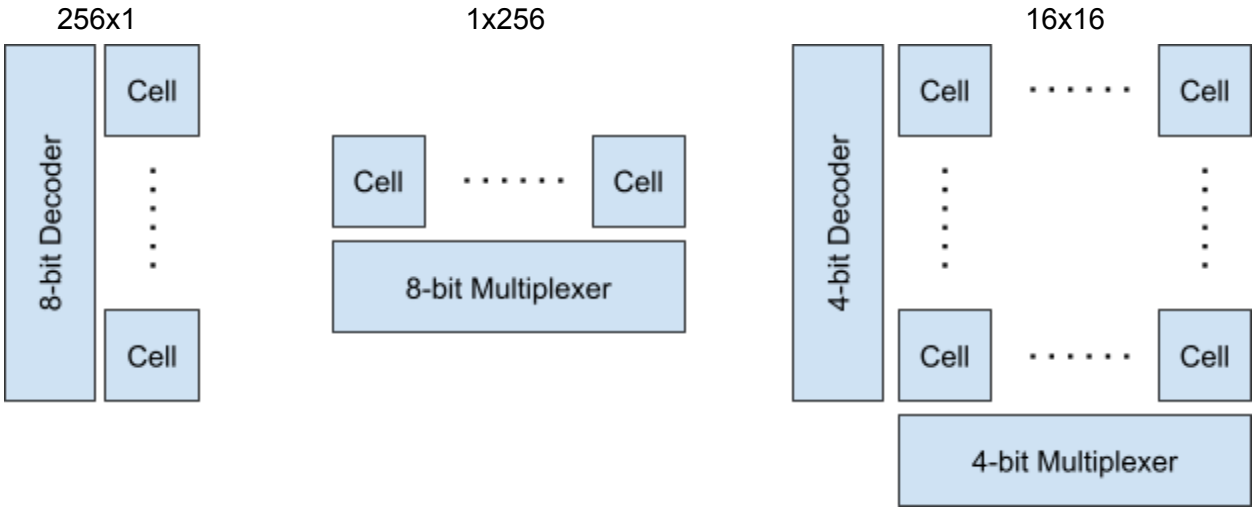
Tutorial 2

Question 1.

For a 256x1 RAM, you have addresses 0, 1, …, 255 totalling to 256 distinct addresses. To represent these addresses, you need at least 8 bits.

Consider the following shapes for the cell matrix: (lines and sense circuit omitted)



You have very long bitlines for 256x1 and very long wordlines for 1x256. Long lines have high capacitance, large RC time constant, and high delay.

In the 256x1 case, your address decoder is huge. As an estimate, the circuitry needed for it is on the order of 256 (since you have at least one wire for each wordline). Similarly, for the 1x256 case, your multiplexer is huge for a similar reason as the 256x1 case.

The 16x16 layout has shorter wordlines and bitlines compared to the worst cases of 256x1 and 1x256. It also replaces one big decoder / multiplier with two smaller ones, each on the order of 16.

That is to say, a square cell matrix is better because:

- Reduces bitline and wordline capacitance
- Reduces delay
- Is easier to fit

Tutorial 2

Question 2.

Decimal	Binary (8-bits)		
	Sign magnitude	1's complement	2's complement
0	0000'0000 / 1000'0000	0000'0000 / 1111'1111	0000'0000
25	0001'1001	0001'1001	0001'1001
-25	1001'1001	1110'0110	1110'0111
-127	1111'1111	1000'0000	1000'0001

Correction:
This is what happens when you force 137 to be an 8-bit signed integer.

Binary (8-bits)	Decimal			
	Unsigned	Sign magnitude	1's complement	2's complement
1000'1001	137	-9	-118	-119

Notes:

- Sign magnitude and 1's complement have different representations for ±0
- From here on out, assume integers to be either unsigned or 2's complement
- Sign magnitude is still used in floating point numbers

Question 3.

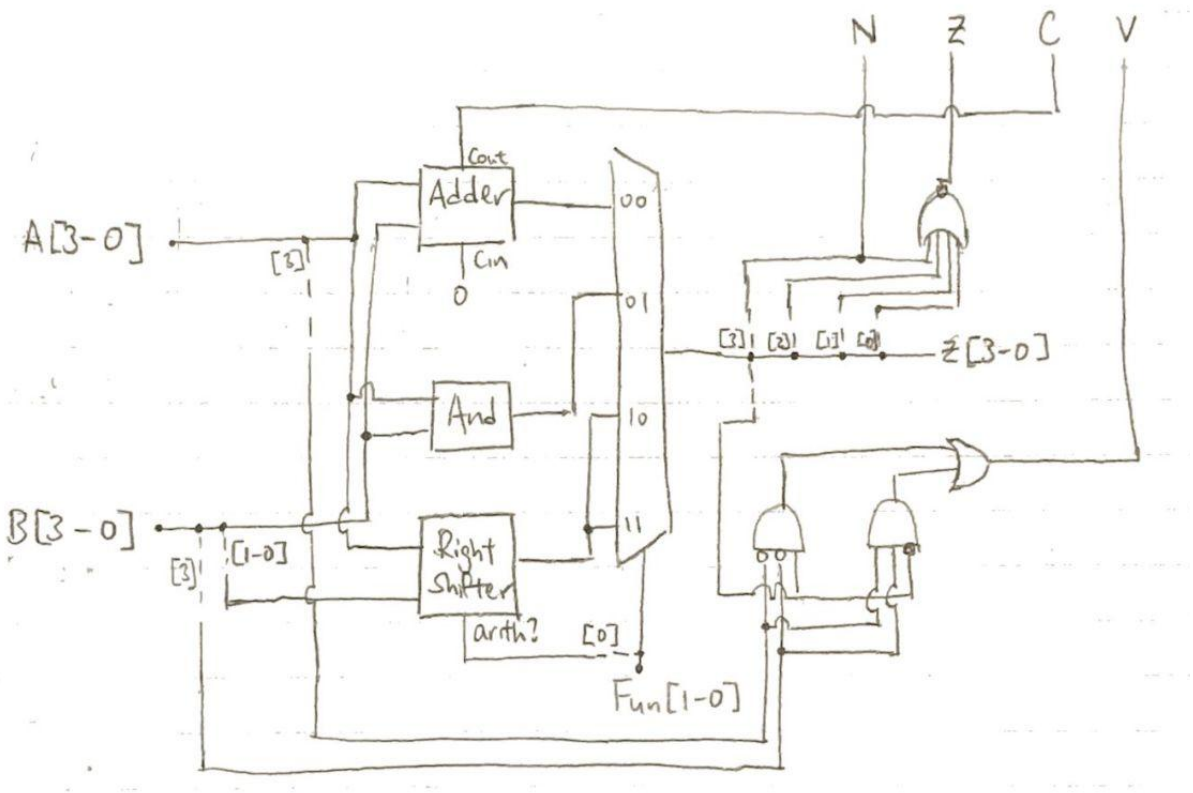
Flip all the bits then add 1.

Note:
Overflow **can** happen during the addition. For example: take -8, which is the most negative number possible in a 4-bit signed integer. You would start with 1000 (in binary), flip the bits to get 0111, then add 1 to end up with 1000, which is the same as what you started with!

Tutorial 2

Question 4.

ALU:



Correction:
The V flag for signed overflow depends on the sign of inputs to the adder and the sign of the result.

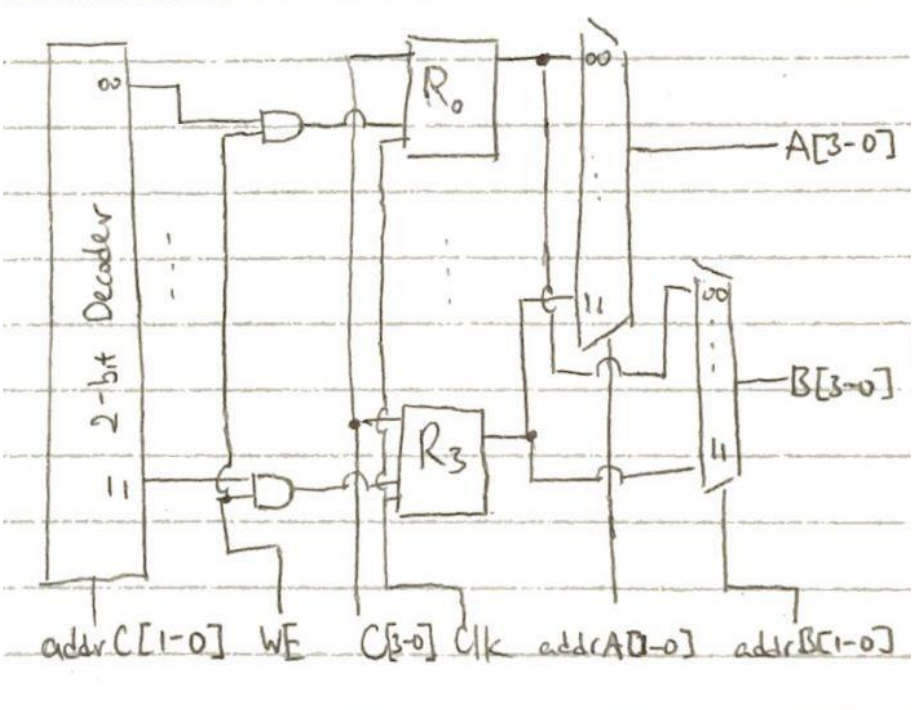
Notes:

- B[1-0] means we consider the 1st and 0th bit of B, which is effectively the lowest two bits of B.
- Right shift moves the bits to the right, creating empty spaces on the MSB side and dropping bits on the LSB side. Whereas logical (>>>) fills the empty spaces with 0's, arithmetic (>>) fills the empty spaces with the MSB of the original value.
- Right shifter is attached at the end of this file.

Tutorial 2

Question 5.

Register file: (the middle two 4-bit registers omitted)



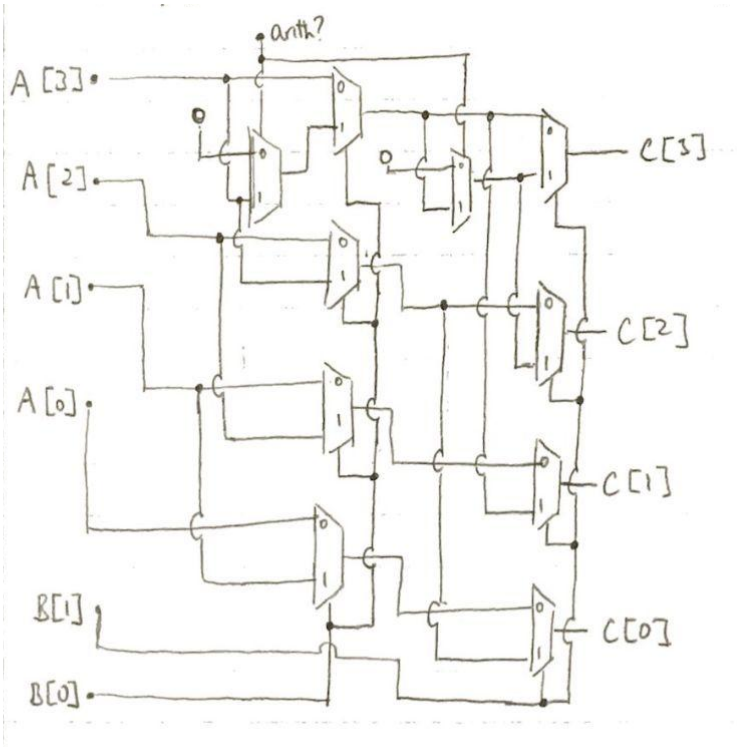
Notes:

- Consult the lecture slides for a more clear diagram
- An individual register is attached at the end of the file

Tutorial 2

Extras.

Right shifter: (the “arith?” input decides if it is arithmetic or logical)



Each individual register: (the middle two D flip-flops omitted)