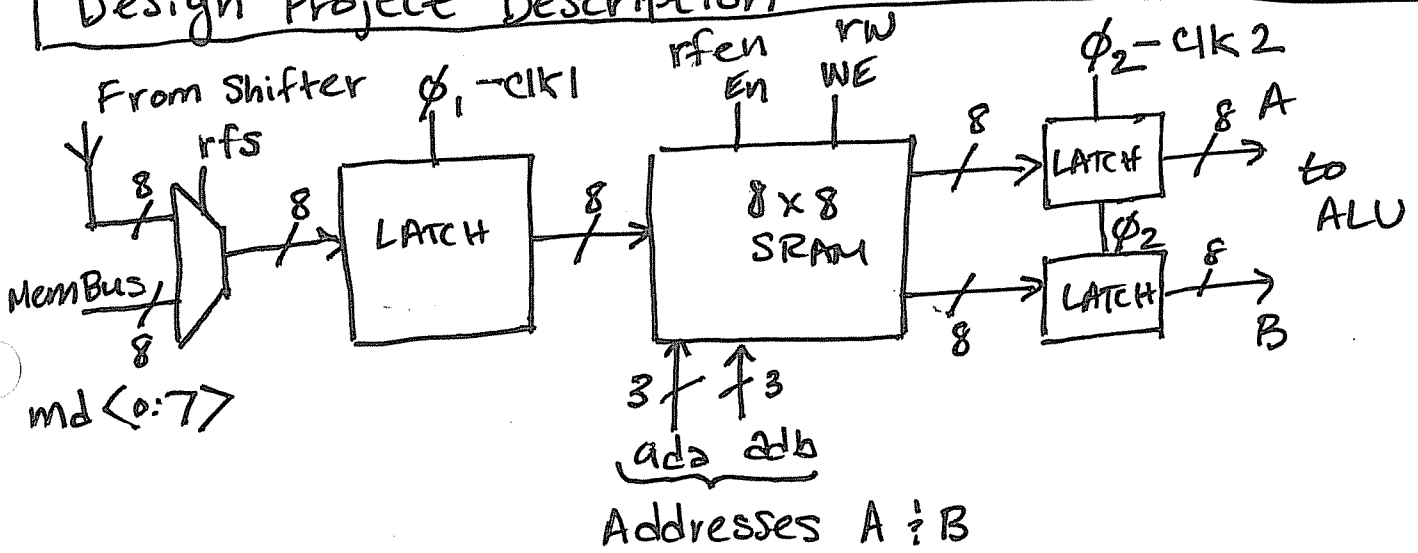
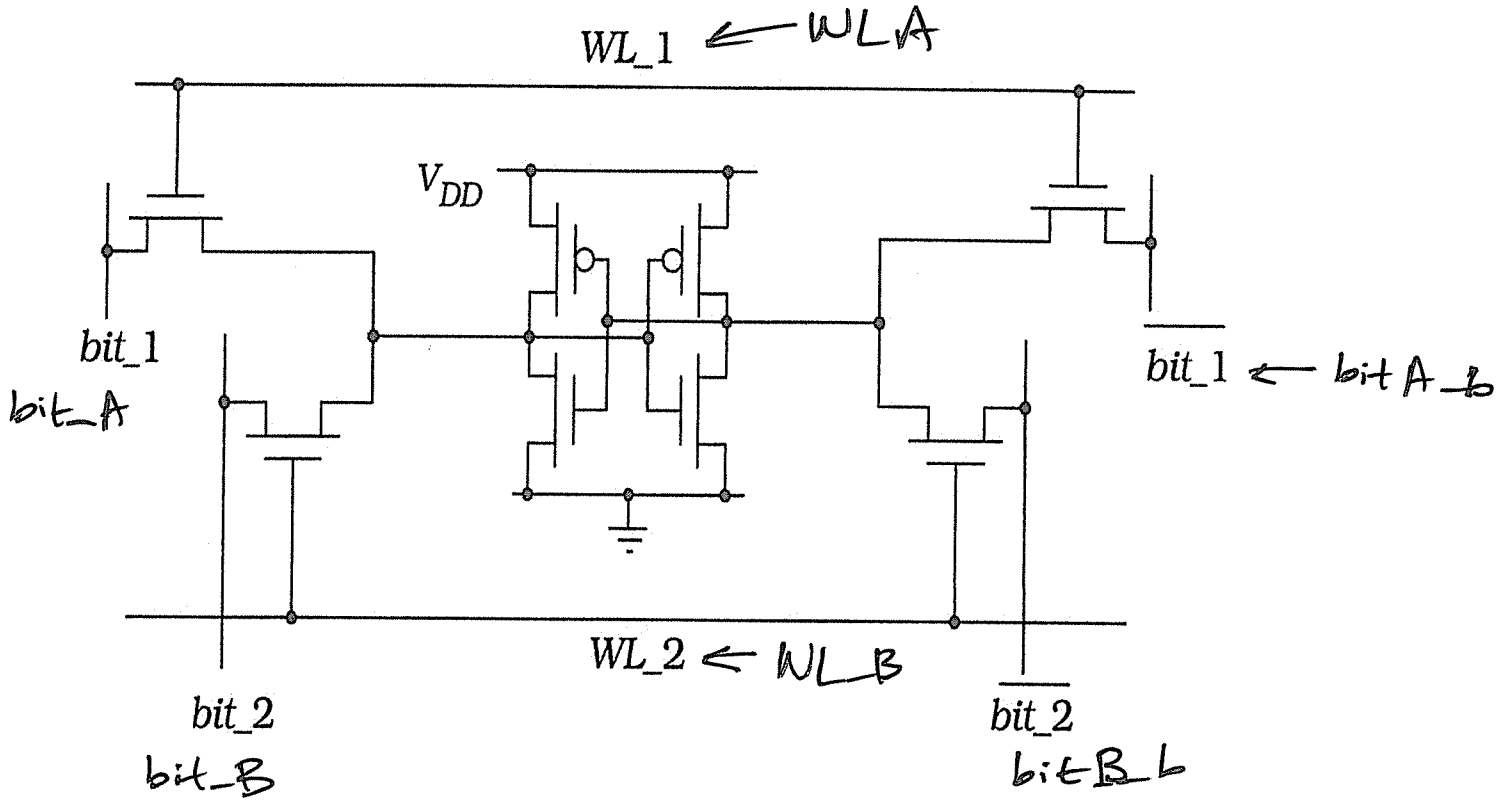
4/2/12 Please turn in your check-ins and your Group's Final Project Proposal.

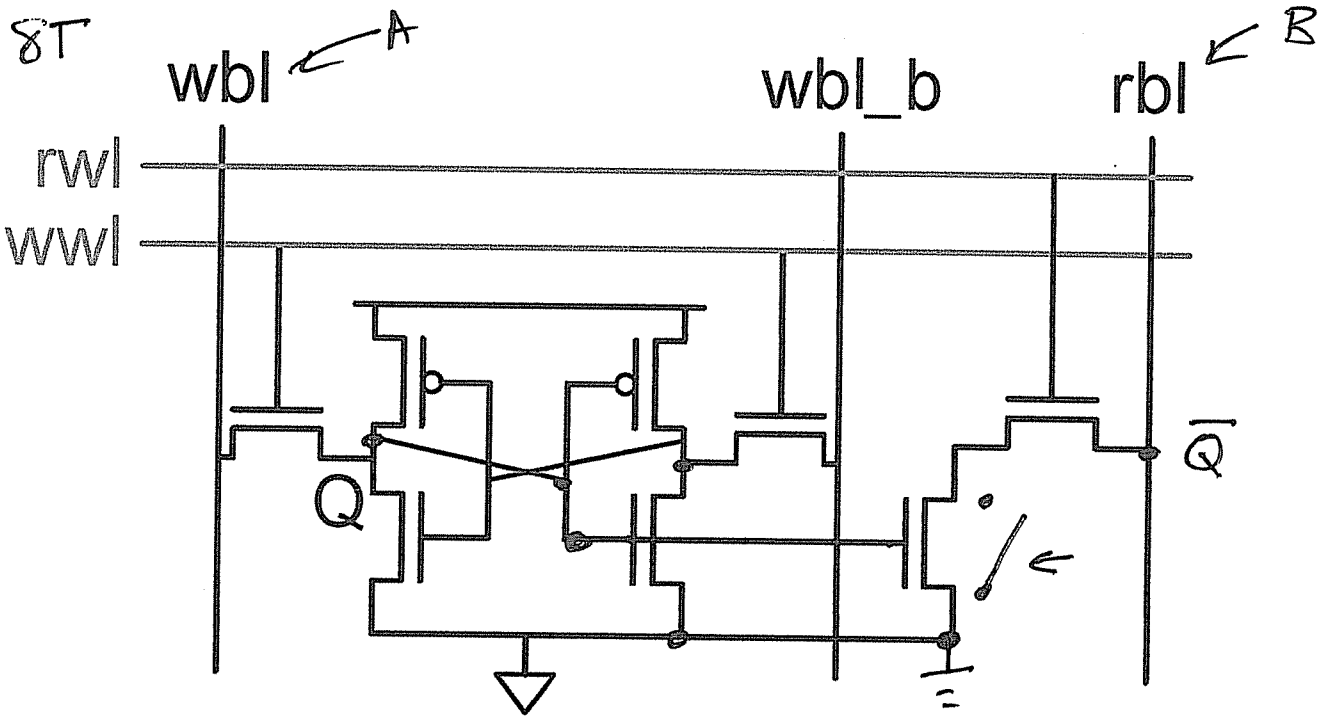
Reminders:
- Homework 8 due Wed
- Check-off Lab 9 by Friday, 4 PM
- No lecture Friday (Lab Day)

WARM UP: Label the SRAM and peripheral input/outputs with the appropriate signal names from the Final Design Project Description

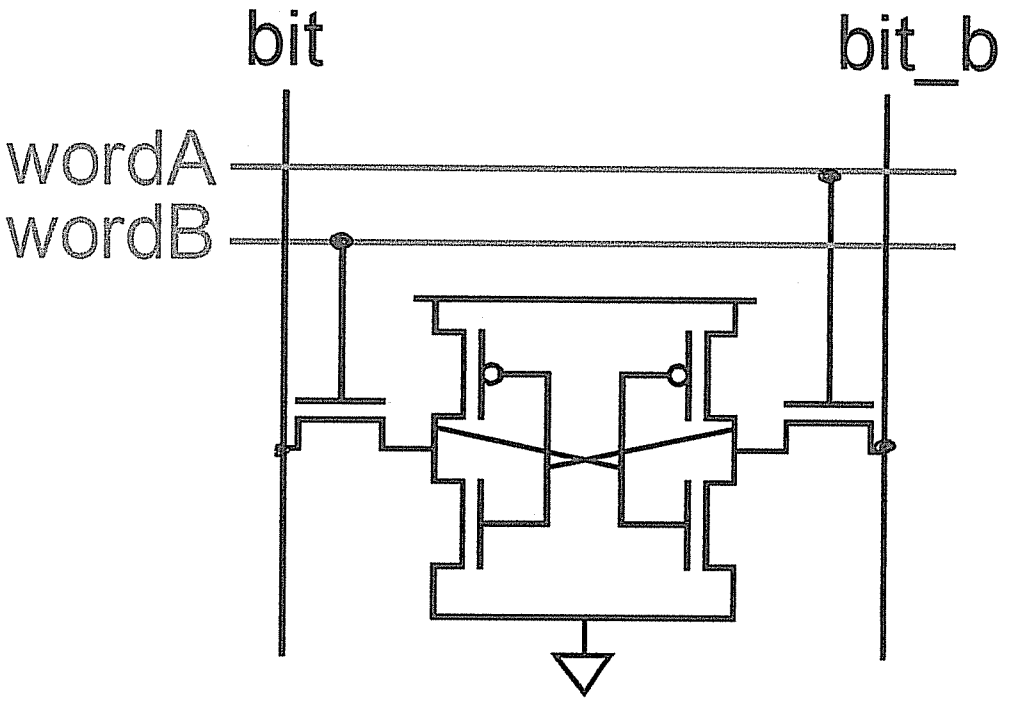


Correction from last lecture: In order to have two outputs A & B, you do need to use a multiport cell. Consider how to implement 2 read/write or 1 read/write, one read port cells using the multiport designs discussed last lecture:


WL_1 ← WL_A
V_{DD}
bit_1
bit_A
bit_1 ← bit_A_b
WL_2 ← WL_B
bit_2
bit_B
bit_2
bit_B_b

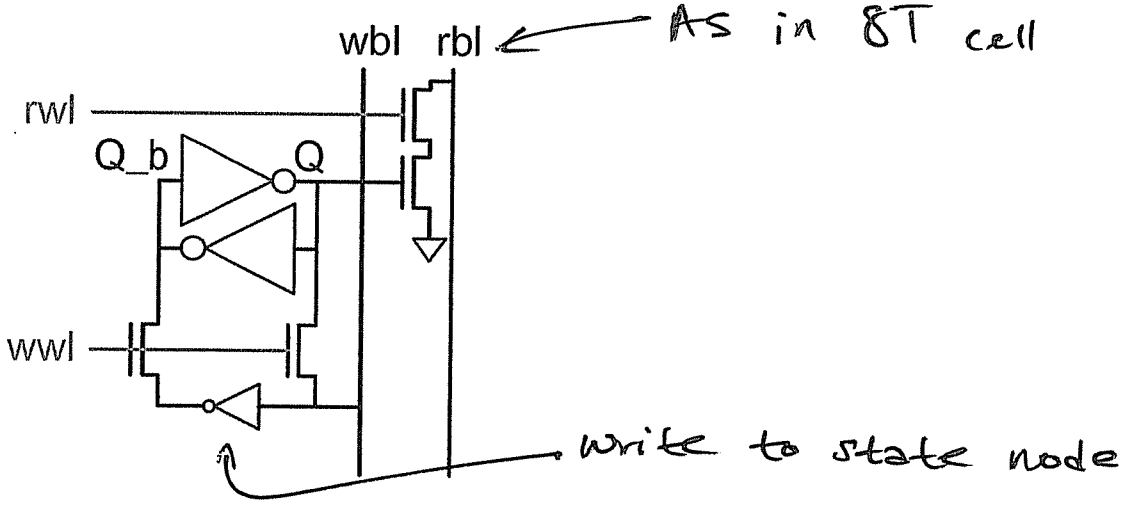

8T
wbl ← A
wbl_b
rbl ← B
rwl
wwl
Q
$\overline{Q}$

Other Multiport Designs:

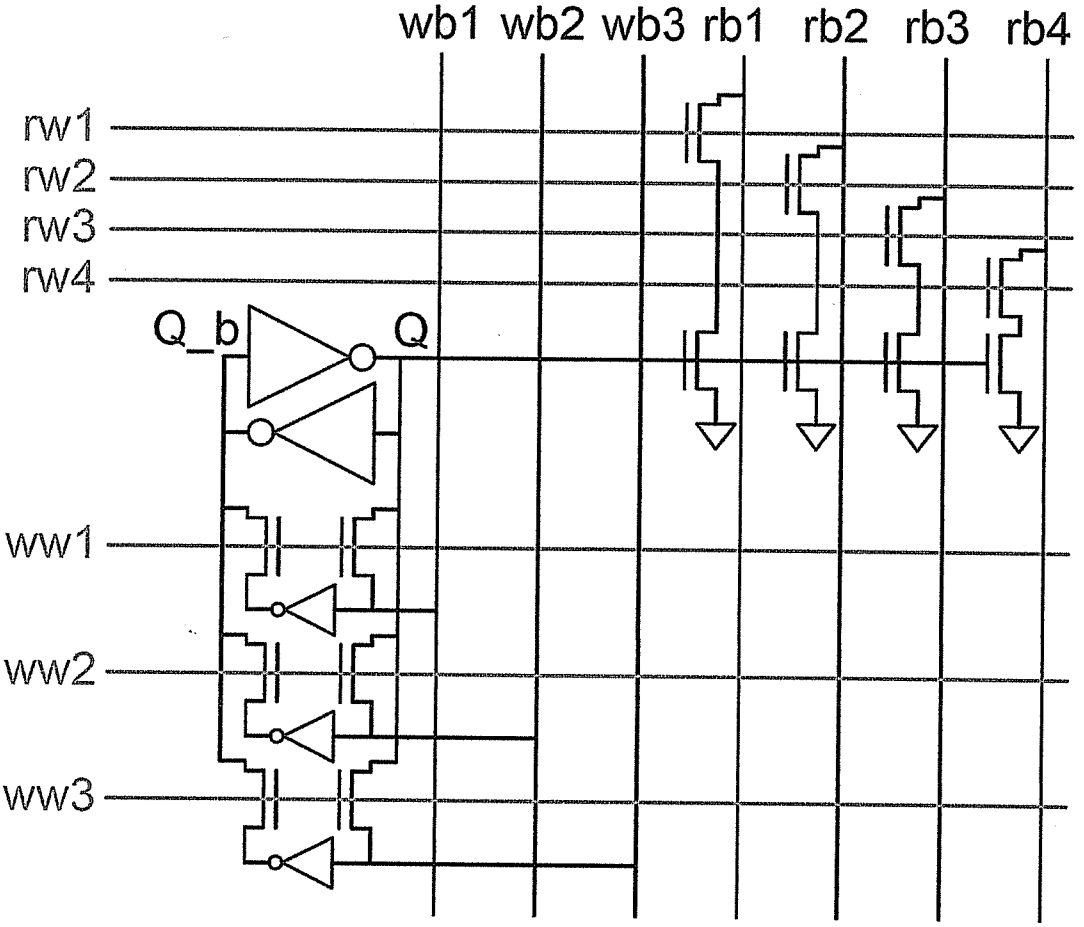


6T cell with 2 word lines. One read appears on bit and the other appears as the complement of the output, on bit_b. In the example shown above, wordA would need to be inverted as an output, since it is connected to the Q_b node. Write still requires both bit and bit_b, so only a single write can occur.

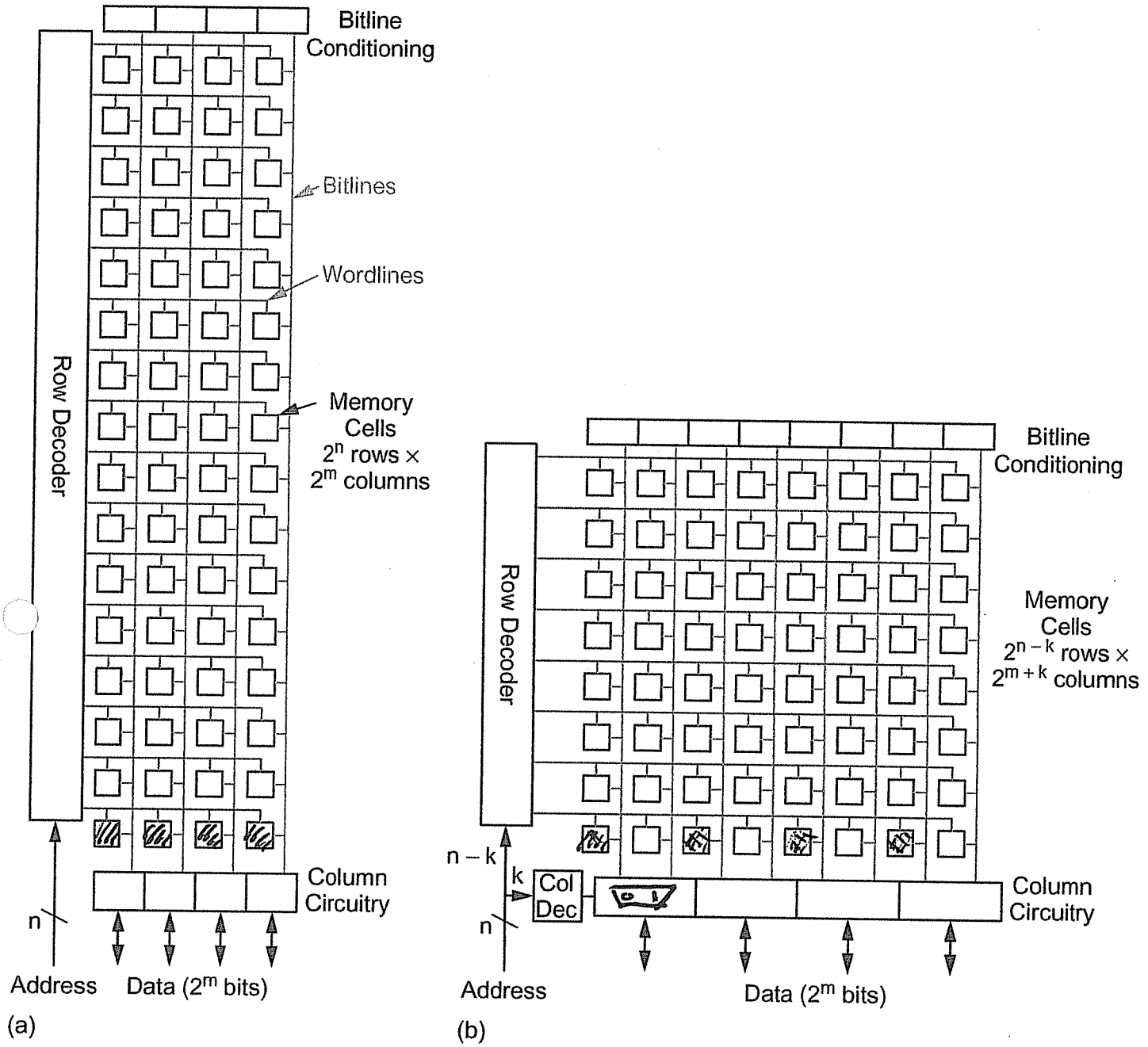
SRAM cells often require multiple input and output ports in modern register files. For example, the Itanium 2 processor issues up to 6 integer instructions in a cycle, each of which requires two source registers and a destination. The register requires four more write ports for late cache returns, for a total of 12 read ports and 10 write ports. How can an SRAM cell accommodate that many ports?



Example: 4 read port, 3 write port cell:



13.2 SRAM Arrays

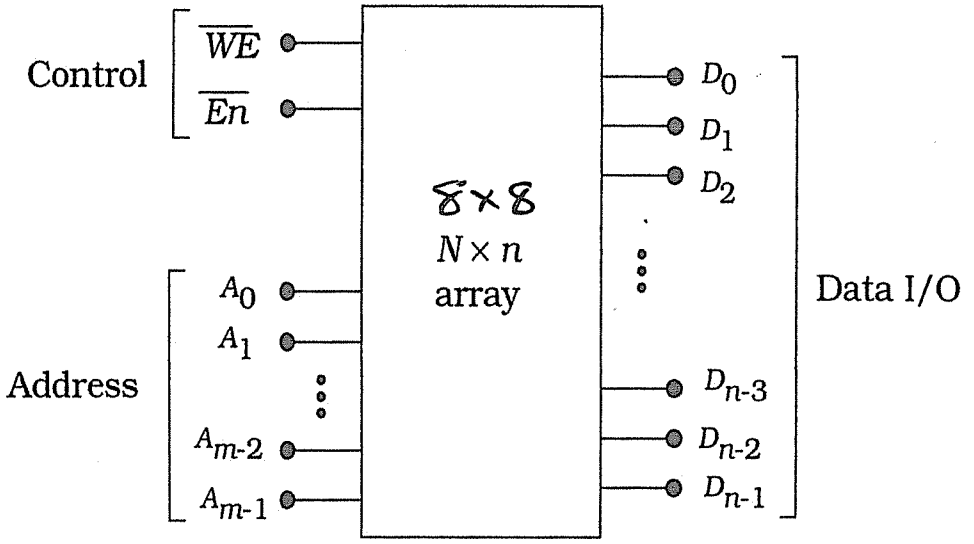


Two possible arrays for 16 4-bit words

To implement an array we need:

1. Row circuitry (address decoder and word line drivers)
2. Column circuitry (bitline conditioning, write driver and column multiplexers)

High Level View:



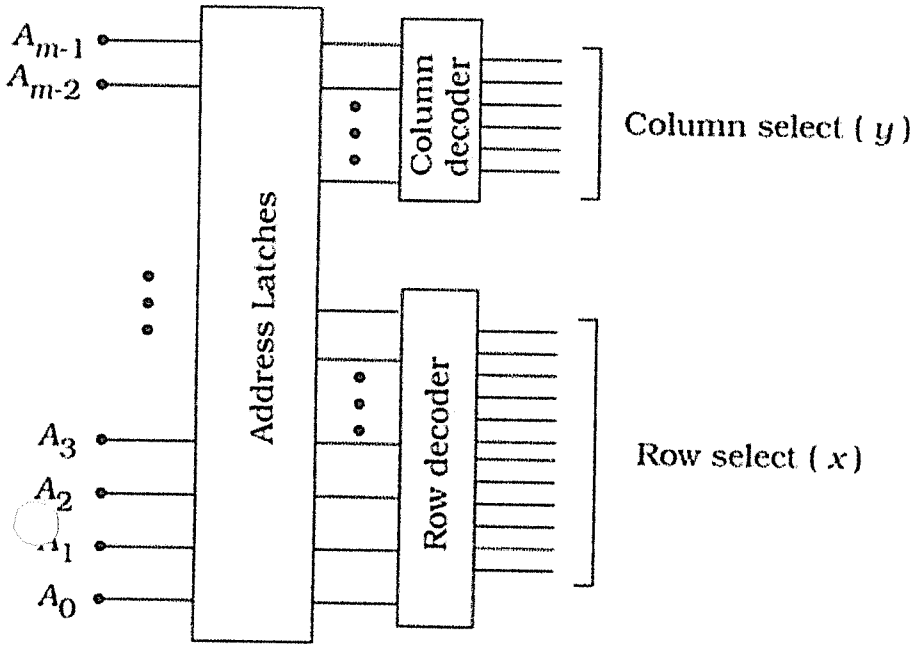
$$N = 2^m$$

WE : Write Enable, assert low control, allows write when WE=0, WE=1 indicates a read operation.

En: Assert low enable signal. When $En = 1$, read and write are both disabled and memory is in a hold state.

Row Circuitry

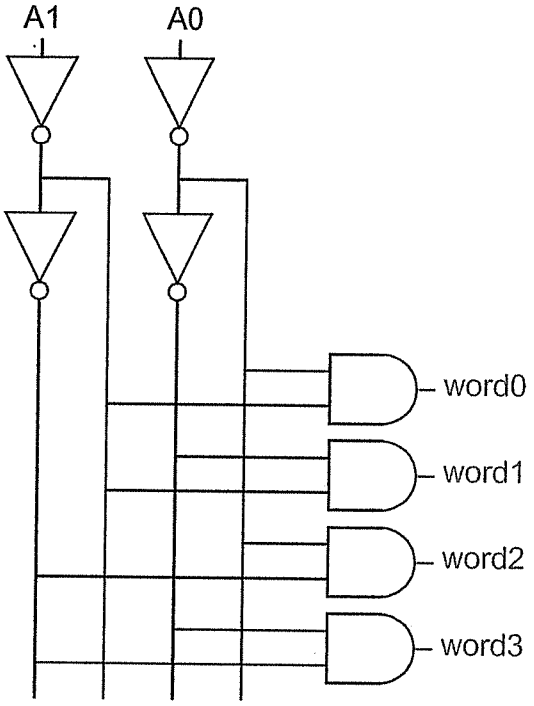
Address decoder

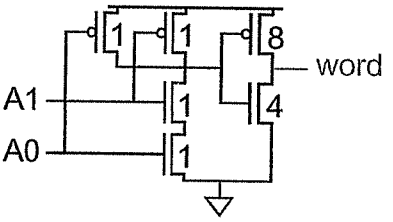


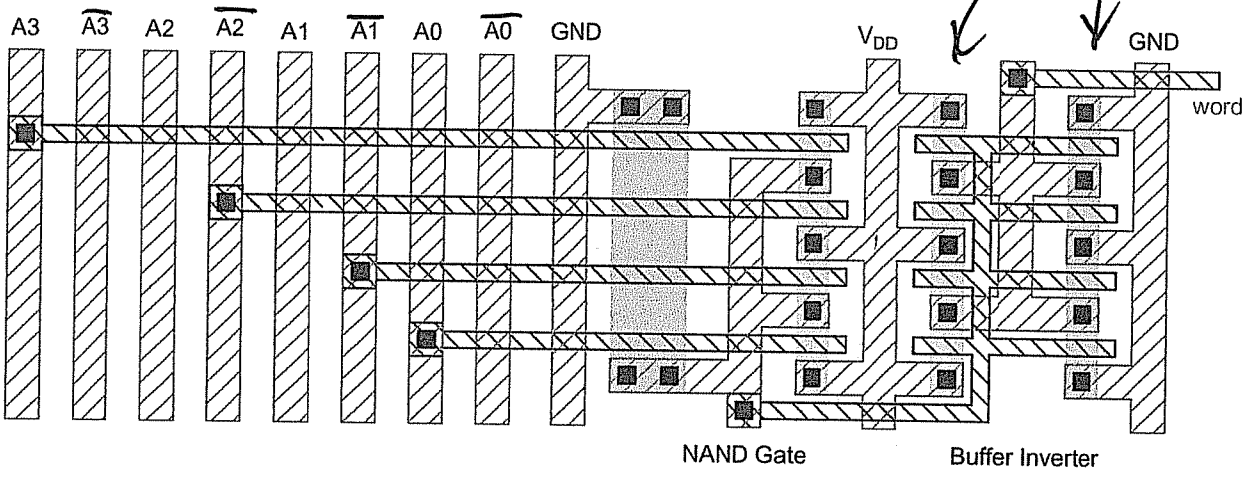
Consider an m-bit address word $A = A_{m-1} \ldots A_2 A_1 A_0$ in an array divided into x rows and y columns with $x + y = m$.

To decode the row address, the simplest decoder is an active-high binary decoder composed of true and complementary inputs and AND gates.

To make the AND, the design shown here is a minimum sized NAND followed by an inverter. The NAND can be made minimum size to reduce the load on the address lines, because there are 2^{m-y} transistors on every address line in the row decoder. The buffer inverter is larger to drive the word line.



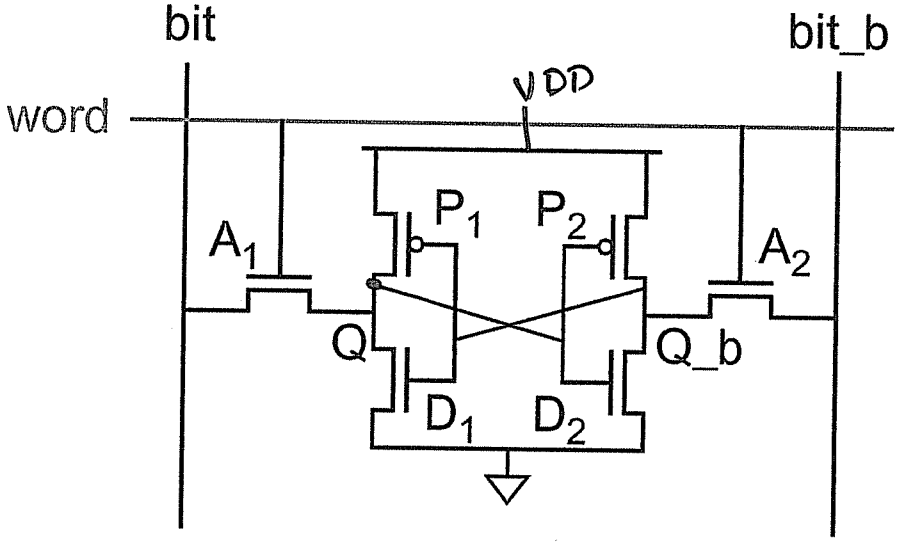




Column Circuitry

Bitline Conditioning

Refresher: How is an SRAM able to both read and write?



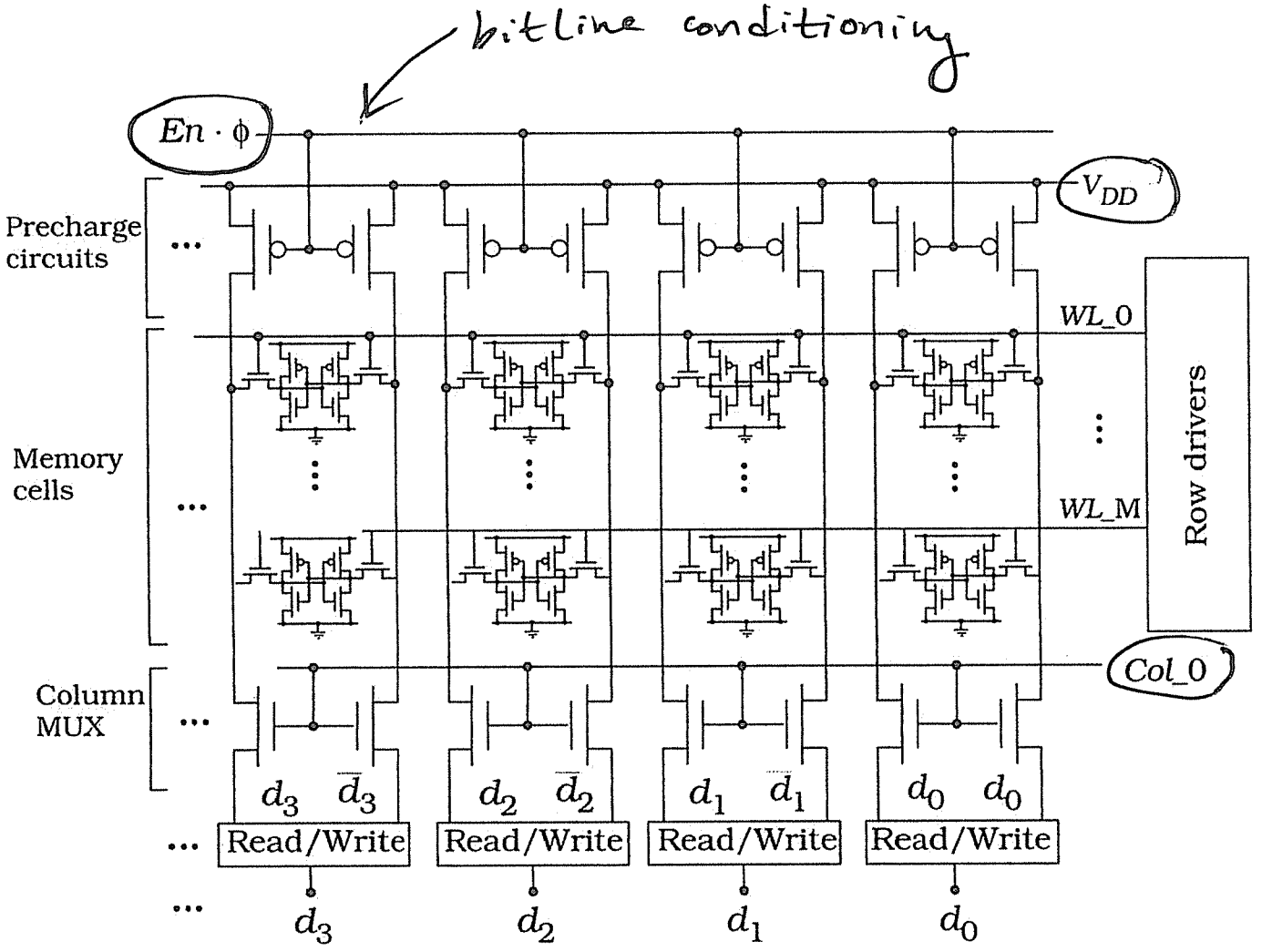
D > A ⇒ bit cannot set Q high

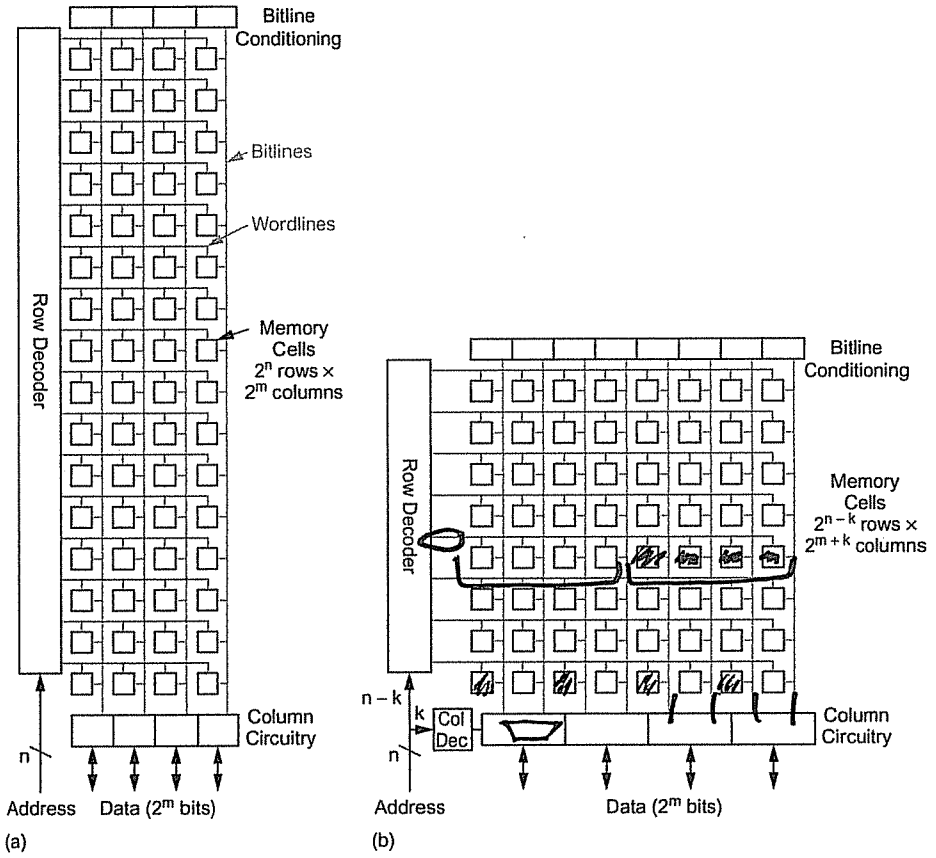
Always need bit or bit_b to get pulled low

P < A ⇒ So bit CAN set Q low

Write: Either Q or $\overline{Q}$ to be pulled low

So, as a result of this analysis we can see that in order for bit or bit_b to get pulled low during a read cycle, they must be kept always HIGH, then left to FLOAT during read. This is called bitline conditioning, and is easily accomplished with pFETs gated by the EN signal.

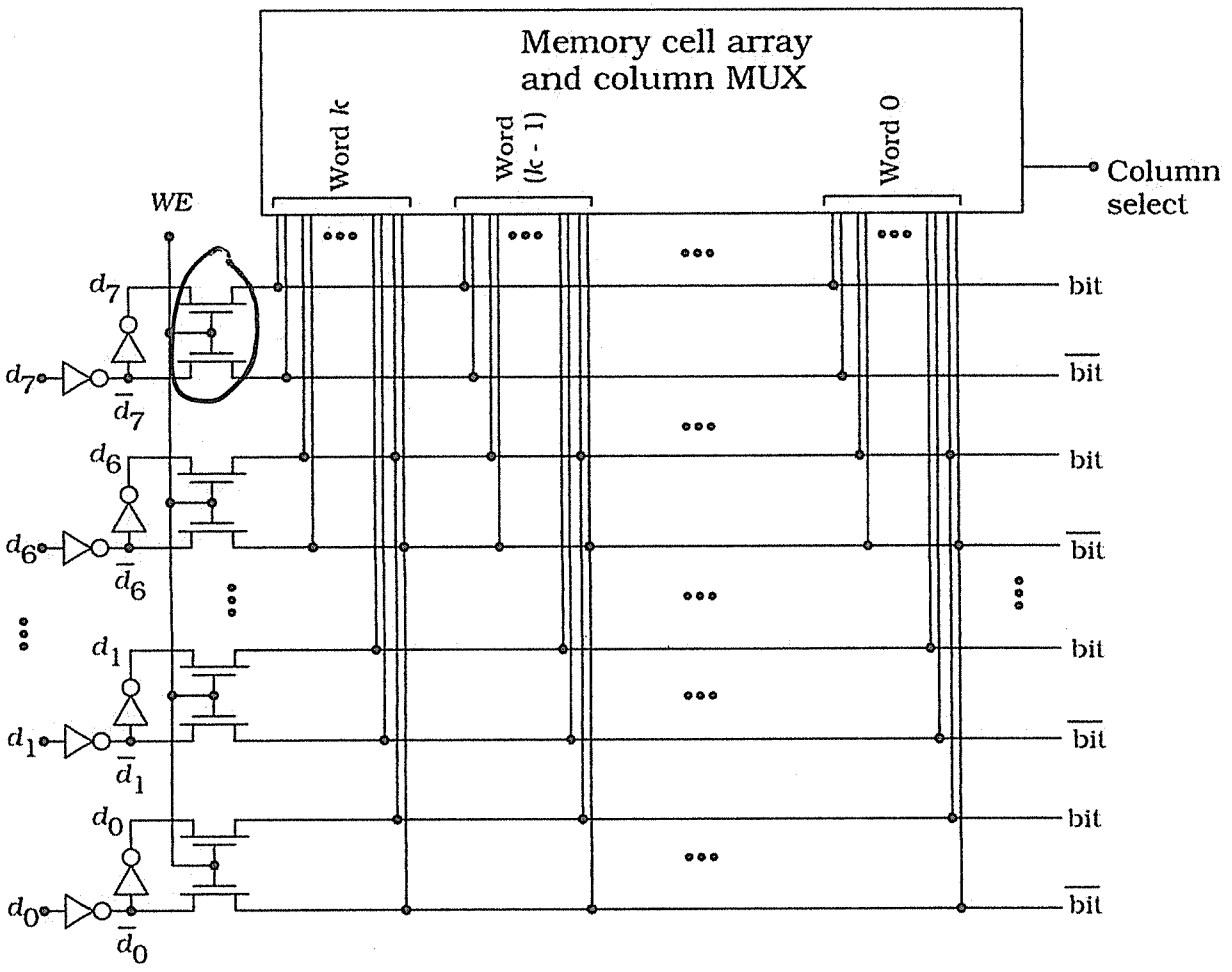




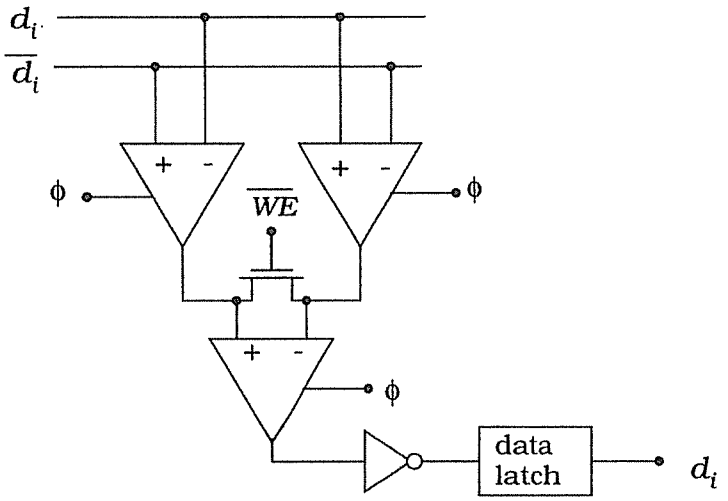
The column MUX selects the appropriate column to read/write, (if there is more than one bitline per column)

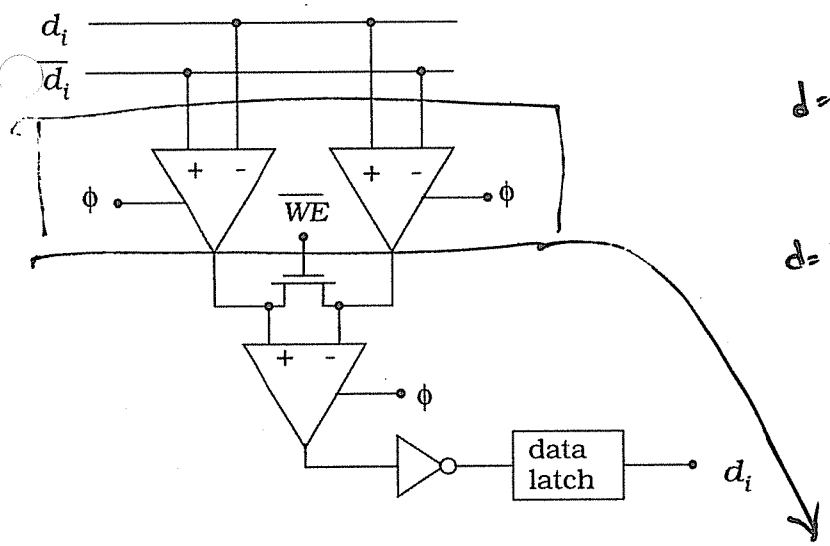
Read/Write:

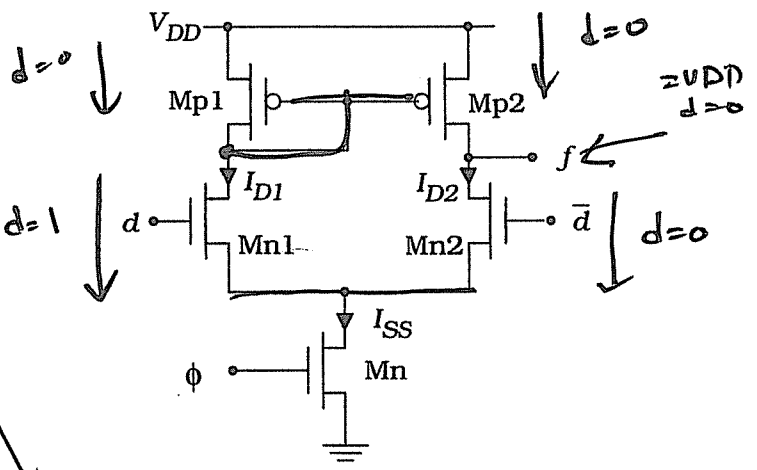
Write:

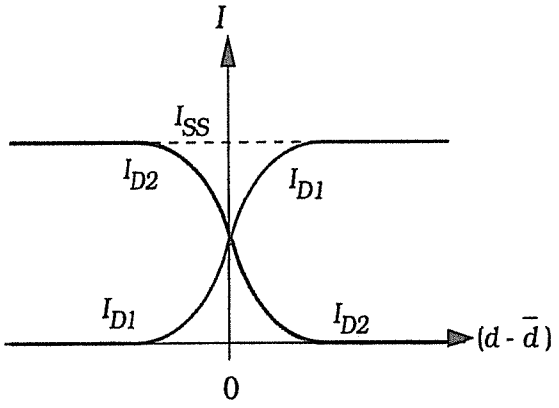


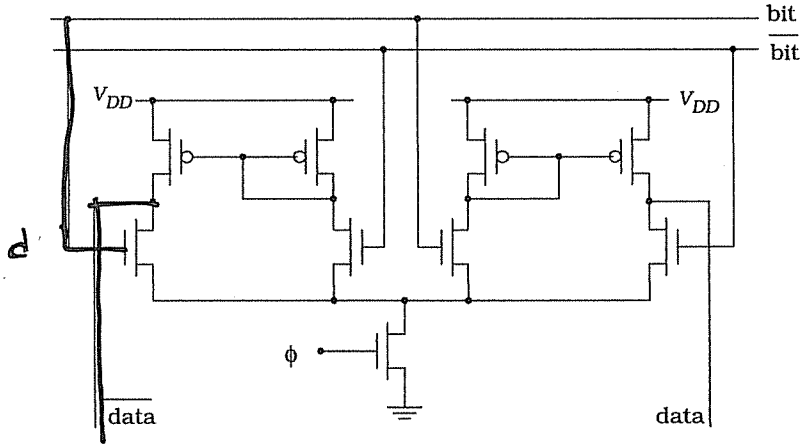
Read: Sense amplifier network











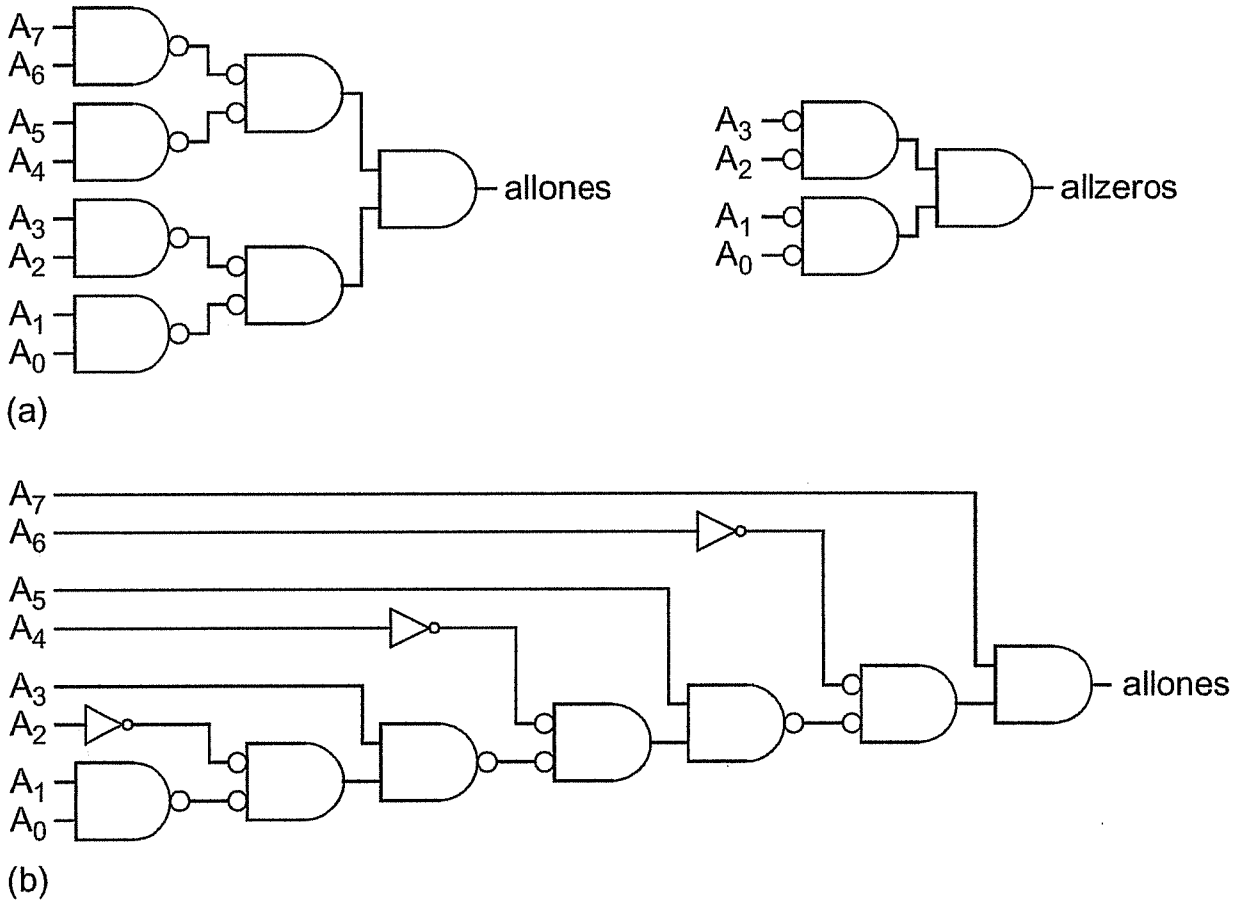
The sense amplifier network increases the sensitivity of the detection circuit. For small SRAM arrays, simply buffering the d_i outputs may also be able to give the desired sensitivity and speed more simply.

Some Remaining Datapath Topics:

One/Zero Detectors

Easily implemented with very simple AND or NOR logic.
Trees of NANDs and INVs are used to minimize logical effort.

Examples:



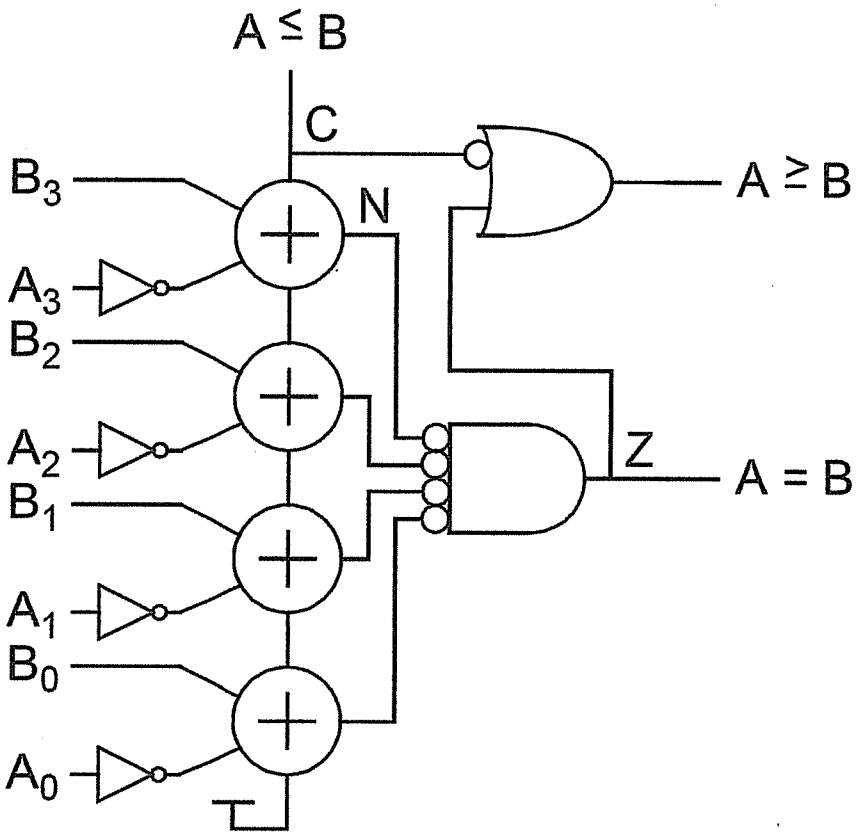
Zero and ones detectors are useful, since many instructions rely on bits in a **Condition Code Register**, which contains useful information about the instructions that were just performed by the processor.

Frequently used bits in a CCR:

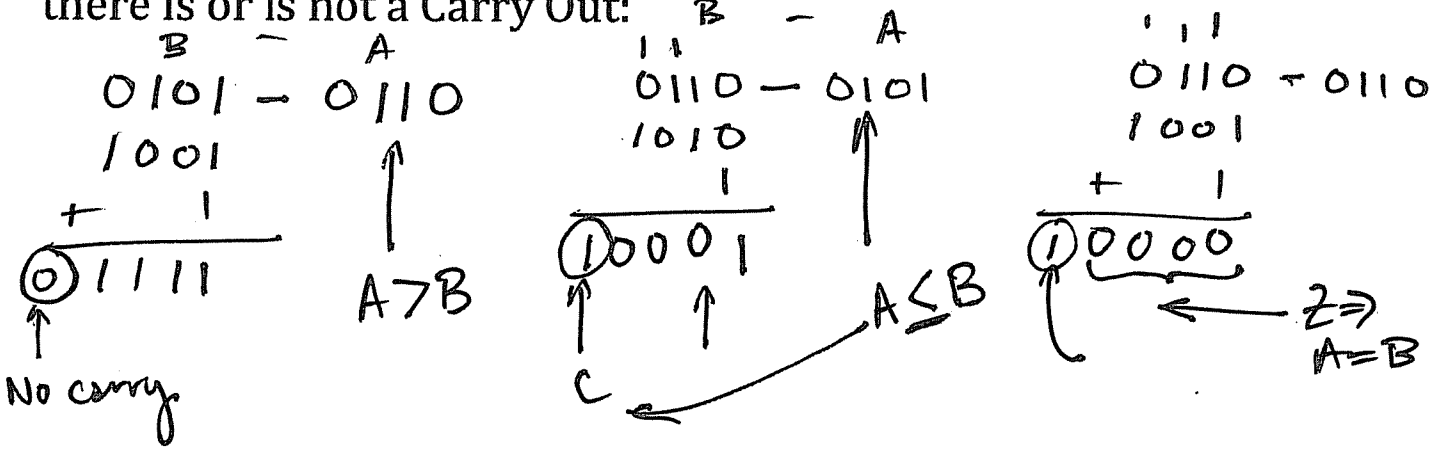
Flag	Name	Description
Z	Zero flag	Indicates that the result of an arithmetic or logical operation (or, sometimes, a load) was zero.
C	Carry flag	Enables numbers larger than a single word to be added/subtracted by carrying a binary digit from a less significant word to the least significant bit of a more significant word as needed. It is also used to extend bit shifts and rotates in a similar manner on many processors (sometimes done via a dedicated **X** flag).
S / N	Sign flag / Negative flag	Indicates that the result of a mathematical operation is negative. In some processors,[1] the N and S flags are distinct with different meanings and usage: One indicates whether the last result was negative whereas the other indicates whether a subtraction or addition has taken place.
V / O / W	Overflow flag	Indicates that the signed result of an operation is too large to fit in the register width using twos complement representation.
P	Parity flag	Indicates whether the number of set bits of the last result is odd or even.

Magnitude and Equality Detection

We have now covered everything we need to find the C and Z the flags in the CCR. Magnitude detection can now be determined simply by adding a Zero detector to the outputs of an Adder:



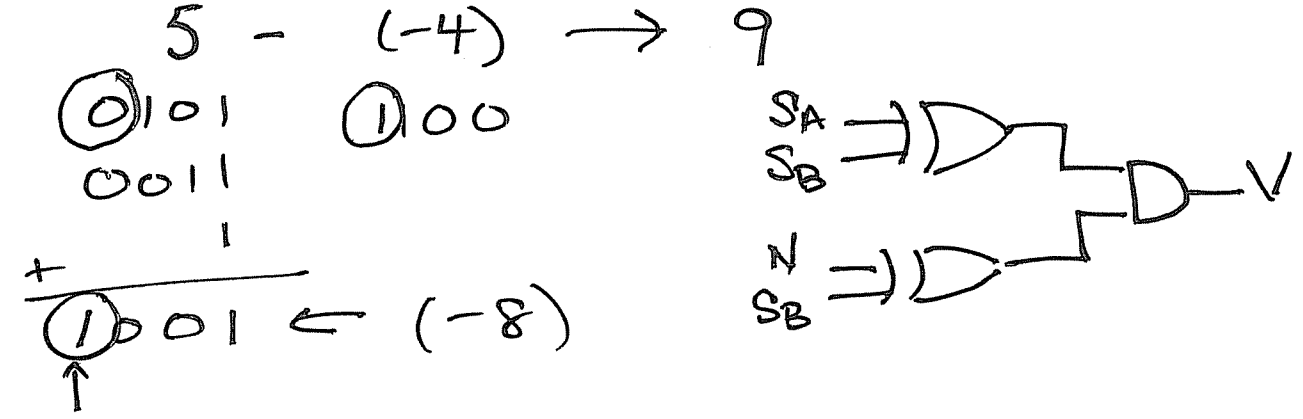
Consider how each of these outputs works. To compare two unsigned numbers, compute $B - A = B + \bar{A} + 1$. Consider if there is or is not a Carry Out:



If there is a carry out (C), then $A \leq B$. The Zero bit (Z) can be used to determine whether it is $A < B$ or $A = B$.

Relation	Unsigned Comparison	Signed Comparison
$A = B$	Z	Z
$A \neq B$	$\overline{Z}$	$\overline{Z}$
$A < B$	$C \cdot \overline{Z}$	$\overline{S} \cdot \overline{Z}$
$A > B$	$\overline{C}$	S
$A \leq B$	C	$\overline{S}$
$A \geq B$	$\overline{C} + Z$	$S + Z$

For signed comparisons, let N be the MSB of the output ($N = 1$ if the output is Negative) The overflow V is true if the inputs had different signs (most significant bits) and the output sign is different from the sign of B.

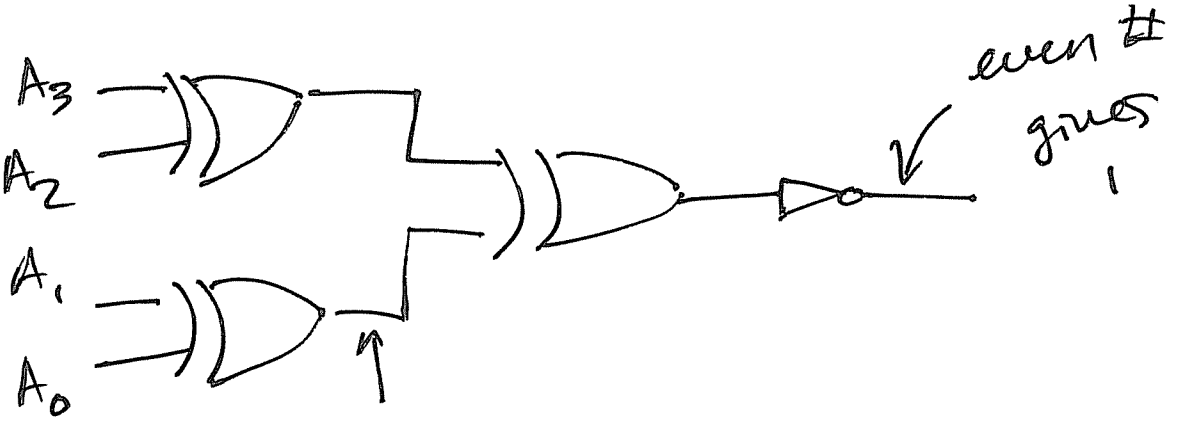


So, overflow flips the sign, which gives the corrected sign of the output as $S = N \oplus V$. If $S = 1$ (negative) then $A > B$.

Parity

Is useful for detecting single bit errors. The Parity Flag $P = 1$ if the number of ones is even in the output. Error-correcting codes are beyond the scope of this course, but consider that if you expect an output with an even number of ones, and instead have one with an odd number of ones that a bit somewhere must have switched. Consider the case where additional bits ("check bits") are added to every word so that only $P = 1$ words are valid. That means that a $P = 0$ word indicates that an error has occurred.

How to determine Parity?



Reading Assignment: 9.2, 9.5, 9.6 for Wed
13.3–13.5 for next Mon.

(Reason for this change in the schedule:
13.4 & 13.5 use pseudo-nMOS circuitry,
which we haven't covered yet)

Check-In: Describe the topology of
a pseudo-nMOS gate.