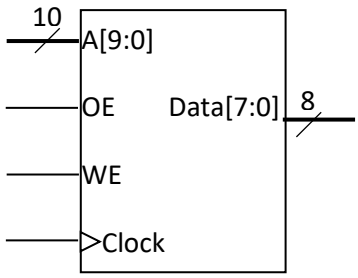
Memories and Datapath

I. Memories (5.6)

A memory is a device that acts like a big array. You supply an index (called an address) and it supplies the data at that location. You can also write to a given address.



On the left is a 1024x8 memory. That is, it has 1024 entries and each entry is 8 bits wide. If Write Enable (WE) is a 1, that means we want to write the value on the data lines into the memory location specified by the address (A) lines. If WE is a 0 and Output Enable (OE) is a 1 that means we want the data at location A to be put on the Data lines. If OE is a zero, Data is set to high-Z.

Notice that data is sometimes an input and sometimes an output!

Questions:

1. What values should we apply if we want to write 0x12 to memory location 0x044?

   A[9:0]= _____ OE=_____ WE=_____ Data[7:0]=_____

2. What values should we apply if we want to read memory location 0x44?

   A[9:0]= _____ OE=_____ WE=_____ Data[7:0]=_____

Types of memories

There are a number of different types of memories each with a different set of properties.

- ROM: _____
- RAM: _____
- Volatile vs. Non-volatile: _____
- Static vs. Dynamic: _____

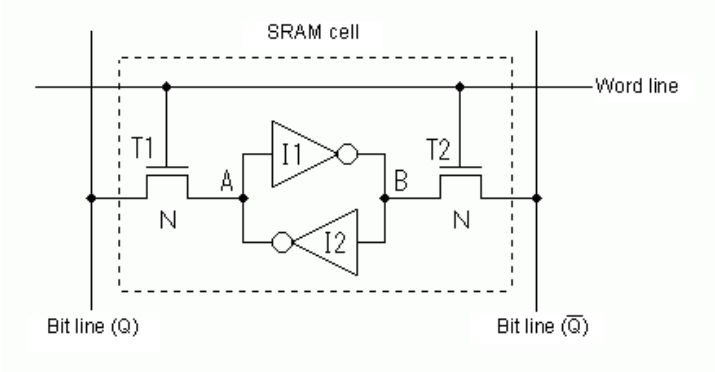
And we commonly find the following common memory types:

- DRAM: _____
- SRAM: _____
- Flash*:_____

  *Note that flash is a particular technology. There are other things (e.g. Magnetoresistive RAM (MRAM)) that have similar properties.

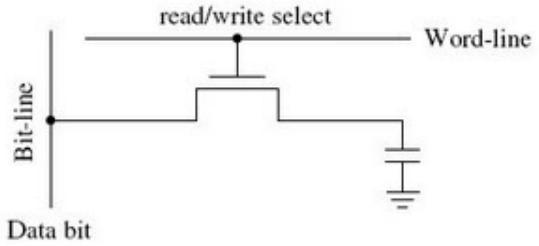
Inside the memories—the little picture

Memories are generally large arrays with small 1-bit cells in them. Let's look at the cell in an SRAM[1] and DRAM[2].



On the left is a single SRAM cell. The inverter pair should be familiar to you—it's the same bi-stable device we started with as a starting point for a latch. The other devices are transistors. We'll cover them in a few weeks, but in this context, you can basically think of them as being a door. If the word line is high, A connects to Q and B connects to Qbar. Otherwise they don't. To read from the device, we simply set the word line high and read the bit line. To write, we "strongly" drive the bit lines and set the word line to be one. That will override the inverter pair's relatively weak signal.

DRAM works in a very similar way. The main difference is that a capacitor is used to store the value. That capacitor will only hold the value for a short time (on the order of 1 to 10 ms!). (A capacitor is a device that holds a certain voltage level for a while). It also suffers from a feature called a "destructive read"—when you read the data, the data is lost.



To deal with the issues associated with the value going away on a DRAM we have to "refresh" the capacitor on a regular basis. That means we may need to read and re-write the data hundreds of times a second…

SRAM vs. DRAM

Area/Density:

- You can generally put about 5-20 DRAM cells in place of one SRAM cell. This translates into DRAM being a lot cheaper per bit.

Cost:

- Cost tends to be fairly proportional to area, so DRAM is significantly cheaper per bit.

Speed:

- SRAM tends to be a lot faster (say on the order of 1-3ns compared to 10-60ns for DRAM).

Power:

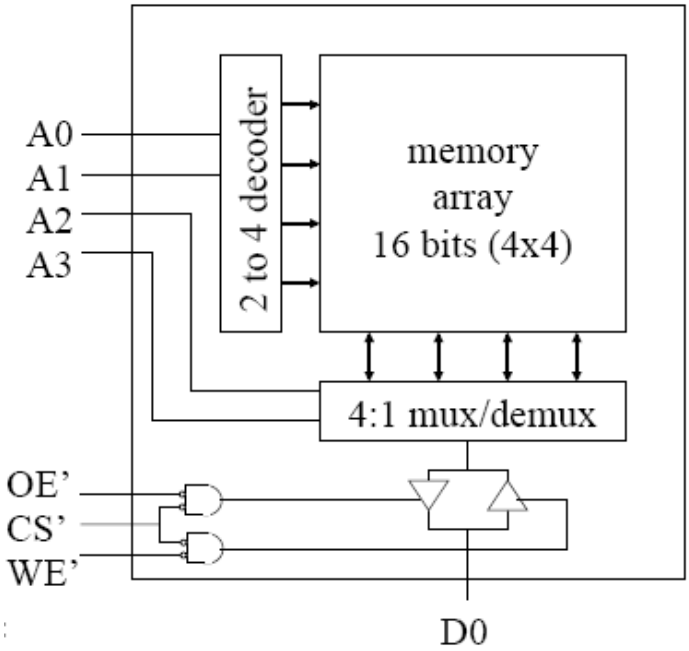
- SRAM tends to eat a lot more power (4GB of DRAM is around 10 Watts in one case, 4MB of SRAM is about 0.5 Watts in another, so a factor of 50/bit in this case. That's not quite a fair comparison; a factor of 10 might be closer…)

We tend to use DRAM when we need lots of cheap memory. We use SRAM when we need to go fast. A microprocessor uses DRAM for the main memory and SRAM for the cache (small fast memory that keeps most likely to be needed data).

[1] Figure taken from *necel.com.*

[2] Figure taken from 1.bp.blogspot.com

Inside the memories—the big picture



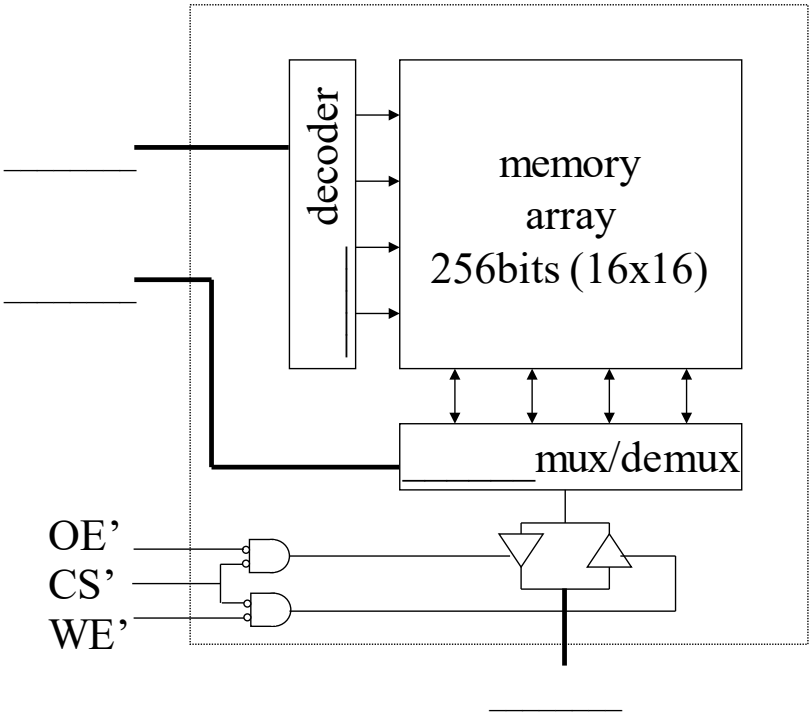
The figure on the left shows how to use a 4x4 memory block to create a 16x1 memory (16 addresses, each one bit). In this diagram we have 4 address lines and one data line. There is also an OE' (meaning active low) and a WE' (also active low).

Questions:

1. What is the decoder doing?
2. What is the MUX doing?
3. What is CS' doing?
4. Memory blocks tend to be square (4x4 in this case). Why do you think that is?

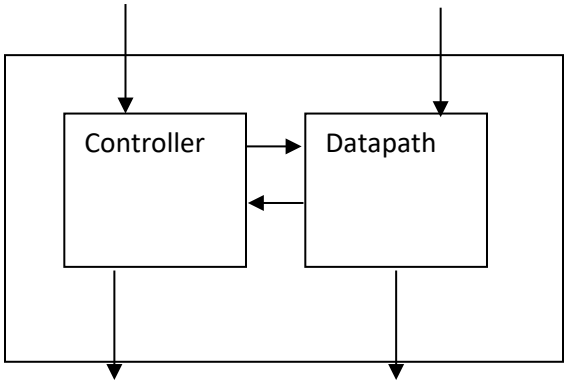
5. Given a 16x16 memory array in which we want to have 4-bit words, how many different words would there be?

6. Fill in the five blanks found in the figure to the right if we want to have 8-bit words.



II. Register-Transfer Level Design (RTL, Chapter 5)

For the most part, we've been tackling fairly simple problems. And for large ones, using a state machine seems like it could get very complex quite quickly. So how *do* we manage complex designs? The answer is a liberal sprinkling of MSI parts and a state machine to control the whole thing.



Consider the figure to the left. The Controller is a state machine. The Datapath is a collection of devices (generally MSI devices like adders, registers, etc.) with control points that lets you choose what to do. You could think of your lab 5 as a datapath where the operation (add, subtract, absolute value) might be controlled by a state machine.

Our text argues that the best way to learn this is to consider a problem you want to solve, then figure out a "high-level state machine". Then design the datapath and controller. See the table below.

	Step	Description
Step 1	*Capture a high-level state machine*	Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more than just Boolean operations on bit inputs and outputs.
Step 2	*Create a datapath*	Create a datapath to carry out the data operations of the high-level state machine.
Step 3	*Connect the datapath to a controller*	Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.
Step 4	*Derive the controller's FSM*	Convert the high-level state machine to a finite-state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath.

"Simple" example—Fire a laser.

Problem specification:

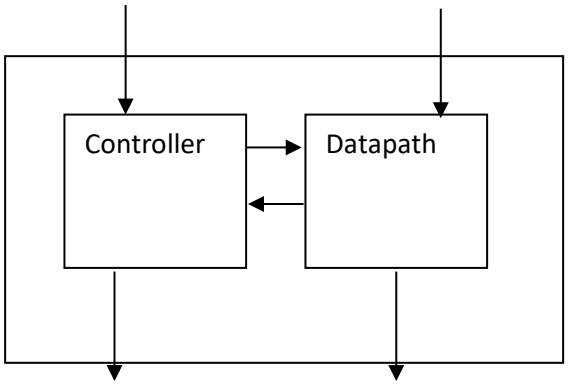
- We want to fire a laser for "x" milliseconds.

Inputs:

- "Fire" button (active high)
- 8-bit binary number ("x" in milliseconds)
- 1 millisecond clock.

Outputs

- Laser control (active high)



	Step	Description
Step 1	*Capture a high-level state machine*	Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more than just Boolean operations on bit inputs and outputs.
Step 2	*Create a datapath*	Create a datapath to carry out the data operations of the high-level state machine.
Step 3	*Connect the datapath to a controller*	Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.
Step 4	*Derive the controller's FSM*	Convert the high-level state machine to a finite-state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath.

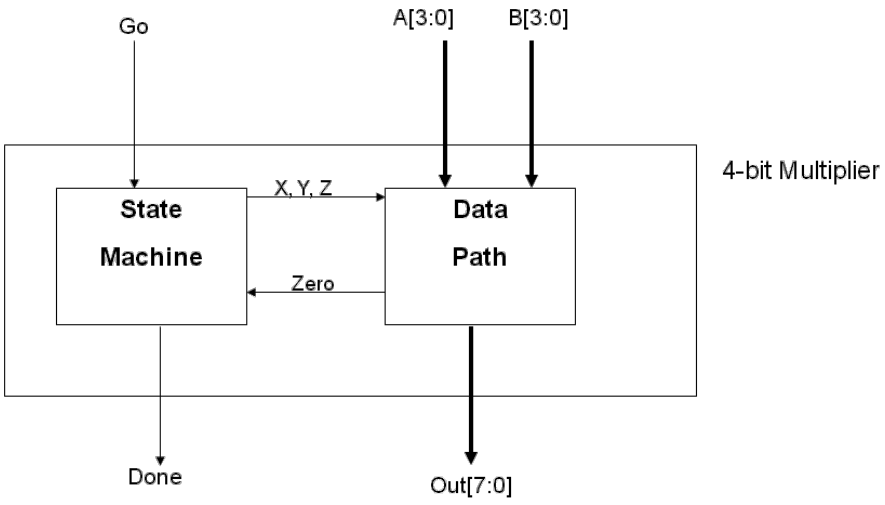
More complex example: multiplier

While that table is the way you want to design these, it's pretty hard to *start* there with a more complex problem. So, we're going to design the controller for a multiplier given the datapath. First, we'll need to review just what it means to multiply numbers (and binary numbers at that).

```
  1001
* 1011
======
```

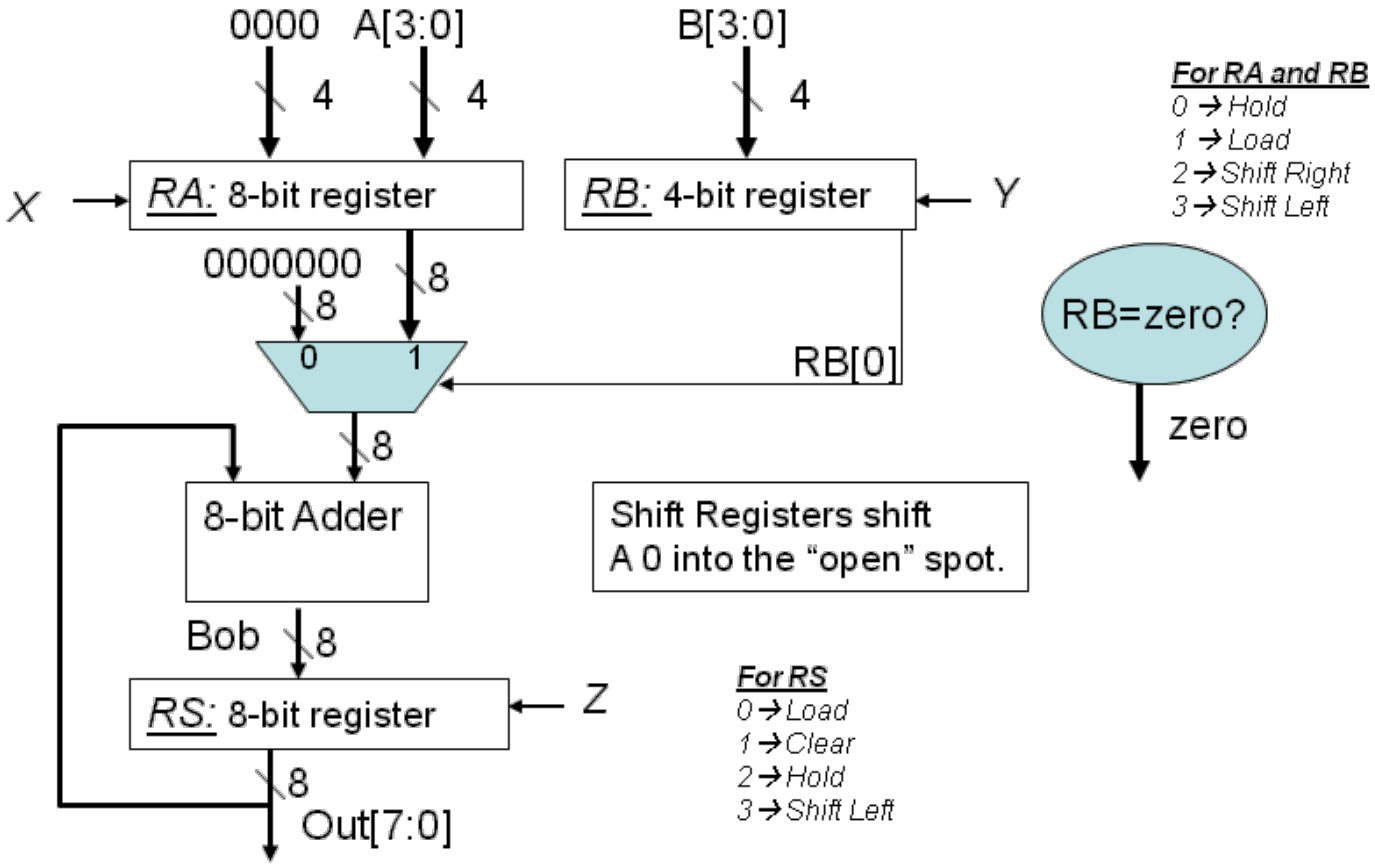
First, note we could build a *combinational* device which does the multiplication. We could brute force it and just build the truth table (for a 4-bit device we'd get _____ rows and _____ output columns). But that wouldn't scale very well. We could certainly come up with something trickier (like we did for N-bit adders). But it's not obvious what that would be. So let's build a sequential device (that takes multiple clock ticks to solve the problem).

Say our multiplier is going to take some time (clock ticks) to solve the problem. In that case, we're going to want a signal to tell it to start and one to tell us when it's done. In addition, we need the two inputs (each N bits say) and one output _____ bits wide.

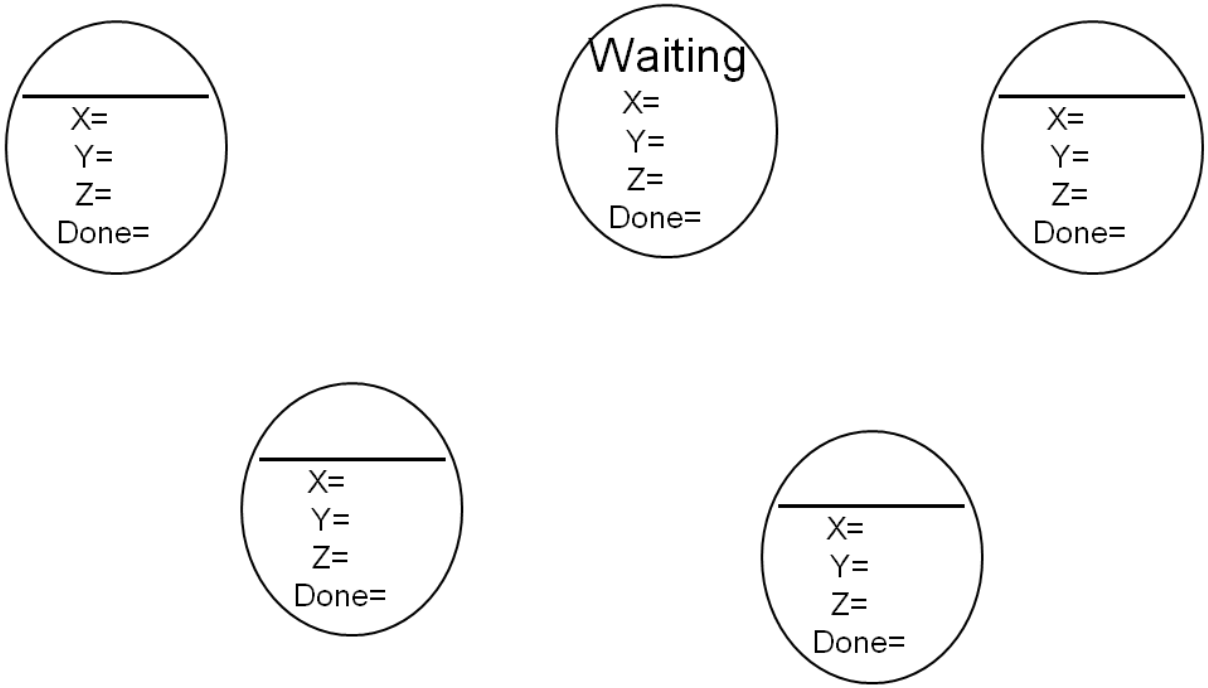


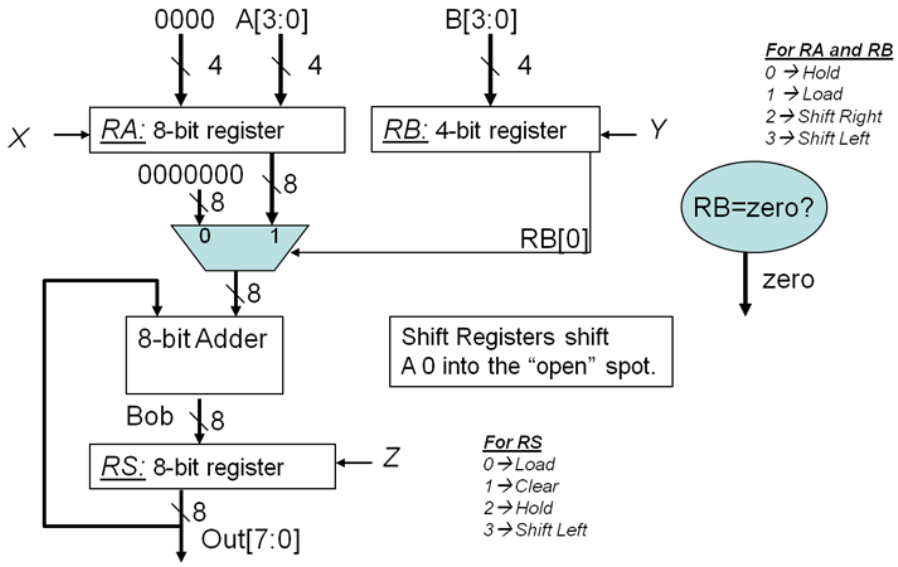
Let's say we use the following datapath:

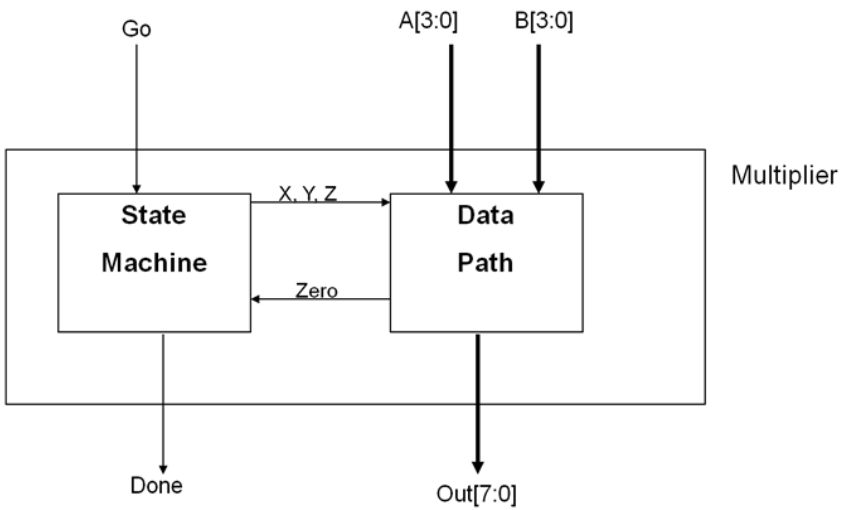
Now we need a state machine to control this…

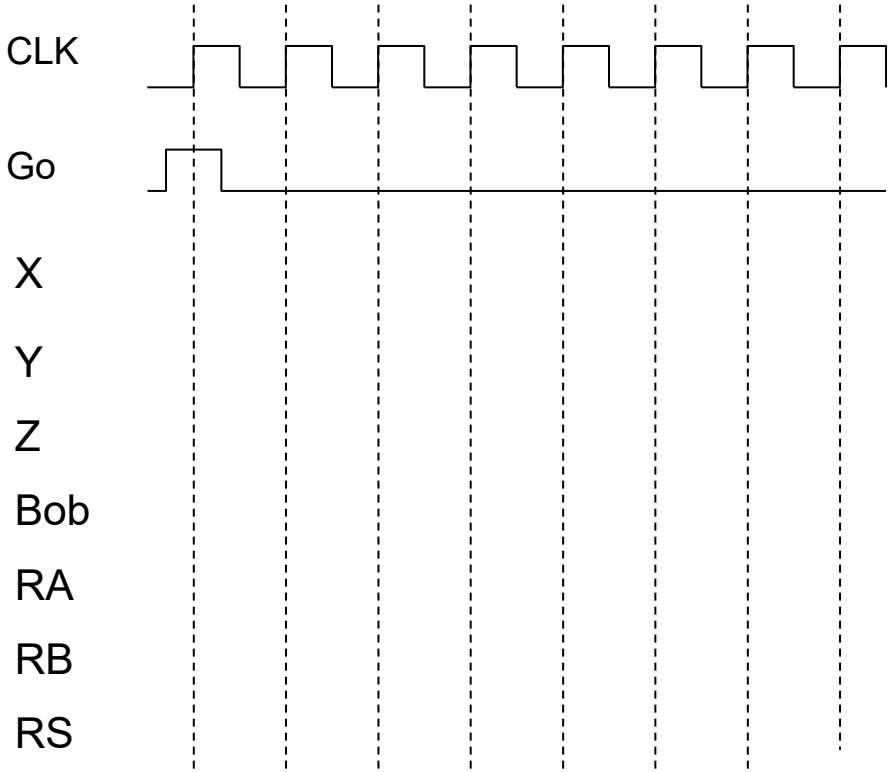

0000
A[3:0]
B[3:0]
4
4
4
For RA and RB
0 → Hold
1 → Load
2 → Shift Right
3 → Shift Left
X
RA: 8-bit register
RB: 4-bit register
Y
0000000
8
8
8
0
1
RB[0]
RB=zero?
zero
8
8-bit Adder
Shift Registers shift
A 0 into the "open" spot.
Bob
8
RS: 8-bit register
Z
For RS
0 → Load
1 → Clear
2 → Hold
3 → Shift Left
8
Out[7:0]

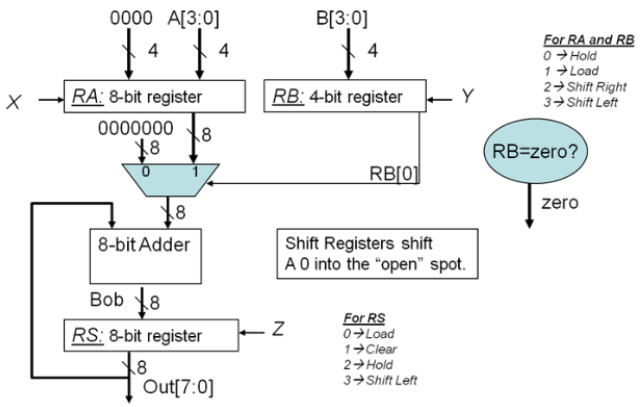
G is Go
Z is Zero

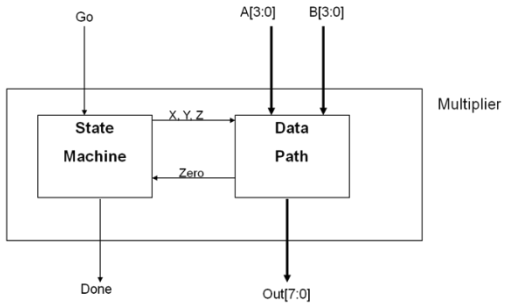


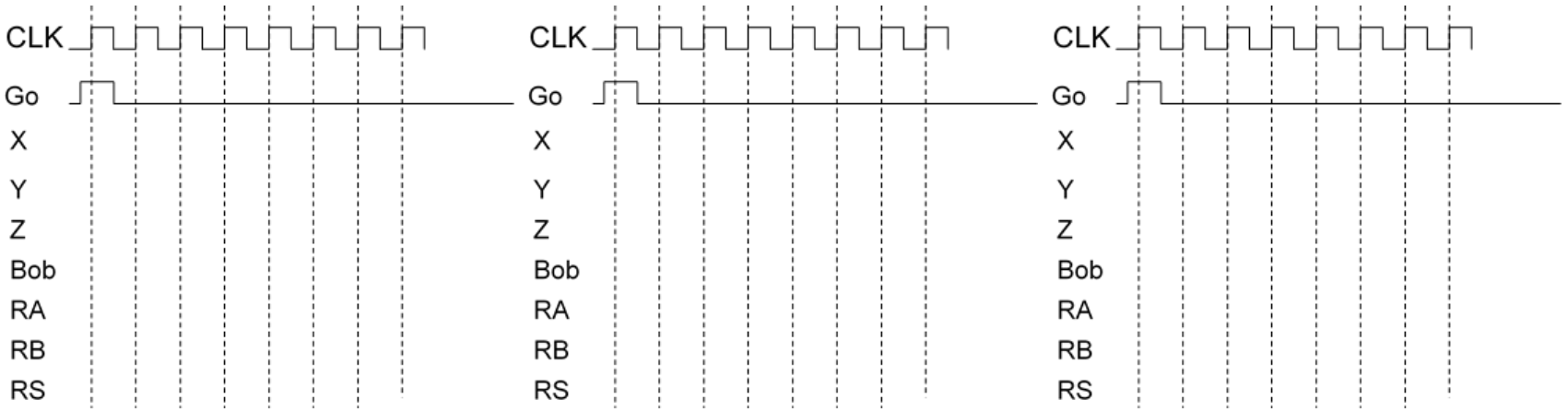




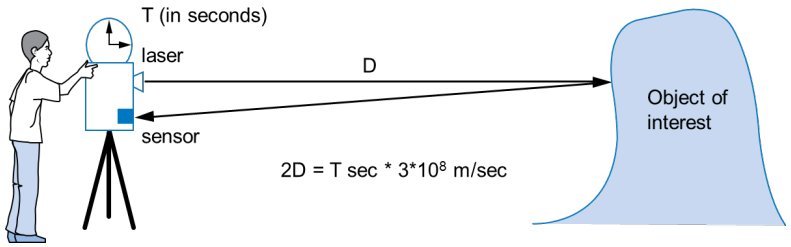

CLK
Go
X
Y
Z
Bob
RA
RB
RS

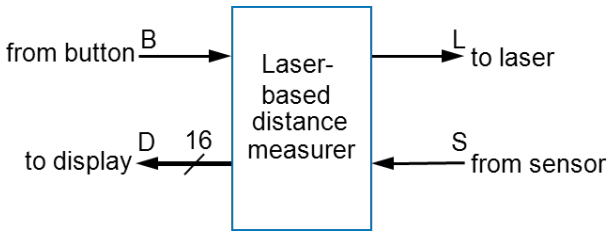

0000 A[3:0]
B[3:0]
4
4
4
X
RA: 8-bit register
RB: 4-bit register
Y
0000000
8
8
0
1
RB[0]
8
8-bit Adder
Shift Registers shift
A 0 into the "open" spot.
Bob
8
RS: 8-bit register
Z
8
Out[7:0]
For RA and RB
0 → Hold
1 → Load
2 → Shift Right
3 → Shift Left
RB=zero?
zero
For RS
0 → Load
1 → Clear
2 → Hold
3 → Shift Left


Go
A[3:0]
B[3:0]
State
Machine
X, Y, Z
Zero
Data
Path
Multiplier
Done
Out[7:0]


CLK
Go
X
Y
Z
Bob
RA
RB
RS
CLK
Go
X
Y
Z
Bob
RA
RB
RS
CLK
Go
X
Y
Z
Bob
RA
RB
RS

Another example: laser distance sensor.



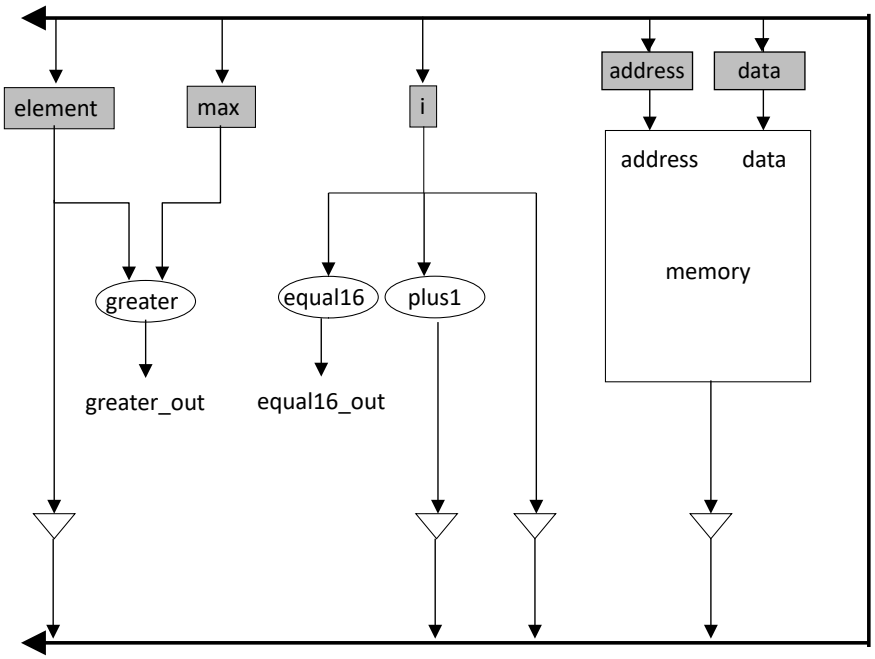


- Laser light travels at speed of light, $3*10^8$ m/sec
- Distance is thus D = T sec * $3*10^8$ m/sec / 2
- Inputs/outputs
  - ***B***: bit input, from button to begin measurement
  - ***L***: bit output, activates laser
  - ***S***: bit input, senses laser reflection
  - ***D***: 16-bit output, displays computed distance

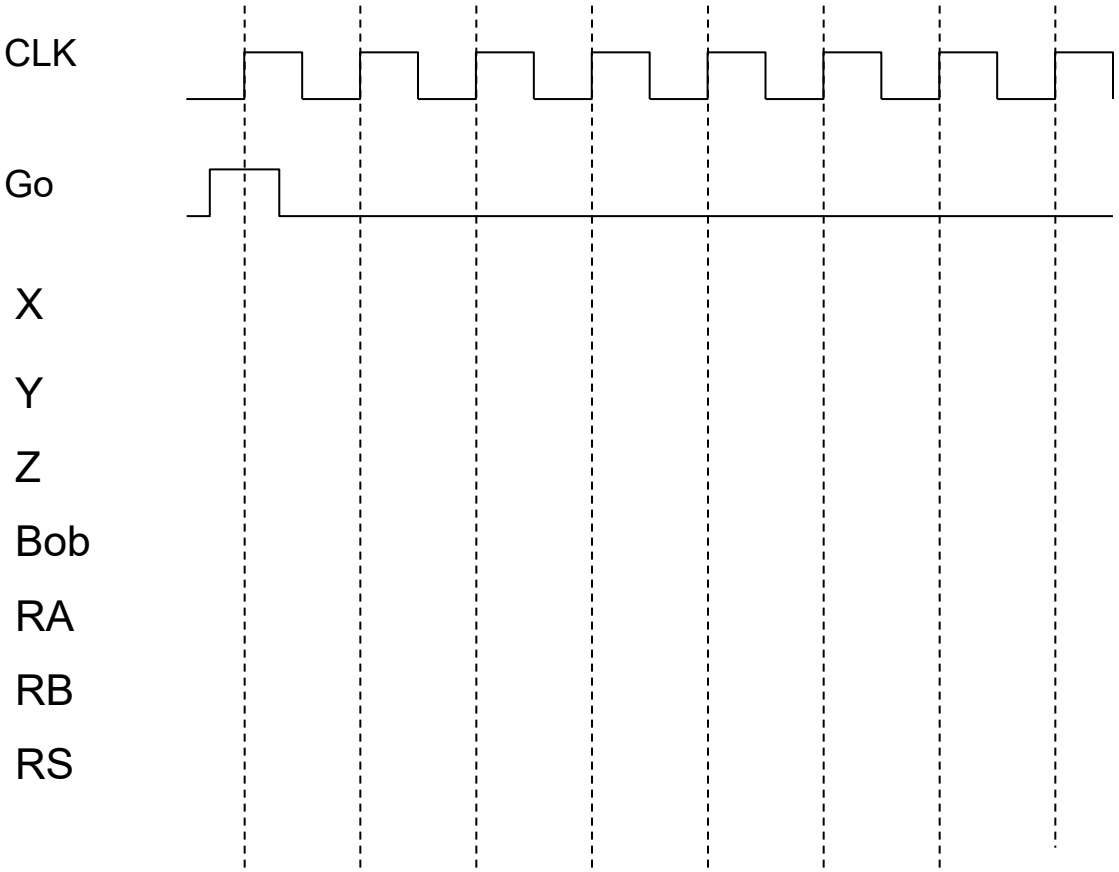
	Step	Description
Step 1	*Capture a high-level state machine*	Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more than just Boolean operations on bit inputs and outputs.
Step 2	*Create a datapath*	Create a datapath to carry out the data operations of the high-level state machine.
Step 3	*Connect the datapath to a controller*	Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.
Step 4	*Derive the controller's FSM*	Convert the high-level state machine to a finite-state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath.

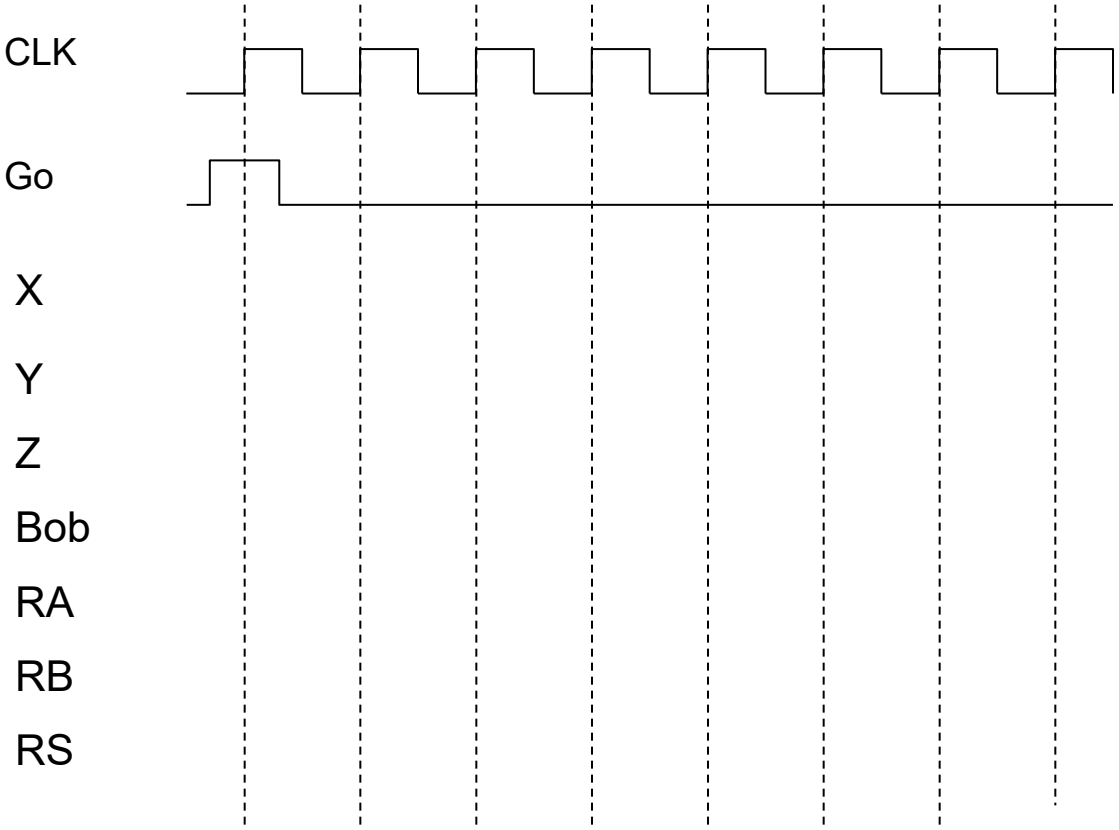
Yet another example

Using the following datapath, design a machine which finds the maximum value of the first 16 memory elements and puts the result in the 17th one.



Later we'll use a similar "one bus" datapath to build a simple computer.


CLK
Go
X
Y
Z
Bob
RA
RB
RS


CLK
Go
X
Y
Z
Bob
RA
RB
RS