Storage Elements & Memory

CS 350: Computer Organization & Assembler Language Programming

[4/23 p.4: *D*-latch: Rename *a* to *Q*, move NOT from D to WE; p.8: Swap *WE*/*D* inputs on figure at bottom]

A. Why?

- Storage elements are the basic circuits that store data, which is used in logic circuits that have a state (i.e., use memory).

B. Outcomes

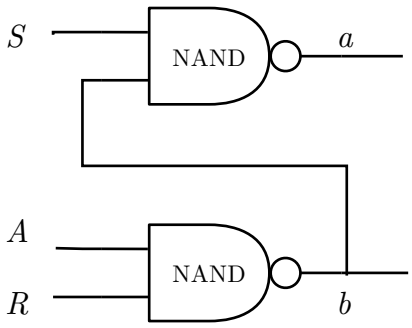
After this lecture, you should

- Know how *R*-*S* latches, *D*-latches, and registers work.
- Know how memory combines address decoders and data storage/update.

C. Storage Element - the R-S Latch

- When input signals change in circuit without loops, the change ripples forward through the circuit to the outputs.
- If a circuit has a loop, then some gate's output is some gate's input and the changes that occur can depend on the past value of the input and output gates.
- In this way, the circuit can have a **state** that depends on past history.
- The *R*-*S* latch is a simple example; it stores one bit of state: It can remember a bit, and you can modify that bit by changing the input.
  - To be said to remember a bit, a circuit needs to have two states (we'll call them 0 and 1), and we have to be able to read the state.
  - To be a storage element, we need a circuit that remembers a bit and also allows us to **set** that bit (to 1 unless otherwise specified) or **reset/unset** that bit (set it to 0).
  - In an *R*-*S* latch, we can reset the bit using input *R* and set the bit using input *S*. There's an output that lets us see the state of the latch.

- The device below isn't quite an R-S latch. It has 3 inputs R, S, and A and 2 outputs, a and b. "The device is in state a" means $a = \overline{b}$.

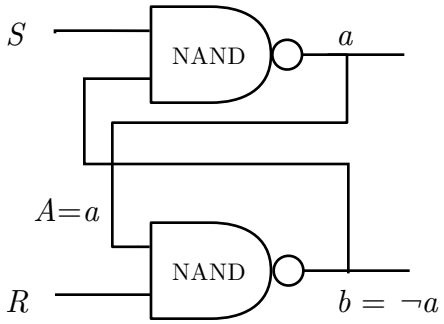


Not Quite an R-S Latch

R S	A	$a = \overline{S} + \overline{b}$	$b = \overline{R} + \overline{A}$
0 0	0	1	1
0 0	1	1	1
0 1	0	0	1
0 1	1	0	1
1 0	0	1	1
1 0	1	1	0
1 1	0	0	1
1 1	1	1	0

- The R-S latch: To get an R-S latch, we have to connect a to A.

- With an R-S latch normally we have $R\,S$ hold (i.e., $R = S = 1$). We can change states by momentarily setting $R\,\overline{S}$ or $\overline{R}\;S$, then we go back to $R\,S$.



An R-S Latch

- If $R\,S$ then $a\,\overline{b}$ and $\overline{a}\,b$ are stable (neither a nor b change values.)

R S	a b	New $a = \overline{S} + \overline{b}$	New $b = \overline{R} + \overline{a}$
1 1	0 1	0	1
1 1	1 0	1	0

- Changing R (*Reset*) to 0 while keeping $S = 1$ changes the state to $\overline{a}\ b$.
  - If $\overline{R}\,S$, then $\overline{a}\ b$ is stable, but $a\,\overline{b} \rightarrow a\,b \rightarrow \overline{a}\,b$. (The state $a\,\overline{b}$ momentarily changes to $a\,b$ and then to $\overline{a}\ b$.)
  - Below, an arrow $\rightarrow x$ means the value is changing from $\overline{x}$ to x.

$R\ S$	$a\ b$	New $a = \overline{S} + \overline{b}$	New $b = \overline{R} + \overline{a}$
0 1	0 1	0	1
0 1	1 0	1	$\rightarrow 1$
0 1	1 1	$\rightarrow 0$	1

- Changing S (Set) to 0 while keeping $R = 1$ changes the state to 1 (i.e., $a\,\overline{b}$).
  - If $R\,\overline{S}$, then $a\,\overline{b}$ is stable, but $\overline{a}\,b \rightarrow a\,b \rightarrow a\,\overline{b}$.

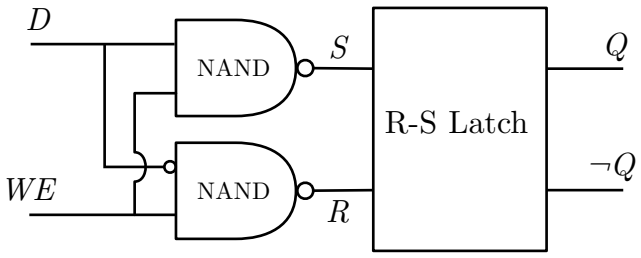
$R\ S$	$a\ b$	New $a = \overline{S} + \overline{b}$	New $b = \overline{R} + \overline{a}$
1 0	0 1	$\rightarrow 1$	1
1 0	1 0	1	0
1 0	1 1	1	$\rightarrow 0$

- We don't set $\overline{R}\ \overline{S}$ because it establishes $a\ b$, which represents neither 0 nor 1.

$R\ S$	$a\ b$	New $a = \overline{S} + \overline{b}$	New $b = \overline{R} + \overline{a}$
0 0	0 1	$\rightarrow 1$	1
0 0	1 0	1	$\rightarrow 1$
0 0	1 1	1	1

D. Gated D-Latch

- Gated D-latches extend R-S latches and makes them easier and safer to use.
- The extension is the pair of $NAND$ gates connected to the R and S inputs of the R-S latch.

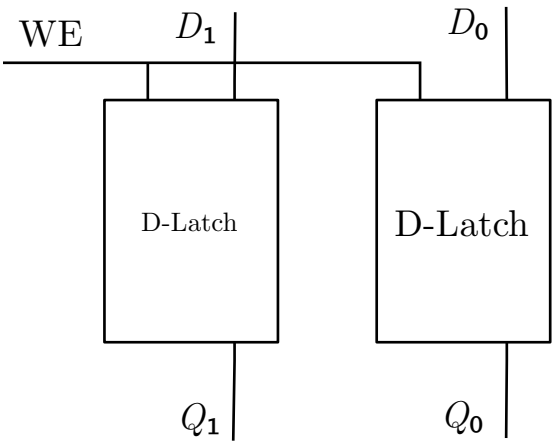


Gated D-Latch

- The $NAND$ gates ensure safety: We can't have R and S both $= 0$.
- The **Input Data** D is copied into the R-S latch exactly when the **Write Enable** (WE) bit is 1.
  - If $WE = 0$ then $R = S = 1$ (D is ignored; the latch state is unchanged)
  - If $WE = 1$ then set Latch $\leftarrow D$:
    - If $D = 1$ then $S = 0$ and $R = 1$, so the latch is set to 1
    - If $D = 0$ then $S = 1$ and $R = 0$, so the latch is set to 0
- [Apr 23: Note that the latch output has been renamed from a to Q.]

E. Registers Hold Bitstrings

- An ***n*-bit wide register** is a circuit that can store an n-bit bitstring.
- We can build an n-bit register by combining n D-latches.
- The register has
  - n input data bits $D[0:n–1]$
  - n output data bits $Q[0:n–1]$
  - One 1-bit **Write Enable** input line (a "**control**" bit).
  - The i'th D-latch has input D_i and output Q_i.
  - All n D-latches share the WE line: If WE $= 0$ then the latches produce their stored values to form Q. If $WE = 1$, then the latches copy the bits of D.
- Below is a diagram for a 2-bit register.



F. Memory Bank

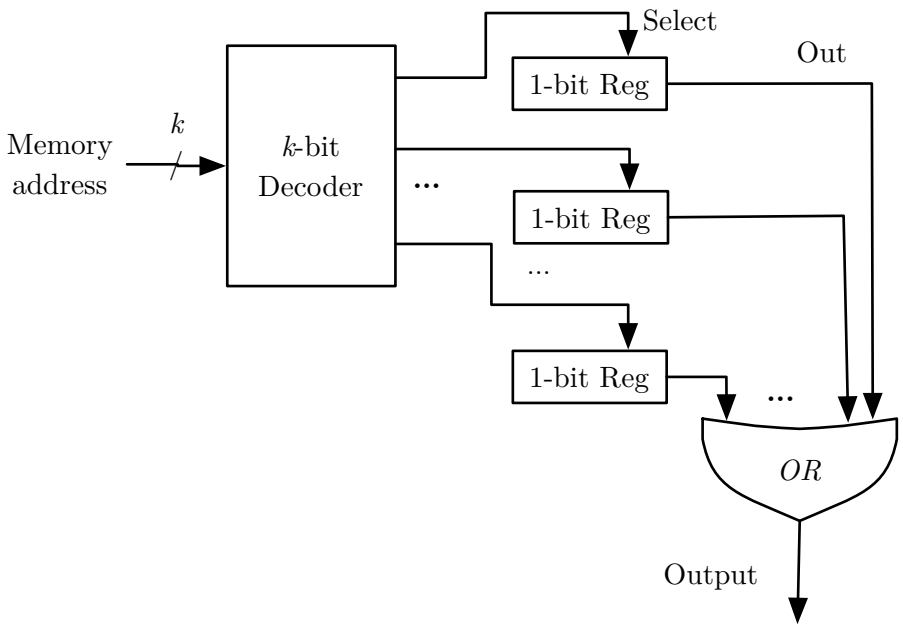
- We can combine multiple registers to produce a memory bank.
- We access individual registers by their name, i.e. **address**, which is typically an unsigned bitstring.
- Some memory address properties:
  - The **address size** or **width** is the number of bits in an address.
  - The **address space** is the set of legal addresses.
    - Unless some addresses aren't used, the address space has size a power of 2.
- Some common sizes of memory registers:
  - A **byte** is an 8-bit string.
  - A **word** is typically the amount of memory needed to store an integer.
    - The **word length** is the width (number of bits) of a word.
    - Nowadays word length is typically 16 or 32 bits.
  - A **double word** is two words long, etc.
- Addressability
  - The **addressability** of a memory bank is the number of bits stored at each address.
  - A memory bank has **byte addressability** if its addressability is 8 bits.

- A memory bank that isn't byte-addressable is typically **word-addressable**: its addressability equals its word length.
- If a memory bank is byte-addressable, the addresses of words are often restricted to be multiples of the addressability.
  - E.g., if a word is 4 bytes wide, then the "**aligned**" word addresses are 0, 4, 8, 12, etc. (Such addresses are **word aligned**.)
- Trying to access an **unaligned** word can cause an error or a slowdown in execution (depends on the hardware design).

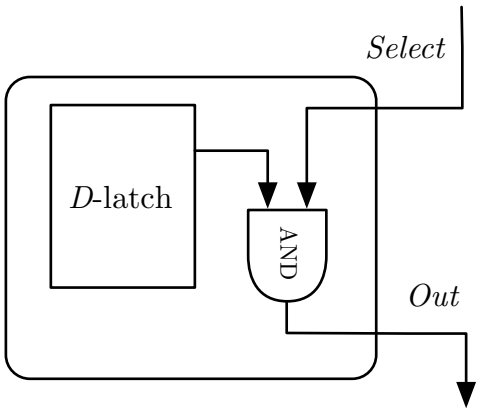
- **Addressability vs Address Size**
  - The address size and the addressability **don't have to be related**.
  - We'll see later that a machine instruction typically is too small to store a whole address, which means we'll have to construct addresses as part of executing instructions.

G. Reading a Memory Bank

- With the goal of understanding how a read/write memory bank works, let's look at a simple case: k-bit addresses where each address holds 1 read-only bit.

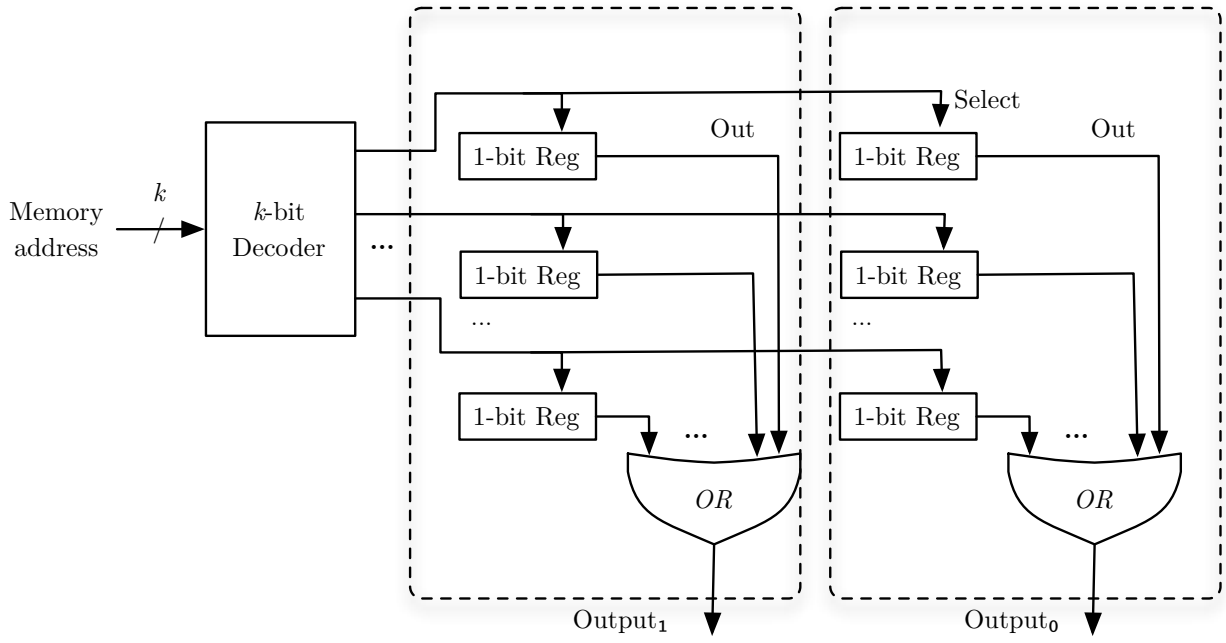


- The k-bit memory address feeds into a decoder with 2^k outputs, exactly one of which will be 1 (the others will be 0). Which output is 1 is specified by the value of the k bit address.
- Each decoder output goes to a 1-bit register. Each register looks at this value as its input "select" line: a 1-bit register is just a D-latch whose output is ANDed with the select line, so if the select line is 1, the register outputs its stored value, otherwise it outputs 0.



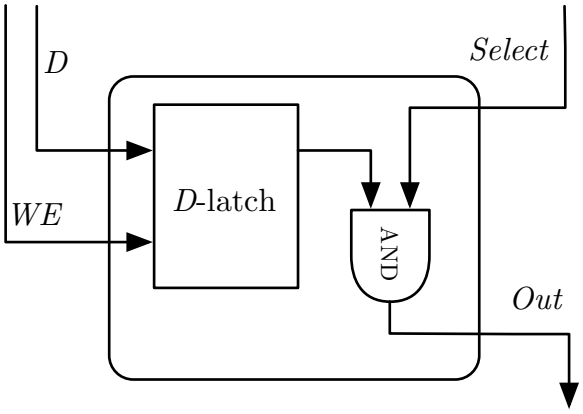
Reading a 1-bit Register

- The outputs of the 2^k registers get ORed together to form the output of our memory bank: The output matches the value of the register selected by the address.
- To store multiple bits of data at each address, we have to repeat the circuitry for each bit of output. (Only one decoder is necessary, however.) The diagram below supports 2 bits of data per address. The dashed rectangles indicate the circuitry being repeated for each "column" of output. Note the registers for each address share the same select line.



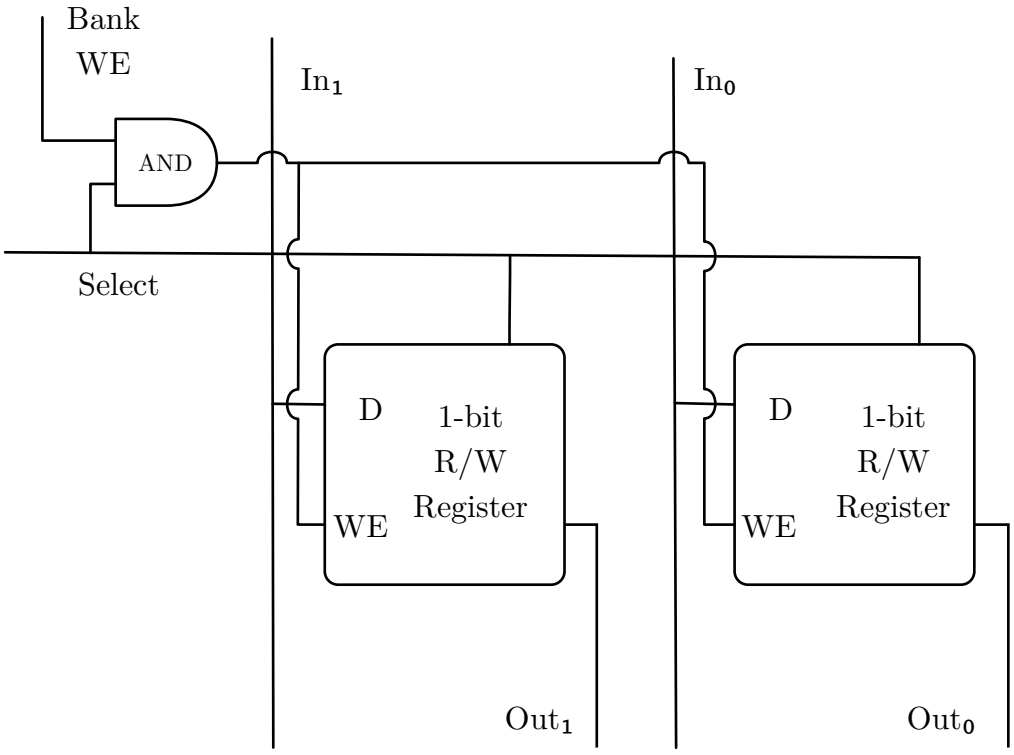
H. Writing To Memory

- With read-write memory, the 1-bit registers gain two inputs: a write enable line and an input data line. The latch state is set to the input if the write enable line is on, otherwise the input is ignored.
- As before, the output of the latch is only sent out if the select line is on, otherwise a 0 is output.



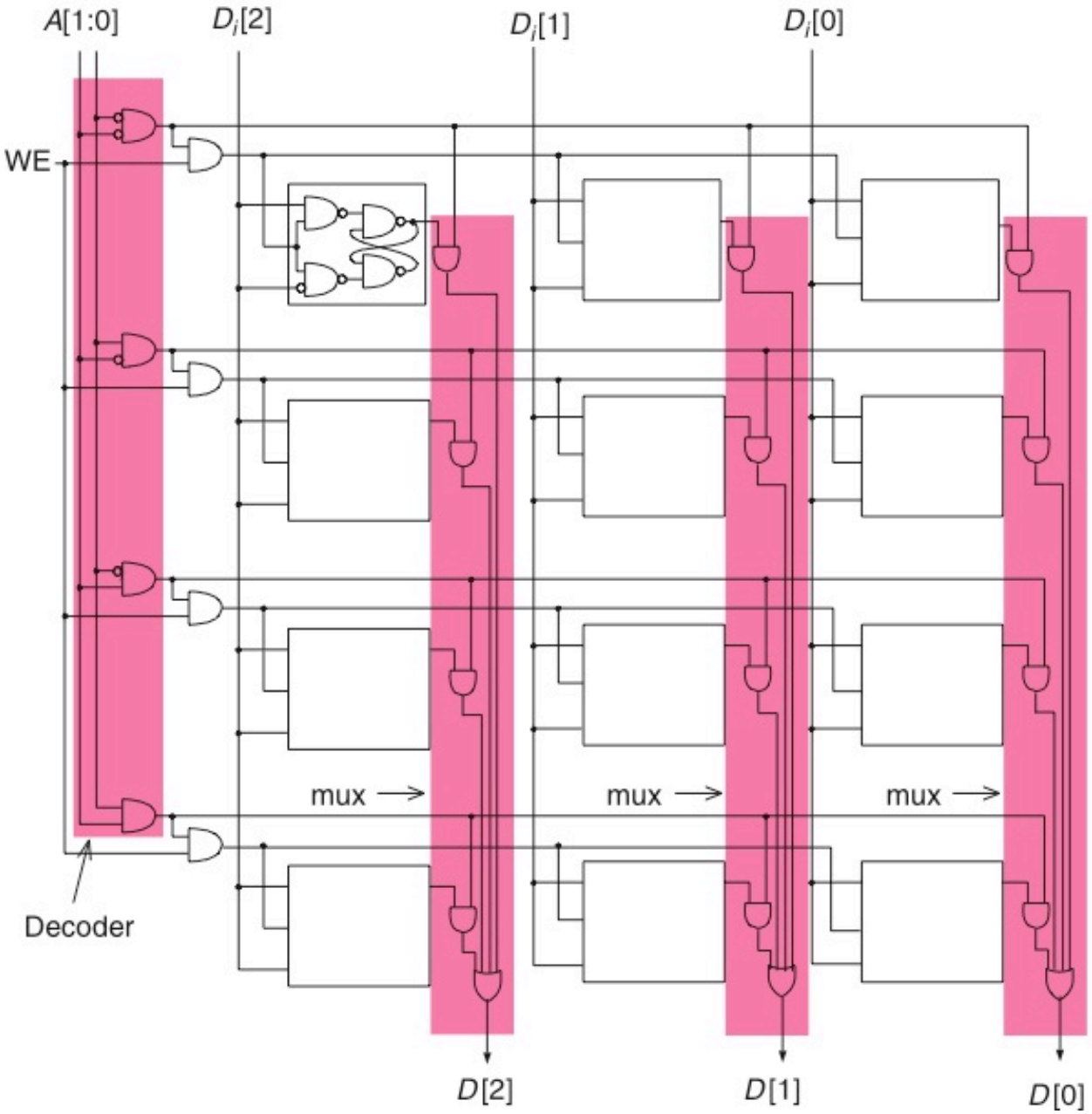
A 1-bit Read/Write Register

- If we concentrate on all the bits stored at one address, we only write data to these bits if this address is the one selected and the memory bank is supposed to write its input data into the bank.



The 2 bits At an Address

- For a fuller example, the textbook's Figure 3.21 shows a memory bank with 2-bit address size and 3-bit addressability.



Storage Elements and Memory

CS 350: Computer Organization & Assembler Language Programming

A. Why?

- Storage elements are the basic circuits that store data, which is used in logic circuits that have a state (i.e., use memory).
- Memory is used to store data for I/O and computations.

B. Outcomes

After this activity, you should be able to:

- Be able to identify and trace the execution of basic storage elements.
- Know how memory combines address decoders, multiplexers, and data storage/update.

C. Questions

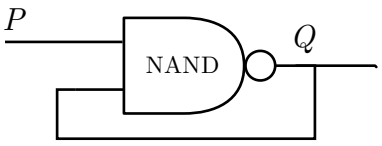
1. The notes discussed 11 R-S Latch configurations. Complete the table below, which shows the five remaining configurations.

R S	a b	New $a = \overline{S} + \overline{b}$	New $b = \overline{R} + \overline{a}$
0 0	0 0		
0 1	0 0		
1 0	0 0		
1 1	0 0		
1 1	1 1		

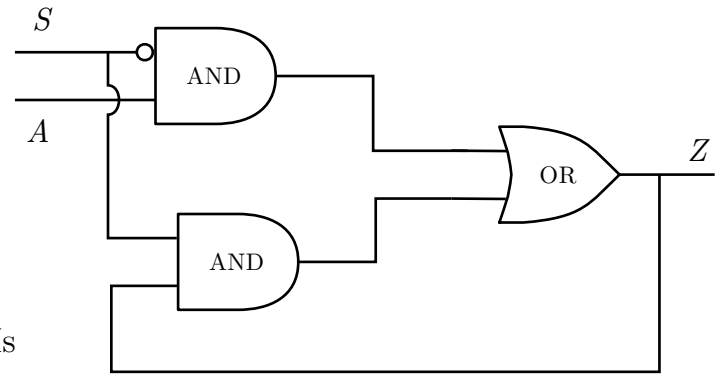
2. Say an R-S latch powers up with $\overline{a}$ $\overline{b}$ and $R = S = 1$. What happens? Is this a problem? If so, suggest a fix.

3. Say an R-S latch powers up with a and b having random values and $R = S = 0$. What happens? Is this a problem? If so, suggest a fix.

4. Does this circuit have a stable (non-oscillating) logical value when $P = 0$? When $P = 1$? Does this circuit remember a bit?



5. (Exercise 3.27) (a) Describe the output of this logic circuit when the select line S is a logical 0. That is, what is the output Z for each value of A? (b) If the select line is switched from a logical 0 to 1, what will the output be? (c) Is this circuit a storage element?



6. Say we have a machine with k-bit addresses and m-byte addressability. (a) What is the address space and how large is it? (b) How many bytes of memory can we have in total? (c) What, if any, is the relationship between k and m?

7. Say a byte-addressable machine has 32-bit memory addresses. How many gigabytes of memory can we access? What if memory addresses are 64-bits? Hint: The standard prefixes are kilo-, mega-, giga-, tera-, peta-, exa-, zetta-, and yotta-.

Solution

1. Here's a full truth table for an R-S latch; the rows we wanted are checked off.

	R S	a b	New $a = \overline{S} + \overline{b}$	New $b = \overline{R} + \overline{a}$
√	0 0	0 0	→ 1	→ 1
	0 0	0 1	→ 1	1
	0 0	1 0	1	→ 1
	0 0	1 1	1	1
√	0 1	0 0	→ 1	→ 1
	0 1	0 1	0	1
	0 1	1 0	1	→ 1
	0 1	1 1	→ 0	1
√	1 0	0 0	→ 1	→ 1
	1 0	0 1	→ 1	1
	1 0	1 0	1	0
	1 0	1 1	1	→ 0
√	1 1	0 0	→ 1	→ 1
	1 1	0 1	0	1
	1 1	1 0	1	0
√	1 1	1 1	→ 0	→ 0

2. If an R-S latch powers up with $R\,S$ then $a\,b$ is stable, which is a problem, since it's not a configuration we're using to denote true or false. We can fix the problem by using the R or S switch ($\overline{R}\ S$ takes $a\,b \rightarrow a\,\overline{b}$, and $R\,\overline{S}$ takes $a\,b \rightarrow a\,\overline{b}$.)

3. If an R-S latch powers up with a and b having random values and $\overline{R}\ \overline{S}$, then we go to $a\,b$ as in the previous question. Again, it's a problem and again it

can be fixed using the technique in the previous problem (use $\overline{R}\ S$ or $R\ \overline{S}$ and then $R\ S$).

4. When $P = 0$, the NAND gate produces 1 regardless of its other input, so Q becomes 1 and stays there. When $P = 1$, the NAND gate produces the NOT of its other input, so Q's logical value flips back and forth and is not stable. This circuit doesn't remember a bit; it's more like a combinatorial function that breaks on certain inputs.

5. The new $Z = A\ \overline{S}\ +\ S\ Z$, so if $S = 0$ then the new $Z = A$. If $S = 1$ then new $Z = Z$ (whether $Z = 0$ or 1). This circuit is a storage element, with Z as its state: If $S = 1$ then Z remains unchanged. To change Z, set A to the value of Z you want, then set $S = 0$ to change Z, and then set $S = 1$ to maintain Z.

6. The address space is the set of legal addresses. Ours has size 2^k. The total amount of memory is $m\ 2^k$ bytes. In theory, there's no relationship between k and m. A typical computer, however, can fit an address within a word, so in practice, if we have word addressability (i.e., if $m > 1$), then we probably have $k \leq m$.

7. A gigabyte (GB) = 2^{30} bytes, so with 32-bit addresses, we can access 2^{32} bytes of memory = 4 gigabytes (4 GB). With 64-bit addresses we can address $2^4 * 2^{60}$ bytes = 16 exabytes (16 EB).

