CS 2110 - Lab 6

LC-3 Datapath & Tracing

Wednesday, June 8, 2022

Lab Assignment: LC3 Quiz!

1) Go to Quizzes on Canvas
2) Select **Lab 06**, password: **Fetch**
3) Get a 100% to get attendance!
    a) Unlimited attempts
    b) Collaboration is **allowed**!
    c) Ask your TAs for help :)

Homework 3

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

Homework 4

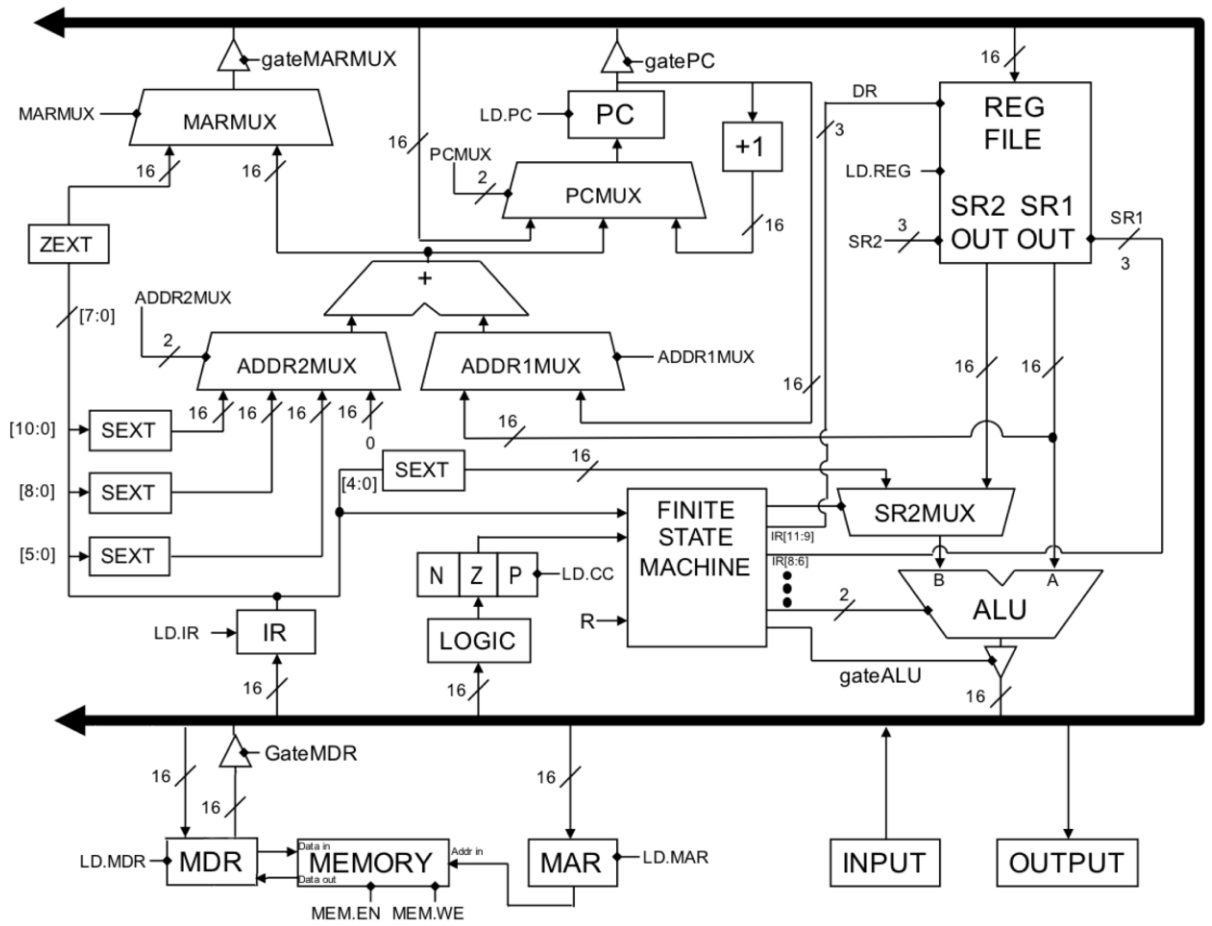
- Released on Friday, June 10th
- **Due Monday, June 20th at 11:59 PM**
- Files available on Canvas
- Submit on Gradescope (unlimited submissions)
- Will be demoed (logistics announced soon)

Quiz 2

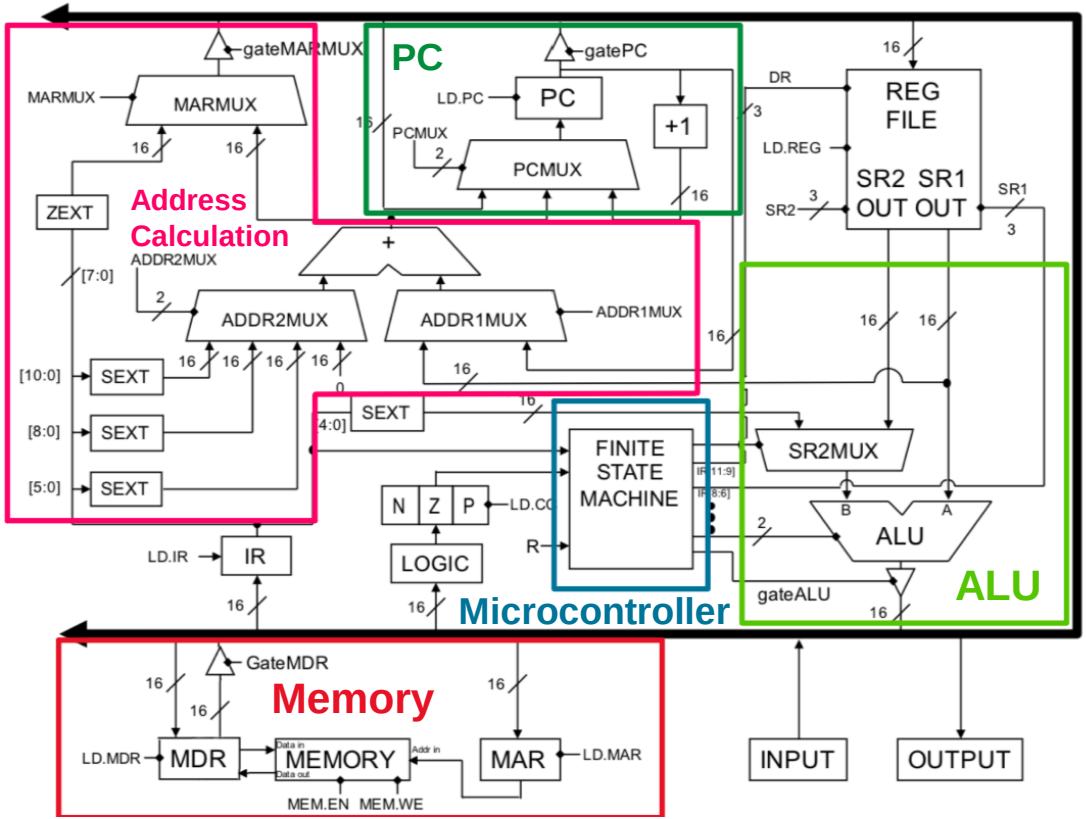
- **Next Wednesday, June 15th**
- Quiz opens at your assigned lab time (unless you have already established a different time with your professor)
- Full 75 minutes to take the quiz!
- After the quiz, you will join lab for the remaining period.
- A topic list will be posted on Canvas

Timed Lab 2

- **Scheduled on Wednesday June 22nd**
- 75 minutes to complete at the beginning of lab, then lab will be resumed.
- Will cover HW4 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.

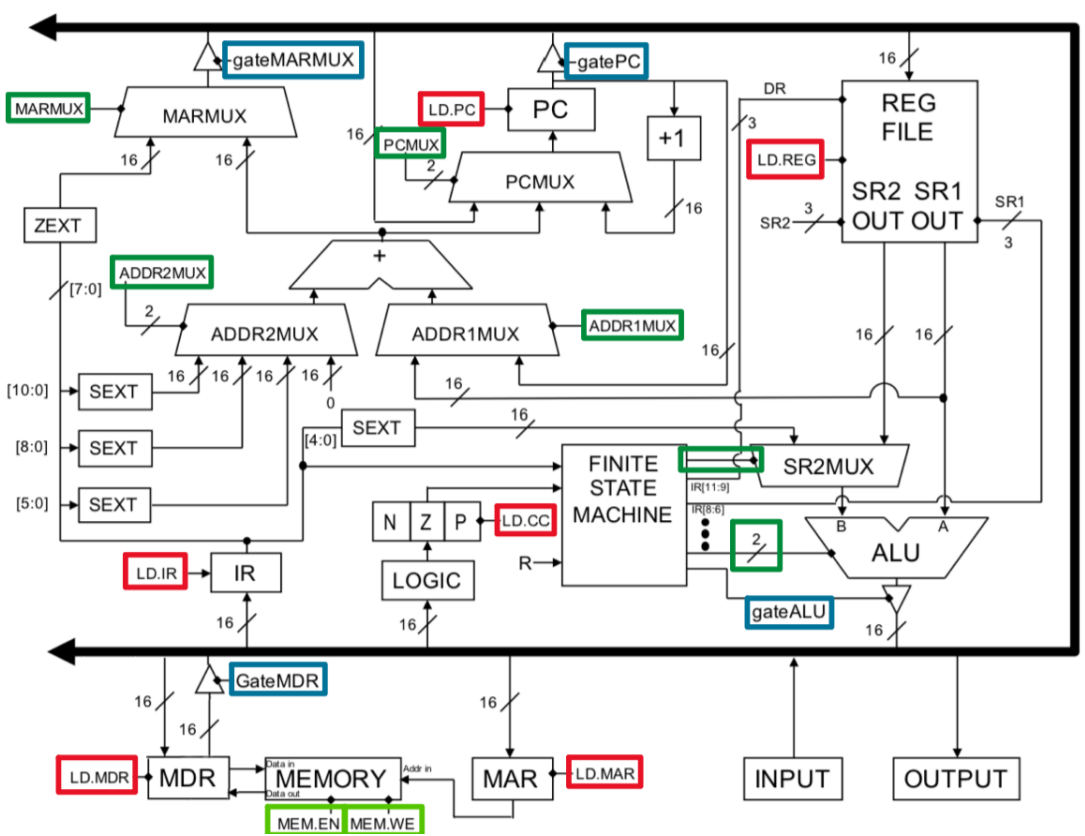

gateMARMUX
MARMUX
MARMUX
gatePC
LD.PC
PC
+1
PCMUX
PCMUX
16
DR
REG
FILE
LD.REG
SR2 SR1
OUT OUT
SR2
SR1
3
ZEXT
[7:0]
ADDR2MUX
ADDR2MUX
ADDR1MUX
ADDR1MUX
+
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
[4:0]
SEXT
0
FINITE
STATE
MACHINE
SR2MUX
IR[11:9]
IR[8:6]
N
Z
P
LD.CC
LD.IR
IR
LOGIC
R
B
A
ALU
gateALU
GateMDR
LD.MDR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

LC-3 Components



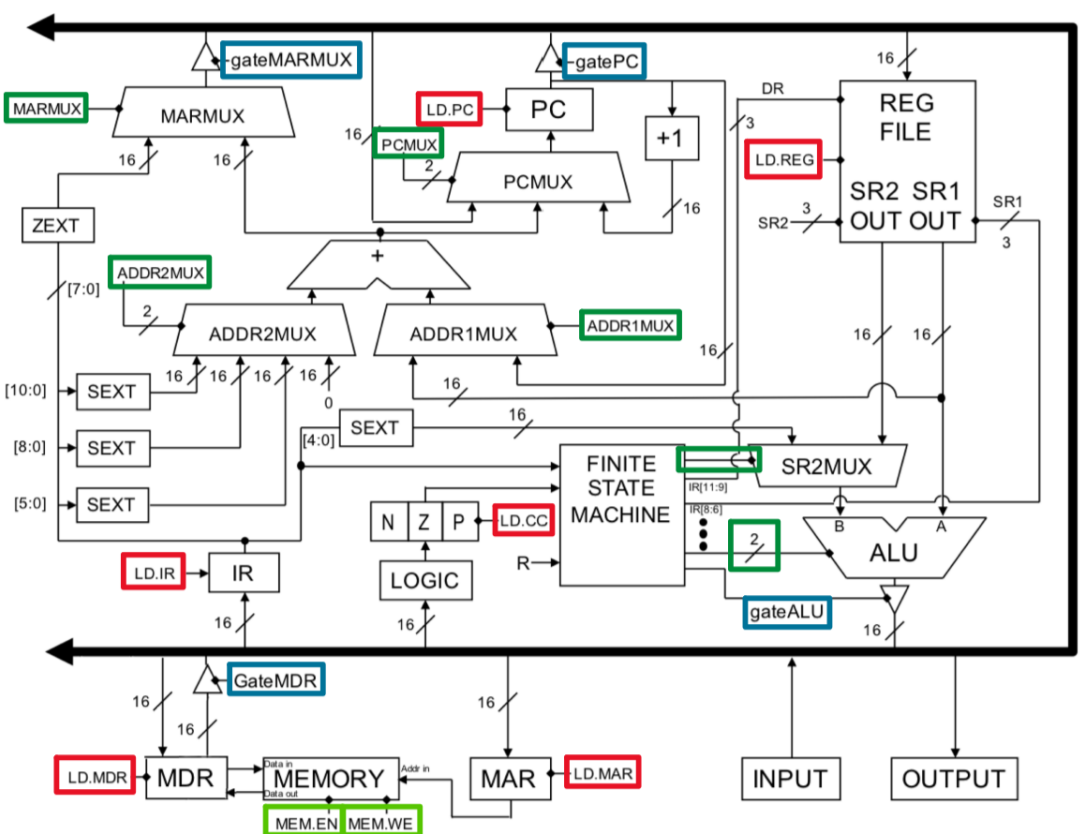
Control Signals

- 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?



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	0b0010000100010000
0x3003	0x2110
...	...
0x300A	#8464

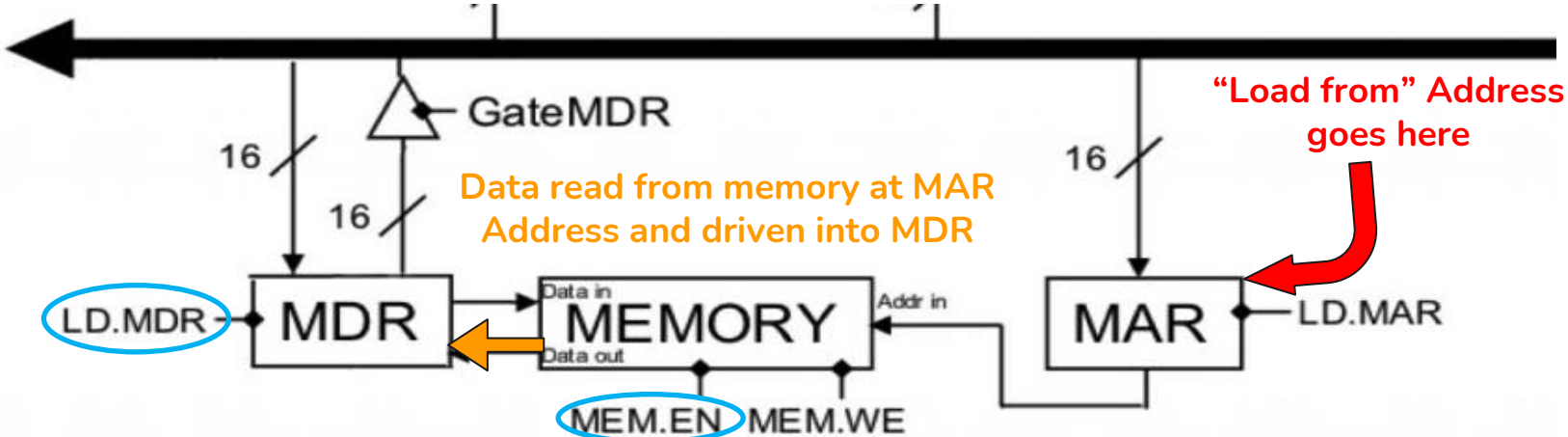
LC-3 Instructions:
Piece of some program in memory.

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

Load data from memory:

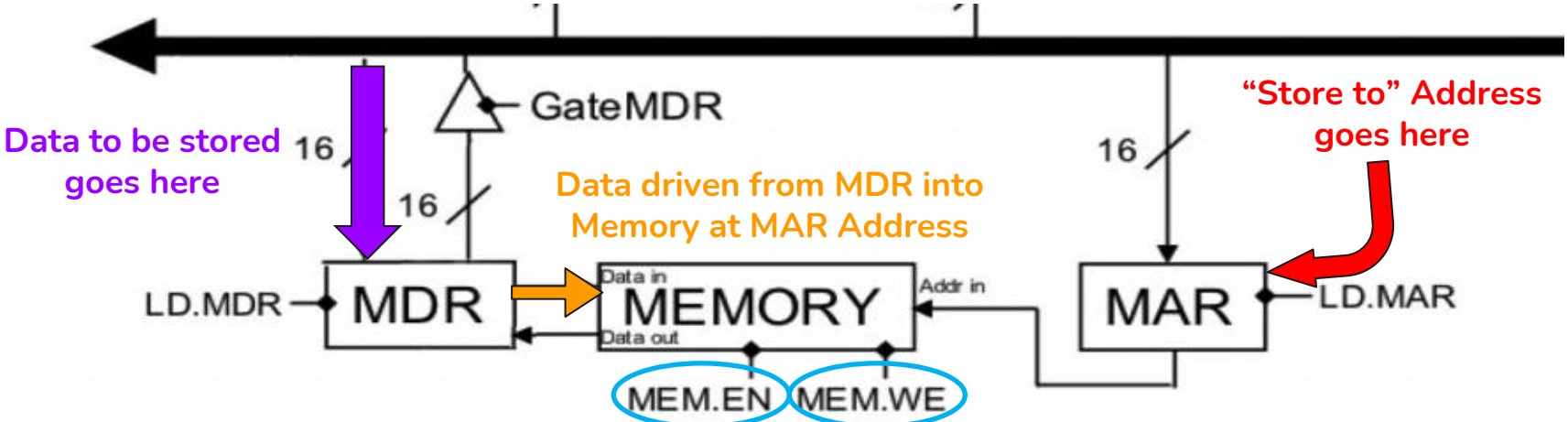
- Place the **address** of where the data is located in **MAR** (**Memory Address Register**).
- Memory is **read** from that address and “driven” to the **MDR** (**Memory Data Register**). After that, the data moves to its destination register.



Storing into Memory

Store data into memory:

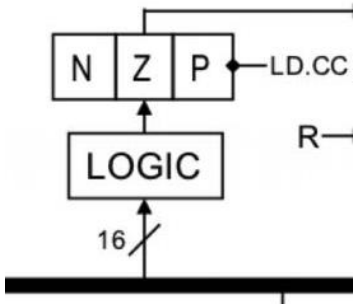
- Data from a register is placed into the **MDR**.
- Address of where to store data is placed into the **MAR**.
- Data is stored in memory at the address specified by the **MAR.**



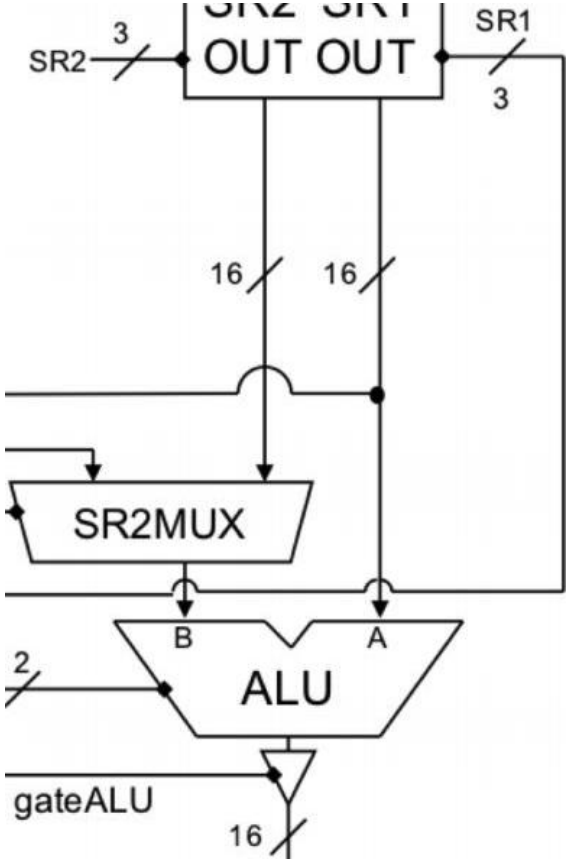
Note

- 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

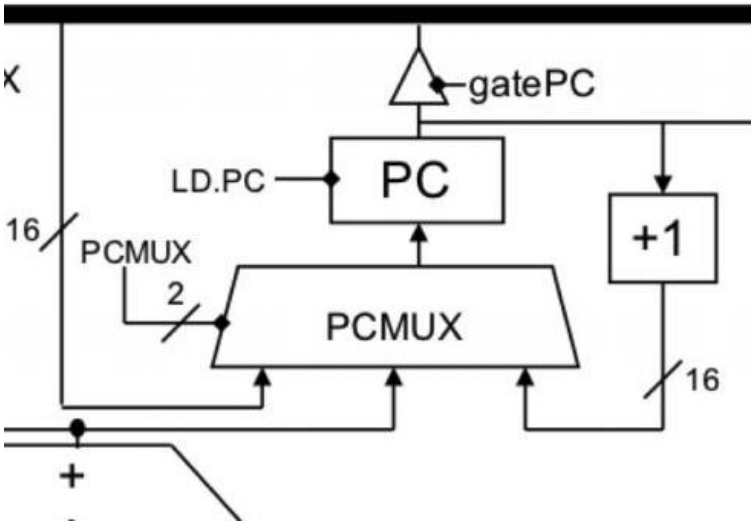


- Two inputs:
  - SR1: Data from register
  - SR2: Data from register **OR** immediate value from the instruction
- Select bits come from Finite State Machine
  - 00 → A + B
  - 01 → A AND B
  - 10 → NOT A
  - 11 → PASS A (unchanged)
- Drives output onto bus
- Condition codes (CC)
  - Set when writing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

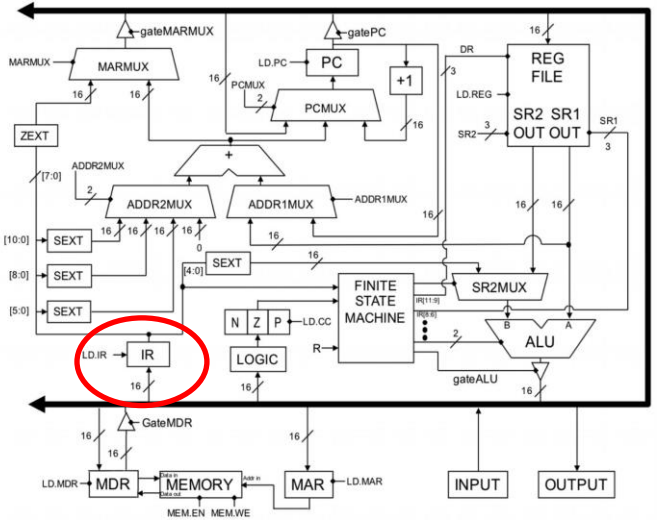


PC

- 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



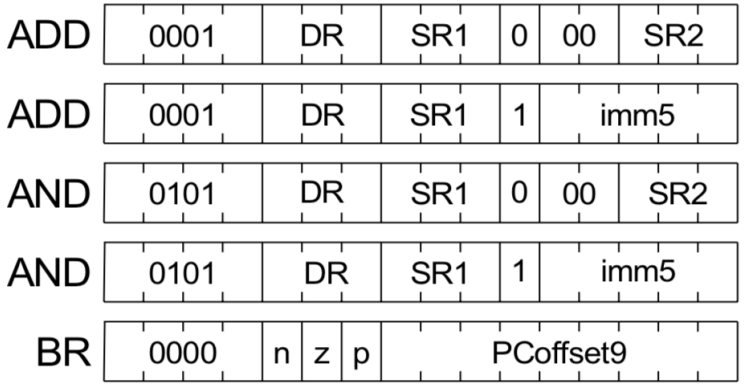
- Instruction register – holds the **value** of the **current** instruction
- For example, the instruction ADD R3, R1, R2 may look like:

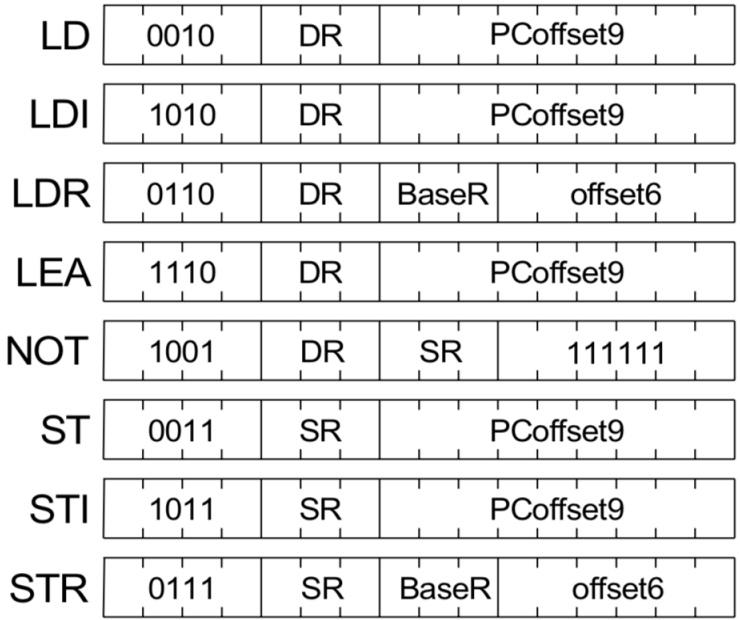
0001 011 001 0 00 010

ADD	0001	DR	SR1	0	00	SR2

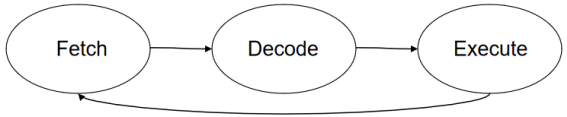
- Having the correct value in the IR helps the FSM determine the current state and send the correct control signals
- A new value can be read from the bus (with signal LD.IR) and the current value is directly connected to relevant components

LC3 Instruction Set





Executing Instructions



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

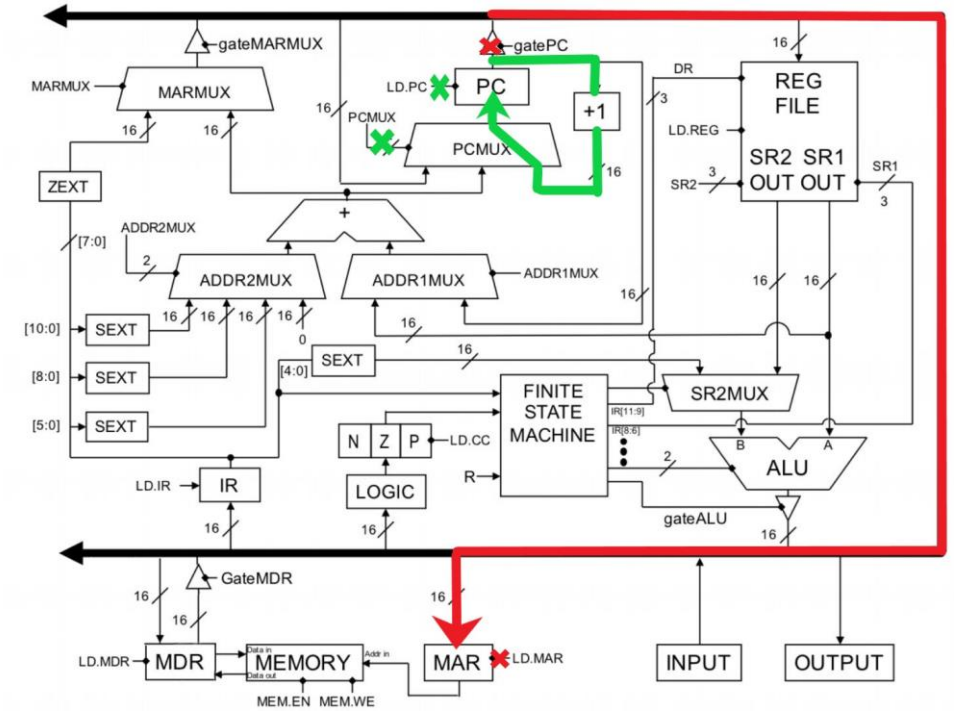
- Each macrostate is made of one or more *microstates* (1 per clock cycle)
- The microstates are the states in the microcontroller's state machine!

The macrostates are:

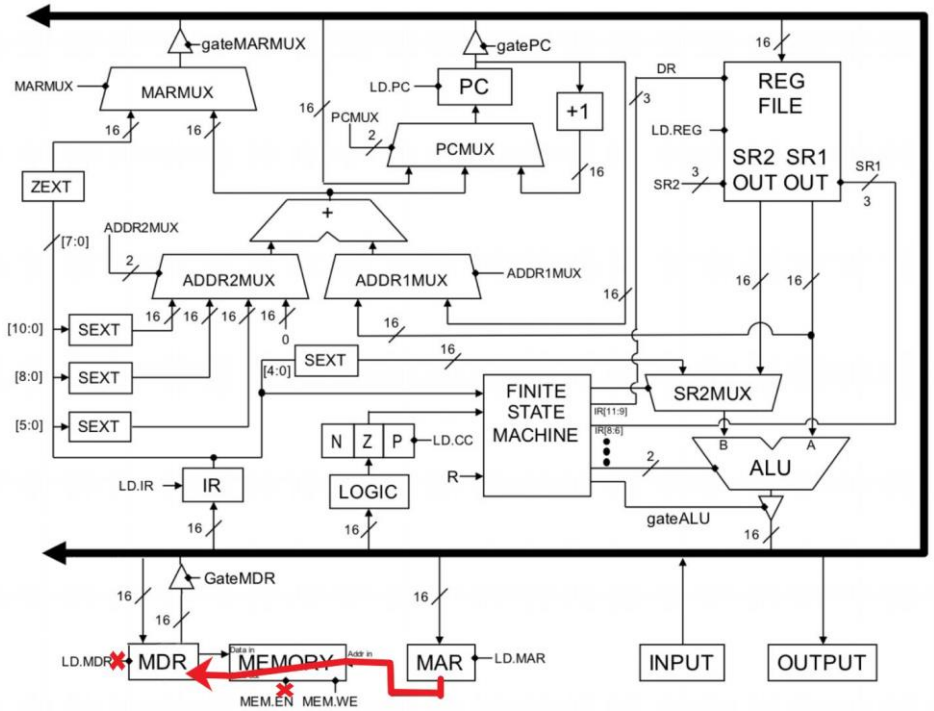
1. Fetch (3 clock cycles)
   - Load the next instruction from memory into the instruction register (IR)
   - Increment the PC
2. Decode (1 clock cycle)
   - The microcontroller looks at the instruction to figure out how to execute it
3. Execute (varies)
   - The microcontroller steps through several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

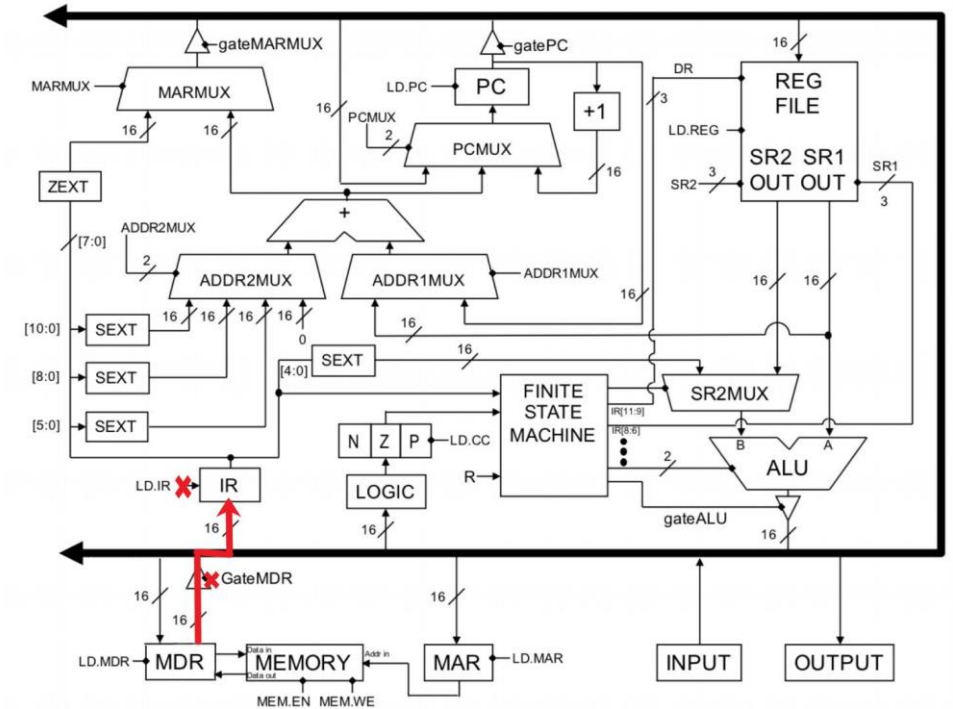
Fetch: Part 1


gateMARMUX
MARMUX
MARMUX
gatePC
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
[4:0]
SEXT
FINITE
STATE
MACHINE
SR2MUX
N
Z
P
LD.CC
LD.IR
IR
LOGIC
R
B
A
ALU
gateALU
GateMDR
LD.MDR
MDR
MEMORY
MAR
LD.MAR
INPUT
OUTPUT
MEM.EN
MEM.WE

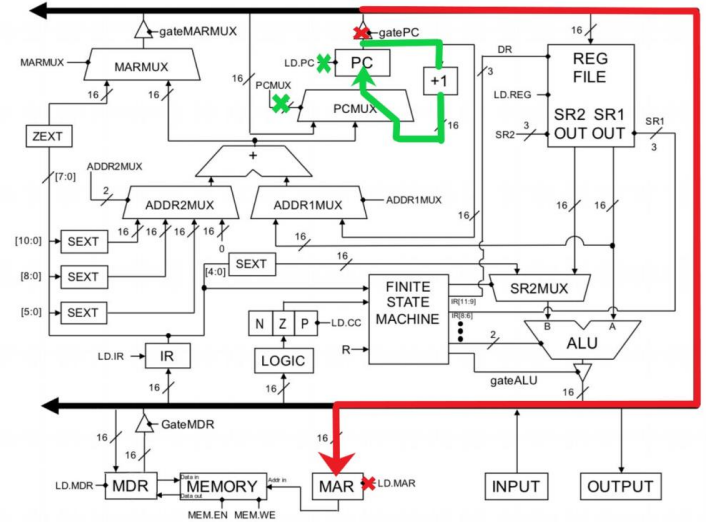
Fetch: Part 2


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

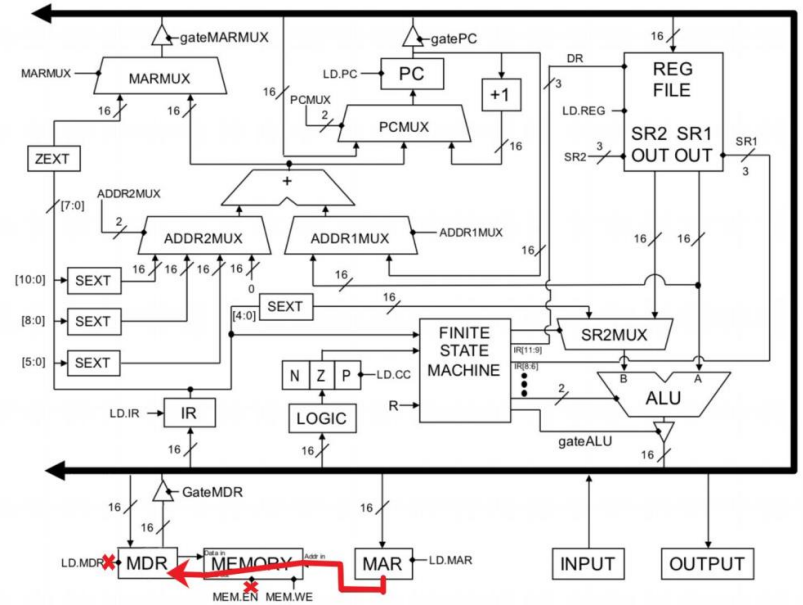
Fetch: Part 3


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

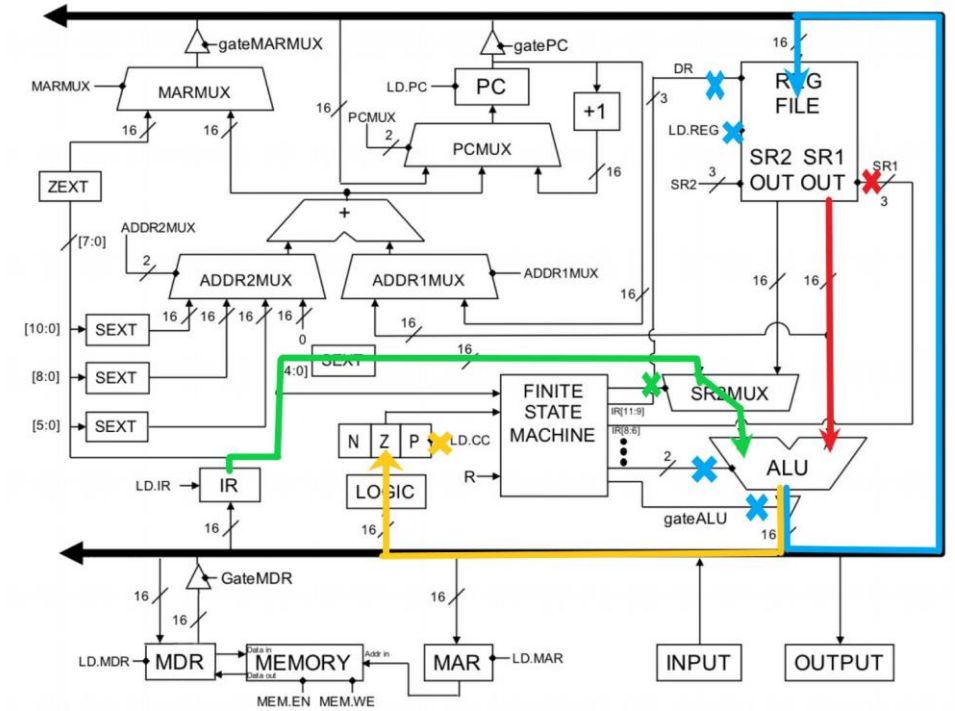
What needs to be done in the first microstate of the FETCH macrostate?



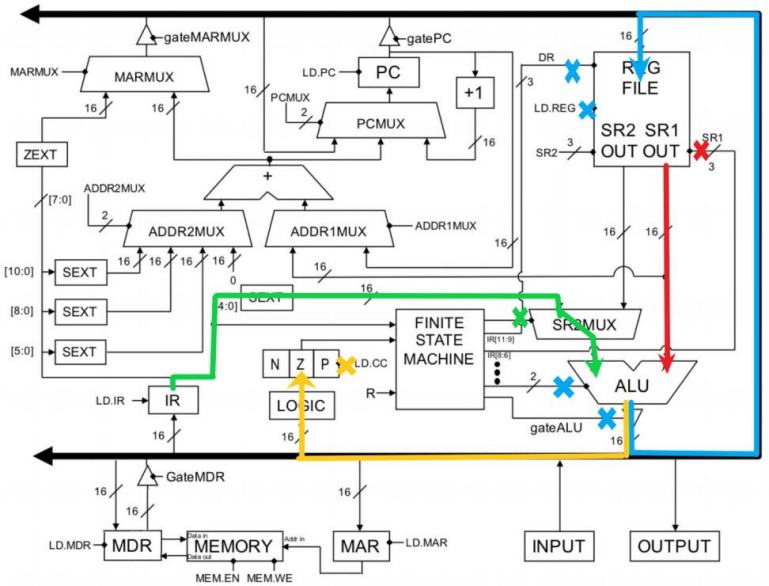
What is MEM.EN used for?



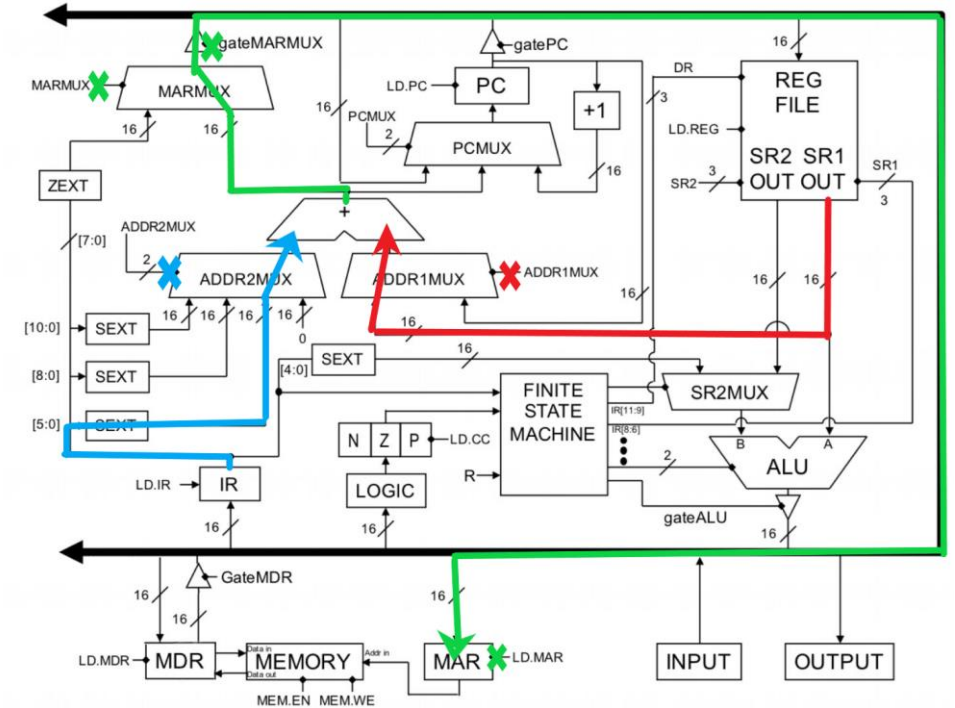
Execute: immediate ADD



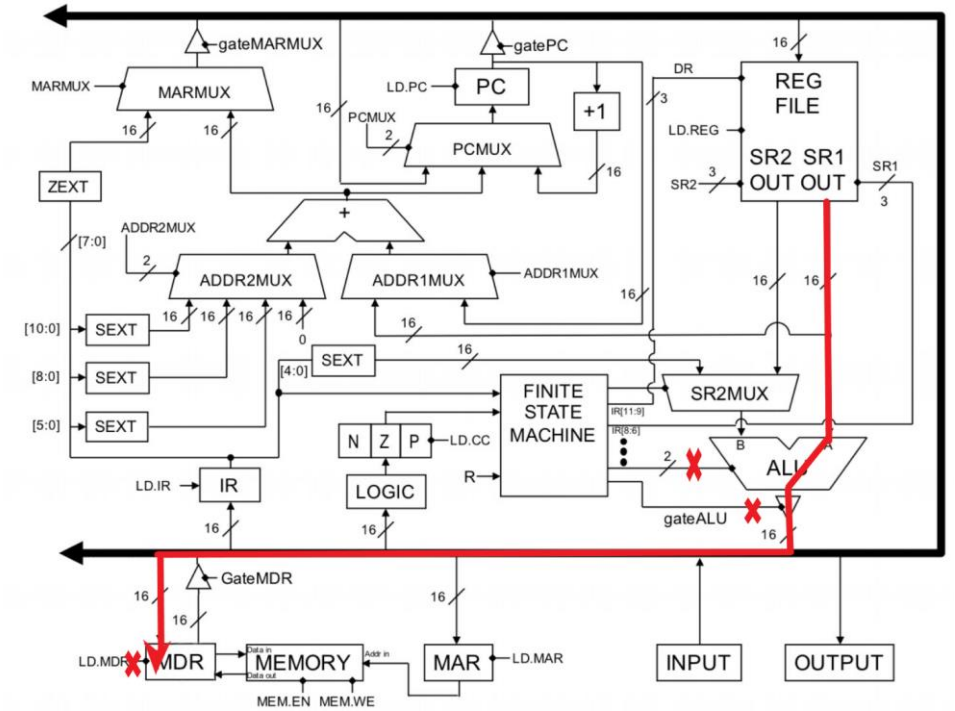
Why do we need the SR2MUX?



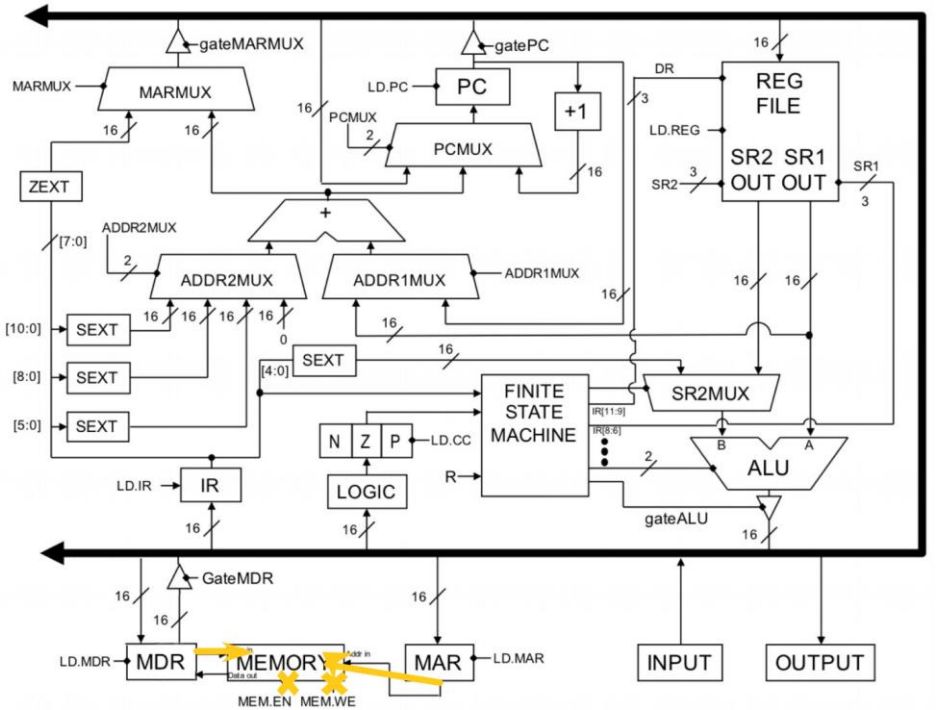
Execute: STR (Part 1)



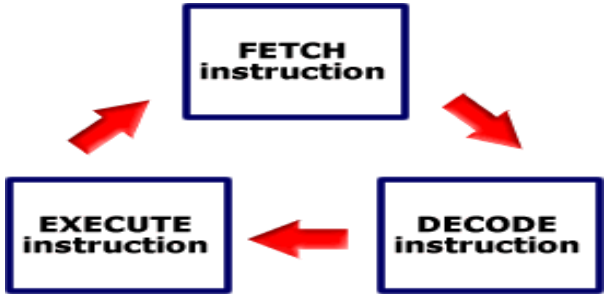
Execute: STR (Part 2)



Execute: STR (Part 3)



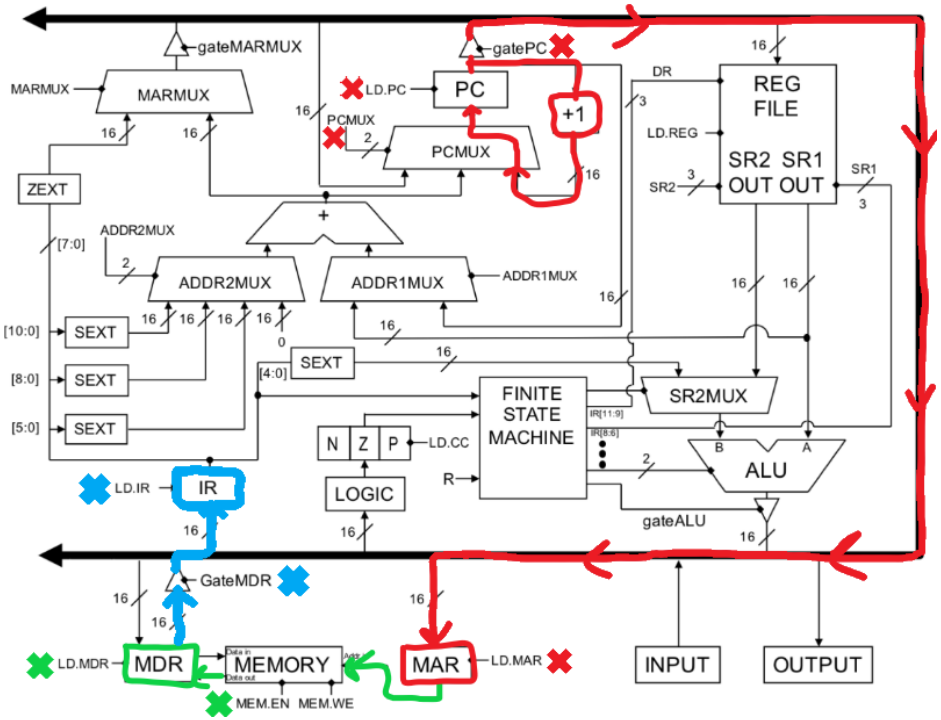
LC-3 Execution Cycle



- The LC-3 cycles between three *macrostates*:
  - **Fetch** (obtaining the instruction at the address stored in the PC)
  - **Decode** (determining which instruction to execute by reading the opcode)
  - **Execute** (doing whatever the instruction says to do)
- The current macrostate is tracked inside the FSM. Some take multiple clock cycles to execute.

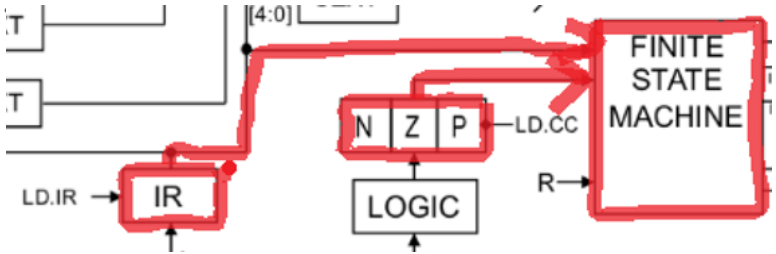
FETCH

- Use PC value (address of next instruction) to read the next instruction from memory
- Load this value into the IR so that it can be decoded and executed later
- Increment the value of PC – it must always point to the **next** instruction to be run



DECODE

- Now that the IR contains the instruction, the FSM can read the opcode.
- If the instruction is BR (branch), the FSM will also check the condition codes to ensure that the branching condition is met.
- The FSM will change to a specific state depending on the opcode. This takes a clock cycle in the LC-3, but some computers can combine DECODE with the last cycle of FETCH.



EXECUTE

- Once the instruction is decoded, the EXECUTE macrostate begins.
- Each instruction causes the EXECUTE phase to act differently.
- The EXECUTE macrostate takes a variable number of clock cycles; some instructions like ADD are quick, while others like LDI take longer.
- Once the EXECUTE macrostate completes, the FETCH macrostate restarts, using a new PC address—generally, this is the next instruction in memory
- You will be implementing the EXECUTE macrostate for many instructions (as well as the FETCH macrostate) in HW4

LC3 Instruction Set