Other Memories

• We showed how to build arbitrarily-large static memories from single-bit RAM cells.
• Today we’ll look at some other kinds of memories.
  - Dynamic RAM is used for the bulk of computer memory.
  - Read-only memories and PLAs are two “programmable logic devices,” which can be considered as special types of memories.
Dynamic memory

- **Dynamic memory** is built with capacitors.
  - A stored charge on the capacitor represents a logical 1.
  - No charge represents a logic 0.
- However, capacitors lose their charge after a few milliseconds. The memory requires constant refreshing to recharge the capacitors. (That’s what’s “dynamic” about it.)
- Dynamic RAMs tend to be physically smaller than static RAMs.
  - A single bit of data can be stored with just one capacitor and one transistor, while static RAM cells typically require 4-6 transistors.
  - This means dynamic RAM is cheaper.
Types of RAMs

• **SDRAM:** *Synchronous DRAM* is one of the most common types of PC memory. Memory chips are organized into “modules” that are connected to the CPU via a 64-bit (8-byte) bus.

• **DDR-RAM:** *Double Data Rate* is very similar to regular SDRAM, except data can be transferred on both the positive and negative clock edges. DDR-RAM has lower power consumption, using 2.5V instead of 3.3V like SDRAM. This makes it good for notebooks and other mobile devices.

• **RDRAM:** Another type of memory called RDRAM is used in the Playstation 2 as well as some Pentium 4 computers. Data can be transferred on both the positive and negative clock edges.

• **XDR-DRAM:** *EXtreme Data Rate Dynamic RAM* is designed to be effective in small, high-bandwidth consumer systems, high-performance memory applications, and high-end GPUs. It eliminates the unusually high latency problems that plagued early forms of RDRAM.
A read-only memory, or ROM, is a special kind of memory whose contents cannot be easily modified.

- The WR and DATA inputs that we saw in RAMs are not needed.
- Data is stored onto a ROM chip using special hardware tools.

ROMs are useful for holding data that never changes.

- Arithmetic circuits might use tables to speed up computations of logarithms or divisions.
- Many computers use a ROM to store important programs that should not be modified, such as the system BIOS.
- Game machines, cell phones, vending machines and other electronic devices may also contain non-modifiable programs.
Memories and functions

• ROMs are actually combinational devices, not sequential ones!
  - You can’t store arbitrary data into a ROM, so the same address will always contain the same data.
  - You can think of a ROM as a combinational circuit that takes an address as input, and produces some data as the output.

• A ROM table is basically just a truth table.
  - The table shows what data is stored at each ROM address.
  - You can generate that data combinationally, using the address as the input.

<table>
<thead>
<tr>
<th>Address $A_2A_1A_0$</th>
<th>Data $V_2V_1V_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>010</td>
<td>110</td>
</tr>
<tr>
<td>011</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>101</td>
<td>000</td>
</tr>
<tr>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>111</td>
<td>011</td>
</tr>
</tbody>
</table>
Decoders

- We can already convert truth tables to circuits easily, with decoders.

- For example, you can think of this old circuit as a memory that “stores” the sum and carry outputs from the truth table on the right.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>C</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
ROM setup

• ROMs are based on this decoder implementation of functions.
  - A blank ROM just provides a decoder and several OR gates.
  - The connections between the decoder and the OR gates are “programmable,” so different functions can be implemented.
• To program a ROM, you just make the desired connections between the decoder outputs and the OR gate inputs.
Here are three functions, $V_2, V_1, V_0$, implemented with an $8 \times 3$ ROM. Blue crosses (X) indicate connections between decoder outputs and OR gates. Otherwise there is no connection.

$$V_2 = \Sigma m(1,2,3,4) \quad V_1 = \Sigma m(2,6,7) \quad V_0 = \Sigma m(4,6,7)$$
The same example again

- Here is an alternative presentation of the same 8 x 3 ROM, using “abbreviated” OR gates to make the diagram neater.

\[ V_2 = \Sigma m(1,2,3,4) \]
\[ V_1 = \Sigma m(2,6,7) \]
\[ V_0 = \Sigma m(4,6,7) \]
Why is this a “memory”?

- This combinational circuit can be considered a read-only memory.
  - It stores eight words of data, each consisting of three bits.
  - The decoder inputs form an address, which refers to one of the eight available words.
  - So every input combination corresponds to an address, which is “read” to produce a 3-bit data output.

<table>
<thead>
<tr>
<th>Address $A_2A_1A_0$</th>
<th>Data $V_2V_1V_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>010</td>
<td>110</td>
</tr>
<tr>
<td>011</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>101</td>
<td>000</td>
</tr>
<tr>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>111</td>
<td>011</td>
</tr>
</tbody>
</table>
ROMs vs. RAMs

• There are some important differences between ROM and RAM.
  - ROMs are “non-volatile”—data is preserved even without power. On the other hand, RAM contents disappear once power is lost.
  - ROMs require special (and slower) techniques for writing, so they’re considered to be “read-only” devices.

• Some newer types of ROMs do allow for easier writing, although the speeds still don’t compare with regular RAMs.
  - Many devices allow you to upgrade programs stored in “flash ROM.”
Programmable logic arrays

• A ROM is potentially inefficient because it uses a decoder, which generates all possible minterms. No circuit minimization is done.

• Using a ROM to implement an n-input function requires:
  - An n-to-$2^n$ decoder, with n inverters and $2^n$ n-input AND gates.
  - An OR gate with up to $2^n$ inputs.
  - The number of gates roughly doubles for each additional ROM input.

• A programmable logic array, or PLA, makes the decoder part of the ROM “programmable” too. Instead of generating all minterms, you can choose which products (not necessarily minterms) to generate.
A blank 3 x 4 x 3 PLA

- This is a 3 x 4 x 3 PLA (3 inputs, up to 4 product terms, and 3 outputs), ready to be programmed.
- The left part of the diagram replaces the decoder used in a ROM.
- Connections can be made in the “AND array” to produce four arbitrary products, instead of 8 minterms as with a ROM.
- Those products can then be summed together in the “OR array.”
Regular K-map minimization

- The normal K-map approach is to minimize the number of product terms for each *individual* function.
- For our three functions, this would result in a total of *six* different product terms.

\[
V_2 = \Sigma m(1,2,3,4) \\
V_1 = \Sigma m(2,6,7) \\
V_0 = \Sigma m(4,6,7)
\]
PLA minimization

- For a PLA, we should minimize the number of product terms for *all functions* together.
- We could express $V_2$, $V_1$ and $V_0$ with just *four* total products:

$$V_2 = xy'z' + x'z + x'yz'$$

$$V_1 = x'yz' + xy$$

$$V_0 = xy'z' + xy$$

$$V_2 = \Sigma m(1,2,3,4)$$

$$V_1 = \Sigma m(2,6,7)$$

$$V_0 = \Sigma m(4,6,7)$$
So we can implement these three functions using a $3 \times 4 \times 3$ PLA:

$$V_2 = \Sigma m(1,2,3,4) = xy'z' + x'z + x'yz'$$

$$V_1 = \Sigma m(2,6,7) = x'yz' + xy$$

$$V_0 = \Sigma m(4,6,7) = xy'z' + xy$$
PLA evaluation

• A k x m x n PLA can implement up to n functions of k inputs, each of which must be expressible with no more than m product terms.
• Unlike ROMs, PLAs allow you to choose which products are generated.
  – This can significantly reduce the fan-in (number of inputs) of gates, as well as the total number of gates.
  – However, a PLA is less general than a ROM. Not all functions may be expressible with the limited number of AND gates in a given PLA.
• In terms of memory, a k x m x n PLA has k address lines, and each of the $2^k$ addresses references an n-bit data value.
• But again, not all possible data values can be stored.
Functions and memories

- ROMs and PLAs give us two more ways to implement functions.
- One difference between expressions/circuits and truth tables:
  - A circuit implies that some calculation has to be done on the inputs in order to arrive at the output. If the same inputs are given again, we have to repeat that calculation.
  - A truth table lists all possible combinations of inputs and their corresponding outputs. Instead of doing a potentially lengthy calculation, we can just “look up” the result of a function.
- The idea behind using a ROM or PLA to implement a function is to “store” the function’s truth table, so we don’t have to do any (well, very little) computation.
- This is like “memoization” or “caching” techniques in programming.
Summary

• There are two main kinds of random access memory.
  - Static RAM costs more, but the memory is faster. Static RAM is often used to implement cache memories.
  - Dynamic RAM costs less and requires less physical space, making it ideal for larger-capacity memories. However, access times are also slower.

• ROMs and PLAs are programmable devices that can implement arbitrary functions, which is equivalent to acting as a read-only memory.
  - ROMs are simpler to program, but contain more gates.
  - PLAs use less hardware, but it requires some effort to minimize a set of functions. Also, the number of AND gates available can limit the number of expressible functions.