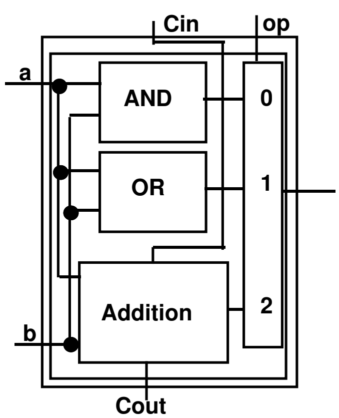
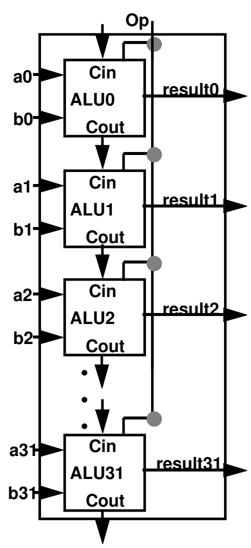
Arithmetic/Logic Unit Design Continued

Having established the basic bit-sliced design, we add functionality to it to implement more instructions.

32-Bit ALU

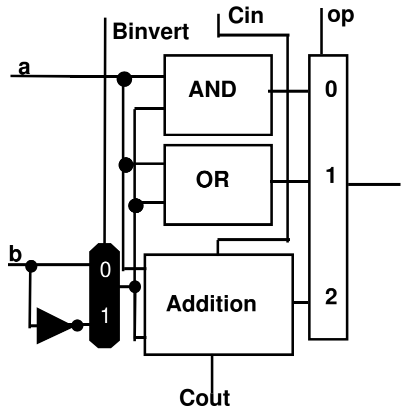
Compose 32 bit-slices





One bit And/Or/Add/Sub ALU

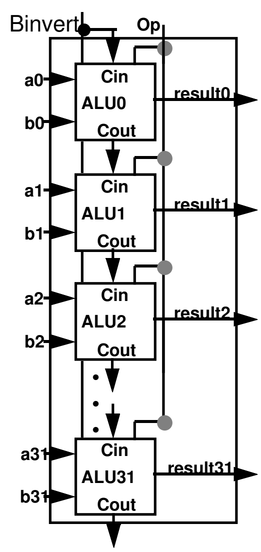
- sub ==> add with negative “b” operand
- Negative “b” ==> complement and add 1



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

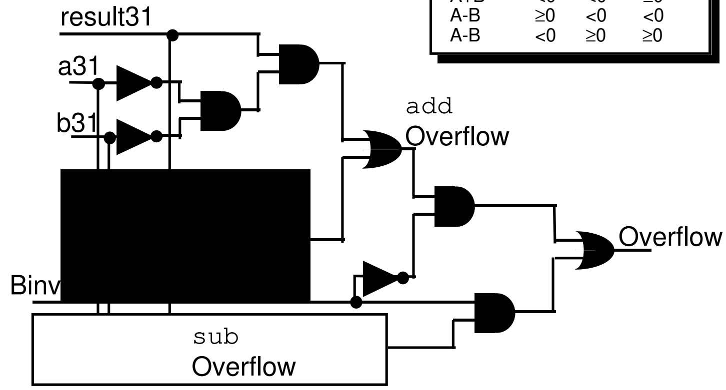
sub => Binvert=1 => Cin=1

so the two lines can be connected.

Overflow Conditions

- Separate add and subtract cases

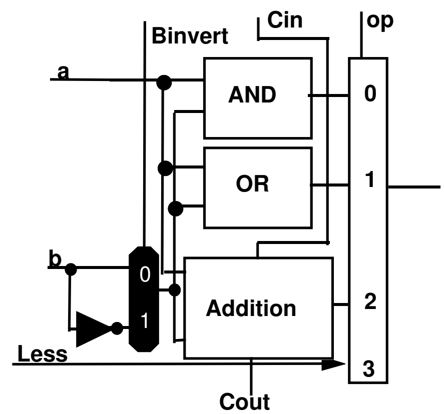
Operation	Op A	Op B	Overflow
A+B	≥0	≥0	<0
A+B	<0	<0	≥0
A-B	≥0	<0	<0
A-B	<0	≥0	≥0



Less-than-test

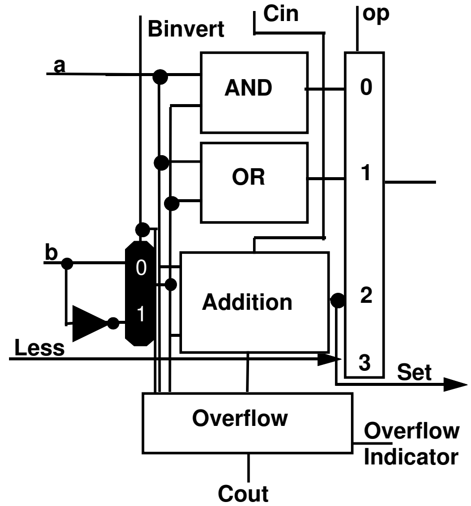
There are three cases:

Bit 0
Bit i
Bit 31

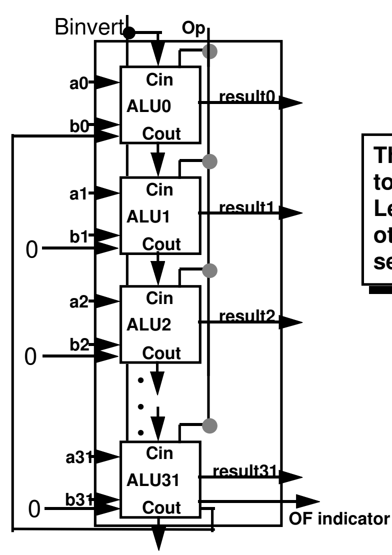


Less Than Test, MSB

- Capture adder output for set bit

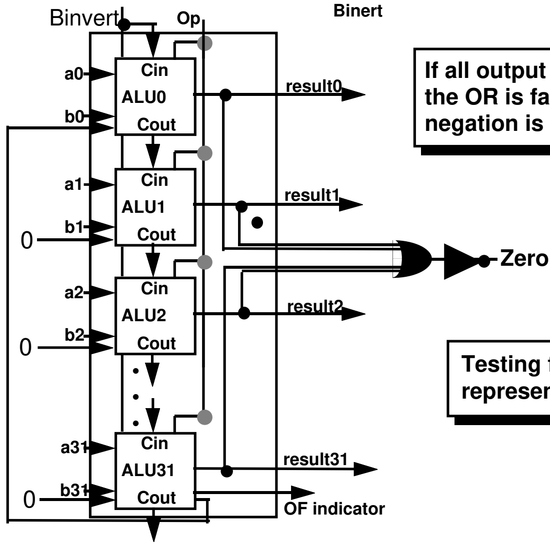


AND-OR-Add-Sub-SLT ALU



The Set line feedsback to become the input to Less for bit-0. All other bits have Less set to zero.

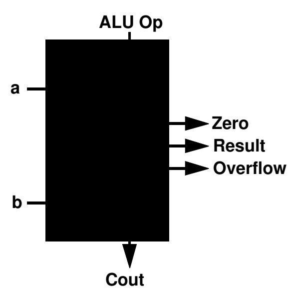
Zero-detecting ALU



If all output bits are zero, the OR is false, so its negation is true.

Testing for alternative representations?

Abstracting ...



ALU Operation	Function
0 00	AND
0 01	OR
0 10	Add
1 10	Sub
1 11	SLT

Binvert **Op**

Using the ALU

The datapath uses the ALU structure several times, though not always in its full generality.