Lecture 6
Integer Arithmetic

Byung-gi Kim

School of Computer Science and Engineering

Soongsil University

3. Arithmetic for Computers

3.1 Introduction
3.2 Addition and Subtraction
3.3 Multiplication
3.4 Division
3.5 Floating Point
3.6 Parallelism and Computer Arithmetic: Associativity
3.7 Real Stuff: Floating Point in the x86
3.8 Fallacies and Pitfalls
3.9 Concluding Remarks
3.10 Historical Perspective and Further Reading
3.11 Exercises

3.1 Introduction

- **Operations on integers**
  - Addition and subtraction
  - Multiplication and division
  - Dealing with overflow
- **Floating-point real numbers**
  - Representation and operations

3.2 Addition and Subtraction

- **The binary number**

Least significant bit (LSB)

Most significant bit (MSB)

01011000 00010101 00101110 11100111

represents the quantity

$0 \times 2^{31} + 1 \times 2^{30} + 0 \times 2^{29} + \ldots + 1 \times 2^{0}$

- **Unsigned integer**
  - Assuming that numbers are always positive
  - A 32-bit word can represent 2^{32} numbers between 0 and $2^{32}-1$

Negative Numbers - Signed Magnitude

- 32 bits can only represent 2^{32} numbers
  - If we wish to also represent negative numbers, we can represent 2^{31} positive numbers (including zero) and 2^{31} negative numbers

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2^{31}-1$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -\ 0_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -\ 1_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -\ 2_{ten}$
...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -\ (2^{31}-2)$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -\ (2^{31}-1)$

Negative Numbers - 1's Complement

- Represent -X as 1's complement of X
  - Converting every bit of X ⇨ 1's complement of X

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2^{31}-1$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -(2^{31}-1)$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -(2^{31}-2)$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -(2^{31}-3)$
...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -1$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -0$

Negative Numbers - 2's Complement

- Represent -X as 2's complement of X
  - (1's complement of X) + 1 ⇨ 2's complement of X

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2^{31}-1$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -\ 2^{31}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -\ (2^{31} - 1)$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -\ (2^{31} - 2)$
...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -\ 2$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -\ 1$

Value of $X = x_{n-1} x_{n-2} \cdots x_1 x_0$

1. **Unsigned number**

$$V(X) = \sum_{k=0}^{n-1} x_k \cdot 2^k$$

2. **Signed magnitude = Sign and magnitude**

$$V(X) = (-1)^{x_{n-1}} \cdot \sum_{k=0}^{n-2} x_k \cdot 2^k$$

3. **1's complement (cf) Diminished radix complement**

$$V(X) = -x_{n-1} \cdot (2^{n-1} - 1) + \sum_{k=0}^{n-2} x_k \cdot 2^k$$

4. **2's complement (cf) Radix complement**

$$V(X) = -x_{n-1} \cdot 2^{n-1} + \sum_{k=0}^{n-2} x_k \cdot 2^k$$

Overflow

- **Overflow if result out of range**
  - Adding positive and negative operands, no overflow
  - Adding two positive operands
    - Overflow if result sign is 1
  - Adding two negative operands
    - Overflow if result sign is 0
- **Overflow detection**
  - Ex: 7 + 7 = 0111 + 0111 = 1110 = -2
  - CarryIn to MSB ≠ CarryOut from MSB
- **Conditional branches that test for overflow**
  - ARM: BVS (branch if overflow set) and BVC (branch if overflow clear)
  - IA-32: JO (jump if overflow) and JNO (jump if not overflow)

Arithmetic for Multimedia

- **SIMD (single instruction stream, multiple data stream)**
  - Many graphics and audio applications would perform the same operation on vectors of 8-bit and 16-bit data
  - Use 64-bit adder, with partitioned carry chain
  - Operate on 8×8-bit, 4×16-bit, or 2×32-bit vectors
- **Saturating operations**
  - On overflow, result is largest representable value

    (cf) 2's complement modulo arithmetic
- **Multimedia extensions to modern instruction sets**

Instruction category	Operands
Unsigned add/subtract	Eight 8-bit or Four 16-bit
Saturating add/subtract	Eight 8-bit or Four 16-bit
Max/min	Eight 8-bit or Four 16-bit
Average	Eight 8-bit or Four 16-bit
Shift right/left	Eight 8-bit or Four 16-bit

Figure 3.3

1-Bit ALU with ADD, OR, AND

- Multiplexor selects between ADD, OR, AND operations

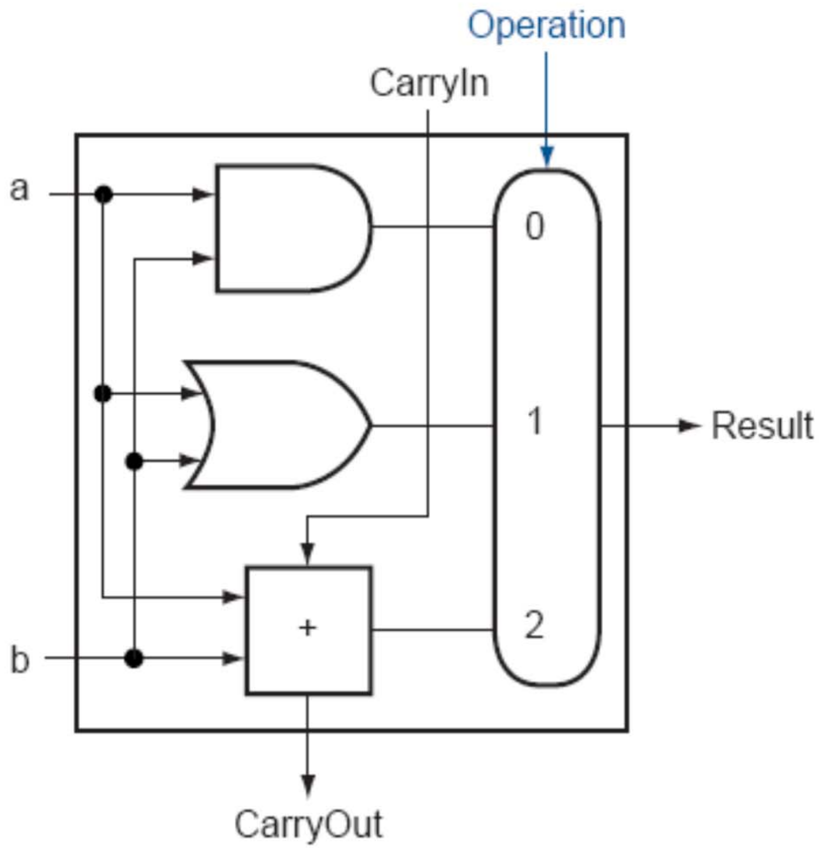


FIGURE B.5.6 A 1-bit ALU that performs AND, OR, and addition (see Figure B.5.5).

32-bit Ripple Carry Adder

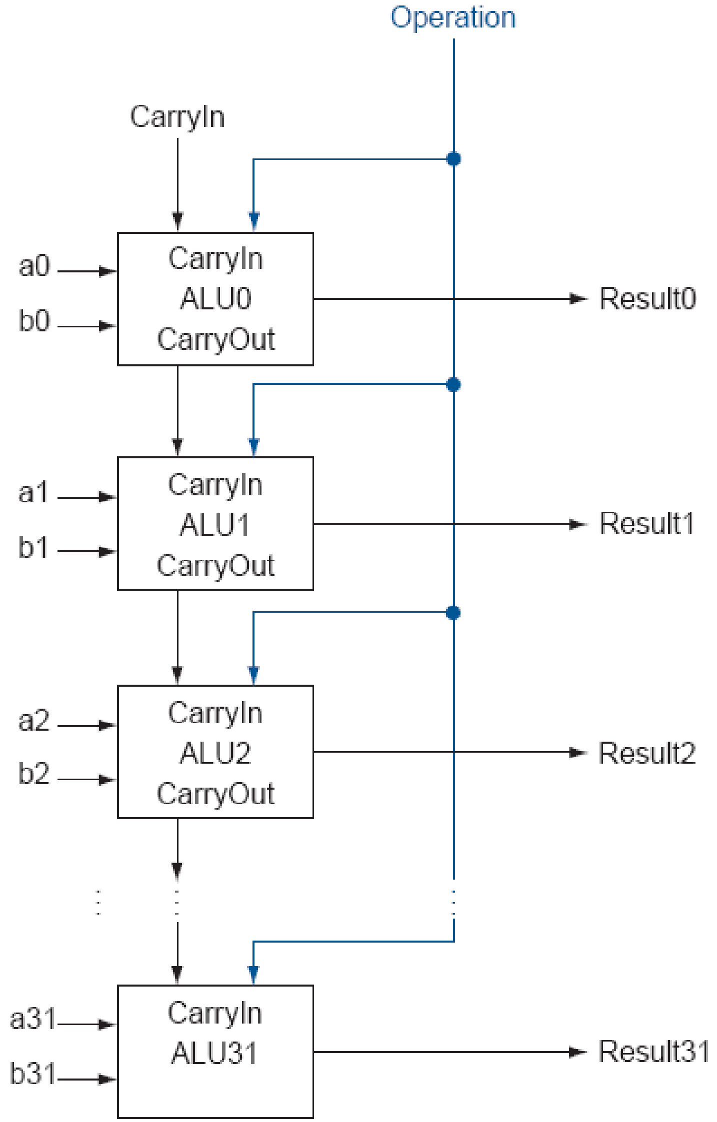


FIGURE B.5.7 A 32-bit ALU constructed from 32 1-bit ALUs. CarryOut of the less significant bit is connected to the CarryIn of the more significant bit. This organization is called ripple carry.

Incorporating Subtraction

- **Must invert bits of B and add a 1**
  - Include an inverter
  - CarryIn for the first bit is 1
  - The CarryIn signal (for the first bit) can be the same as the Binvert signal

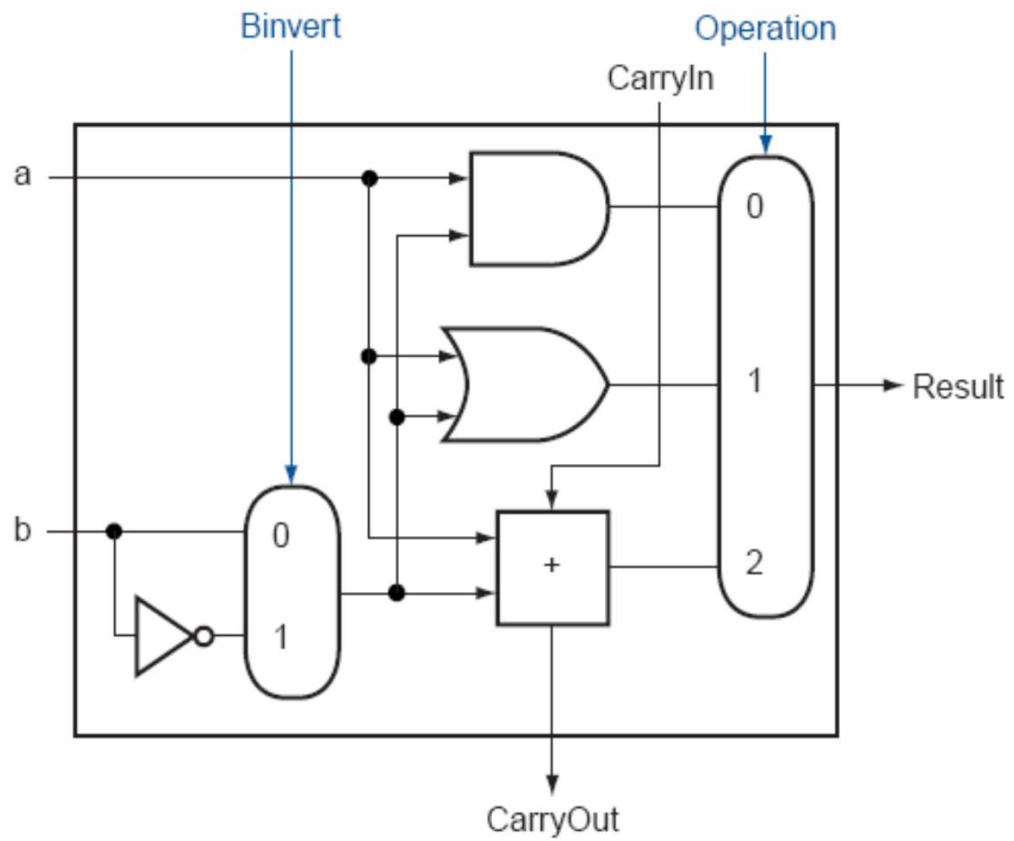


FIGURE B.5.8 A 1-bit ALU that performs AND, OR, and addition on a and b or a and $\overline{b}$. By selecting $\overline{b}$ (Binvert = 1) and setting CarryIn to 1 in the least significant bit of the ALU, we get two's complement subtraction of b from a instead of addition of b to a.

Incorporating NOR

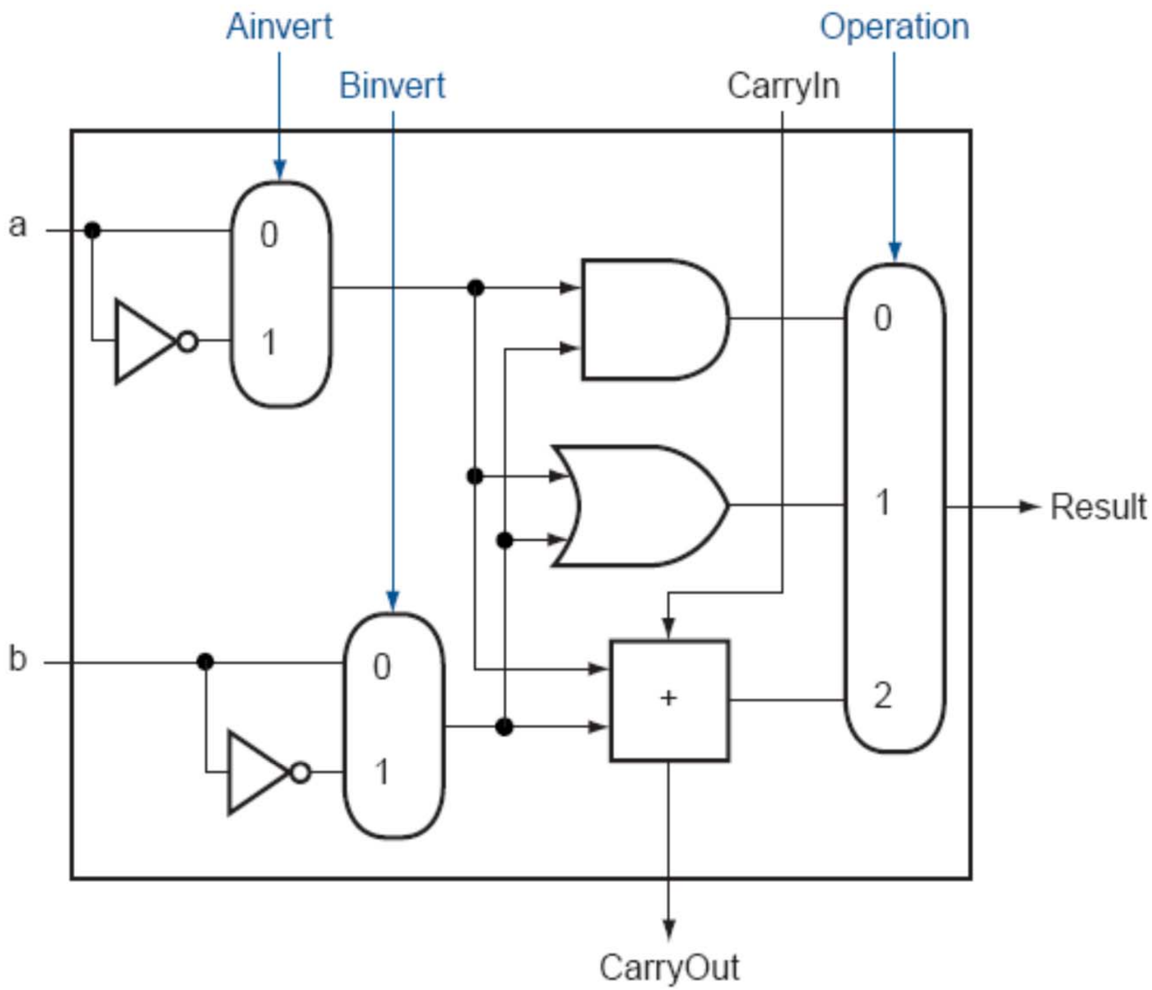
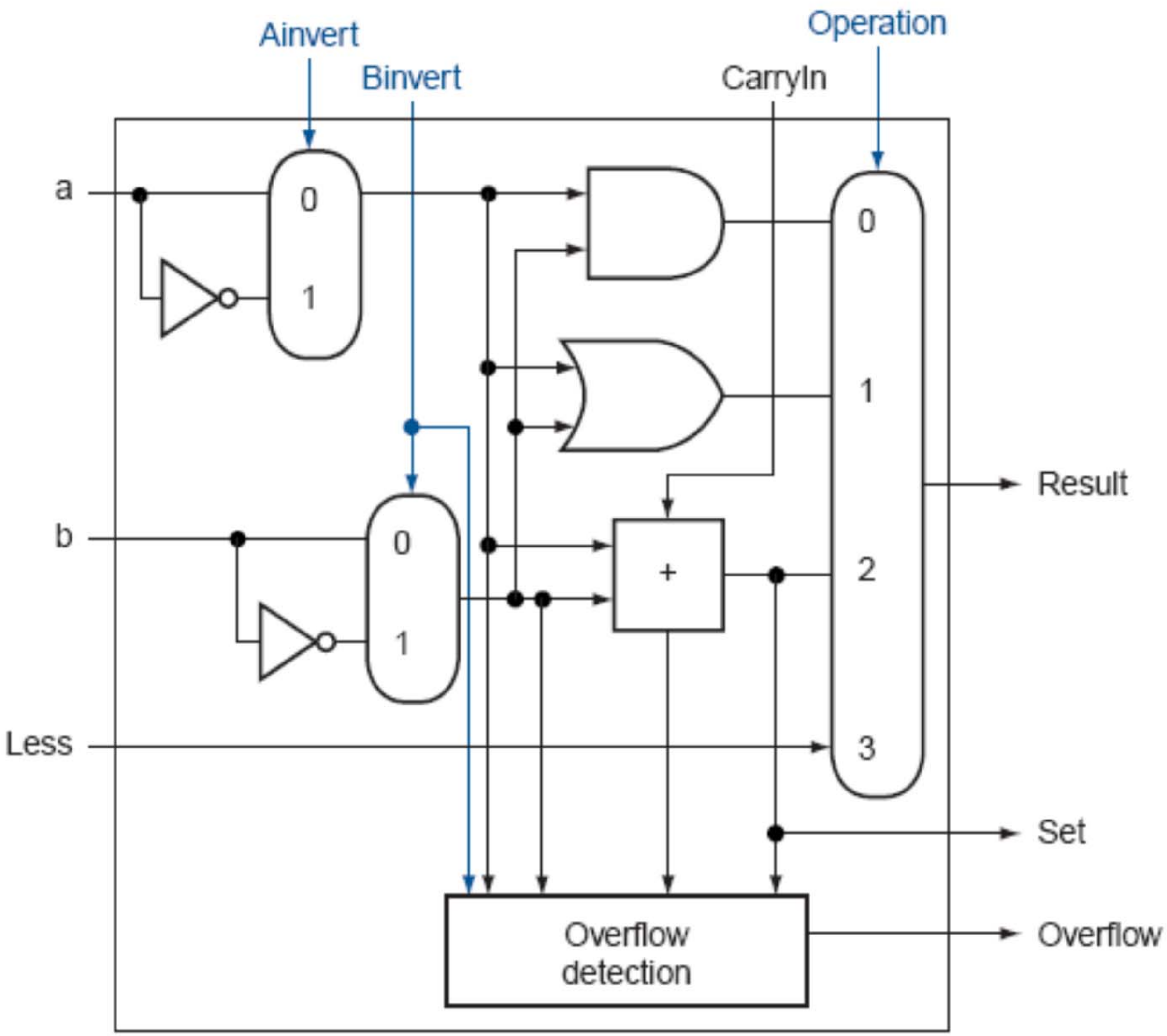


FIGURE B.5.9 A 1-bit ALU that performs AND, OR, and addition on a and b or $\overline{a}$ and $\overline{b}$. By selecting $\overline{a}$ (Ainvert = 1) and $\overline{b}$ (Binvert = 1), we get a NOR b instead of a AND b.

Incorporating slt

- Perform a – b and check the sign
- New signal (Less) that is zero for ALU boxes 1-31
- The 31st box has a unit to detect overflow and sign
  - The sign bit serves as the Less signal for the 0th box



Incorporating beq

- Perform a – b and confirm that the result is all zero's

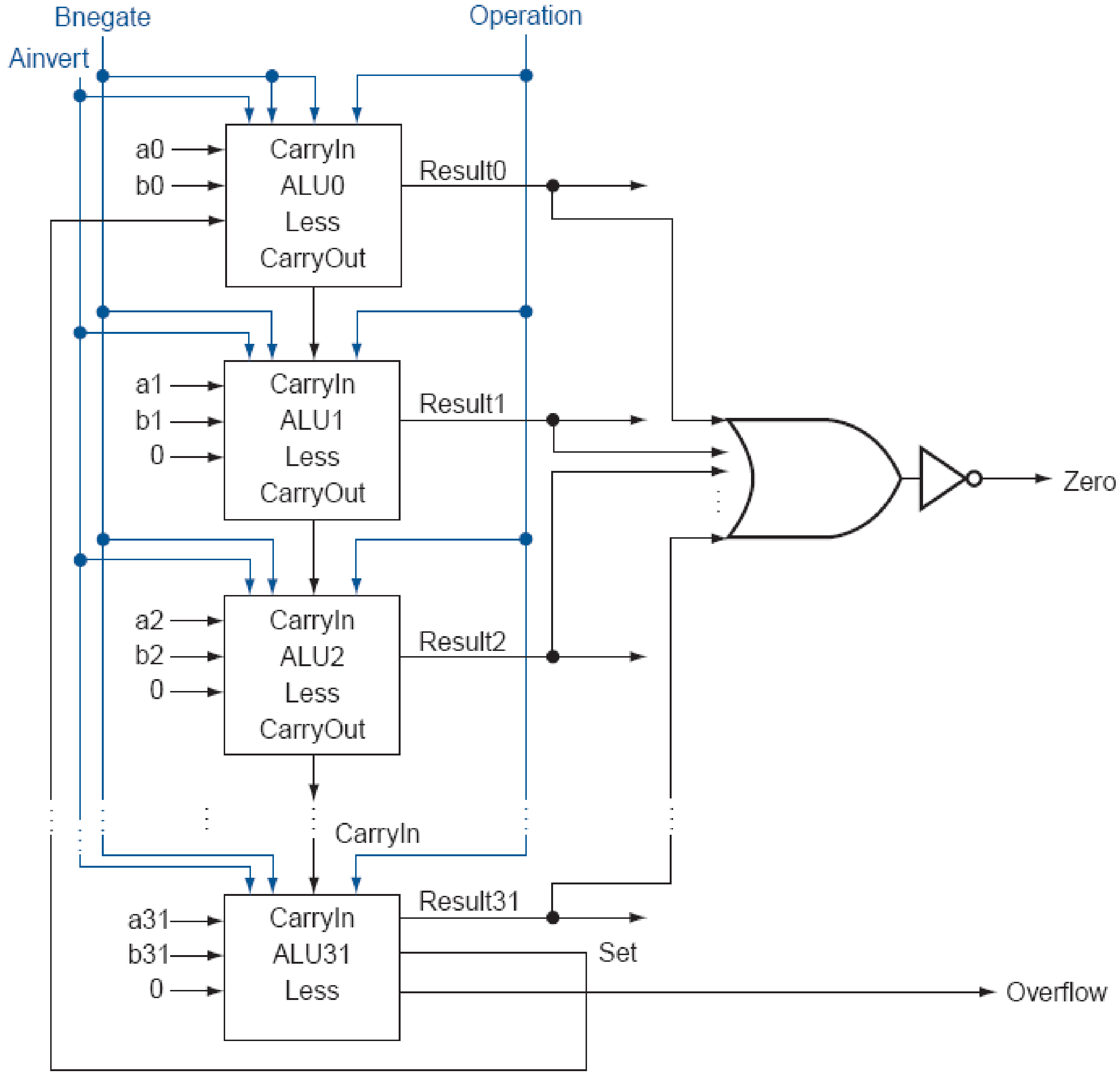
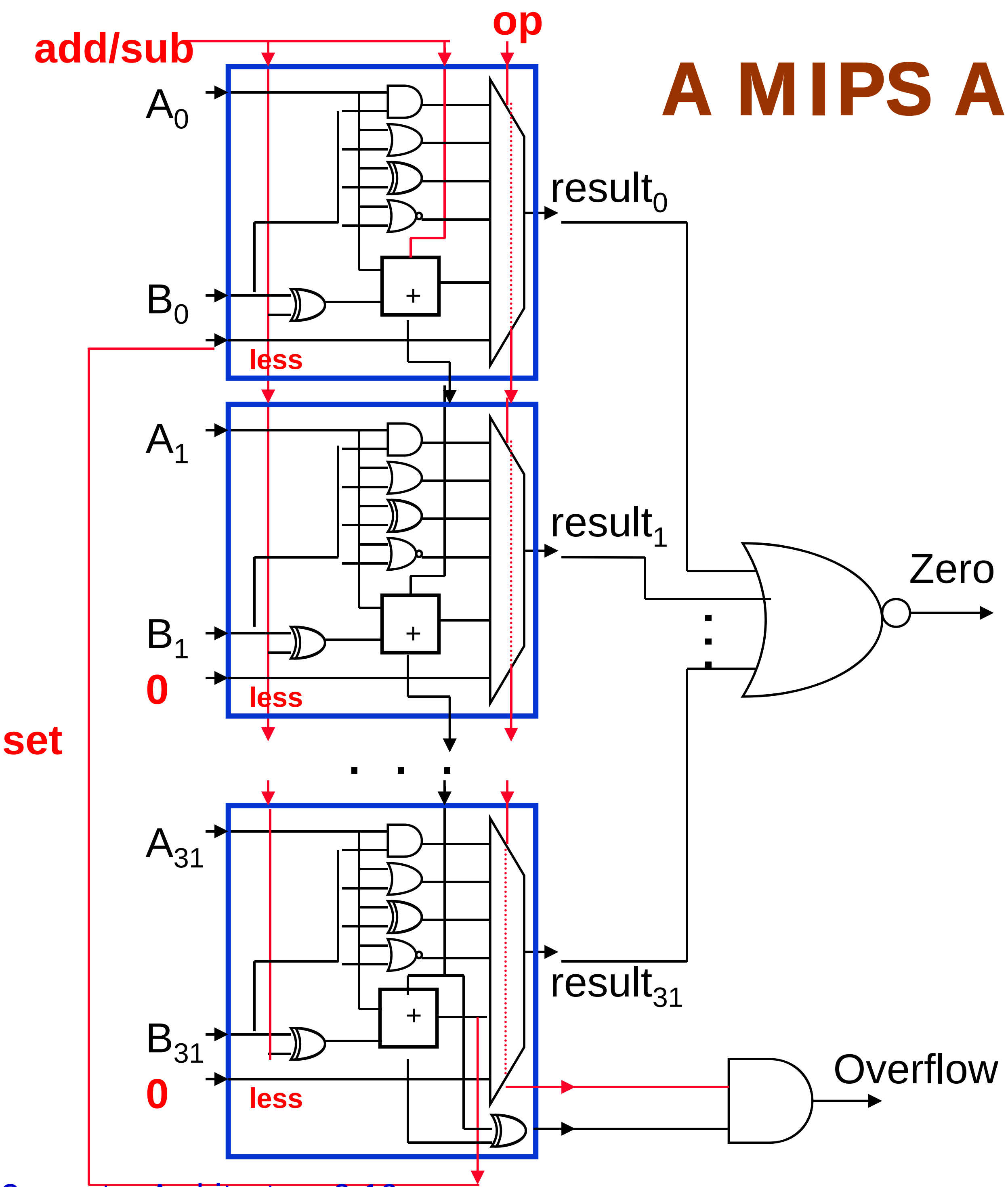


FIGURE B.5.12 The final 32-bit ALU. This adds a Zero detector to Figure B.5.11.

A MIPS ALU Implementation

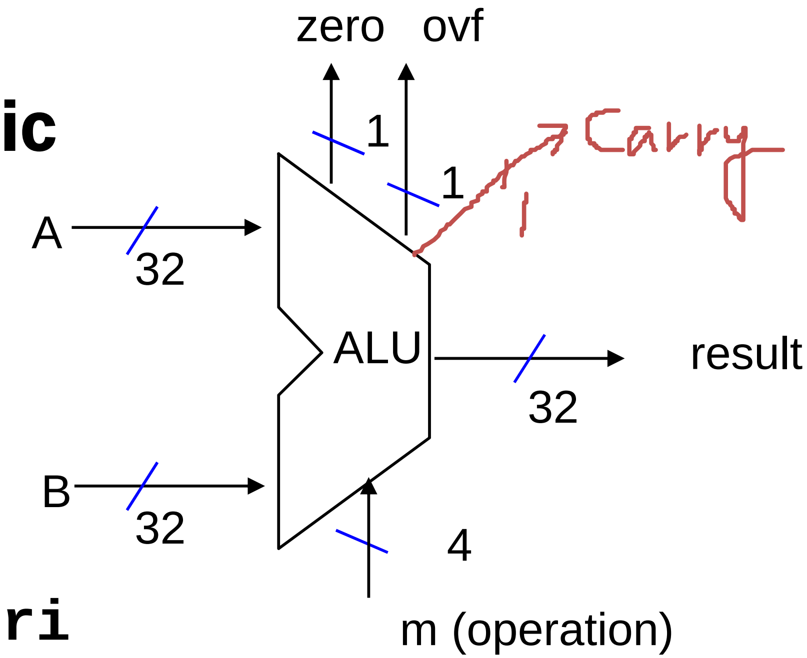


- Zero detect
  - **`slt, slti, sltiu, sltu, beq, bne`**
- Enable overflow bit setting for signed arithmetic
  - **`add, addi, sub`**

MIPS Arithmetic Logic Unit (ALU)

- **Must support the Arithmetic/Logic operations of the ISA**

```
add, addi, addiu, addu
sub, subu
mult, multu, div, divu
sqrt
and, andi, nor, or, ori, xor, xori
beq, bne, slt, slti, sltiu, sltu
```



- **With special handling for**
  - sign extend – **`addi, addiu, slti, sltiu`**
  - zero extend – **`andi, ori, xori`**
  - overflow detection – **`add, addi, sub`** **(cf)** **`addu, addiu, subu`**