

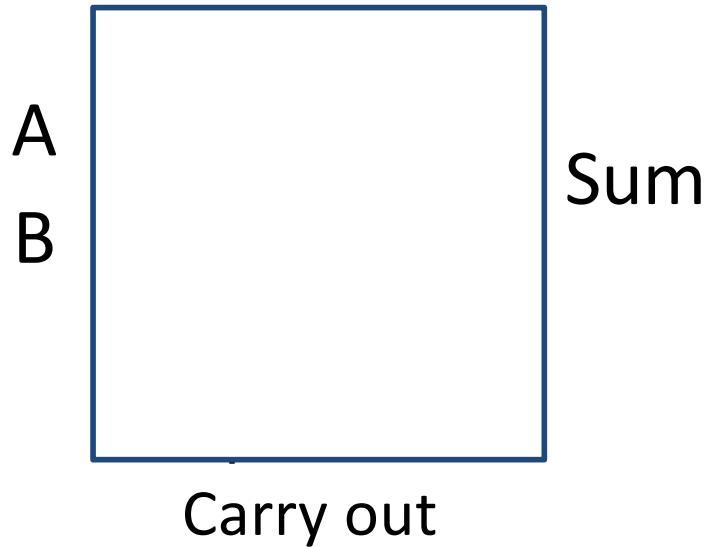
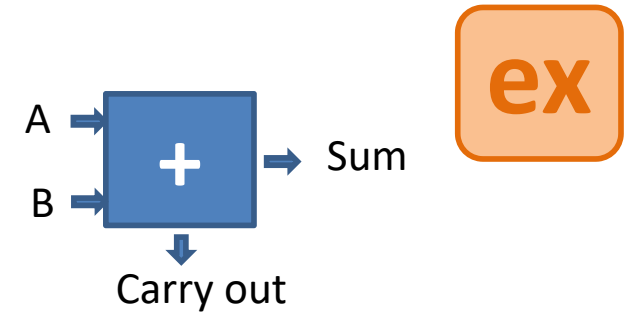


Arithmetic Logic

adders

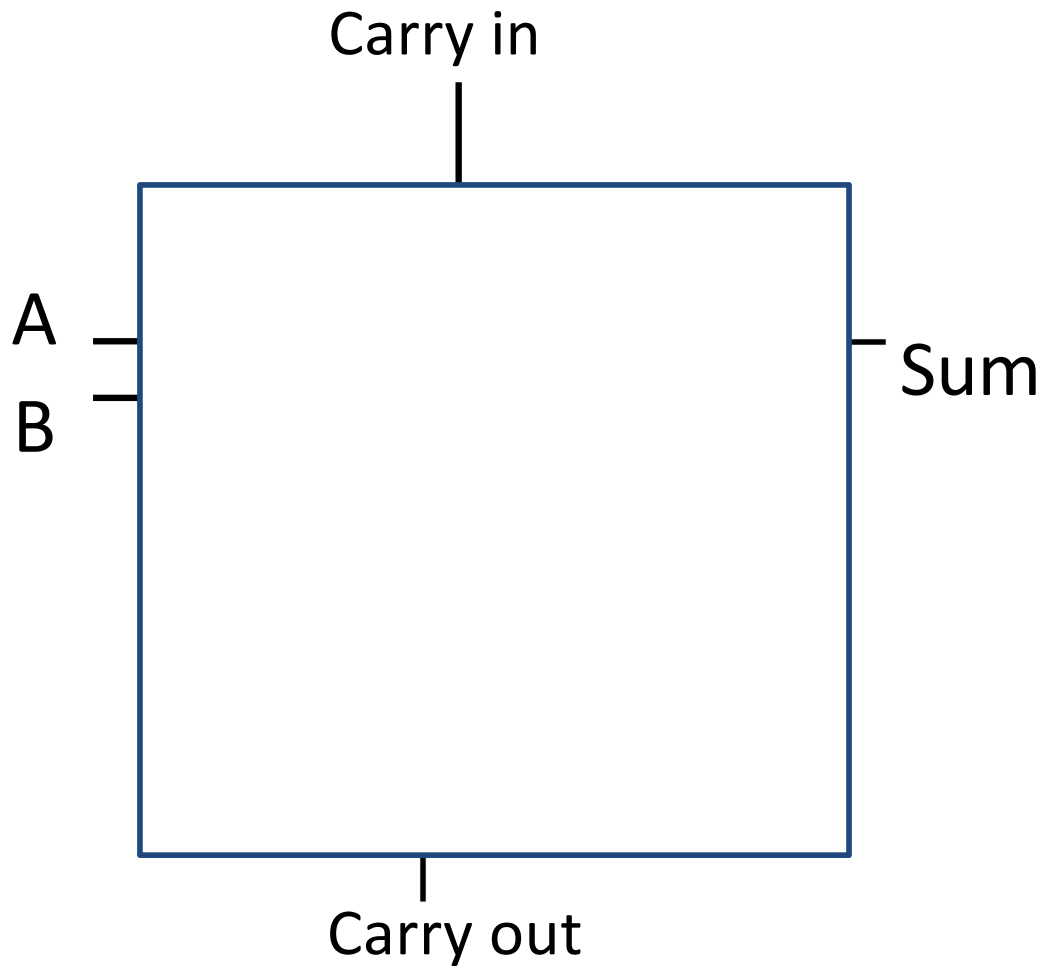
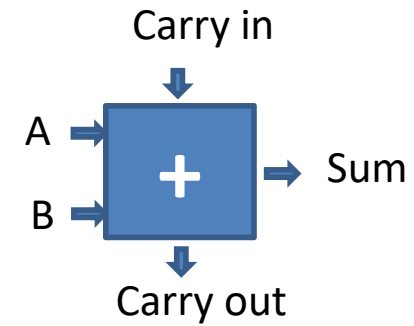
Arithmetic Logic Unit

Addition: 1-bit *half* adder



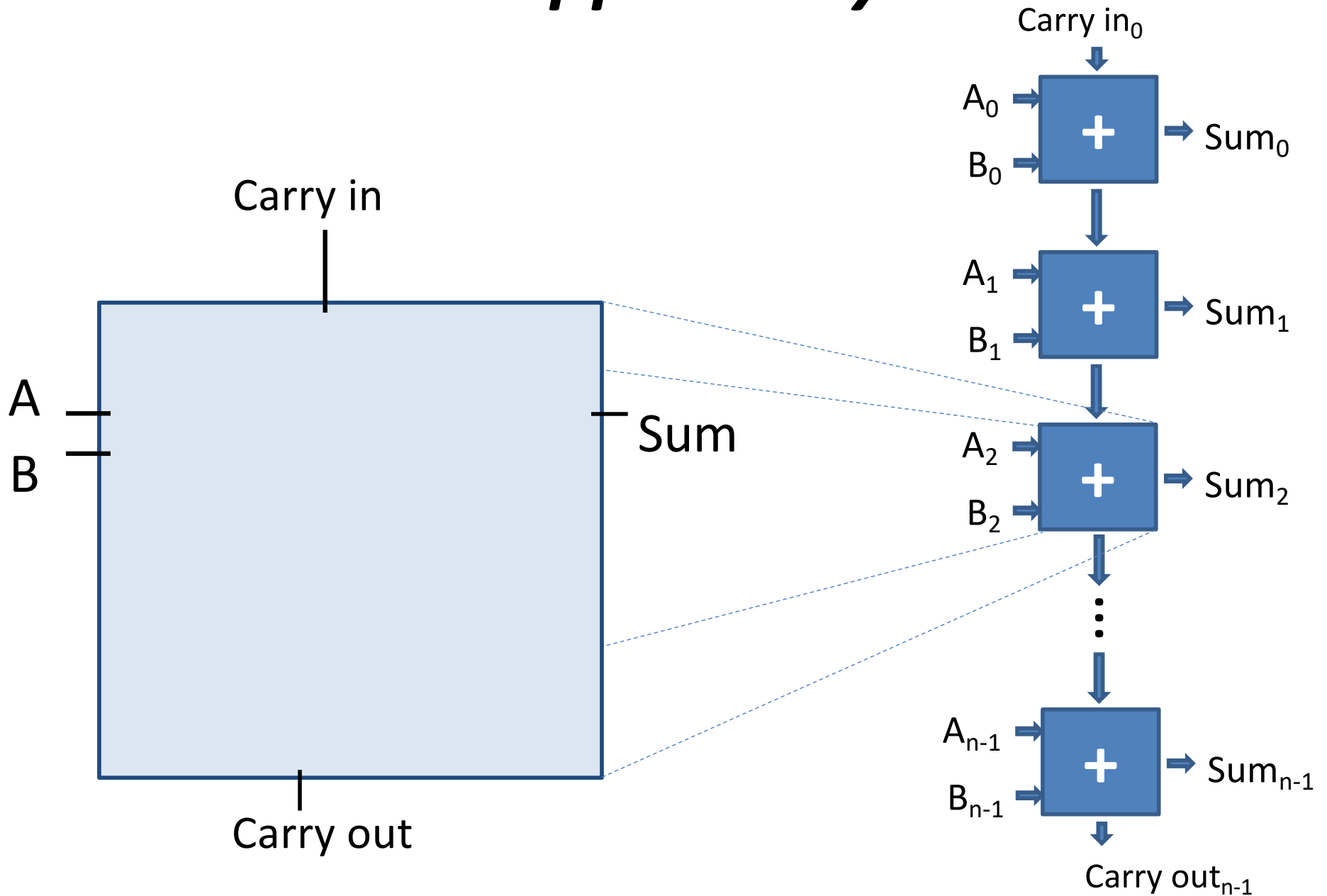
| A | B | Carry out | Sum |
|---|---|-----------|-----|
| 0 | 0 | | |
| 0 | 1 | | |
| 1 | 0 | | |
| 1 | 1 | | |

Addition: 1-bit *full* adder



| Carry in | A | B | Carry out | Sum |
|----------|---|---|-----------|-----|
| 0 | 0 | 0 | | |
| 0 | 0 | 1 | | |
| 0 | 1 | 0 | | |
| 0 | 1 | 1 | | |
| 1 | 0 | 0 | | |
| 1 | 0 | 1 | | |
| 1 | 1 | 0 | | |
| 1 | 1 | 1 | | |

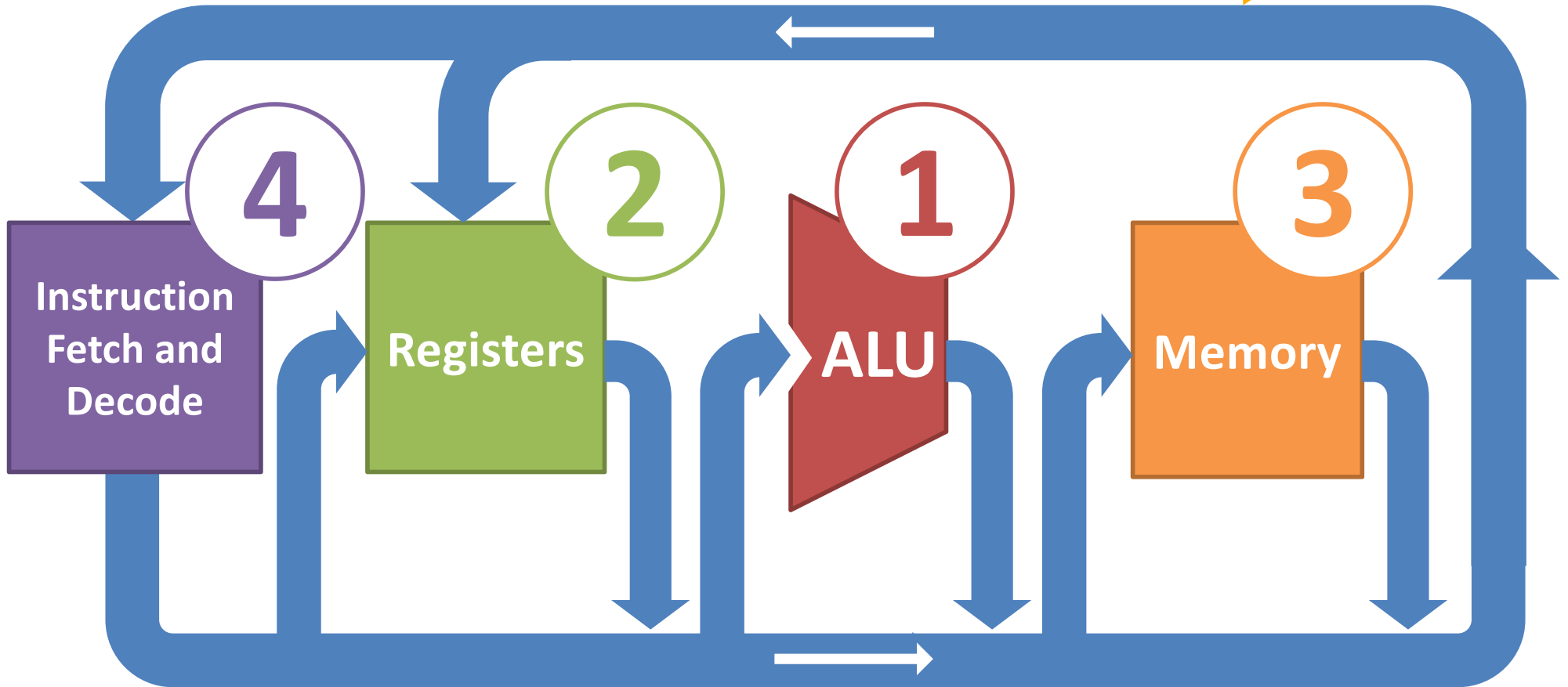
Addition: *n*-bit *ripple-carry* adder



There are faster, more complicated ways too...

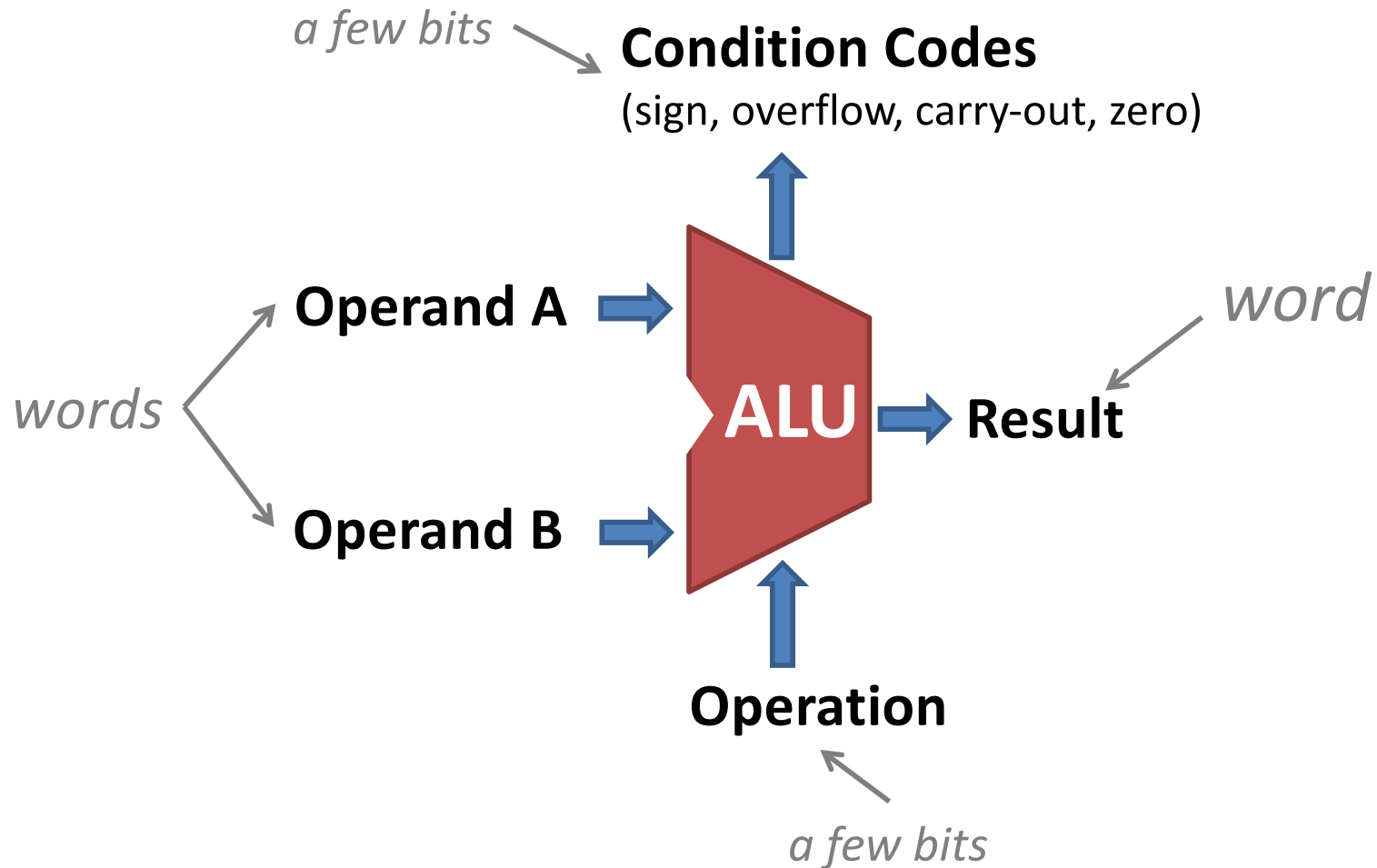
Processor Components

Abstraction!



Arithmetic Logic Unit (ALU)

1



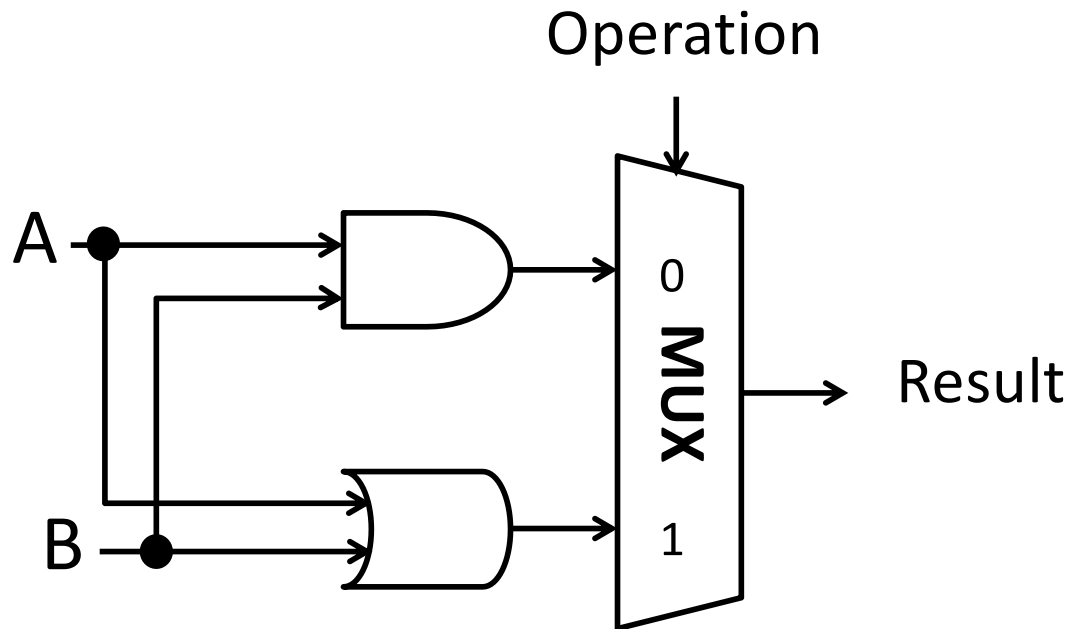
Hardware unit for arithmetic and bitwise operations.

1-bit ALU for bitwise operations



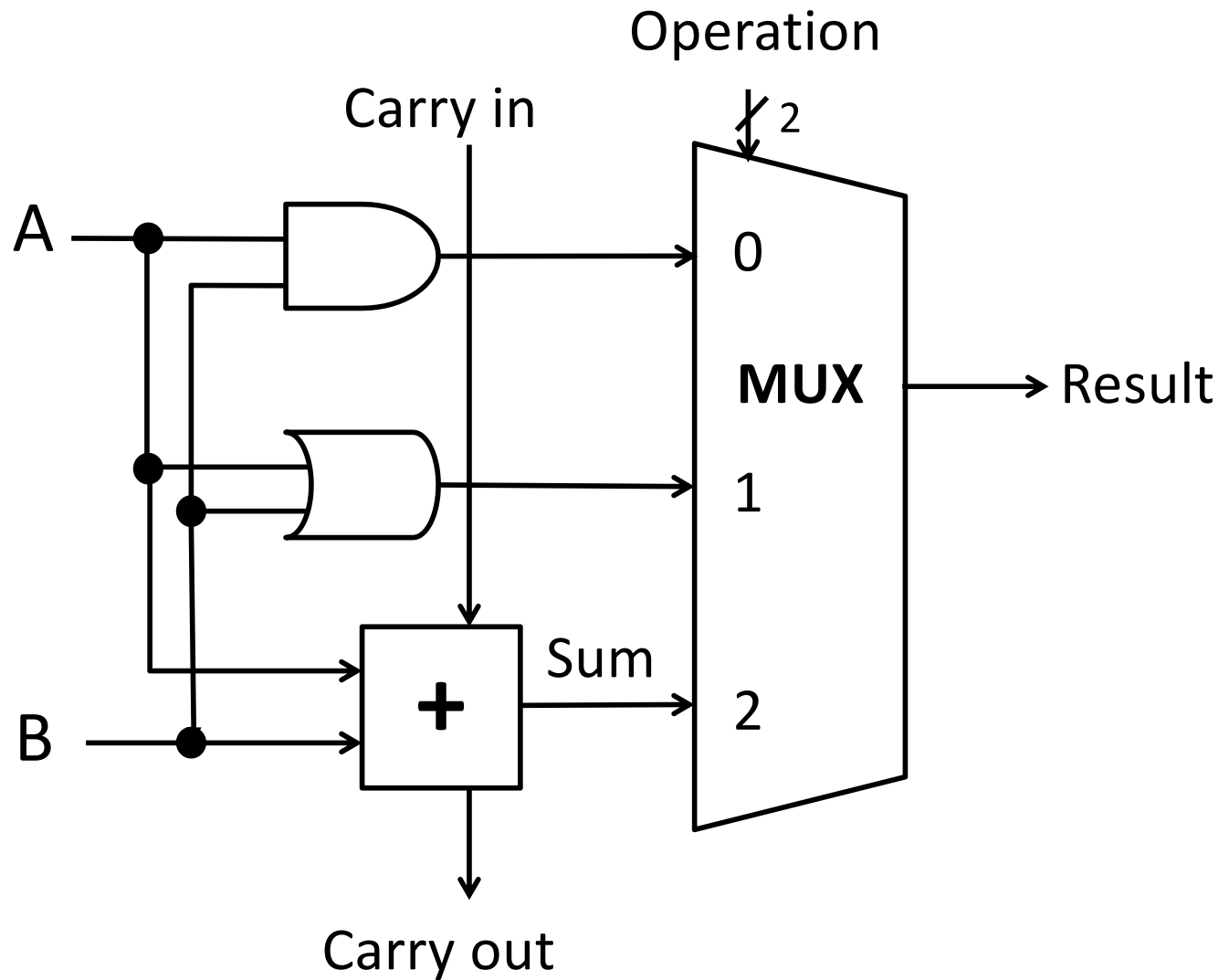
Build an n-bit ALU from n 1-bit ALUs.

Each bit i in the result is computed from the corresponding bit i in the two inputs.

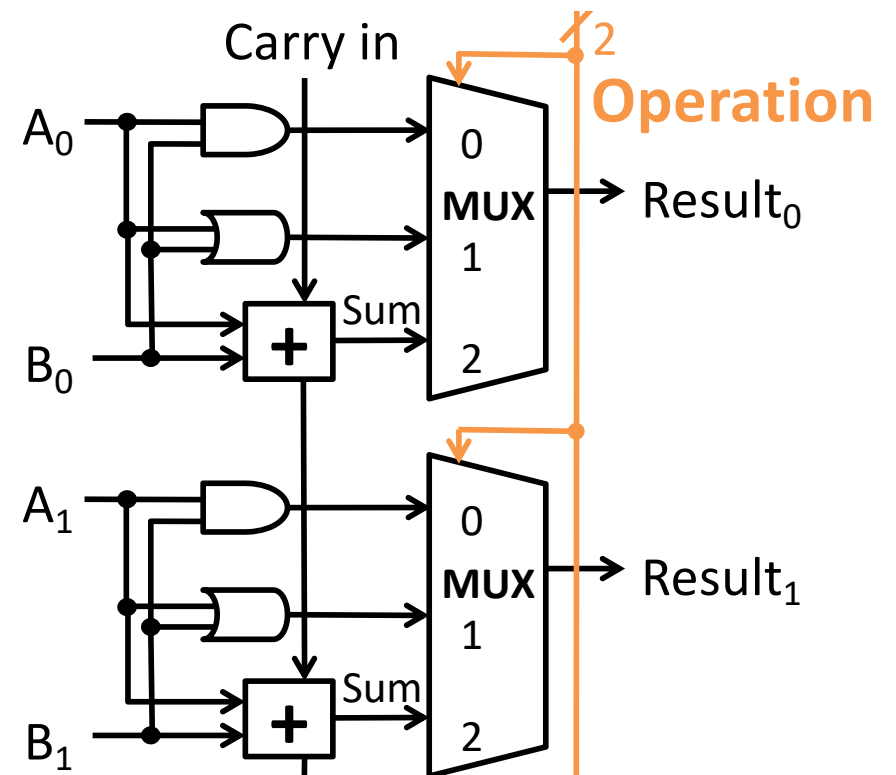
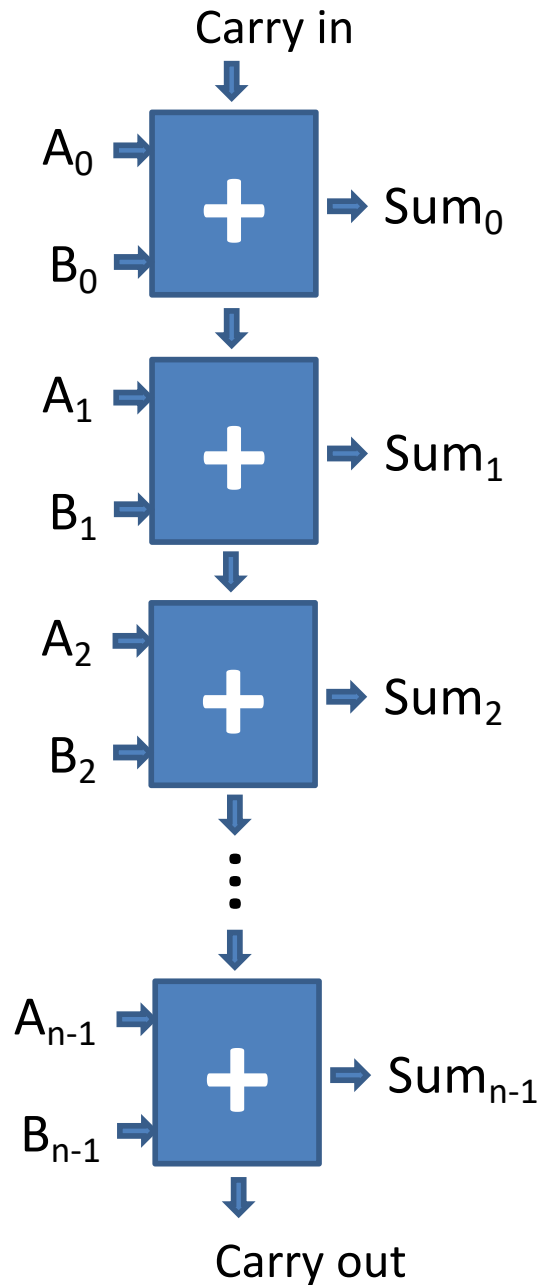


| Op | A | B | Result |
|----|---|---|--------|
| 0 | 0 | 0 | |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | |

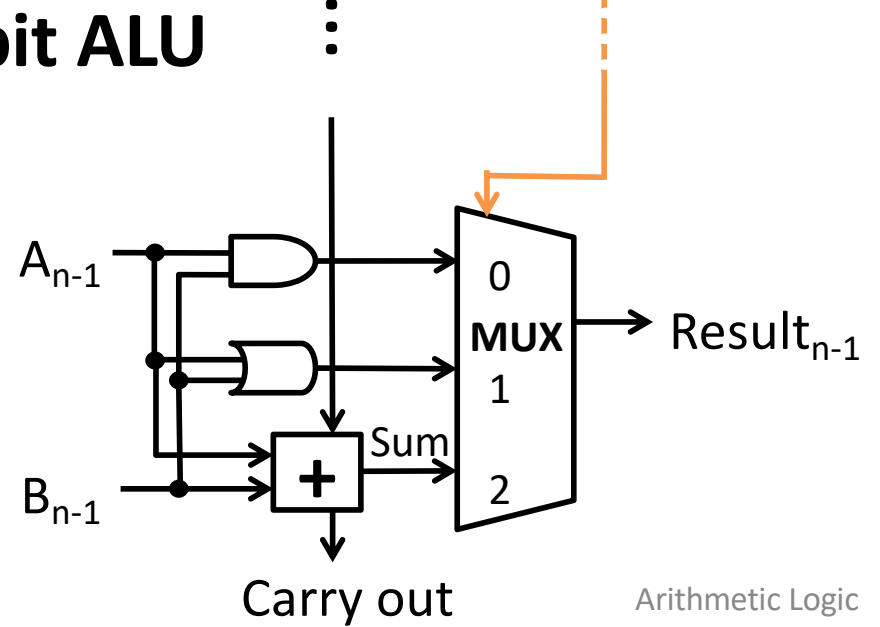
1-bit ALU



n-bit ripple carry adder



n-bit ALU



ALU conditions

Extra ALU outputs

describing properties of result.

Zero Flag: ex

1 if result is 00...0 else 0

Sign Flag: ex

1 if result is negative else 0

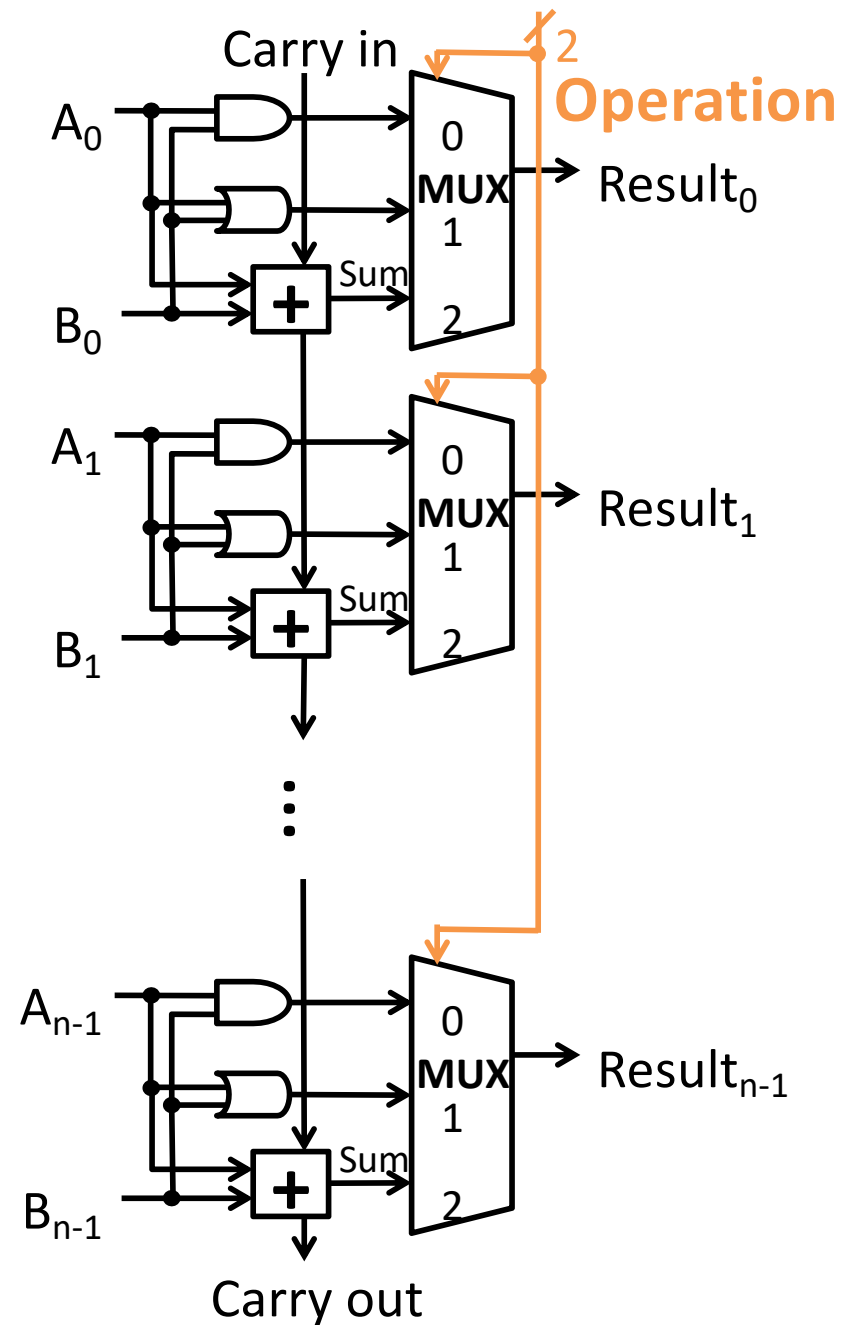
Carry Flag:

1 if carry out else 0

(Signed) Overflow Flag:

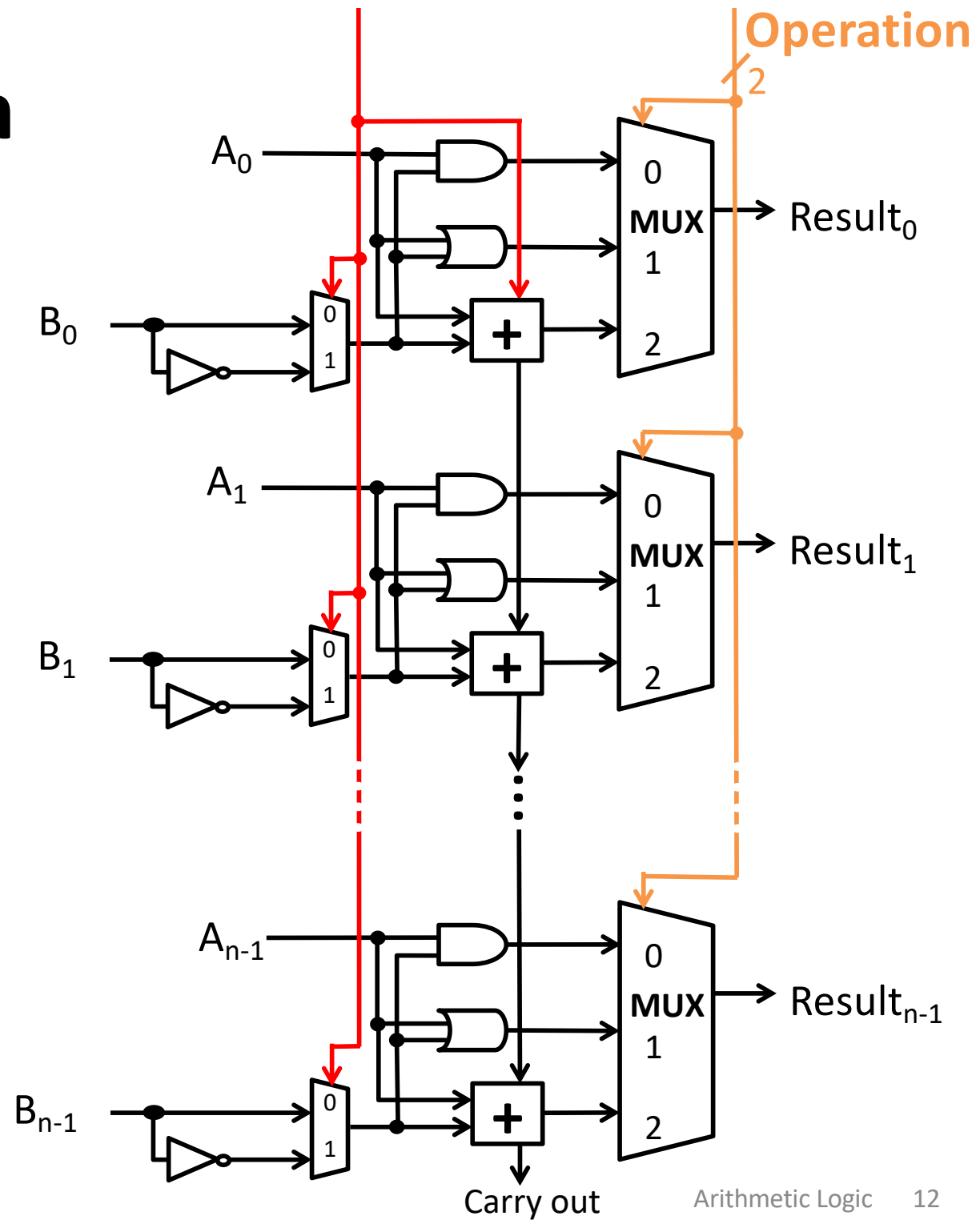
1 if signed overflow else 0

Implement these.



Add subtraction

How can we control ALU inputs or add minimal new logic to **compute A-B**?



ex

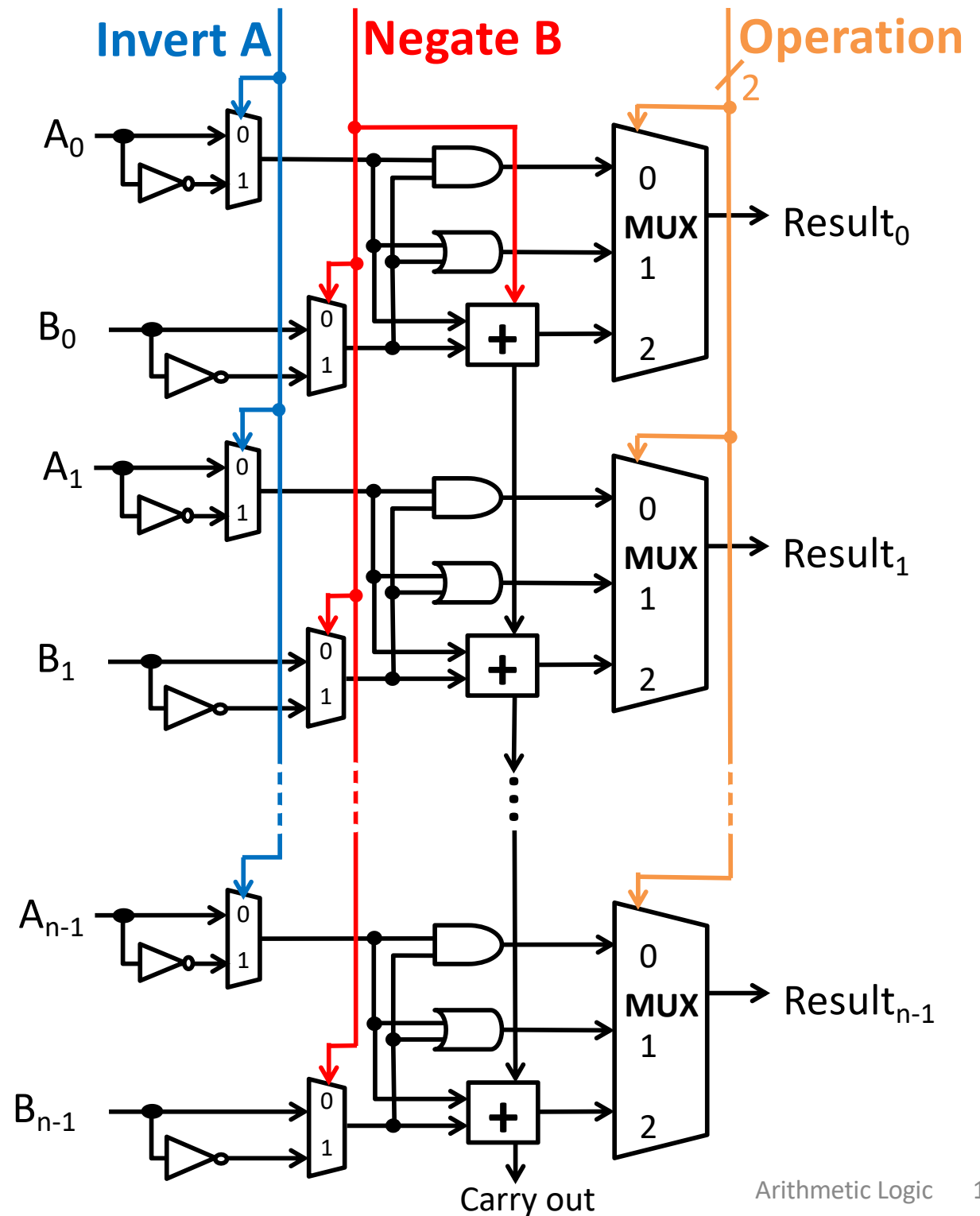
A NAND B

A NOR B

A < B

A == B

How can we control ALU inputs or add minimal new logic to compute each?



Controlling the ALU



| ALU control lines | Function |
|-------------------|----------|
| 0000 | AND |
| 0001 | OR |
| 0010 | add |
| 0110 | subtract |
| 1100 | NOR |

