CSSE132
Introduction to Computer Systems

11 : Basic computational structures

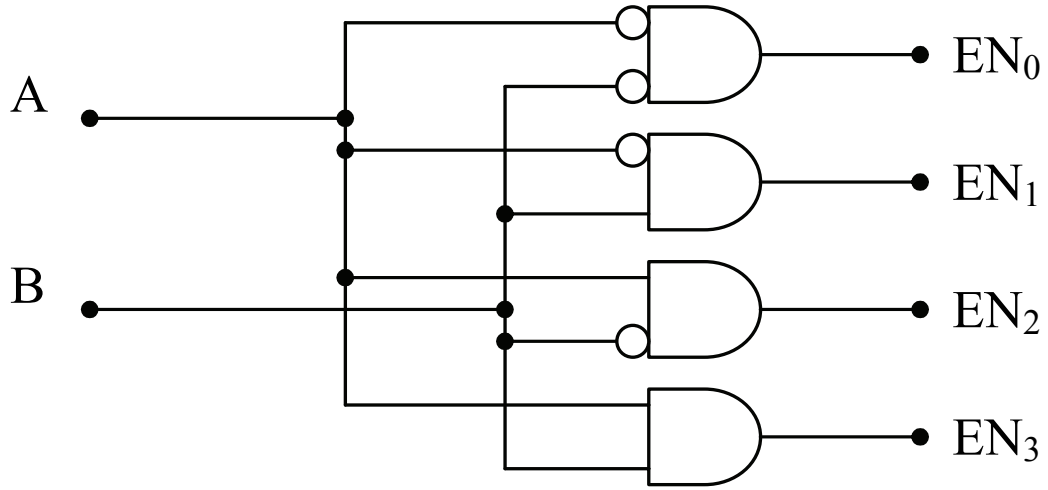
March 20, 2013

Today: Basic computational structures

- **Helpful structures**
  - Decoder/encoder
  - Multiplexor/demultiplexor
  - Sign extender
- **ALU**
  - ALU control
  - Zero detector
  - Set less than

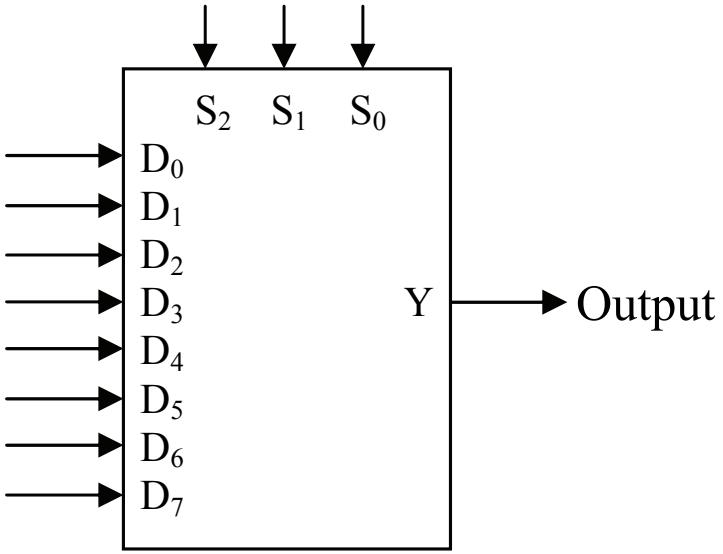
Decoder/encoder

- **Outputs unique signal based on input**
  - Inputs: state of systems
  - Output: unique representative code
  - 2 inputs = 2^2 outputs
- **Encoder reverses the process**



Multiplexor

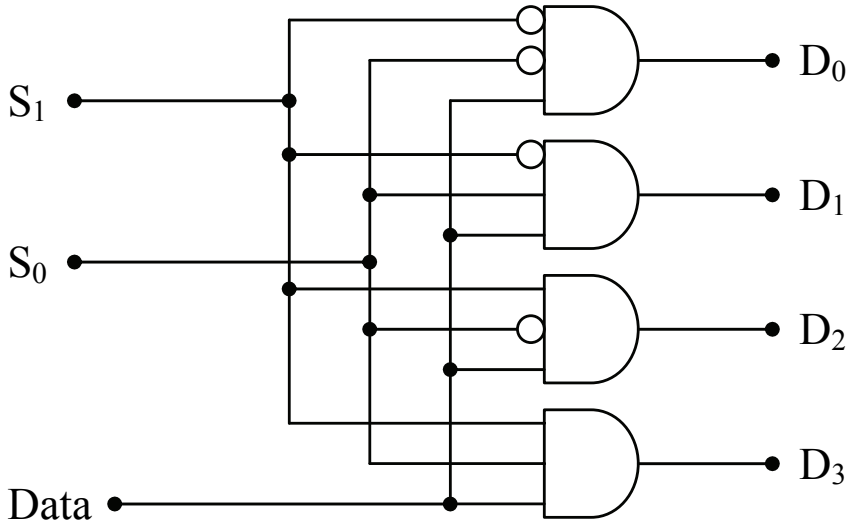
- **Select single data stream from multiple channels**
  - Multiple data inputs
  - Single data output
  - Control S selects single data stream



S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Demultiplexor

- **Outputs data to one of multiple data channels**
  - Single data input
  - Multiple data outputs
  - Control S selects data output



S_1	S_0	Data	D_0	D_1	D_2	D_3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

Sign extender

- **CPUs work with signed numbers**
  - Word size or smaller
  - Often need to convert to word size
- **Need to duplicate (extend) sign bit**
  - Preserves original number in larger container
- **How to do this?**

Sign extender

- **CPUs work with signed numbers**
  - Word size or smaller
  - Often need to convert to word size
- **Need to duplicate (extend) sign bit**
  - Preserves original number in larger container
- **How to do this?**
  - Just connect MSB input to sign extend bits!
  - Only need wires

ALU

- **Arithmetic Logic Unit**
  - Responsible for all computations in computer
  - Supported operations
    - AND
    - OR
    - Add
    - Subtract
    - Is less than
    - Is equal
    - Others: NOT, NOR, NAND…
  - Design is similar to adder
    - Start with 1 bit ALU, expand

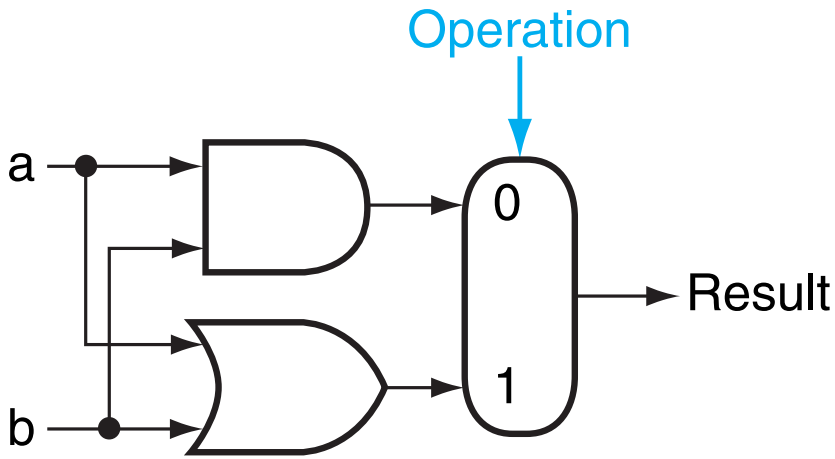
1 bit ALU

- **Start with AND and OR operations**
  - Inputs A and B
  - Select operation by control signal OP
  - Single output R

- **Hint: a multiplexor will help!**
  - Op 0 = AND
  - Op 1 = OR

1 bit ALU

- **Start with AND and OR operations**
  - Inputs A and B
  - Select operation by control signal OP
  - Single output R

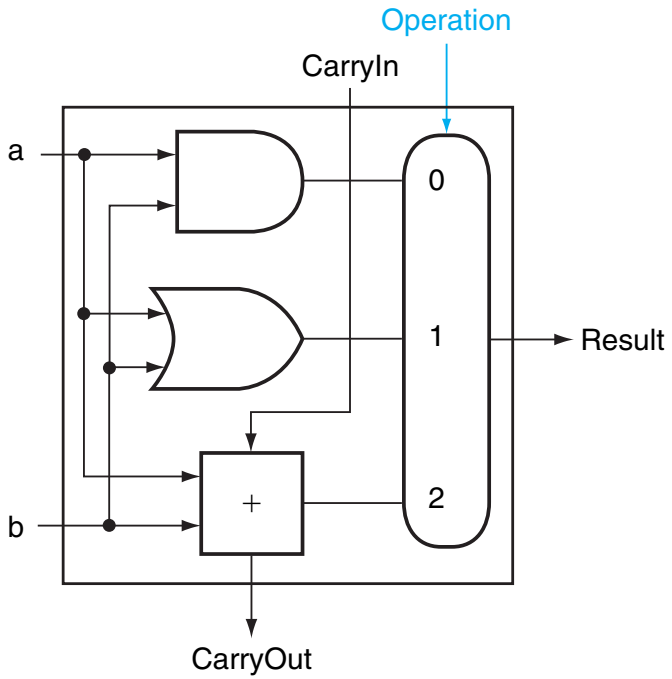


1 bit ALU

- **Add in ADD**
  - We'll use a full adder
  - Inputs A, B, C_{in}
  - Outputs S, C_{out}
  - $S = AB'C_{in}'+A'BC_{in}'+A'B'C_{in}+ABC_{in}$
  - $C_{out} = AB+BC_{in}+AC_{in}$

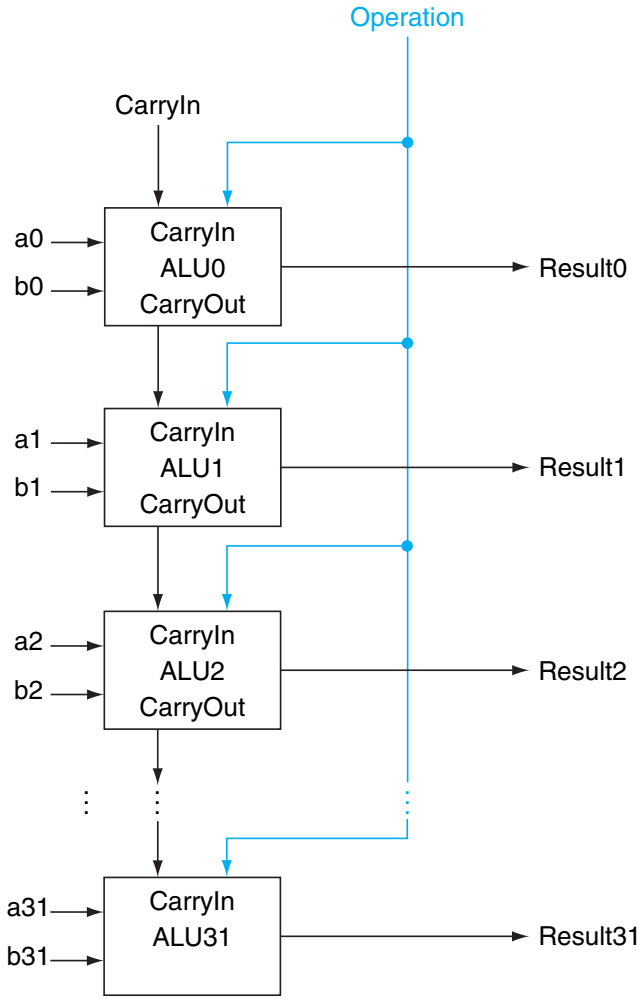
1 bit ALU

- **Add in ADD**
  - We'll use a full adder
  - Inputs A, B, C_{in}
  - Outputs S, C_{out}
  - $S = AB'C_{in}'+A'BC_{in}'+A'B'C_{in}+ABC_{in}$
  - $C_{out} = AB+BC_{in}+AC_{in}$
- **Need to expand mux**



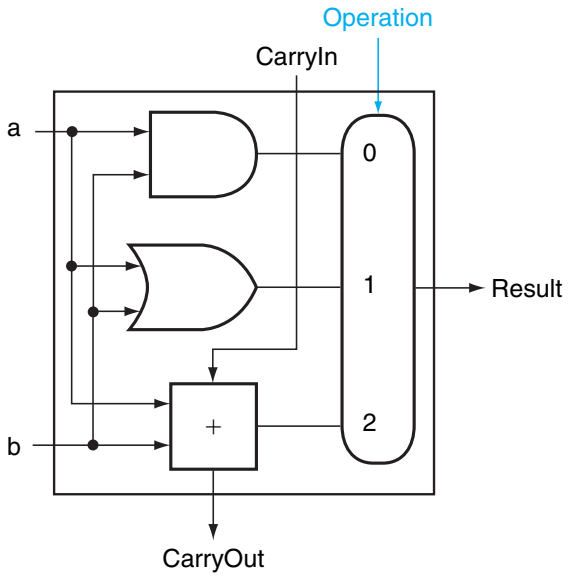
Wider ALU

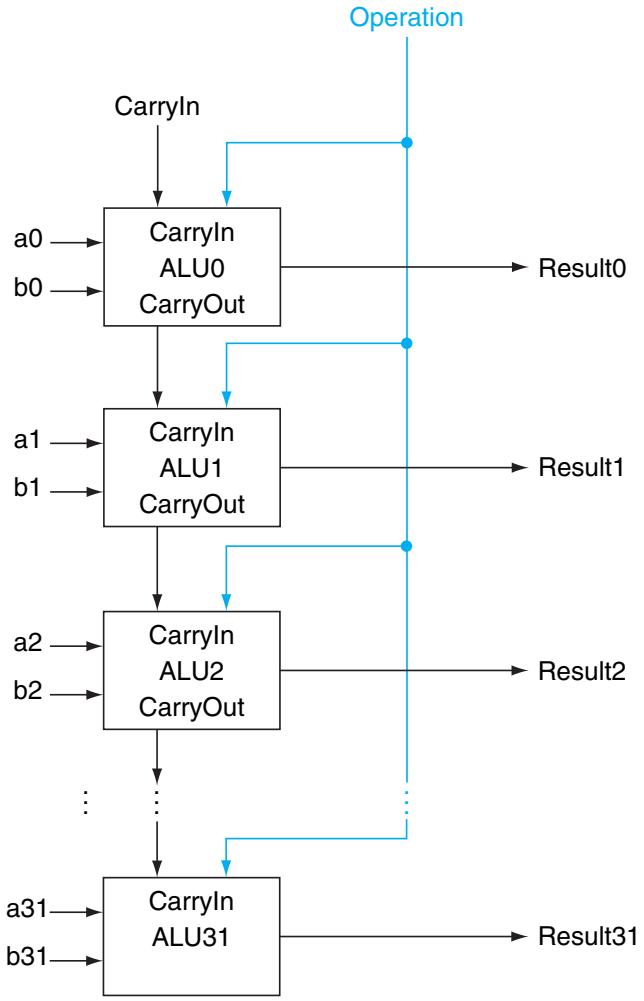
- **Can link 1 bit ALUs together to form large ALU**
  - 32 bit example



Subtract

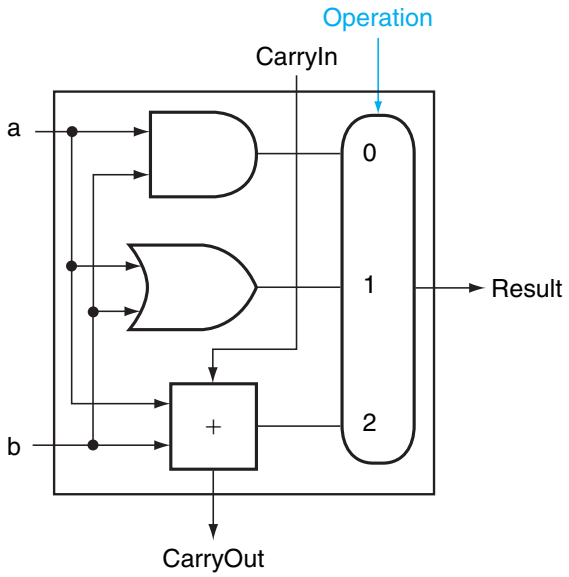
- **Given that 1 bit ALUs form larger ALUs, implement subtract**
  - Remember Two's complement!
    - -x = (~x)+1
- **What do we need?**

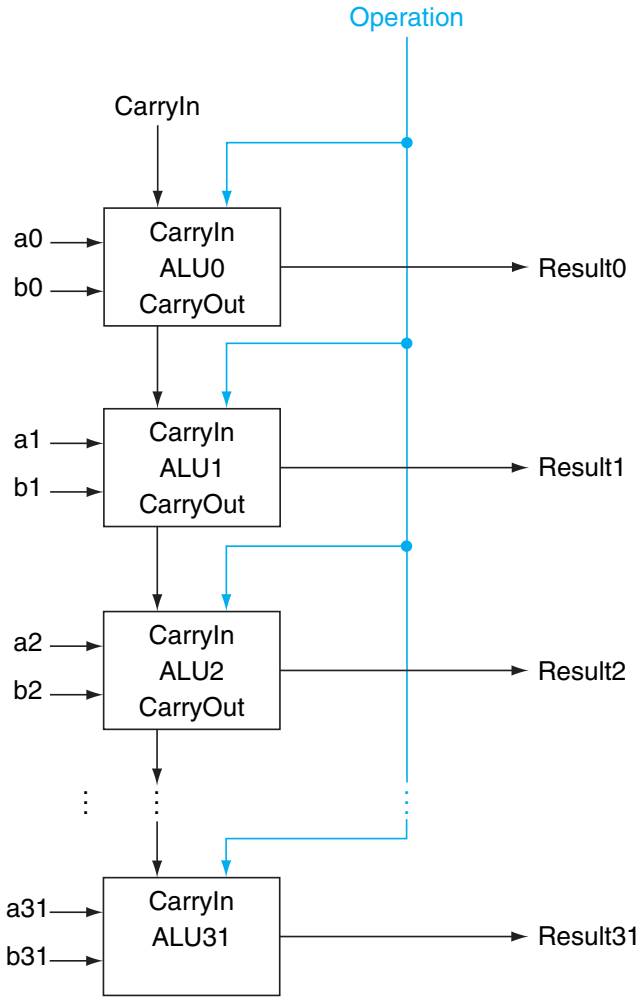




Subtract

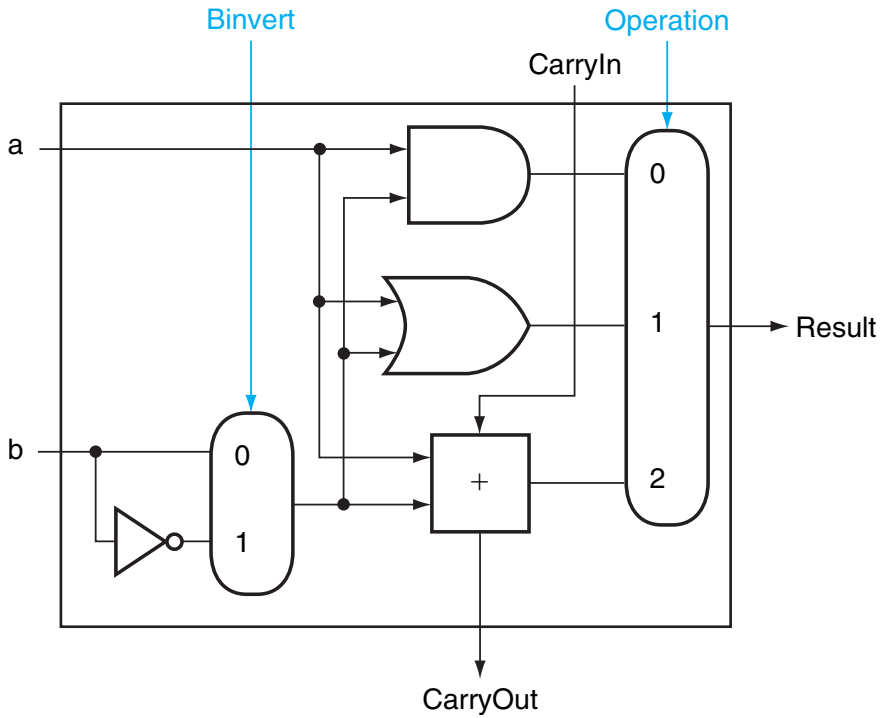
- **Given that 1 bit ALUs form larger ALUs, implement subtract**
  - Remember Two's complement!
    - -x = (~x)+1
- **What do we need?**
  - Invert B or A
  - Add 1 to LSB





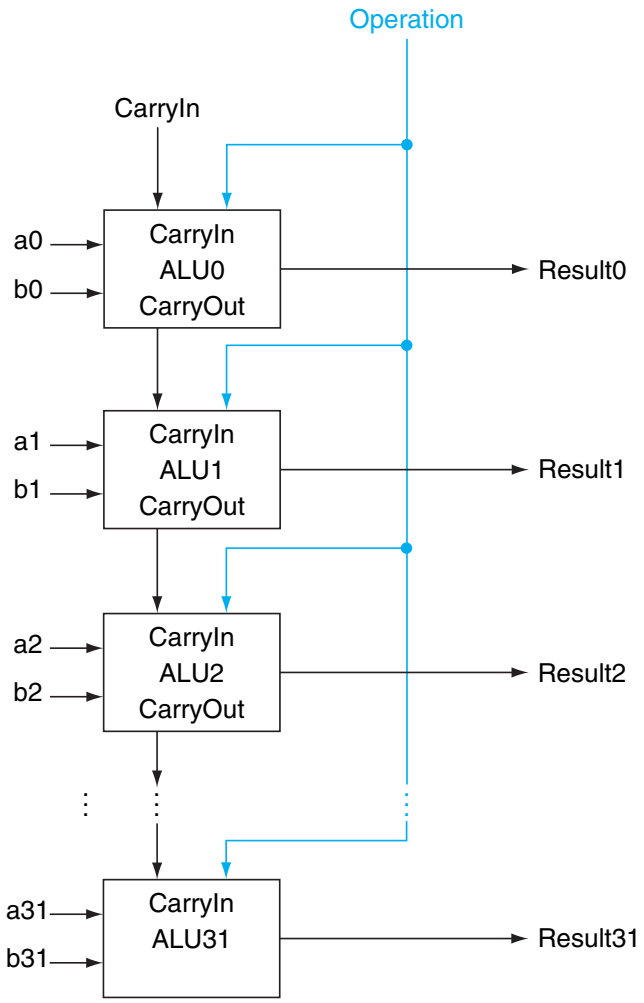
Subtract

- **Invert B**
  - Still need to use adder, so don't expand mux (keep using + op)
  - Add control line to select inverted B



Subtract

- **Add 1 to LSB**
  - If operation is subtract
    - Set LSB carry in to 1
    - Set ALU op to +
- **This incredible convenience is why most computers use two's complement**



Equal

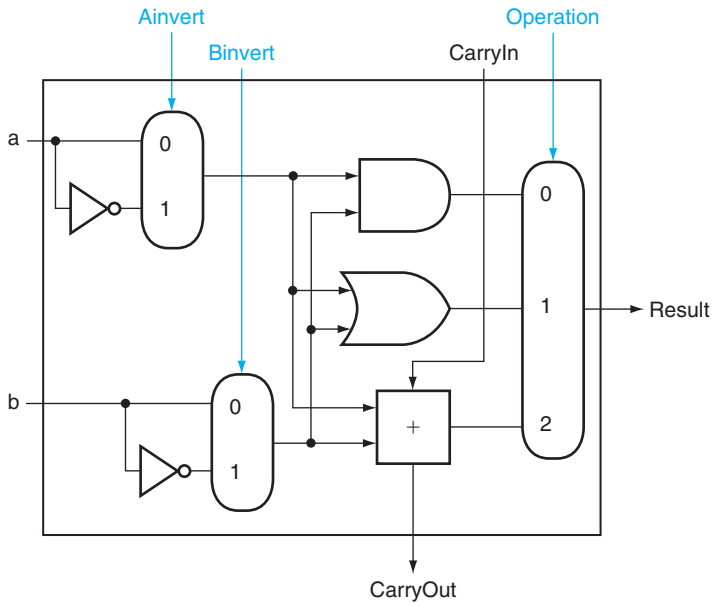
- **Add 1bit ‘zero’ output to ALU**
  - Set to 1 when A and B are equal

- **How to do?**
  - Subtract A and B
  - If all bits are 0, must be equal!
  - OR all bits
  - Invert result

Other operations

- **Could add more operations**
  - NOR (invert A)
  - Shifting (special hardware)
  - Many others…

ALU with NOR support



Set less than

- **If A < B**
  - R = 0x00…01
- **If not A < B (i.e. A >= B)**
  - R = 0x00….00

- **How to do this?**
  - Subtract is useful
  - Sign-bit (MSB) is useful
  - Need to expand mux for new operation

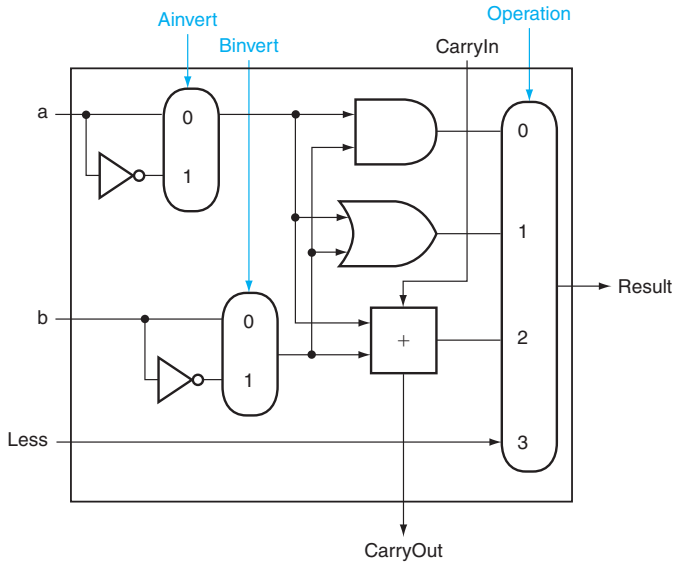
Set less than

- **If A < B**
  - R = 0x00…01
- **If not A < B (i.e. A >= B)**
  - R = 0x00….00

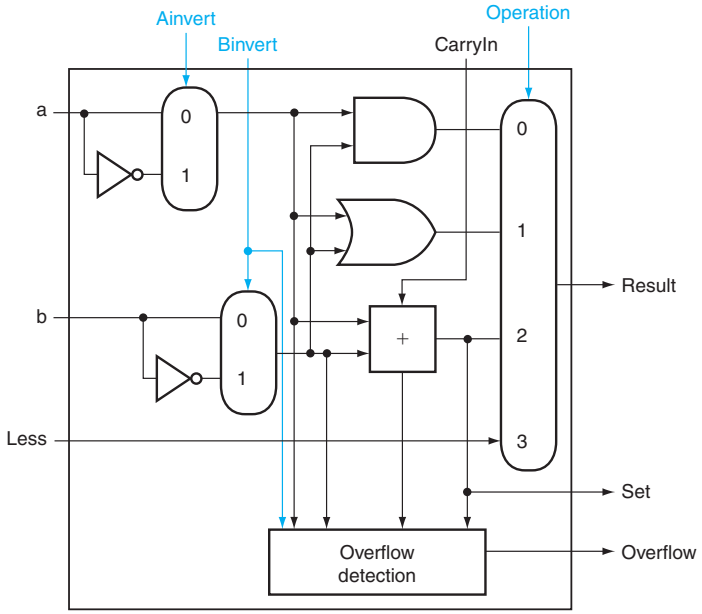
- **How to do this?**
  - Set LSB to MSB (sign bit)
  - Output 0 for all other bits

Set less than

- **If A < B**
  - R = 0x00…01
- **If not A < B (i.e. A >= B)**
  - R = 0x00….00

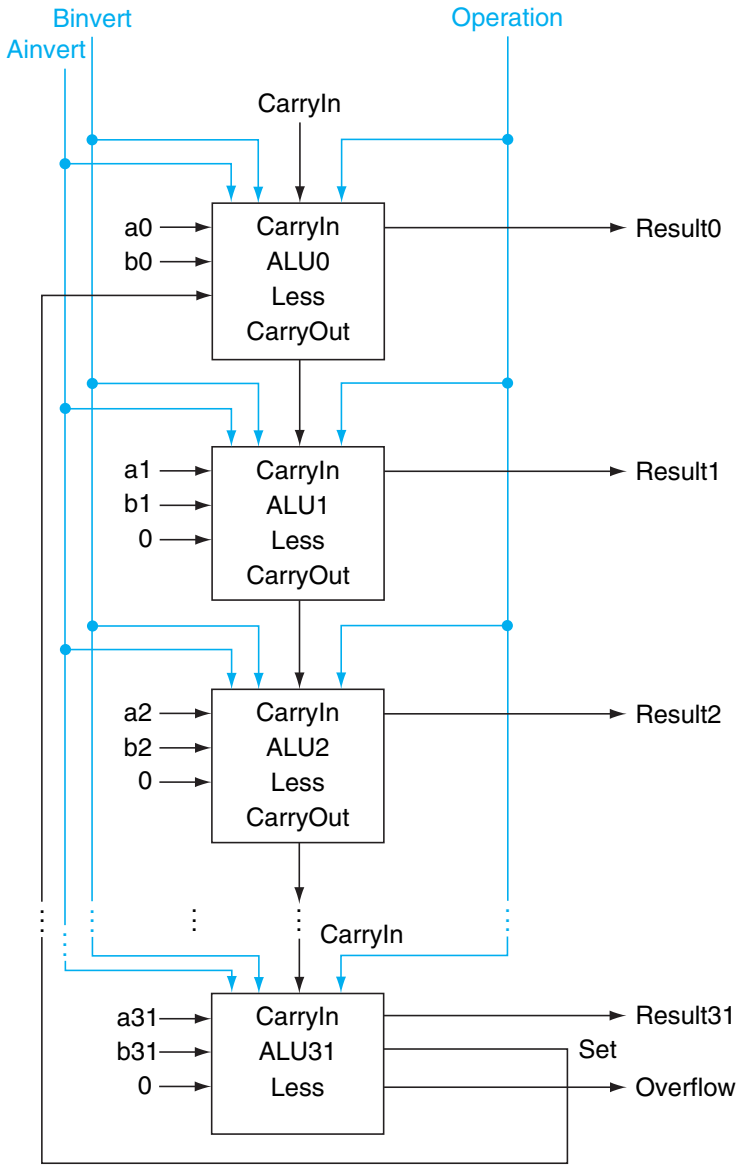


- **How to do this?**
  - Add new input ‘less’
    - Can 0 to result mux
  - Add new output ‘set’ to MSB ALU
    - Output MSB result
    - Use later



Set less than

- **If A < B**
  - R = 0x00…01
- **If not A < B (i.e. A >= B)**
  - R = 0x00….00
- **How to do this?**
  - Set ALUs to subtract
    - MSB ‘set’ is now sign bit
  - Pass MSB ‘set’ to LSB ‘less’
  - Set all other ‘less’ to 0



Set less than

- **If A < B**
  - R = 0x00…01
- **If not A < B (i.e. A >= B)**
  - R = 0x00….00
- **Result**
  - Subtract results in < 0
    - Sign bit of 1 is sent to LSB
  - Subtract results in >= 0
    - Sign bit of 0 is sent to LSB
  - All other bits set to 0