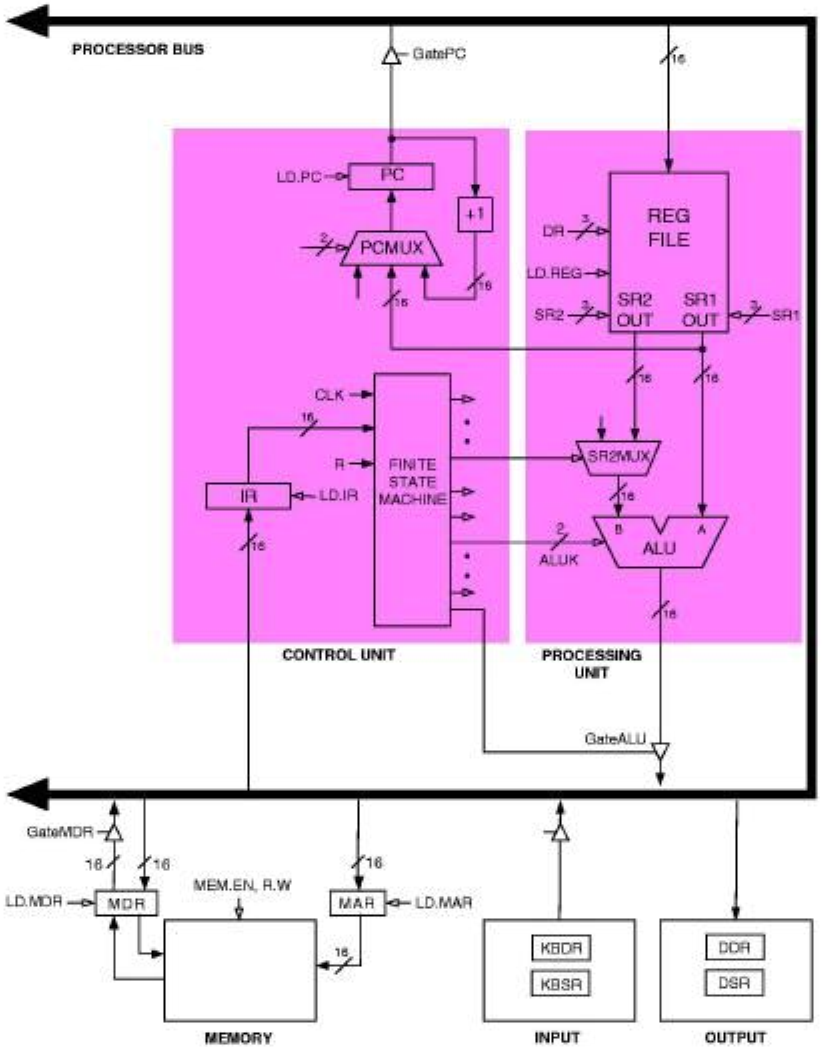
LC-3 Instruction Fetch

How the Finite State Machine controls the Data Path

We already know that the purpose of any Finite State Machine is to control whatever "engine" makes up the system in question, whether it be a garage door opener, a detour sign, or - in our case - the data path of a microprocessor. Each state of the FSM consists of a specific set of control signals activating the system "engine" in some particular way.

We also know that we can regard a computer as ***"an electronic device that takes in information, stores it, and manipulates it to create new information"*** ... we can think of the data path as the part that actually does that "manipulation" of the bits with which our information is represented.

So we need to start by understanding exactly what our LC-3 data path can do.
Remember, we can view the data path as consisting of 3 main modules: Memory, Processing, and Control, plus peripherals for i/o:



As you can see, the FSM is actually part of the control module, but we will view it as the "orchestrating" principle for the entire device.

So, depending on the current state, the FSM can cause data to "flow" from one register to another or from Memory to a register or vice versa, or a particular operation to be carried out in the ALU, etc.
It does this by turning specific control signals on & off in the correct sequence - a bit like you might control the flow of chemical reactants through a reactor by opening & closing valves ...

You already know exactly how many of the individual components of the data path work: registers (controlled by Write Enable (WE) signals); multiplexers (controlled by Select signals); decoders (set up by Address bits); adders (no control signal required!), etc.

There are two components that need a little more explanation: the bus, and gates.

The microprocessor bus:
Every component has to receive data and pass data on. Rather than constructing a separate channel for every such transmission, we have components ***write*** their data ***to*** a bus, and other components ***read*** data ***from*** the bus: you can see why we call it a "bus" - data "gets on" and "gets off" as needed!
It is actually an active circuit - you can think of it as a register with a long "tail" - but we will kind of pretend it is just a bundle of 16 wires to keep things simple.

Now reading from the bus is straightforward: all 16 bits of the bus will be wired to the input of a register that needs the bus data, so all we need to do is assert the register WE at the right instant.

But writing to the bus is rather trickier! A number of different devices have to write their data to the bus at different times - what would happen if two devices just applied their data at the same time?? The whole device would short circuit!!

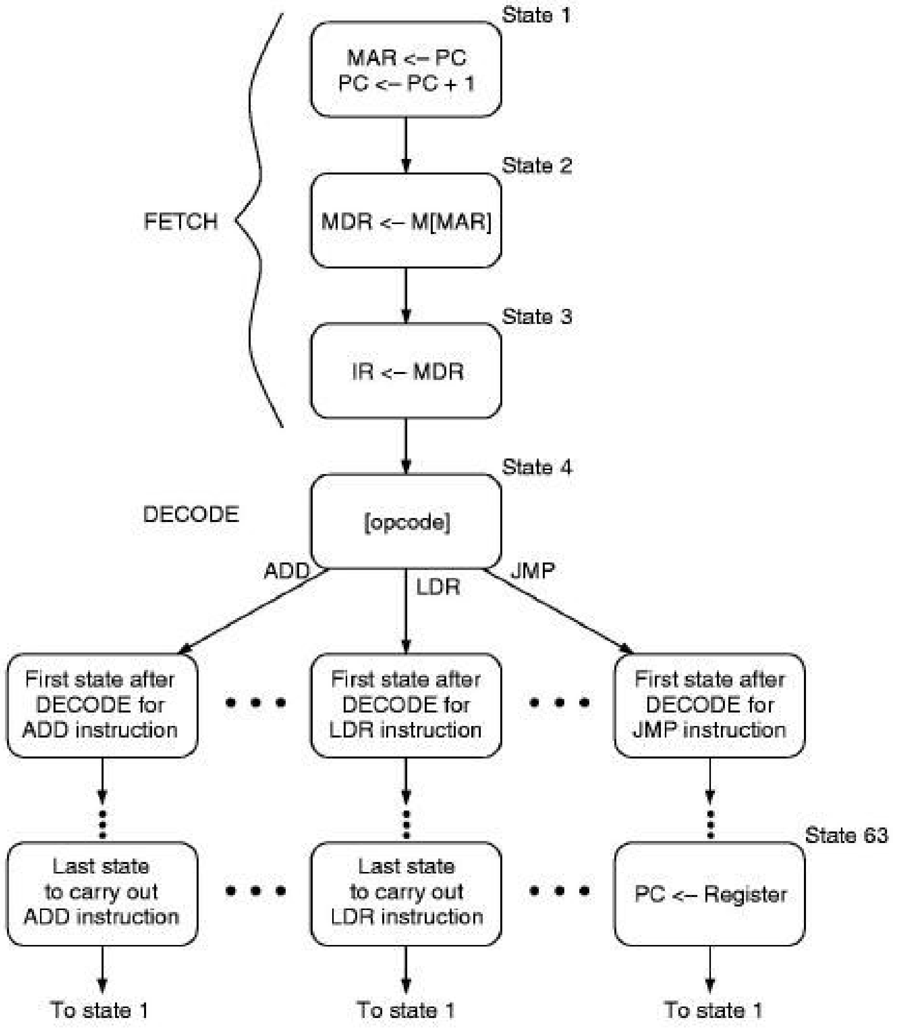
Which brings us to ...
Gates, aka tri-state devices:
Clearly we need a "gate" between each data provider and the bus. We will "open" the gate only at the instant that particular device needs to write its data, and keep it closed otherwise. Only one device will ever be given access to the bus at any given instant.

Gate truth table:

Ctrl	**In**	**Out**
0	0	**Hi Z**
0	1	**Hi Z**
1	0	**0**
1	1	**1**

In other words, when the control signal is off, the output is "high impedance" = disconnected; when the control signal is asserted, the output is equal to the input.

Finally, we can now trace the trajectory through state space that implements the first step of the von Neumann Instruction Cycle = Fetch Instruction.


State 1
MAR <– PC
PC <– PC + 1
State 2
FETCH
MDR <– M[MAR]
State 3
IR <– MDR
State 4
DECODE
[opcode]
ADD
LDR
JMP
First state after DECODE for ADD instruction
First state after DECODE for LDR instruction
First state after DECODE for JMP instruction
State 63
Last state to carry out ADD instruction
Last state to carry out LDR instruction
PC <– Register
To state 1
To state 1
To state 1

The overall purpose of the first three states is to copy the next instruction from its location in memory to the Instruction Register (IR), at which point the opcode, IR[15:12], will be presented to the FSM, triggering a transition to the branch corresponding to that instruction. The figure shows just 3 of the possible 16 trajectories out of the DECODE state.

<u>State 1</u>:	MAR <= (P)	; copy the contents of PC (the address at which the next instruction is stored in memory) to the MAR (via the bus)
	PC <= (PC) + 1	; increment the value of the PC by 1, and write it back to the PC
<u>State 2</u>:	MDR <= Mem[(MAR)]	; copy the contents of memory location (MAR), to MDR
<u>State 3</u>:	IR <= (MDR)	; copy contents of MDR to IR (via the bus)

So what control signals does the FSM need to assert in each of these states in order to implement this sequence of events?

Note that all control signals are *<u>named</u>*:

- *Write enables are called* **`Ld.<name of register>`**
- *Multiplexor select lines are called the name of the multiplexor (***`PCMUX`***, etc.)*
- *Gate controls are called* **`Gate.<name of source device>`**

<u>Follow the control signal names on the data path schematic:</u>

First, as noted above, the overall Register Transfer we are aiming for is:

`IR <= Mem[(PC)]`

Make sure you completely understand and can use this notation:
- take the contents of the PC, treat it as a memory address; read from that location in memory; and copy the value found there to the Instruction Register.
We will accomplish this full transfer in a set of smaller steps, encapsulated in three states:

<u>State 1:</u>	MAR <= (PC)	1. Gate.PC	*- copy PC to bus*
		2. Ld.MAR	*- copy bus to MAR*
	PC <= (PC) + 1	3. PCMUX selects incrementer	
		4. Ld.PC	*- write incrementd PC back to PC*
<u>State 2</u>:	MDR <= Mem[(MAR)]	5. Mem.En/R	*- enable Memory, for reading*
		Note that a memory operation can take many FSM cycles to complete, so the FSM has to wait for a "Ready" signal to come back from memory before proceeding to the next step.	
		Wait for Ready	
		6. Ld.MDR	*- copy Memory read to MDR*
<u>State 3</u>:	IR <= (MDR)	7. Gate.MDR	*- copy MDR to bus*
		8. Ld.IR	*- copy bus to IR*

==

We will be doing this same analysis for 14 of the 15 LC-3 instructions, so make sure you can follow along and understand every step of this tutorial.

As added motivation, one full question on the final exam will be one of these analyses :)