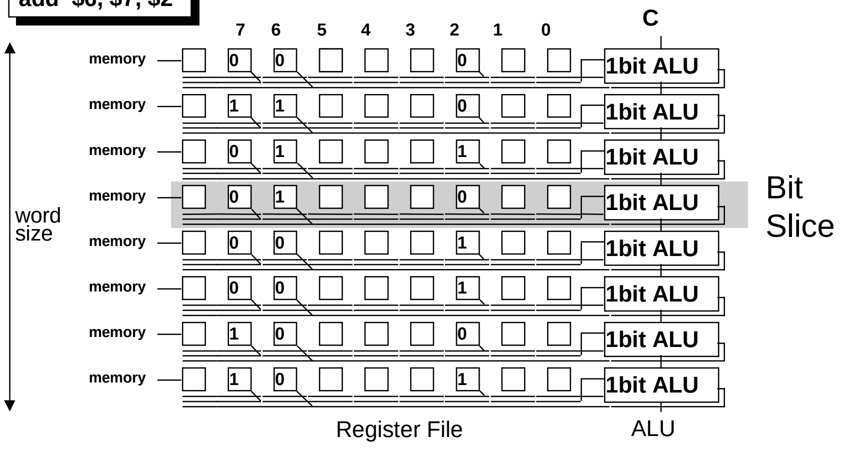
Construction of an Arithmetic/Logic Unit

The ALU performs all of the basic operations of the machine. RISC architectures attempt to make all of the operations have similar complexity

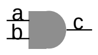
Data Path Structure

add $6, $7, $2



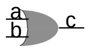
Logic Gates*

AND(a,b): return c = a · b



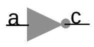
a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

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

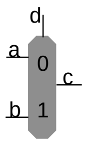


a	c
0	1
1	0

MUX(d,a,b): return c =

(if d=0 then

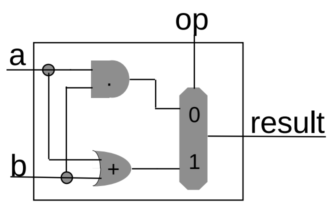
a else b)



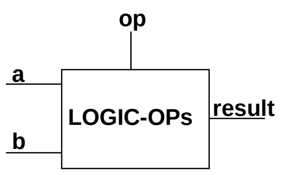
d	c
0	a
1	b

*Bill's nickname in college?

1Bit ALU: Logic Operations



Bitwise Logic Operations	
AND	$result,$a,$b
OR	$result,$a,$b

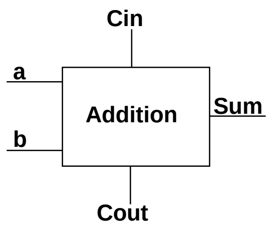


LOGIC-OPs: *MUX*(op, *AND*(a,b), *OR*(a,b))

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



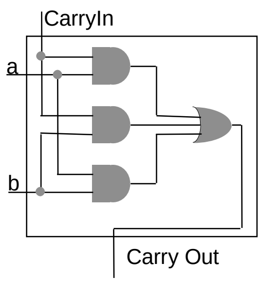
`Cout = (b · Cin) + (a · Cin) + (a · b) + (a · b · Cin)`

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

Logic for Carry Out

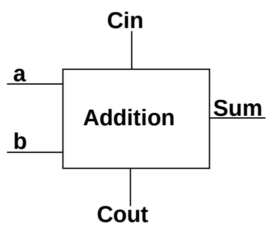
`Cout = (b · Cin) + (a · Cin) + (a · b) + (a · b · Cin)`

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



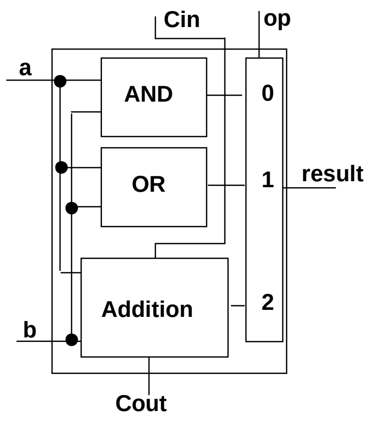
$$\text{Sum} = (a \cdot \overline{b} \cdot \overline{Cin}) + (\overline{a} \cdot b \cdot \overline{Cin}) + (\overline{a} \cdot \overline{b} \cdot Cin) + (a \cdot b \cdot Cin)$$

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

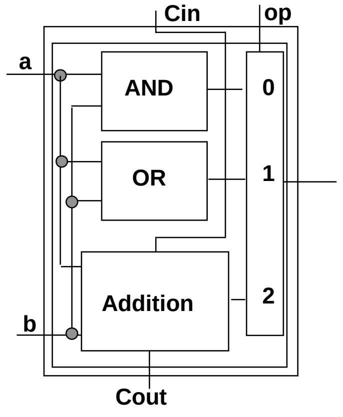
Combine components

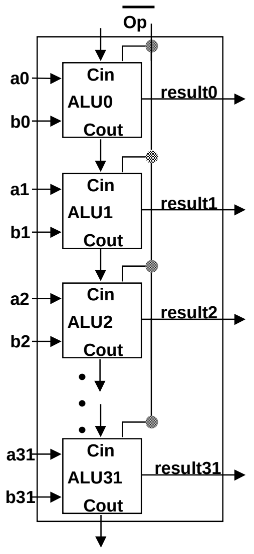
- Set op=1 for and $result, $a,$b
- Set op=2 for or $result, $a,$b
- Set op=3 for add $result, $a,$b



32-Bit ALU

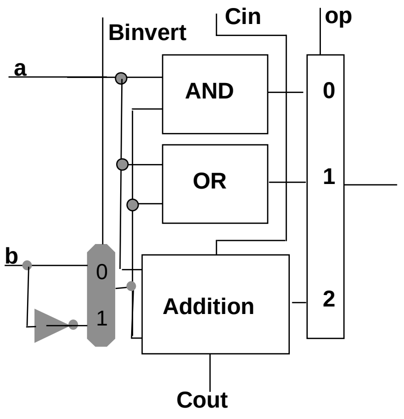
Compose 32 bit-slices





One bit And/Or/Add/Sub ALU

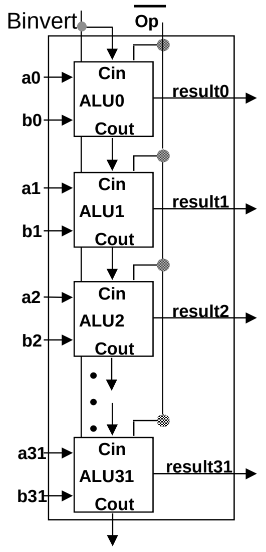
- sub ==> add with negative “b” operand
- Negative “b” ==> complement and add 1



Binvert
0 ==> add
1 ==> sub

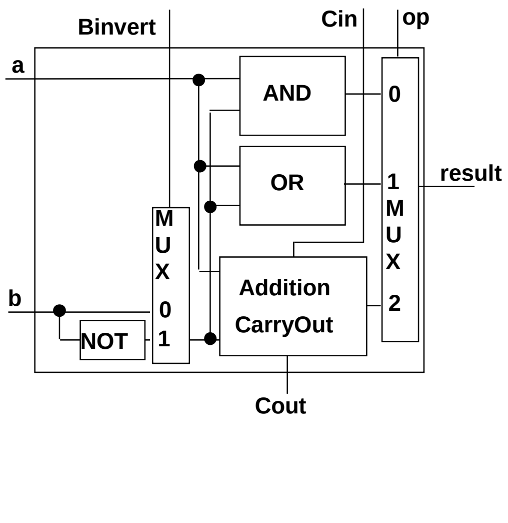
Cin0
0 ==> add
1 ==> sub

32-Bit ALU



The Binvert control signal will always match the carry in:
add => Binvert=0 => Cin=0
sub => Binvert=1 => Cin=1
so the two lines can be connected.

1-Bit AND-OR-Add-Sub ALU



Binvert
Cin
op
a
AND
OR
0
1
M
U
X
2
result
b
NOT
M
U
X
0
1
Addition
CarryOut
Cout