CS 2110 - Lab 05

State Machines & the LC-3

Monday, June 6th, 2021

Lab Assignment: LC3 Quiz!

1) Go to Quizzes on Canvas
2) Select **Lab 05**, password: **Bus**
3) Get a 100% to get attendance!
    a) Unlimited attempts
    b) Collaboration is **allowed**!
    c) Ask your TAs for help :)

Homework 2

- Released!
- **Due Monday, June 6th (today!) at 11:59 PM**
- Files available on Canvas
- Submit on Gradescope
    - **Double check that your grade on Gradescope is your desired grade!**

Homework 3

- Released!
- **Due Monday, June 13th at 11:59 PM**
- Files available on Canvas
- Submit on Gradescope (unlimited submissions)

Timed Lab 1

- **Scheduled this Wednesday, June 8th (next lab!)**
- 75 minutes to complete at the beginning of lab, then lab will be resumed.
- Will cover HW2 Material primarily
  - Please have Docker working on the device you bring to Lab! If you have any trouble, please see a TA ASAP
  - If you need outlet priority, please let a TA know ASAP
  - You may use any submitted homework files during the timed lab
  - Unlimited submissions throughout the TL period.

Quiz 2

- **Next Wednesday, June 15th**
- Quiz opens at your assigned lab time (unless you have already established a different time with your professor)
- Full 75 minutes to take the quiz!
- After the quiz, you will join lab for the remaining period.
- A topic list will be posted on Canvas

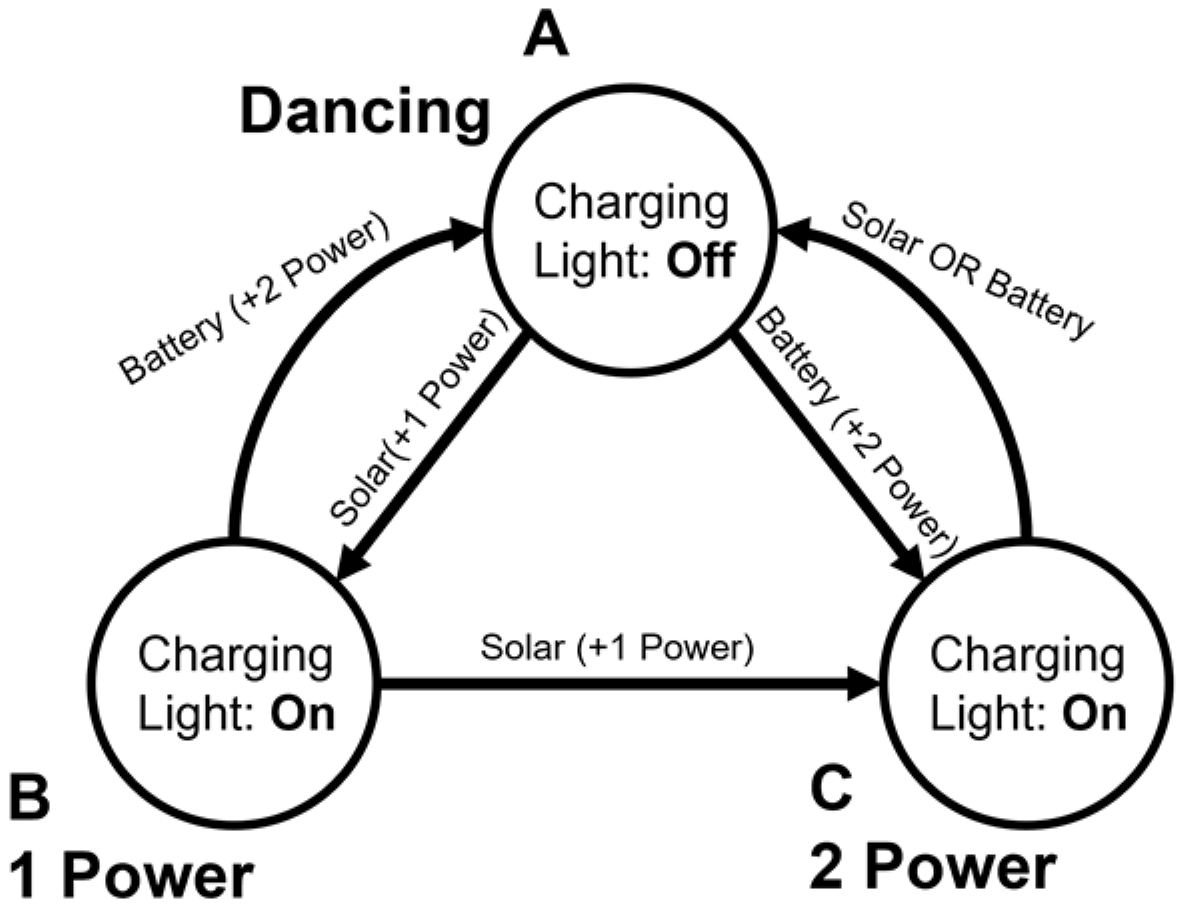
Today's Topics

- State Machines
  - One Hot
  - Binary Reduced
- The LC-3 Datapath
  - The Bus
  - Tri-state buffers
  - Control signals
  - Microcontrollers

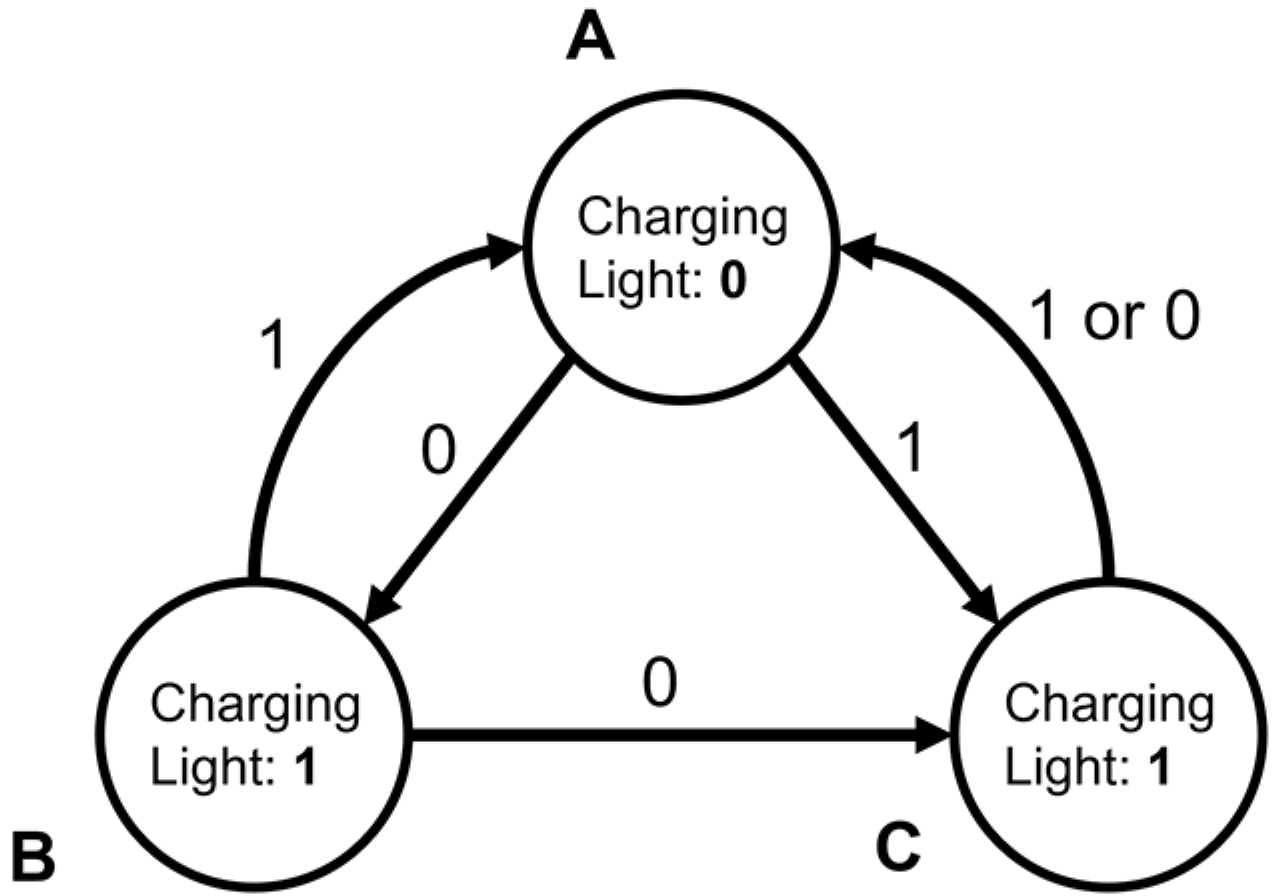
State Machines!

- Example: Dave, the Funky Dancing Robot Crab
    - Dave is a dancing robot crab. Before it can dance it must fully recharge (3 charges). It can recharge slow by solar energy (1 charge) or fast with a battery (2 charges).
    - Dave has 3 possible states:
        1. Dancing (requires 3 charges)
        2. Charging with 1 charge
        3. Charging with 2 charges

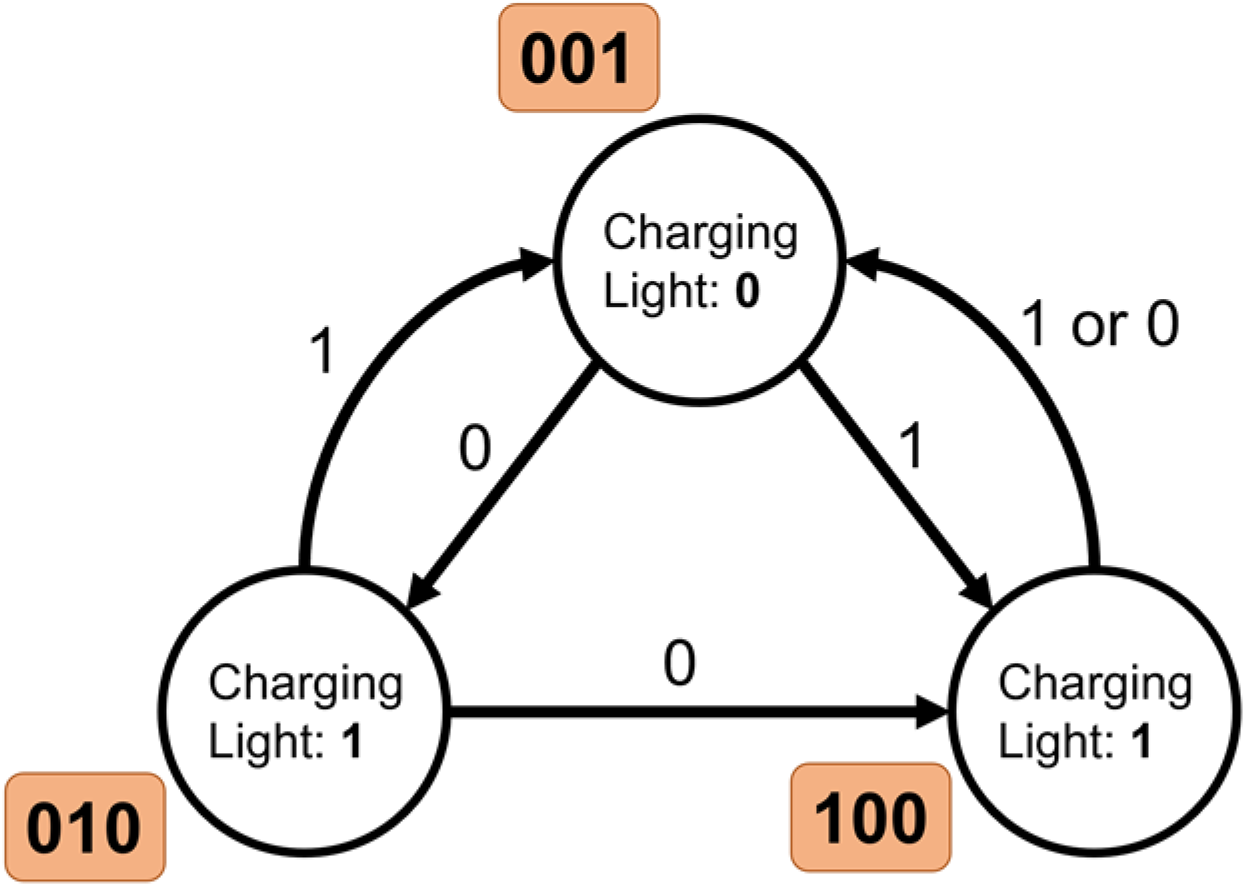
Dave’s State Transition Diagram



State Transition Diagram - Representation

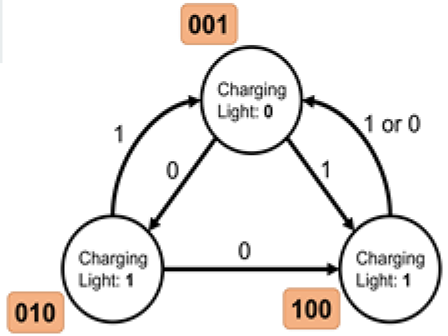


State Machine Diagram - One Hot

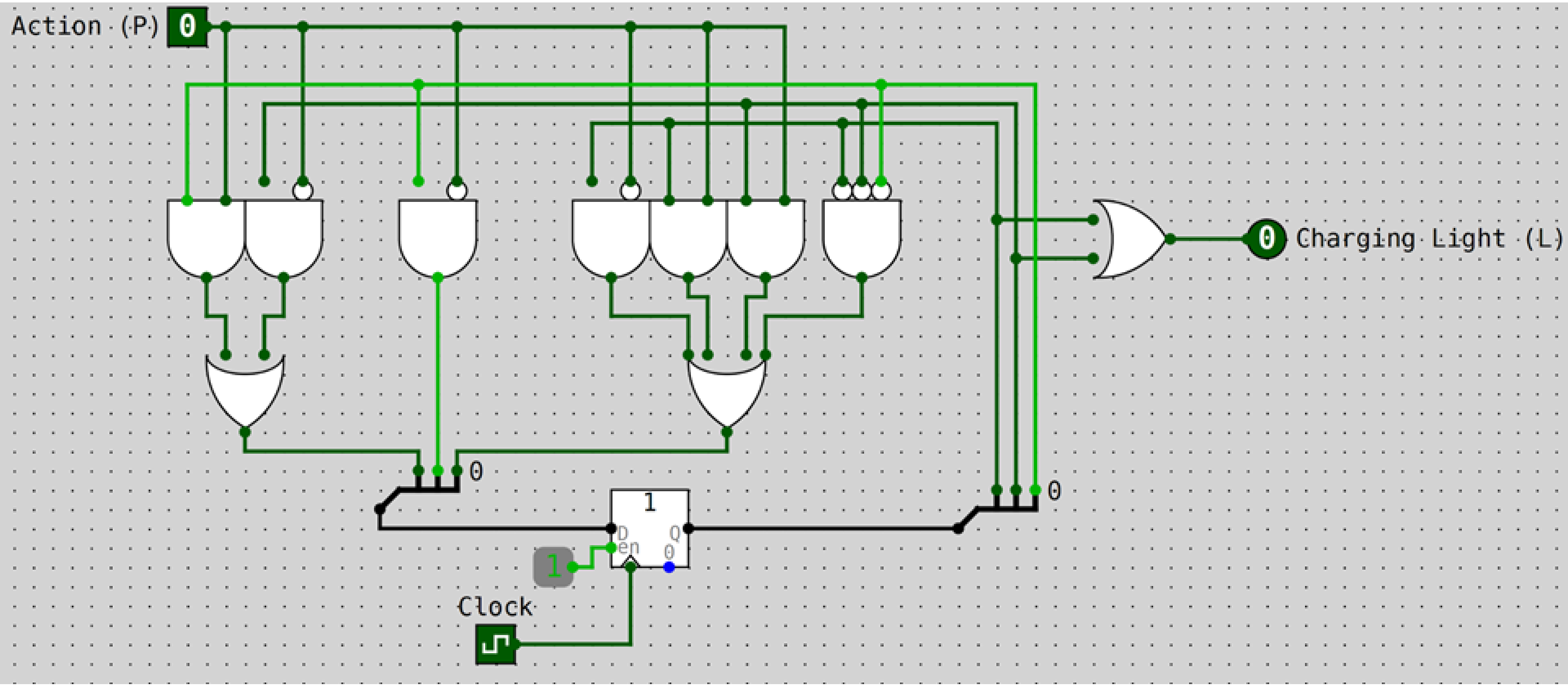


Truth Table - One Hot

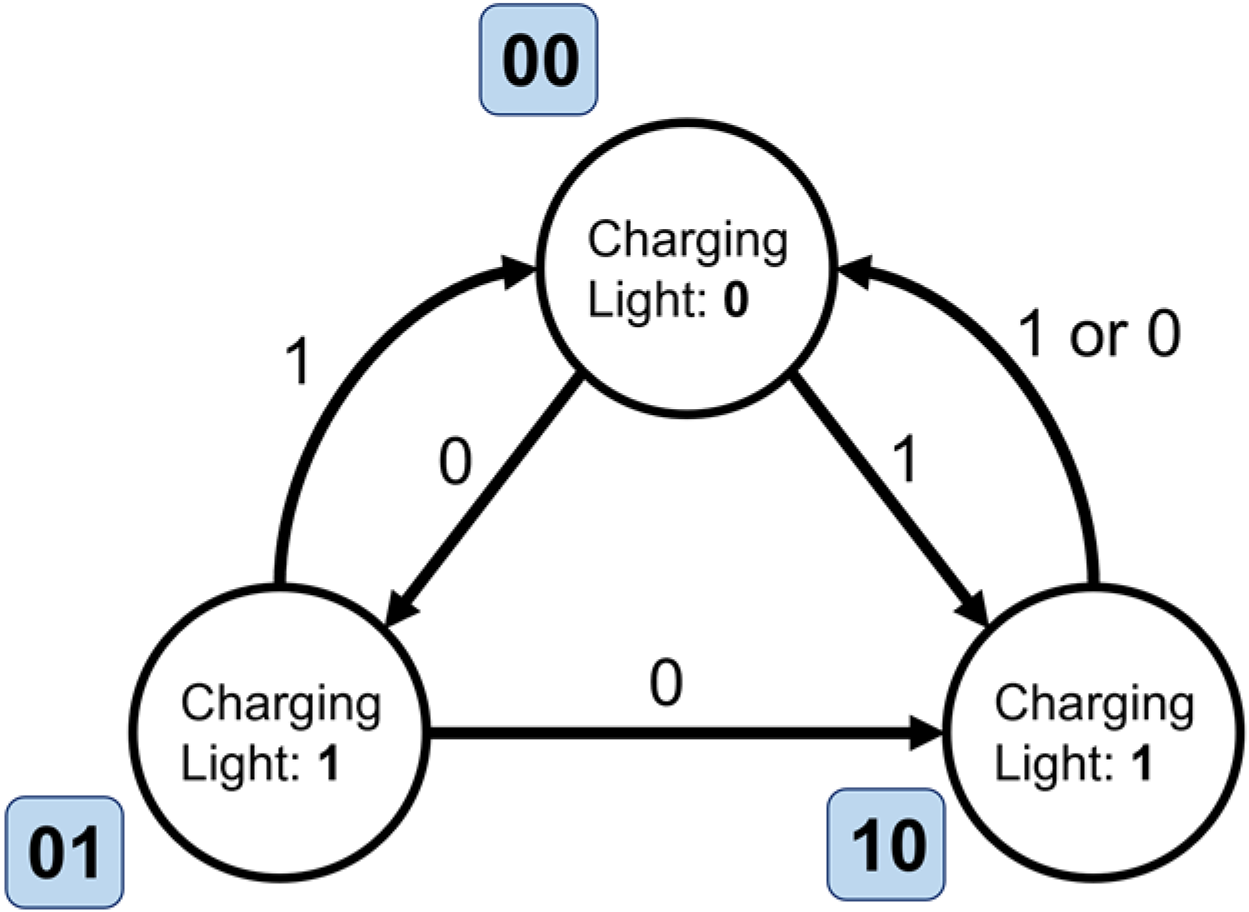
P	$S_2S_1S_0$	$N_2N_1N_0$	L
X	000	001	0
0	001	010	0
1	001	100	0
0	010	100	1
1	010	001	1
0	100	001	1
1	100	001	1



Circuit - One Hot



State Machine Diagram - Binary Encoded



Truth Table - Binary Encoded

P	S_1S_0	N_1N_0	L
0	00	01	0
1	00	10	0
0	01	10	1
1	01	00	1
0	10	00	1
1	10	00	1
0	11	XX	X
1	11	XX	X

Since we only have three inputs (P, S_1, S_0), we can use K-maps to create a simpler circuit. We have to create one K-map for each output.

N_1	$S_1'S_0'$	$S_1'S_0$	S_1S_0	S_1S_0'
P'	0	1	X	0
P	1	0	X	0

$N_1 = S_1'S_0'P + S_0P'$

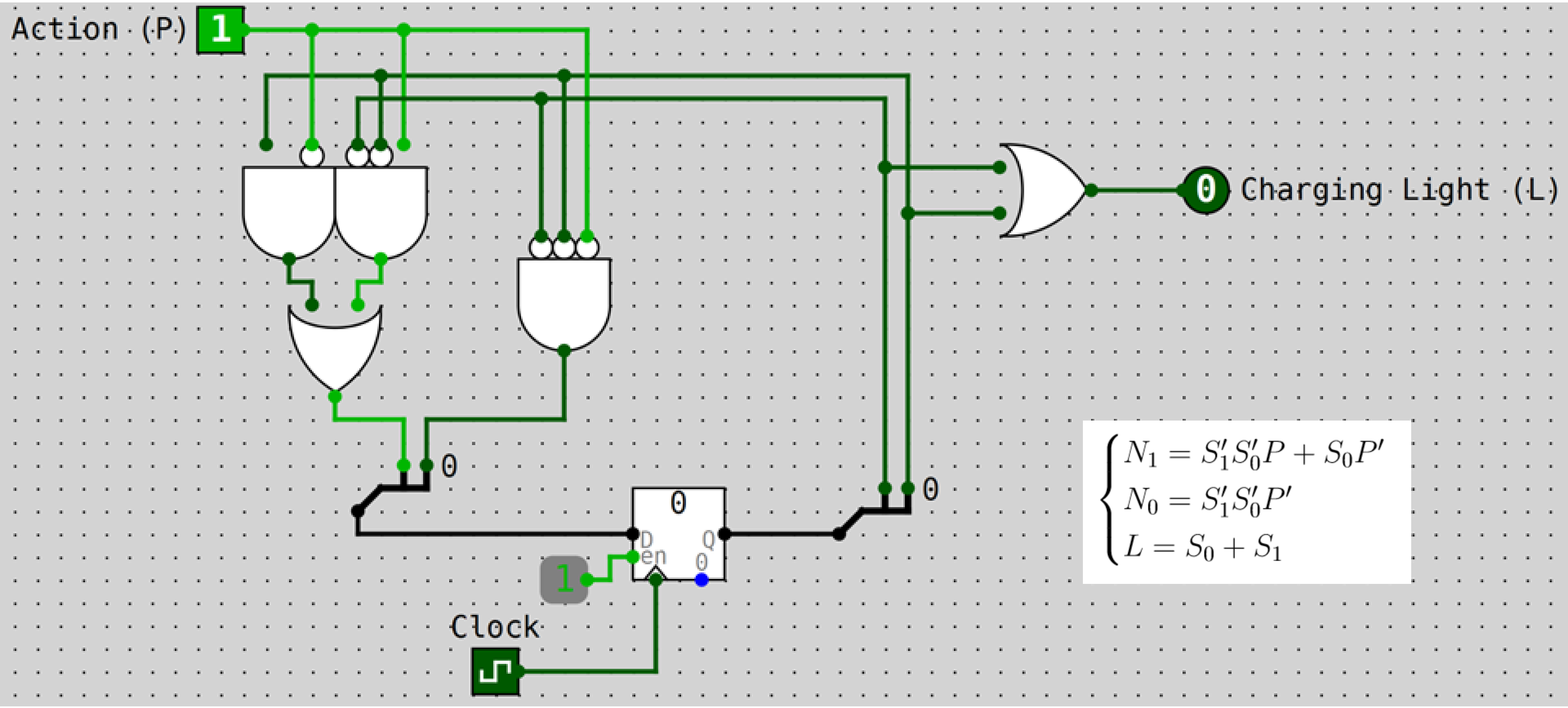
N_0	$S_1'S_0'$	$S_1'S_0$	S_1S_0	S_1S_0'
P'	1	0	X	0
P	0	0	X	0

$N_0 = S_1'S_0'P'$

L	$S_1'S_0'$	$S_1'S_0$	S_1S_0	S_1S_0'
P'	0	1	X	1
P	0	1	X	1

$L = S_0 + S_1$

Circuit - Binary Encoded



$$\begin{cases} N_1 = S_1'S_0'P + S_0P' \\ N_0 = S_1'S_0'P' \\ L = S_0 + S_1 \end{cases}$$

A word of caution...

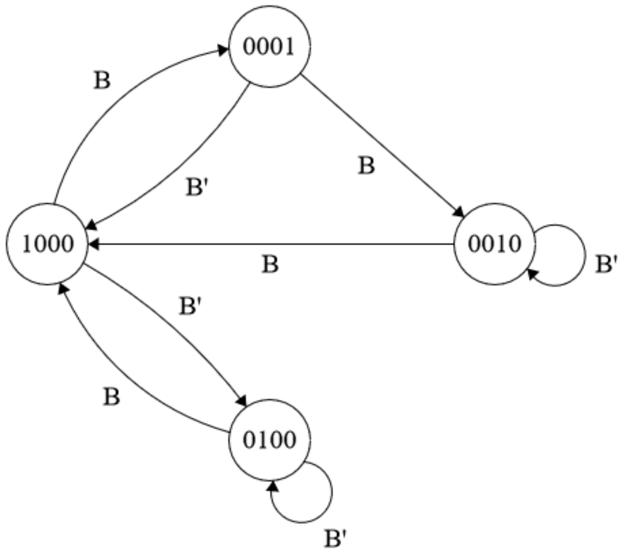
- State machines are NOT event-driven; they are locked to the clock
- The state is always updated once per clock cycle, even if you transition into the same state
- This is true of all sequential logic in modern computers

Given a state machine with 20 states, how many bits do you need to represent it in a one-hot state machine?

Given a state machine with 17 states, how many bits do you need to represent it in a binary encoded state machine?

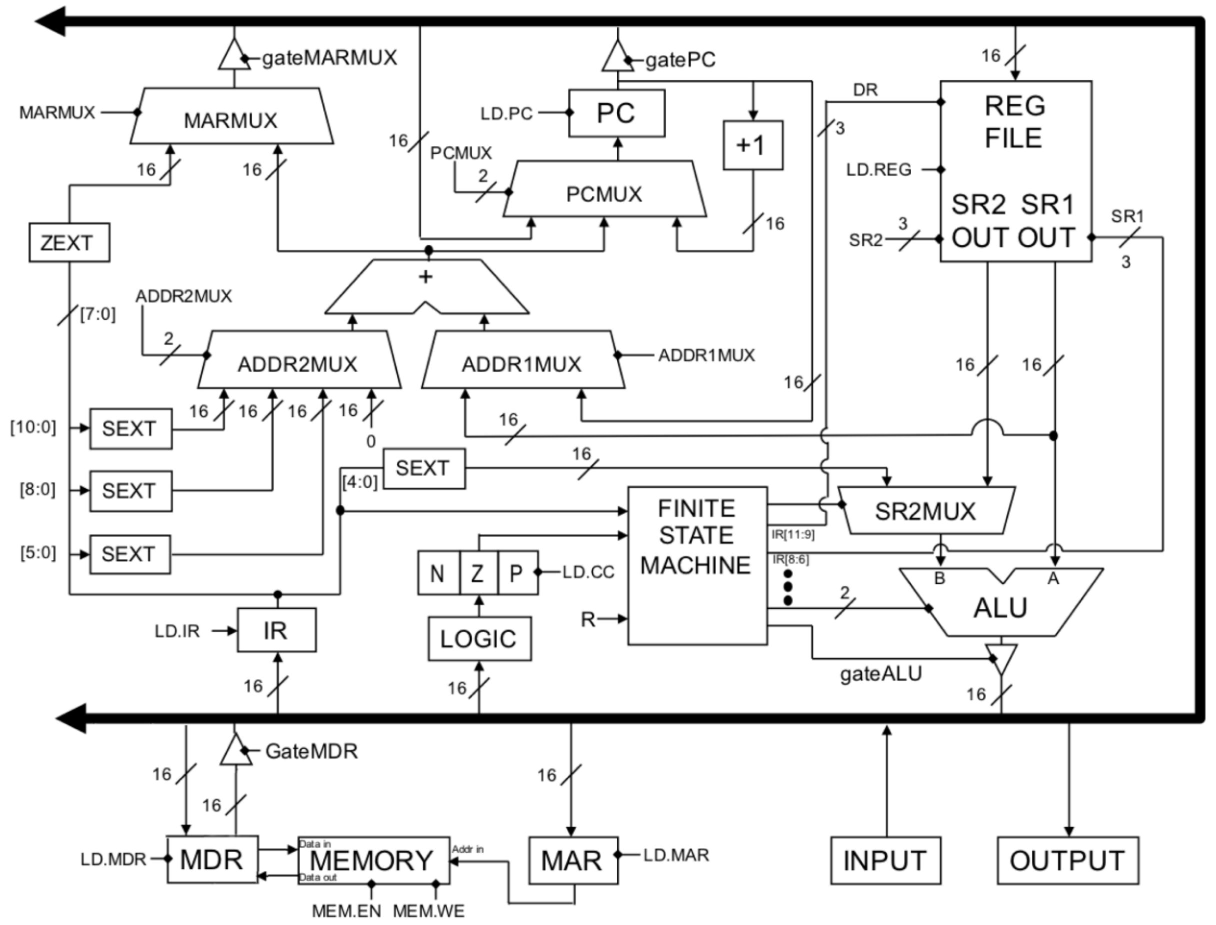
Which of the following is a correct boolean expression for N3?

- S0B' + S1B + S3’
- S0B' + S1B + S2B
- S1B + S2B
- N0B + N1B' + S0B'

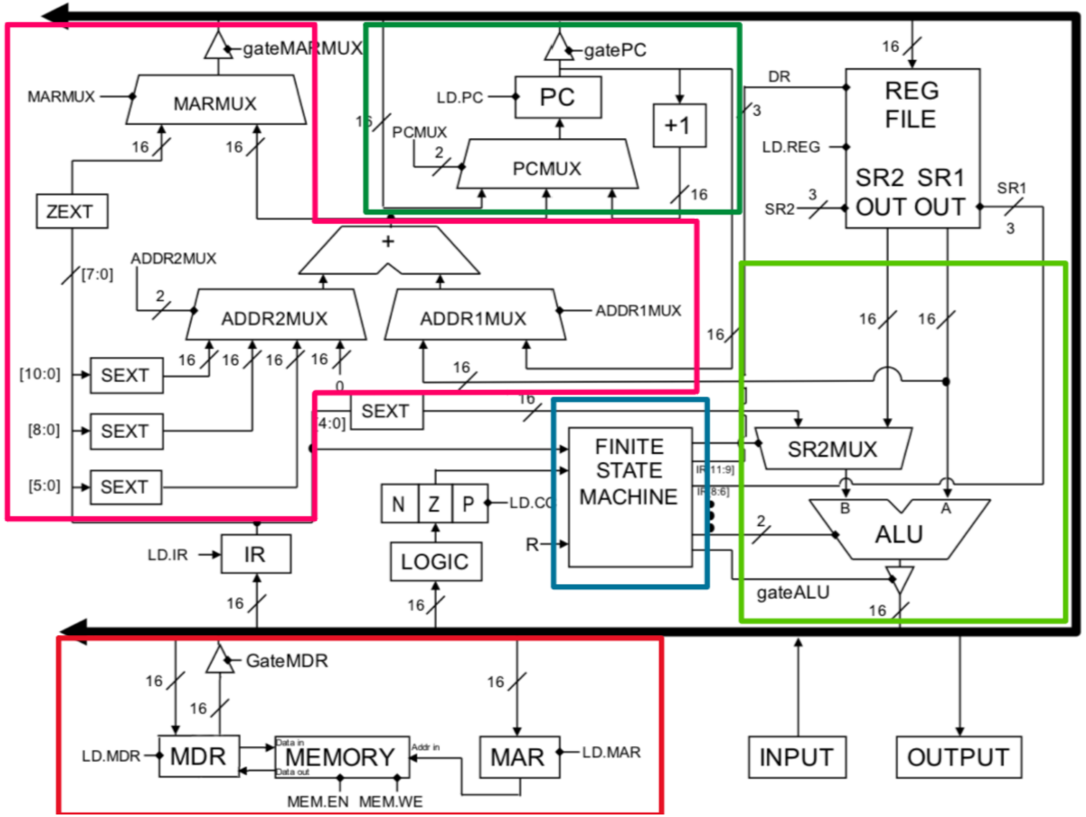


Intro to the LC-3

- A pedagogical processor architecture
- Combines everything we've learned so far
    - ALU, state machine, registers, memory, etc.
- Looks intimidating, but we'll break it down into its components

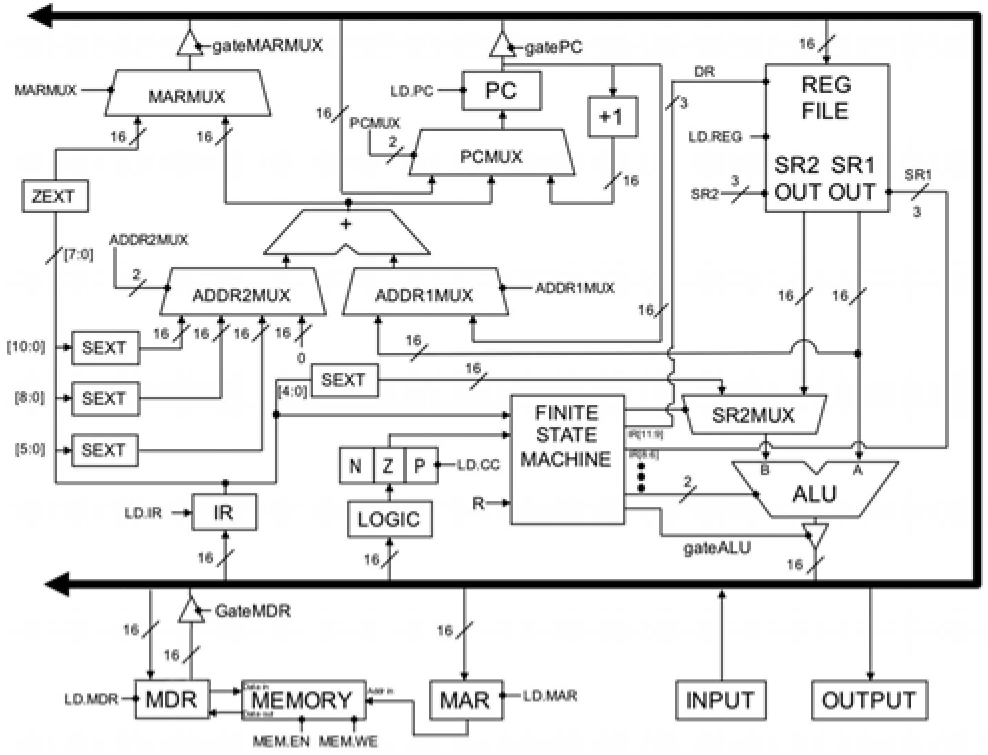

gateMARMUX
gatePC
MARMUX
MARMUX
LD.PC
PC
+1
PCMUX
PCMUX
DR
REG FILE
LD.REG
SR2 SR1 OUT OUT
SR2
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR2MUX
ADDR1MUX
ADDR1MUX
+
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
SEXT
[4:0]
0
FINITE STATE MACHINE
IR[11:9]
IR[8:6]
SR2MUX
N
Z
P
LD.CC
R
ALU
B
A
LD.IR
IR
LOGIC
gateALU
GateMDR
LD.MDR
MDR
MEMORY
Data in
Data out
Addr in
MEM.EN
MEM.WE
MAR
LD.MAR
INPUT
OUTPUT

LC-3 Components



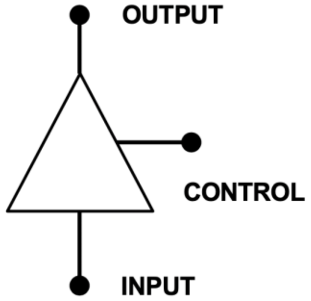
The Bus

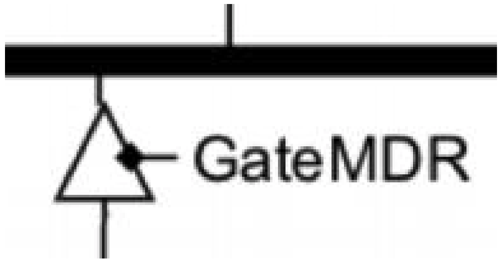
- 16-bit wire that transfers data between many components
- Only **one** value on wire at a time—why?



Tristate Buffer

- Helps in making sure there is only one value on bus
- Driven through signals like `GatePC`, `GateMDR`, etc.

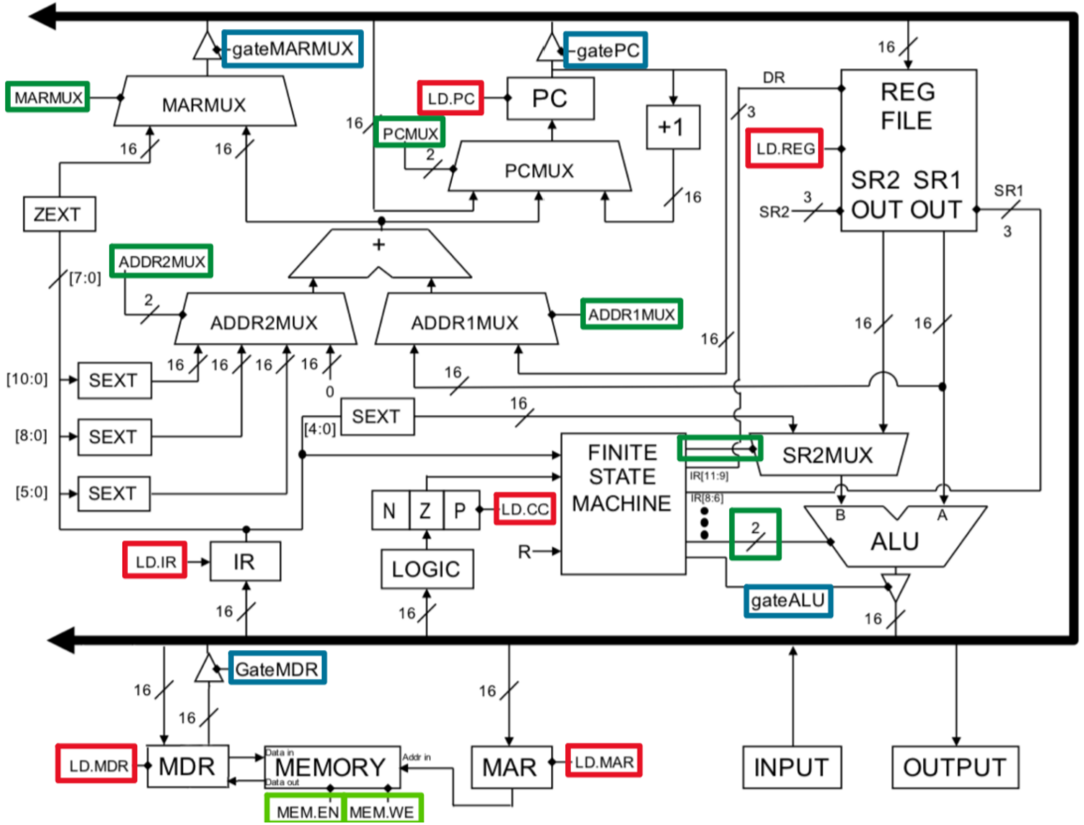




CONTROL	INPUT	OUTPUT
0	0	Z
0	1	Z
1	0	0
1	1	1

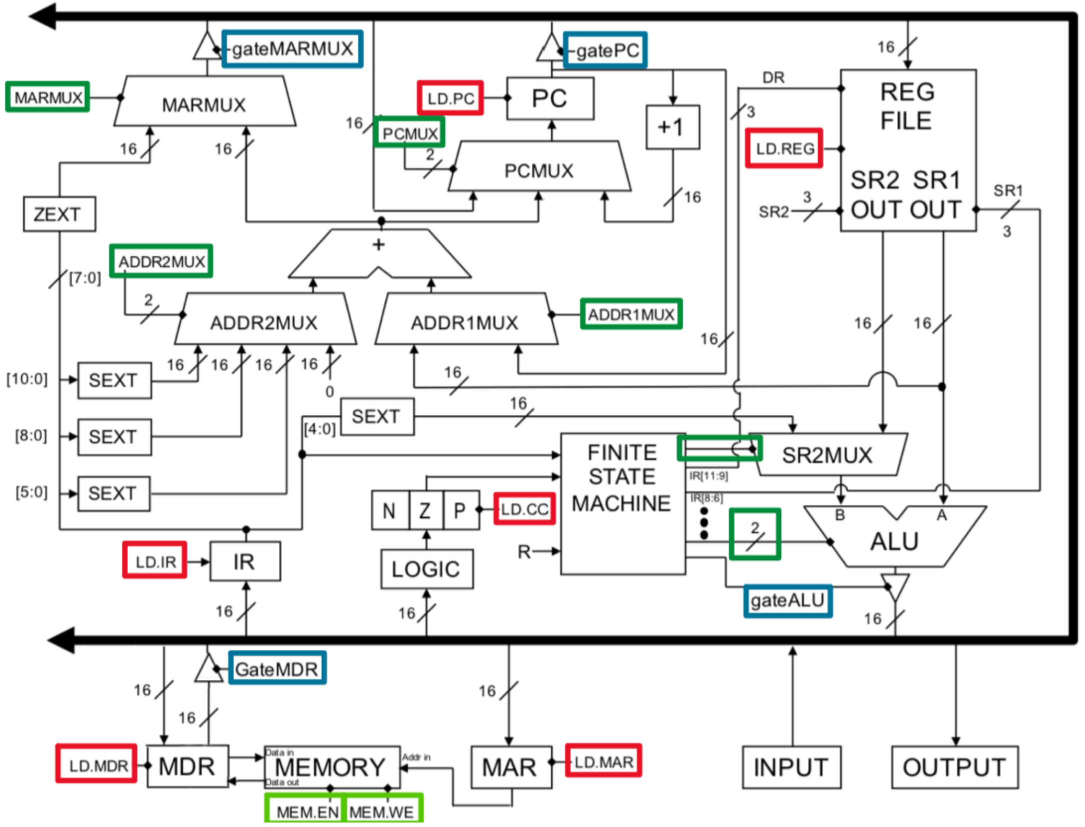
Control Signals

- Used to move data around
  - LD.___ — write enable for a register
  - gate___ — tri-state buffer that allows data to go onto the bus
  - ___MUX — selector for a multiplexer



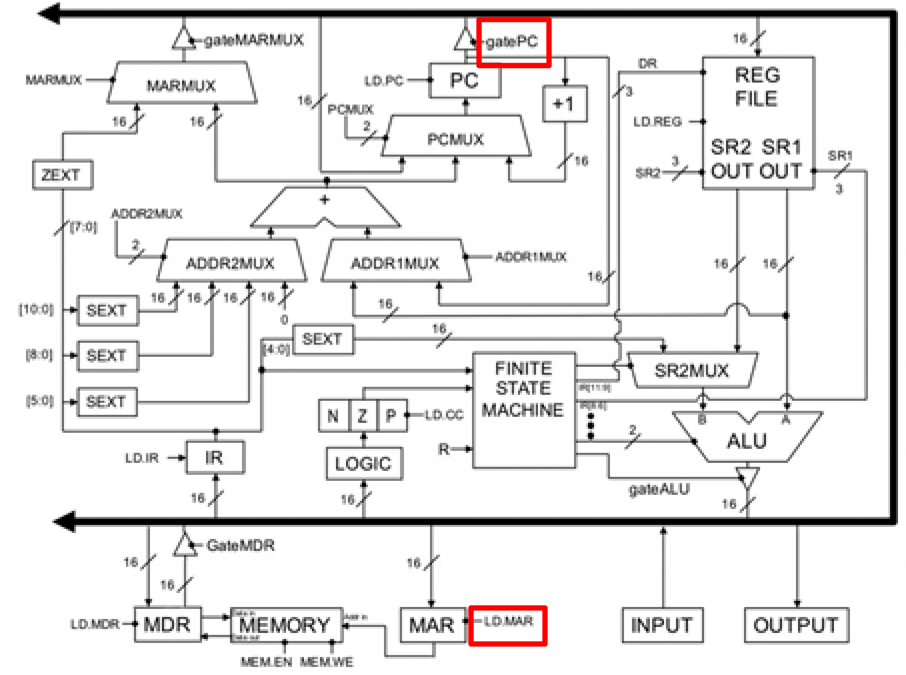
Example

- Say we want to put the data from the PC (a register) into the MAR (another register). What signals do we turn on?

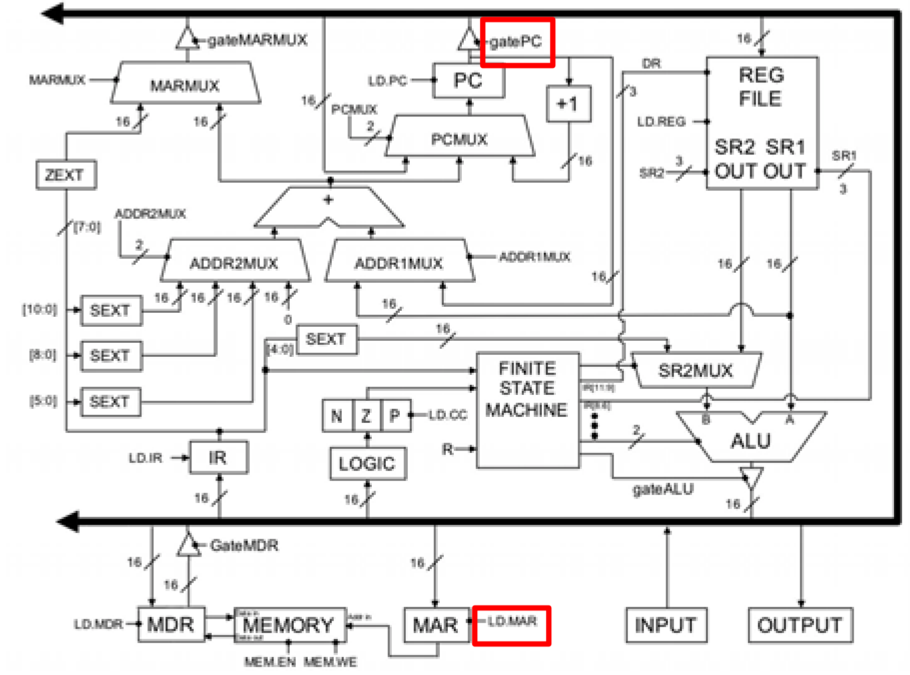


Answer

- `gatePC` to drive the data in the PC onto the bus
- `LD.MAR` to enable writing into the MAR
- On the next clock edge, the data will be saved!



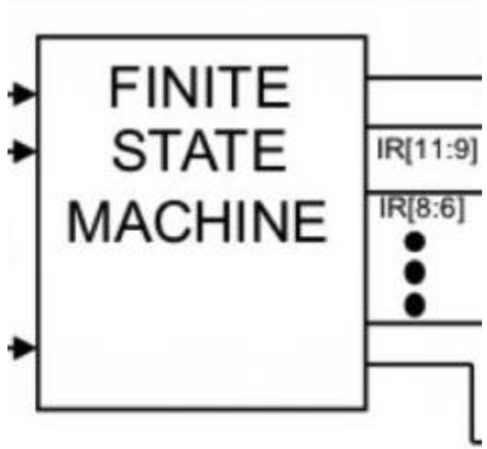
What is the result of GatePC being asserted?



Clock

- Clock cycles
    - Alternates between 0/1’s
- One clock is connected to all the registers, keeping them in sync
- Ensure that we don’t run into short circuits
- When a register has its Write Enable on, the next clock edge saves its current input

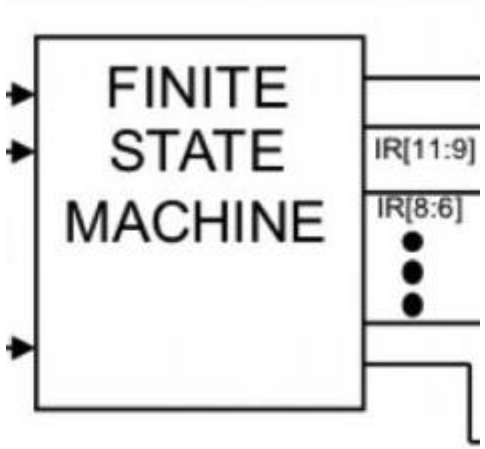
The Microcontroller



- Finite state machine (FSM)
- Every instruction has a series of states to execute it
- Like in HW3, each state has
    - A set of outputs (control signals like ALUK, LdReg, etc.)
    - The next state to go to
- Ensures that correct operations take place, while preventing short circuits
- You'll be filling in parts of the state machine truth table for HW4

The Purpose of the FSM

- Turn control signals on/off which are sent out to the rest of the datapath
- Move between microstates in order to fetch, decode, and execute instructions (more on this soon)



What is Memory? - Registers VS. Memory (RAM)

You have been exposed to **sequential logic**. (i.e. registers)

Registers: Used for quick data storage on the processor.

Random Access Memory:

- Memory is like a really big array
- Contains instructions for programs that are currently being executed.
- Stores any data a program may need.

The two “sides” of memory:

- Address side
- Data side (or Data @ Address)

NOTE: Everything here is technically in binary.

Memory Layout Example

This example is used to show the type of data that can be found in memory.

Addresses	Data @ Addresses
0x3000	ADD R1, R2, R0
0x3001	HALT
0x3002	0**b**0010000100010000
0x3003	0**x**2110
...	...
0x300A	#8464

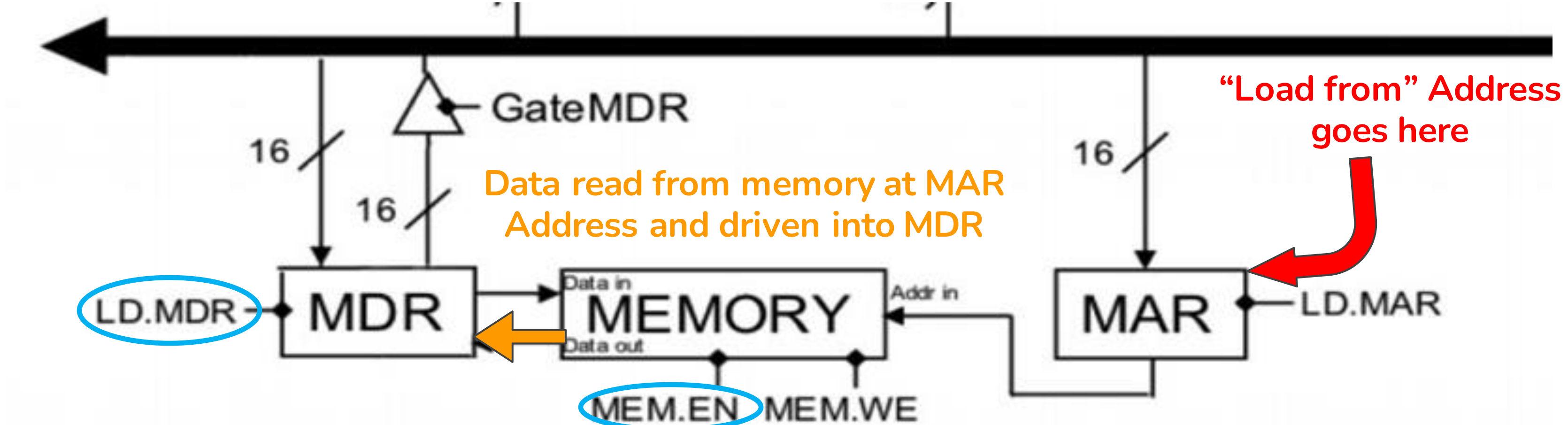
LC-3 Instructions:
Piece of some program in memory.

Random Data in Memory:
Data used by a program currently being executed.
OR
Left over data from an old program previously executed.

Loading from Memory

<u>**Load data** from memory:</u>

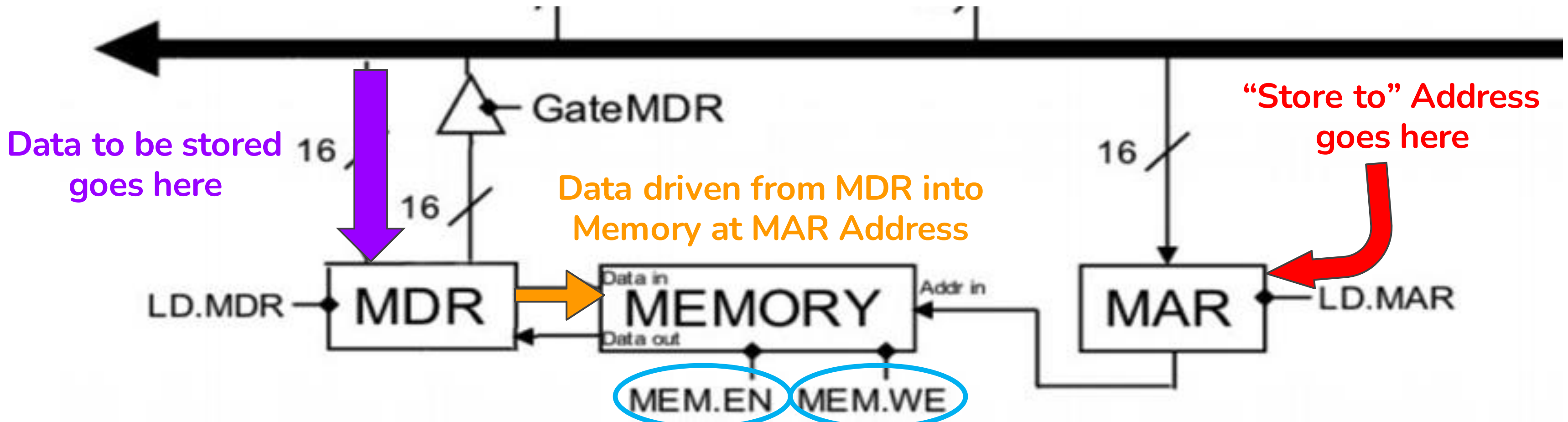
- Place the **address** of where the data is located in **MAR** (**Memory Address Register**).
- Memory is **read** from that address and “driven” to the **MDR** (**Memory Data Register**). After that, the data moves to its destination register.


GateMDR
16
16
16
“Load from” Address goes here
Data read from memory at MAR Address and driven into MDR
LD.MDR
MDR
Data in
MEMORY
Data out
Addr in
MAR
LD.MAR
MEM.EN
MEM.WE

Storing into Memory

Store data into memory:

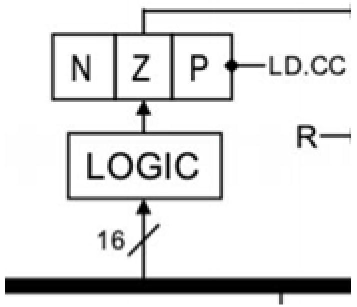
- Data from a register is placed into the **MDR**.
- Address of where to store data is placed into the **MAR**.
- Data is stored in memory at the address specified by the **MAR.**

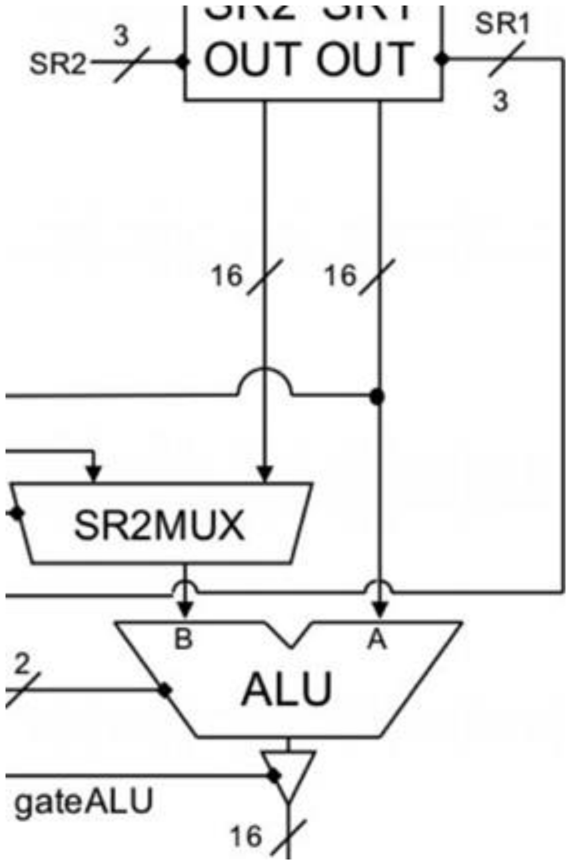

GateMDR
Data to be stored goes here
16
16
16
Data driven from MDR into Memory at MAR Address
"Store to" Address goes here
LD.MDR
MDR
Data in
MEMORY
Data out
Addr in
MAR
LD.MAR
MEM.EN
MEM.WE

Note

- Every piece of data in memory is just 16 bits of binary
- How does the LC-3 know...
  - Which addresses contain data that's an instruction?
  - Which addresses contain data that's an integer?
  - Which addresses contain data that's a character?
- Answer: it doesn’t!

ALU

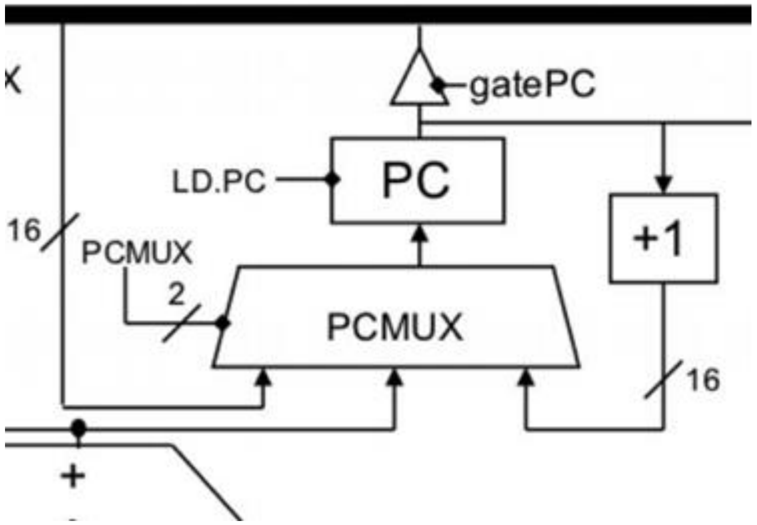




- Two inputs:
  - SR1: Data from register
  - SR2: Data from register **OR** immediate value from the instruction
- Select bits come from Finite State Machine
  - 00 → A + B
  - 01 → A AND B
  - 10 → NOT A
  - 11 → PASS A (unchanged)
- Drives output onto bus
- Condition codes (CC)
  - Set when writing to general purpose registers, LD.CC flag
  - Only three possible states: 001, 010, 100
  - What do they each mean?

PC

- Holds the current program counter – address of next instruction to execute
- Repeat that: "the address of the *next* instruction to execute"
- It does NOT hold:
  - The current instruction
  - The address of the current instruction
- Three ways to update PC
  - PC + offset
  - PC + 1
  - Given a value for the PC via the bus
- The "correct" way is selected through PCMux



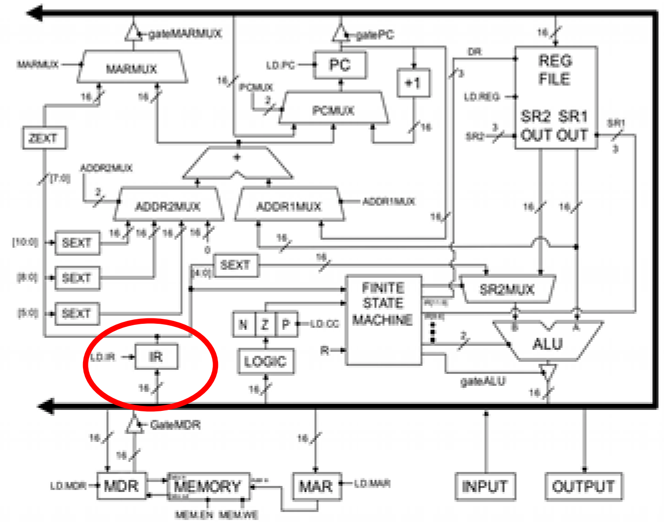
IR

- Instruction register – holds the **value** of the **current** instruction
- For example, the instruction ADD R3, R1, R2 may look like:

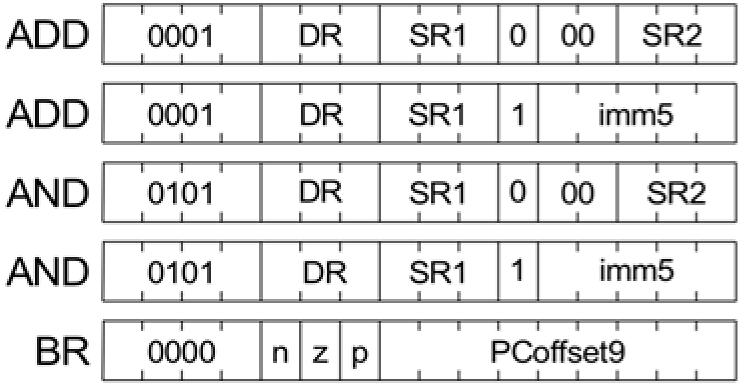
```
     0001   011   001  0  00  010
```

ADD	0001	DR	SR1	0	00	SR2

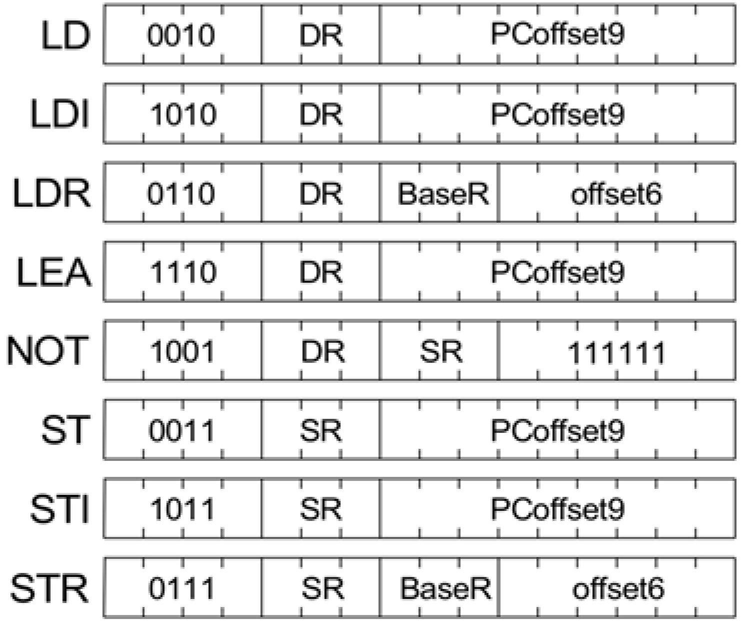
- Having the correct value in the IR helps the FSM determine the current state and send the correct control signals
- A new value can be read from the bus (with signal LD.IR) and the current value is directly connected to relevant components



LC3 Instruction Set

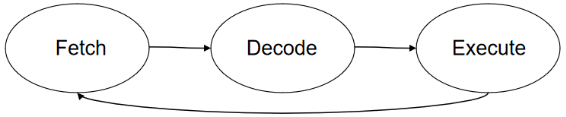


ADD	0001	DR	SR1	0	00	SR2
ADD	0001	DR	SR1	1	imm5	
AND	0101	DR	SR1	0	00	SR2
AND	0101	DR	SR1	1	imm5	
BR	0000	n	z	p	PCoffset9	



LD	0010	DR	PCoffset9	
LDI	1010	DR	PCoffset9	
LDR	0110	DR	BaseR	offset6
LEA	1110	DR	PCoffset9	
NOT	1001	DR	SR	111111
ST	0011	SR	PCoffset9	
STI	1011	SR	PCoffset9	
STR	0111	SR	BaseR	offset6

Executing Instructions


$$\begin{tikzcd}\text{Fetch} \arrow[r] & \text{Decode} \arrow[r] & \text{Execute} \arrow[ll, bend left]\end{tikzcd}$$

Instruction execution has three "stages", called *macrostates*

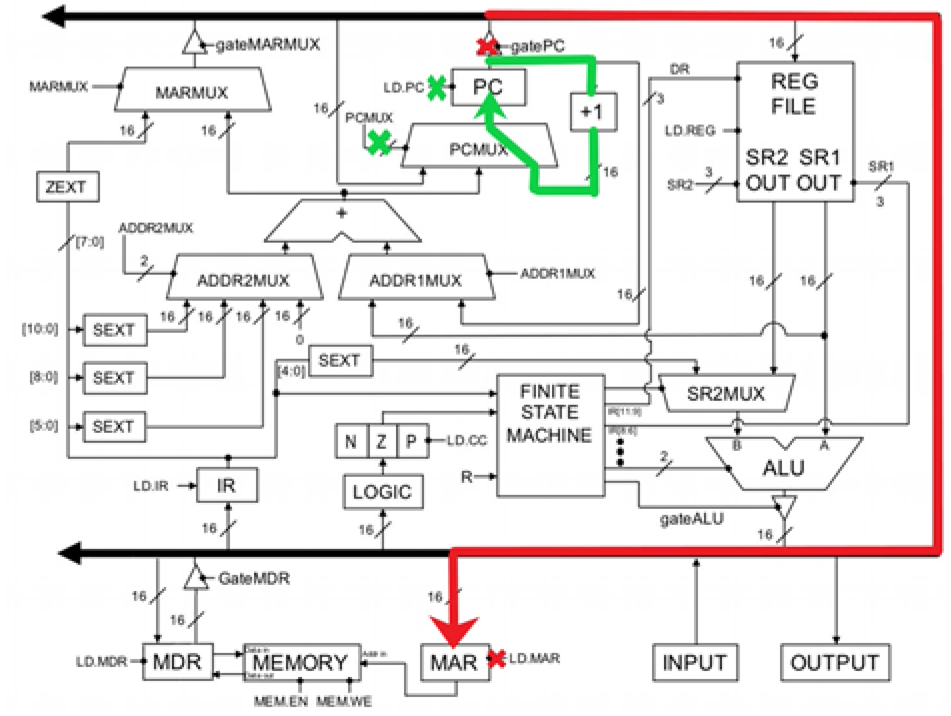
- Each macrostate is made of one or more *microstates* (1 per clock cycle)
- The microstates are the states in the microcontroller's state machine!

The macrostates are:

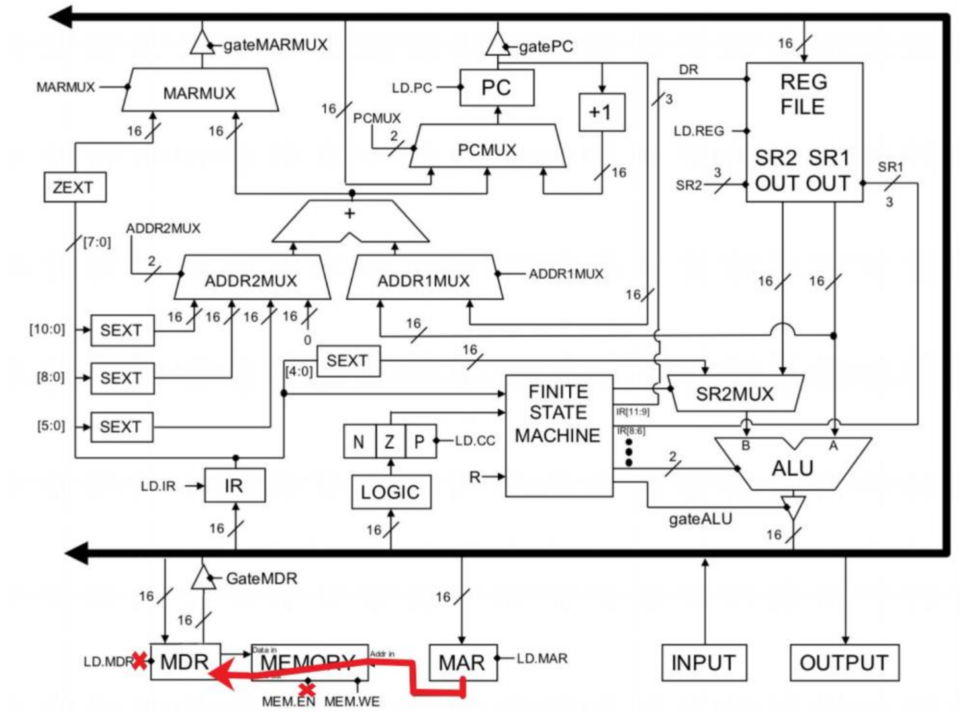
1. Fetch (3 clock cycles)
   - Load the next instruction from memory into the instruction register (IR)
   - Increment the PC
2. Decode (1 clock cycle)
   - The microcontroller looks at the instruction to figure out how to execute it
3. Execute (varies)
   - The microcontroller steps through several microstates and asserts the correct control signals to execute the instruction

The book splits up executing instructions into seven phases. This is a more conceptual overview of how it works and is unrelated to the three macrostates.

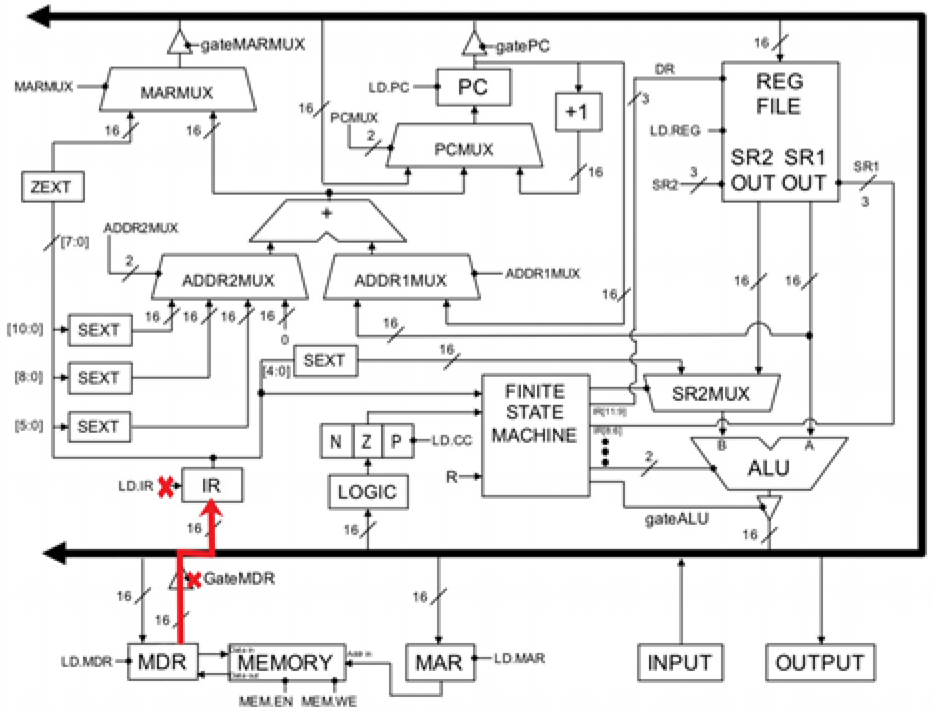
Fetch: Part 1


gateMARMUX
gatePC
MARMUX
MARMUX
LD.PC
PC
+1
PCMUX
PCMUX
REG FILE
DR
LD.REG
SR2 SR1 OUT OUT
SR2
SR1
ZEXT
ADDR2MUX
ADDR2MUX
ADDR1MUX
ADDR1MUX
SEXT
SEXT
SEXT
SEXT
[10:0]
[8:0]
[5:0]
[4:0]
[7:0]
FINITE STATE MACHINE
SR2MUX
N Z P
LD.CC
ALU
IR
LD.IR
LOGIC
R
gateALU
GateMDR
MDR
LD.MDR
MEMORY
MEM.EN
MEM.WE
MAR
LD.MAR
INPUT
OUTPUT

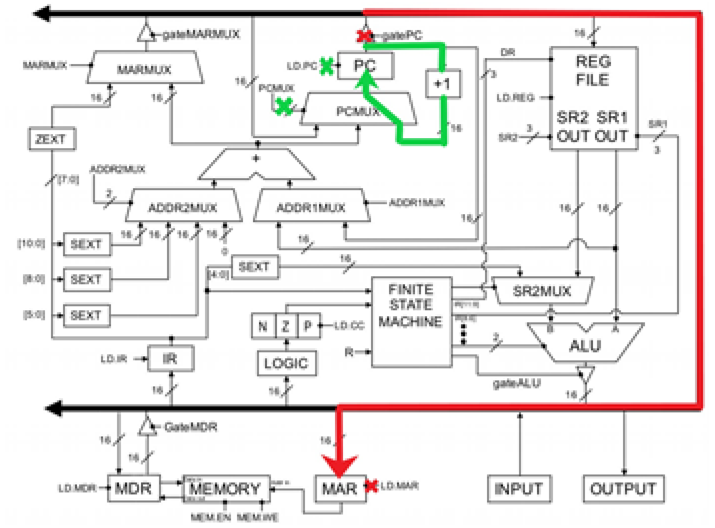
Fetch: Part 2


gateMARMUX
gatePC
MARMUX
MARMUX
LD.PC
PC
16
PCMUX
2
PCMUX
+1
16
DR
3
REG
FILE
LD.REG
SR2 SR1
OUT OUT
SR2
3
SR1
3
ZEXT
[7:0]
ADDR2MUX
2
ADDR2MUX
ADDR1MUX
ADDR1MUX
16
16
16
16
16
0
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
SEXT
[4:0]
FINITE
STATE
MACHINE
SR2MUX
IR[11:9]
IR[8:6]
N
Z
P
LD.CC
B
A
ALU
2
R
LOGIC
LD.IR
IR
gateALU
GateMDR
LD.MDR
MDR
MEMORY
Data in
Addr in
MAR
LD.MAR
MEM.EN
MEM.WE
INPUT
OUTPUT

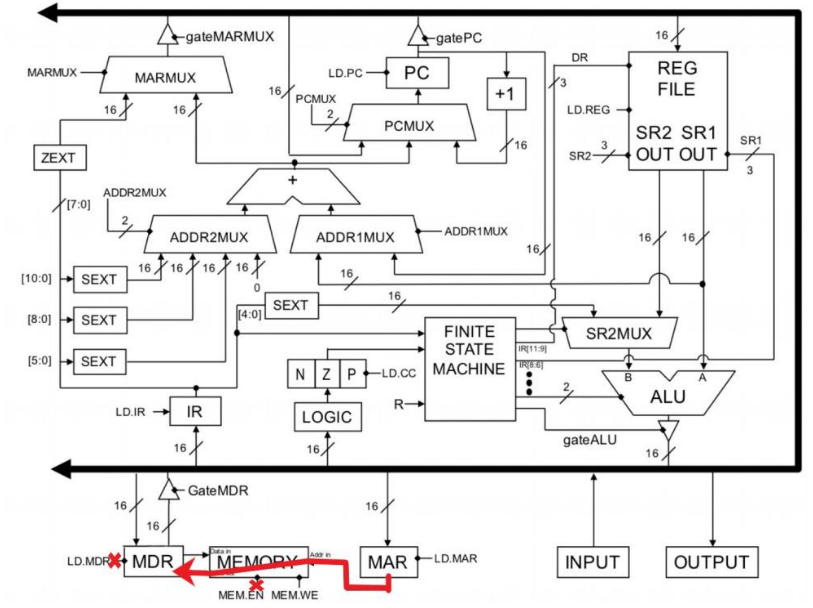
Fetch: Part 3


gateMARMUX
MARMUX
MARMUX
gatePC
LD.PC
PC
+1
PCMUX
PCMUX
REG FILE
DR
LD.REG
SR2 SR1 OUT OUT
SR2
SR1
ZEXT
ADDR2MUX
ADDR2MUX
ADDR1MUX
ADDR1MUX
SEXT
[10:0]
[8:0]
[5:0]
[4:0]
[7:0]
FINITE STATE MACHINE
SR2MUX
N Z P
LD.CC
LOGIC
R
ALU
gateALU
LD.IR
IR
GateMDR
LD.MDR
MDR
MEMORY
MAR
LD.MAR
MEM.EN
MEM.WE
INPUT
OUTPUT

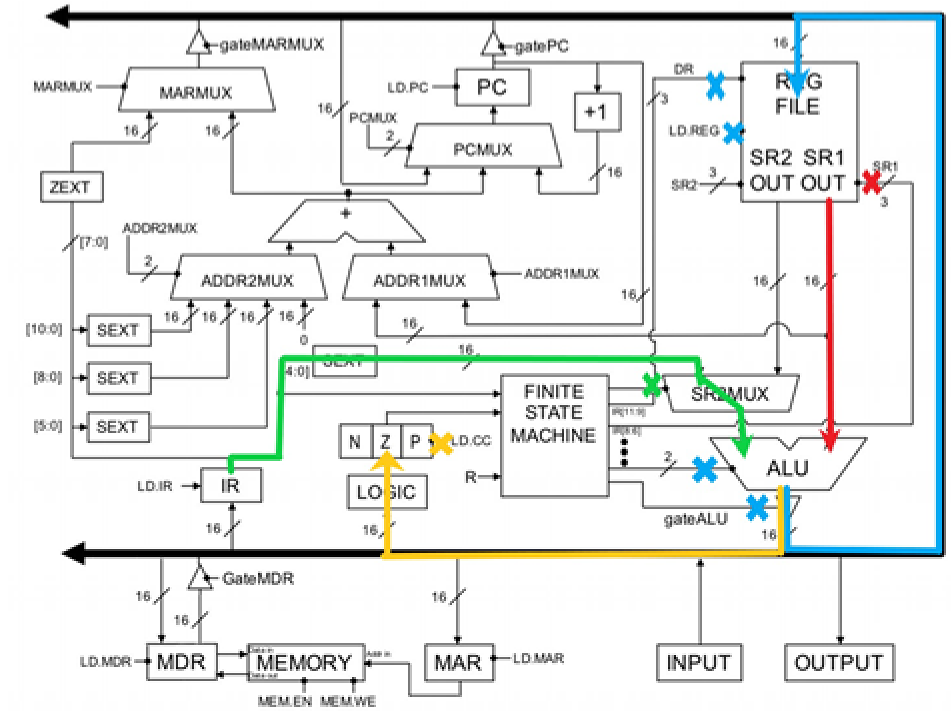
What needs to be done in the first microstate of the FETCH macrostate?



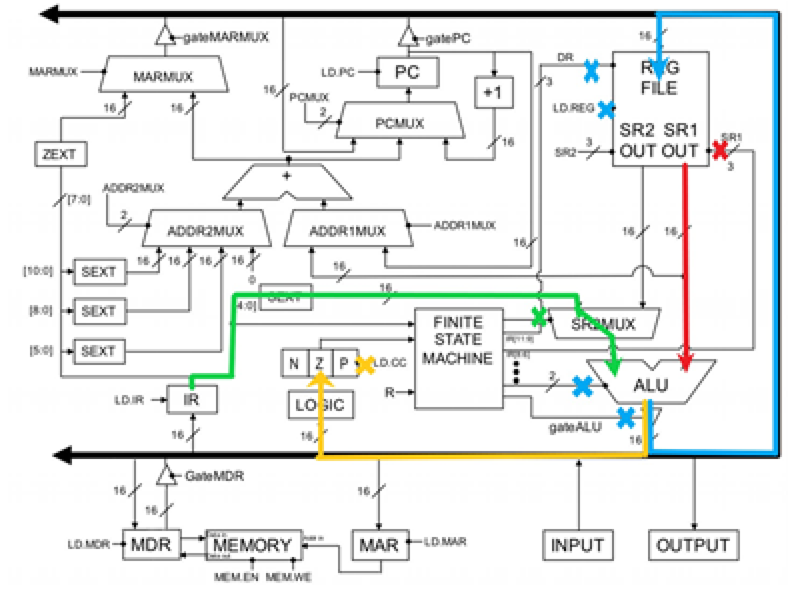
What is MEM.EN used for?



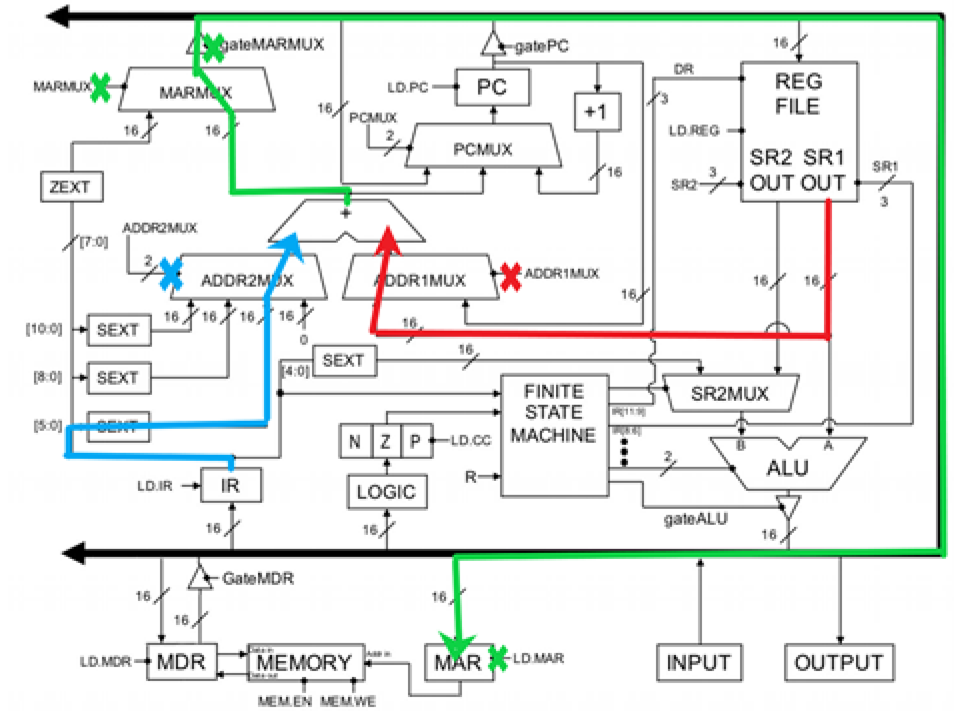
Execute: immediate ADD



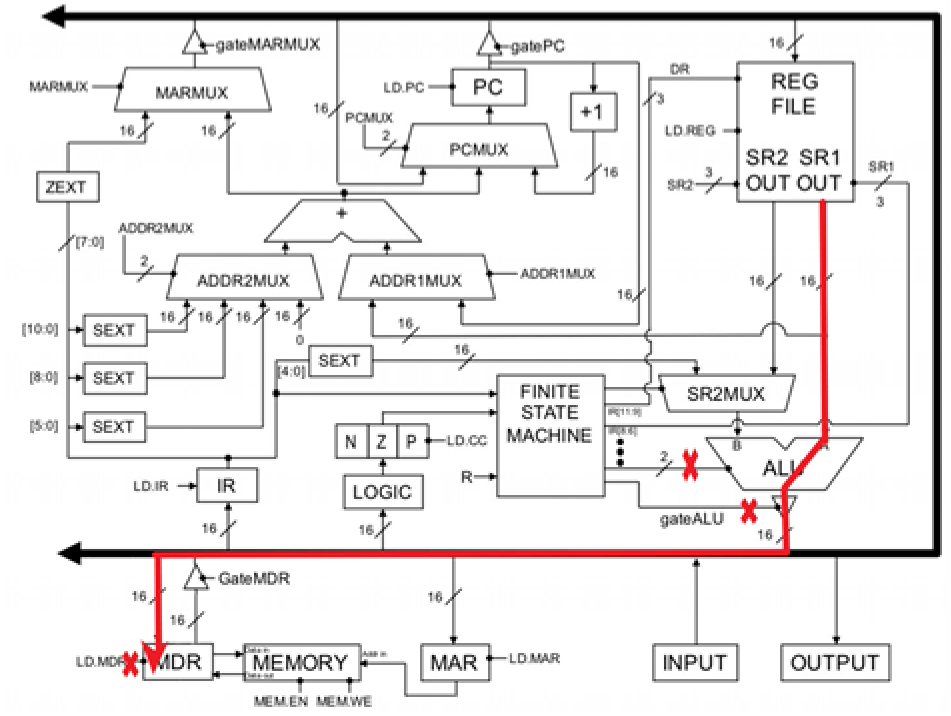
Why do we need the SR2MUX?



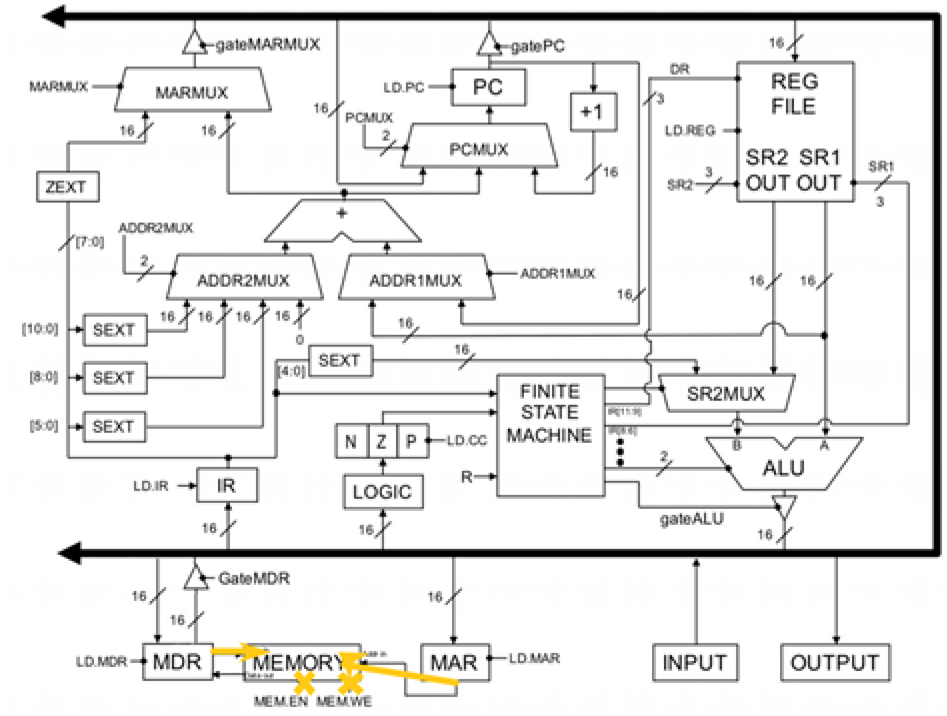
Execute: STR (Part 1)



Execute: STR (Part 2)



Execute: STR (Part 3)


gateMARMUX
MARMUX
MARMUX
gatePC
LD.PC
PC
+1
PCMUX
PCMUX
REG FILE
DR
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR2MUX
ADDR1MUX
ADDR1MUX
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
SEXT
[4:0]
FINITE STATE MACHINE
SR2MUX
N
Z
P
LD.CC
R
LOGIC
ALU
LD.IR
IR
gateALU
GateMDR
LD.MDR
MDR
MEMORY
MAR
LD.MAR
INPUT
OUTPUT
MEM.EN
MEM.WE