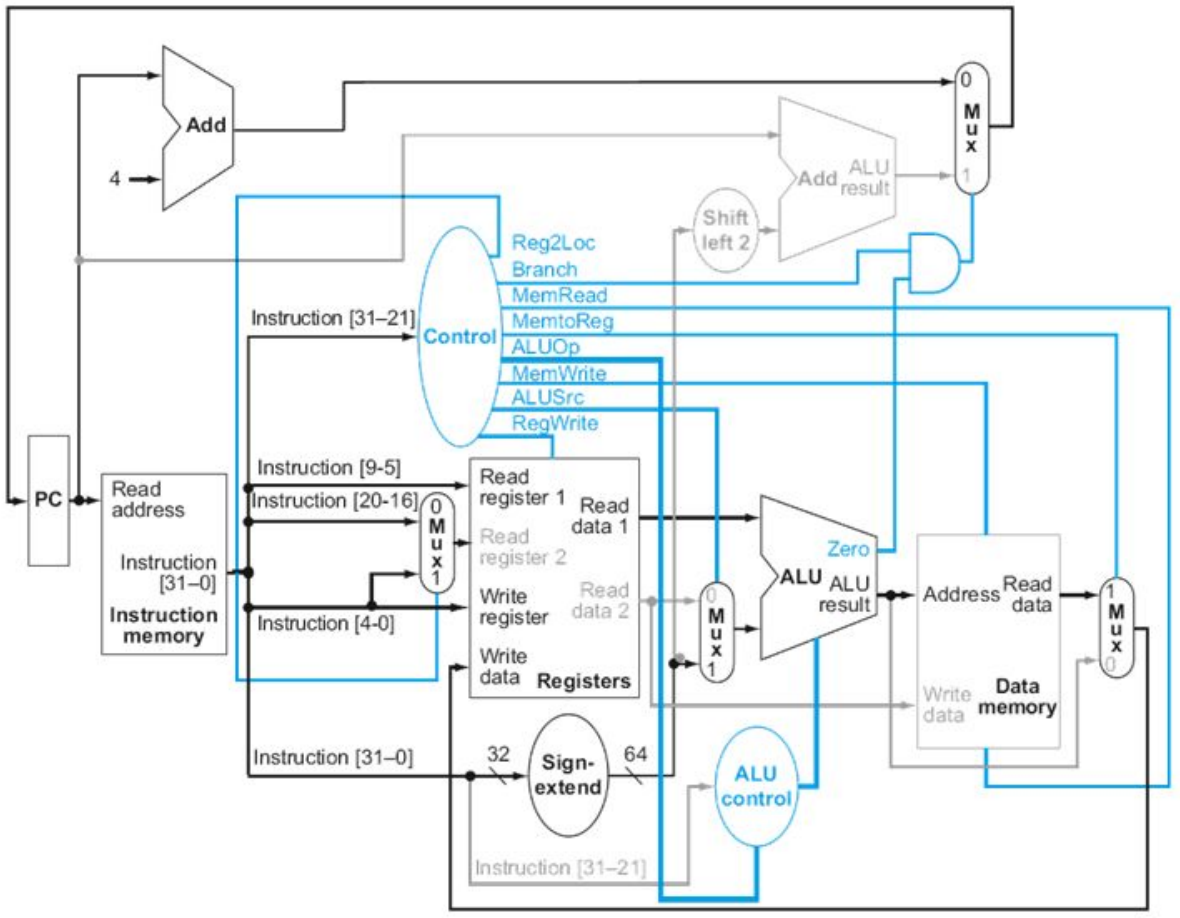
```
LDUR    X9, [X22,#64] // Index 8 requires offset of 64
ADD     X9, X21, X9
STUR    X9, [X22,#96]
```

LEGv8 D-format Instructions



- Instruction fields
  - Rn: base register
  - address: constant offset from contents of base register (-256 to 255)
  - Rt: destination (load) or source (store) register number
  - op2 = “00”

Load Instruction



Branch Operations

Branch to a labeled instruction if a **condition** is true. Otherwise, continue sequentially:

- CBZ register, L1

  if (register == 0) branch to instruction labeled L1;

- CBNZ register, L1

  if (register != 0) branch to instruction labeled L1;

Branch **unconditionally** to instruction labeled L1:

- B L1

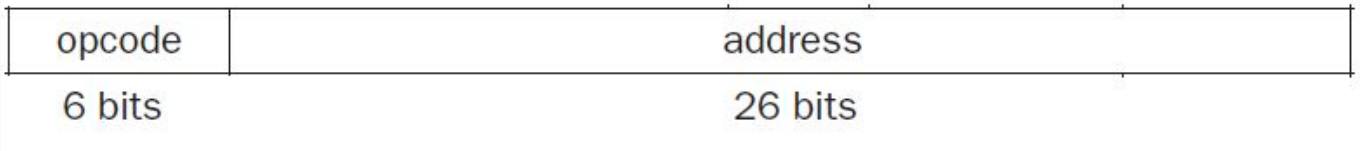
Branch Addressing

- CB-type

```
CBNZ X19, Exit     // go to Exit if X19 != 0
```

- B-type

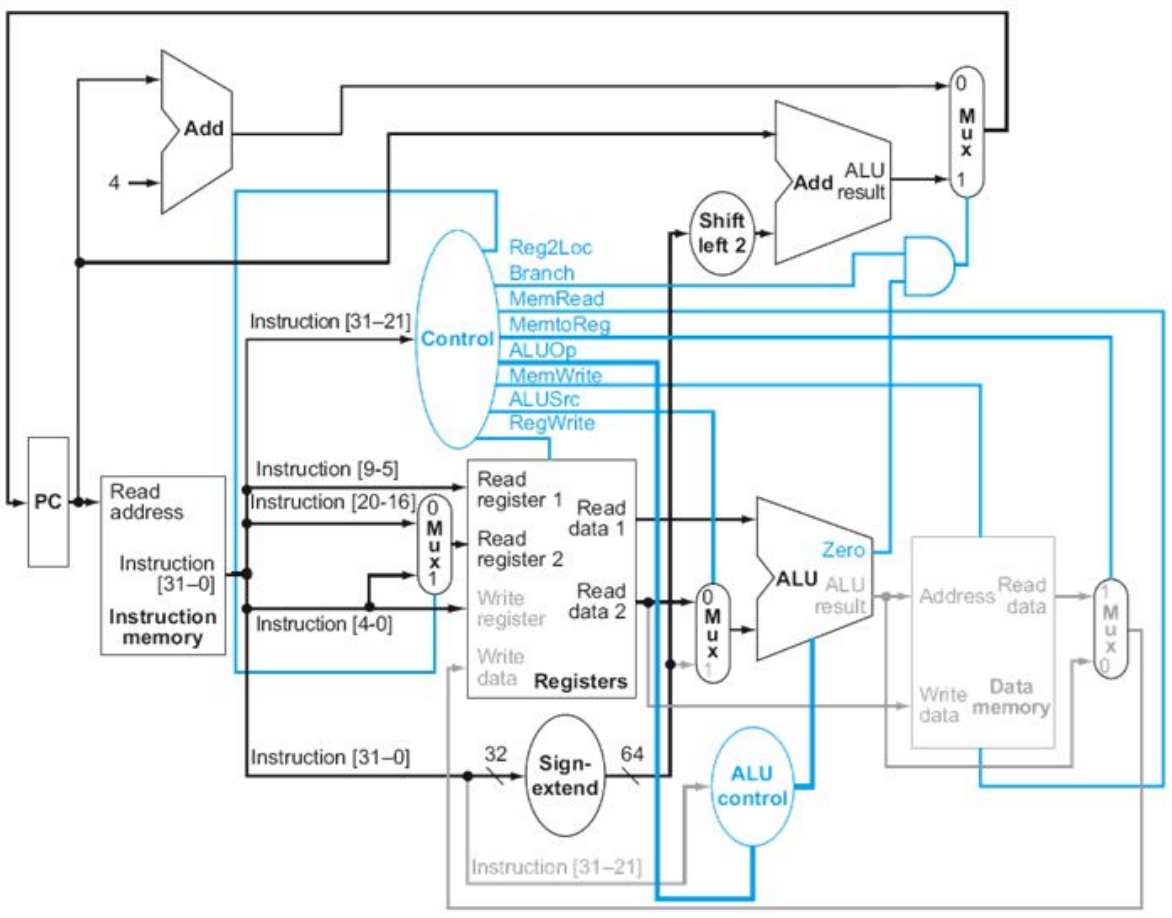
```
B loop      // go to loop
```



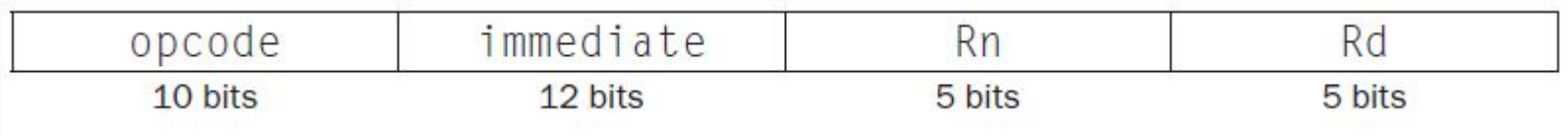
Both addresses (n° instructions) are PC-relative

New_PC = Current_PC + (n° instructions * 4)

CBZ Instruction



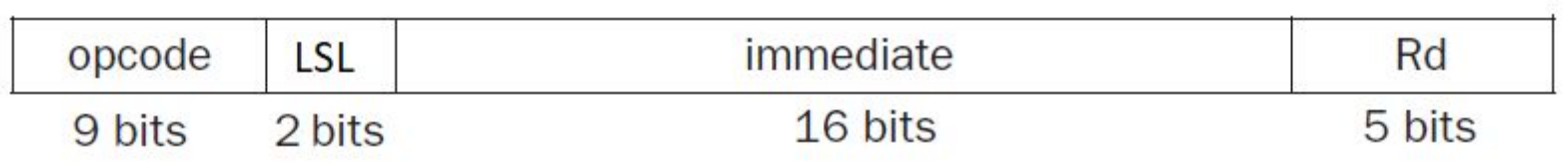
LEGv8 I-format Instructions



- Instruction fields
  - Rn: source register
  - Rd: destination register
  - immediate is unsigned

LEGv8 IW-format Instructions

The instruction MOVZ transfers a 16-bit immediate constant field value into one of the four quadrants leftmost of a 64-bit register, filling the other 48 bits with 0s. The instruction MOVK only changes 16 bits of the register, keeping the other bits the same.



opcode	LSL	immediate	Rd
9 bits	2 bits	16 bits	5 bits

- Instruction fields
  - Rd: destination register
  - immediate is unsigned
  - LSL:

Bit 22	Bit 21	LSL
0	0	0
0	1	16
1	0	32
1	1	48

Move wide with zeros (MOVZ) and move wide with keep (MOVK)

The machine language version of `MOVZ X9, 255, LSL 16`:

110100101	01	0000 0000 1111 1111	01001

Contents of register `X9` after executing `MOVZ X9, 255, LSL 16`:

0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 1111 1111	0000 0000 0000 0000

The machine language version of `MOVK X9, 255, LSL 0`:

111100101	00	0000 0000 1111 1111	01001

Given value of `X9` above, new contents of `X9` after executing `MOVK X9, 255, LSL 0`:

0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 1111 1111	0000 0000 1111 1111