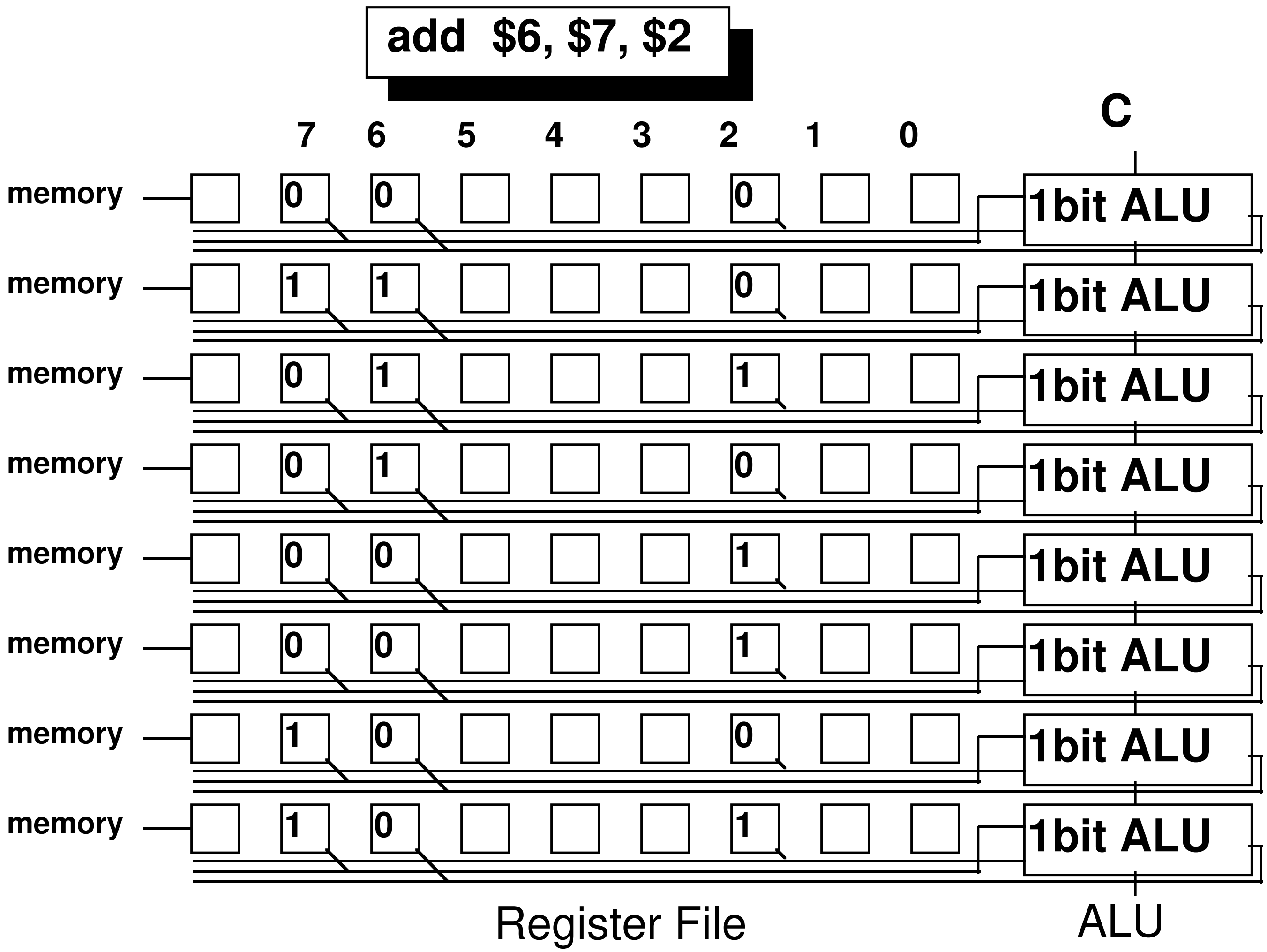
Construction of an ALU: Arithmetic/Logic Unit

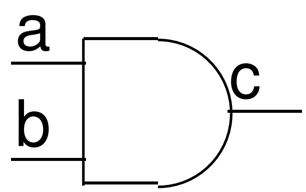
The ALU must be capable of performing all of the basic operations of the machine. RISC architectures attempt to make all of the operations have a similar complexity.
The ALU developed in this lecture will perform a small number of representative operations.

Data Path Structure



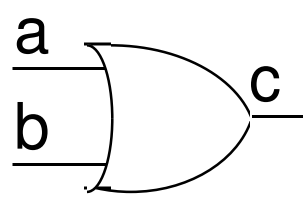
Logic Gates*: Computer Building Blocks

AND(a,b): return c = a · b



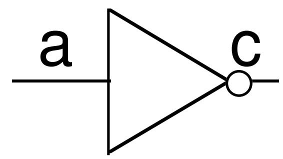
a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

OR(a,b): return c = a + b



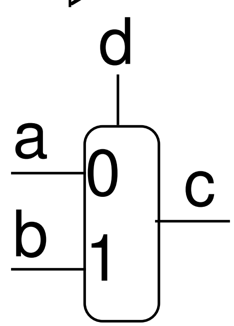
a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

NOT(a): return c = ~a



a	c
0	1
1	0

MUX(d,a,b): return c = (if d=0 then a else b)



d	c
0	a
1	b

*Bill's nickname in college?

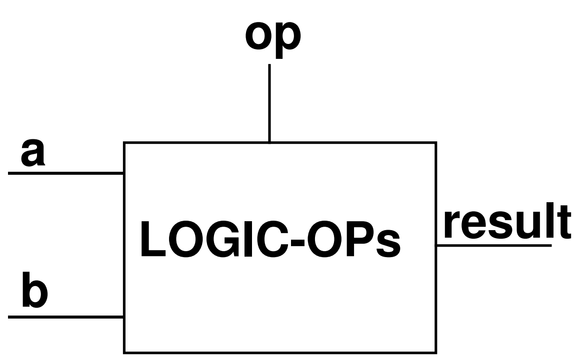
1Bit ALU: Logic Operations

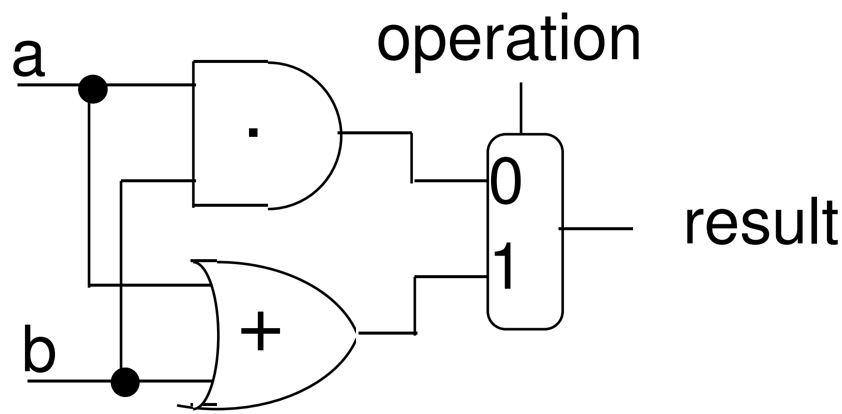
Bitwise Logic Operations

AND $x, $y, $z

OR $x, $y, $z

LOGIC-OPs: *MUX*(op, *AND*(a,b), *OR*(a,b))

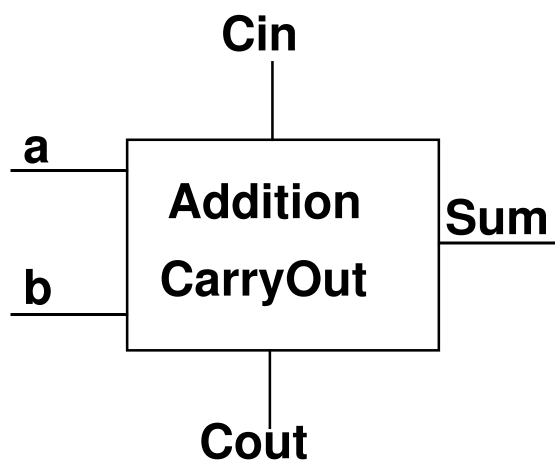




1Bit Adder, CarryOut

3 Inputs (a, b, Cin), 2 outputs (Sum, Cout)

a	b	cin	cout	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

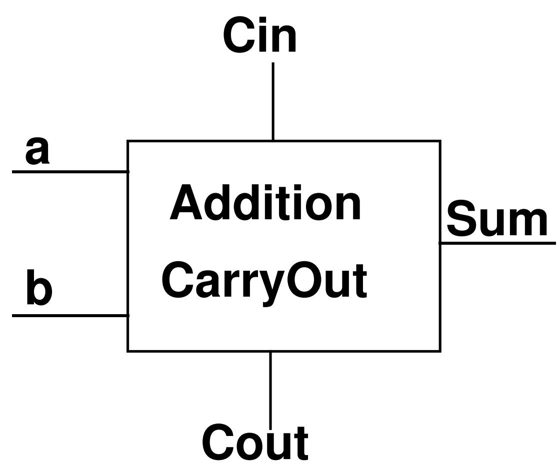


`Cout = (b · Cin) + (a · Cin) + (a · b) + (a · b · Cin)`

Cout: *OR*(*AND*(b,Cin), *AND*(a, Cin), *AND*(a,b))

1Bit Adder, Sum

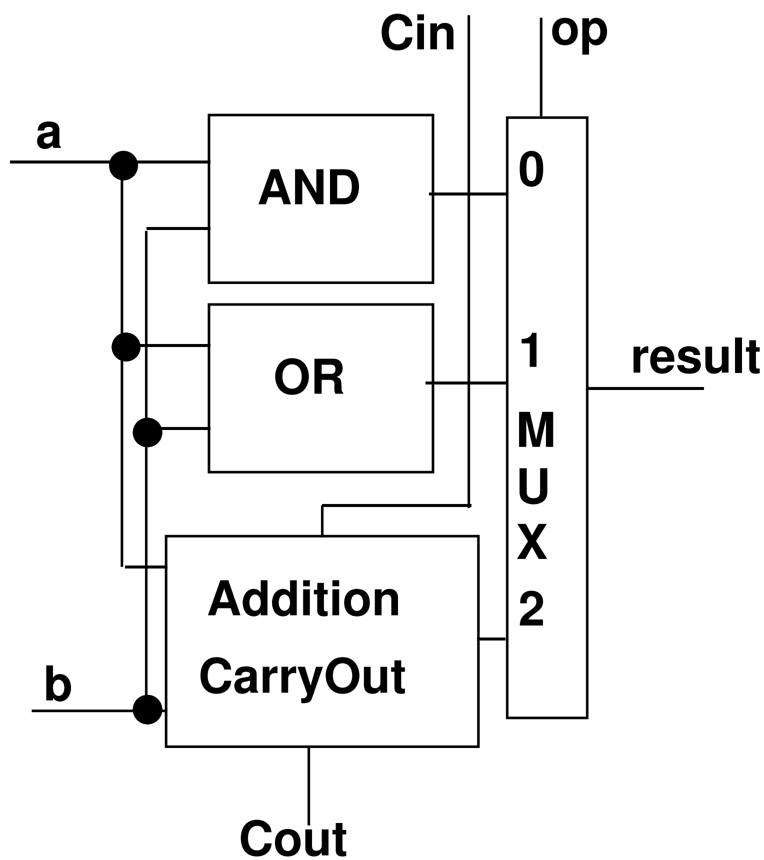
a	b	cin	cout	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



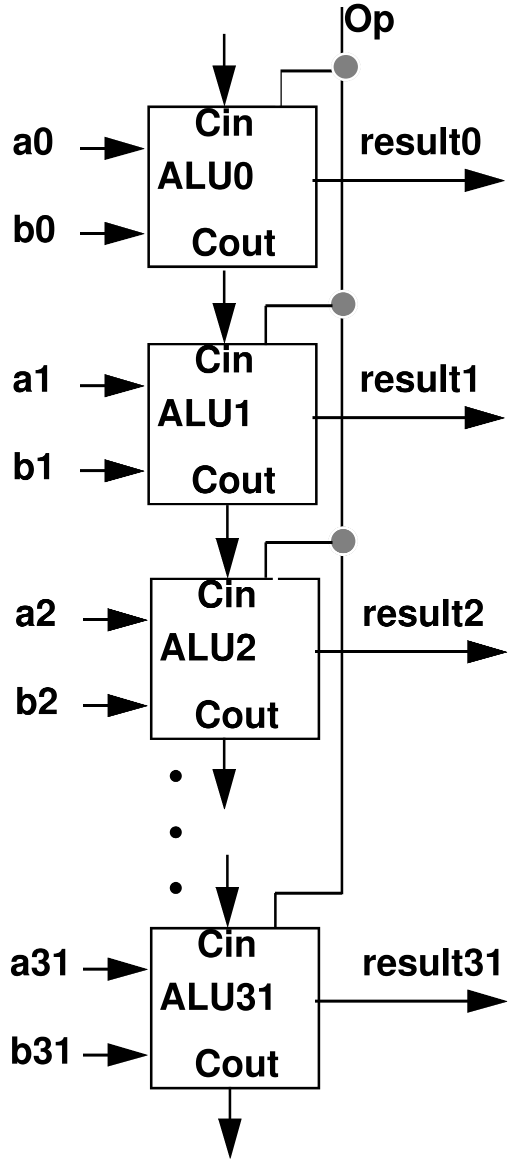
$$\mathrm{Sum} = (a \cdot \overline{b} \cdot \overline{\mathrm{Cin}}) + (\overline{a} \cdot b \cdot \overline{\mathrm{Cin}}) + (\overline{a} \cdot \overline{b} \cdot \mathrm{Cin}) + (a \cdot b \cdot \mathrm{Cin})$$

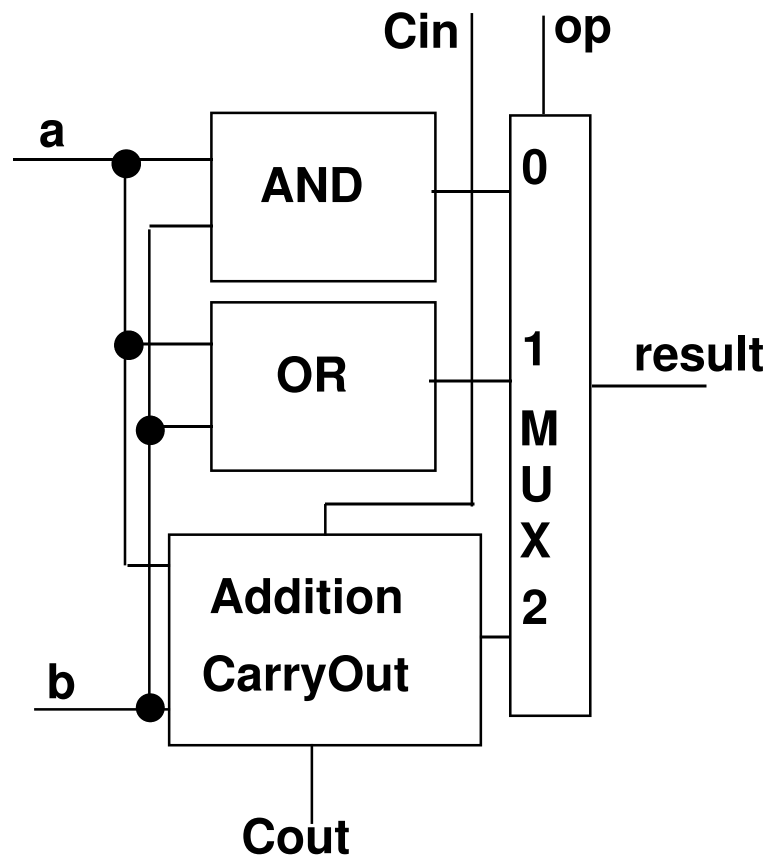
Sum: *OR*(*AND*(a, *NOT*(b), NOT(Cin)), *AND*(*NOT*(a), b, NOT(Cin)), *AND*(*NOT*(a), *NOT*(b), Cin), *AND*(a,b,Cin))

1-Bit AND-OR-Add ALU



32-Bit ALU


Op
a0
b0
Cin
ALU0
Cout
result0
a1
b1
Cin
ALU1
Cout
result1
a2
b2
Cin
ALU2
Cout
result2
a31
b31
Cin
ALU31
Cout
result31


Cin
op
a
b
AND
OR
Addition
CarryOut
0
1
MUX
2
result
Cout

1-Bit AND-OR-Add-Sub ALU