Memory & LC4 Start

Introduction to Computer Systems, Fall 2022

Instructor: Travis McGaha

TAs:

Ali Krema	Andrew Rigas	Anisha Bhatia
Audrey Yang	Craig Lee	Daniel Duan
David LuoZhang	Eddy Yang	Ernest Ng
Heyi Liu	Janavi Chadha	Jason Hom
Katherine Wang	Kyrie Dowling	Mohamed Abaker
Noam Elul	Patricia Agnes	Patrick Kehinde Jr.
Ria Sharma	Sarah Luthra	Sofia Mouchtaris

How is HW2 going?

Logistics

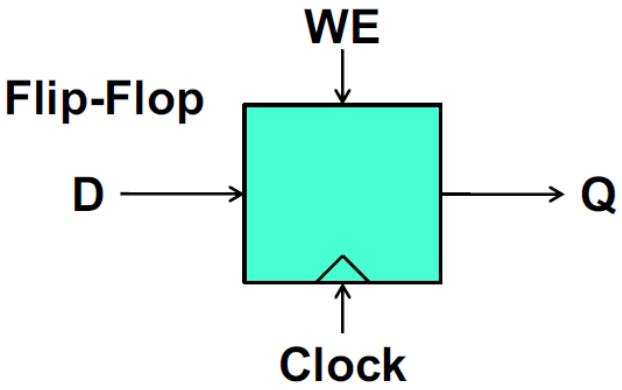
- HW02 Combinational Logic: **This Friday** 9/30 @ 11:59 pm
  - Written Homework, submitted to gradescope
  - **NO EXTENSIONS OVER 72 HOURS**
  - Should have everything you need
  - Practice in Recitations this week
  - **Please read the Clarifications and FAQ Post on ED**
- HW03 Combinational Logic: to be released this week
  - Written Homework, submitted to gradescope
  - **NO EXTENSIONS OVER 72 HOURS**

Lecture Outline

- **D Flip Flops & Registers**
- Memory at a high level
- Memory using flip flops
- Memory Hierarchy
- LC4 (start)

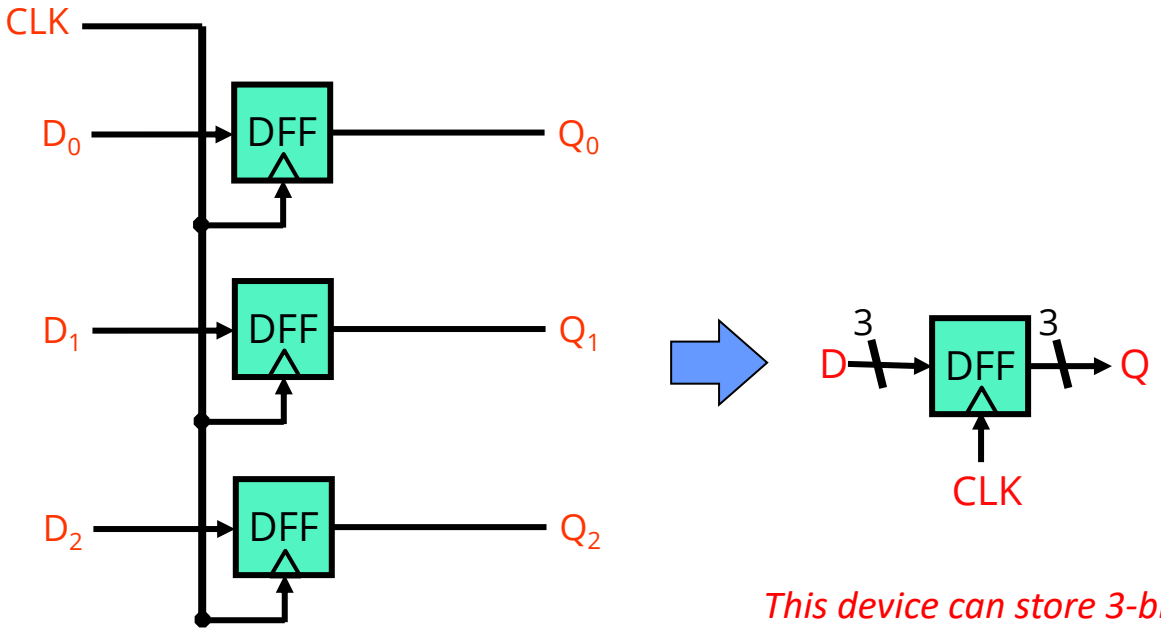
Flip Flops Summary

- We can abstract away the details of a Flip Flop as a 1-bit **storage container**.
  - Takes in an input D
  - Has an output or stored value "Q"
  - Takes a clock input (often represented with a triangle)
  - Usually has a WE to control if it will update on the next rising edge
  - A set of D flip flops can be grouped together to form a register (storage for a multiple-bit value, more on this next lecture)



Register Made of Flip Flops

- A collection of D Flip-Flops, controlled by a common CLK signal can be called a register
  - A register is a fixed size multi-bit fast storage location used by a Processor. (More on this over the next few weeks)
  - (WE not shown, but assume that WE is connected to each DFF)



This device can store 3-bits!

Lecture Outline

- D Flip Flops & Registers
- **Memory at a high level**
- Memory using flip flops
- Memory Hierarchy
- LC4 (start)

Memory as an array

- Memory is like a huge array…
  - Stores almost all the information needed to run a program (Code, variables, strings, …)
  - In a 64-bit machine, this array has **18,446,744,073,709,551,616** indexes
  - An index corresponds to a location in memory that contains data
    - (An index in this context is called an **address**)
  - A location in memory is of fixed size bits
    - Usually 8-bits, but 16-bits for LC4 in this class
  - These addresses could be storing variables

```
int a = 0;
int b = 9;
```

Index # *(Address)*	Information *(Data)*
0	0xFFEC
1	0x000A
2	0xFFF1
3	0x0008
4	0xFFFF
5	0x0000
6	0x0102
7	0x0000
8	0xFF32
9	0x2400
10	0x0000
11	0x0009
12	0xF308

Memory as an array

- Address Space: The range of possible addresses. Usually, some power of 2.
  - In LC4, we have 2^{16} (65,536) possible addresses
    that range from 0 - 65535
- Addressability: number of bits per location
  - 16-bits for LC4, usually 8-bits on modern computers
  - "16-bit addressability" for LC4

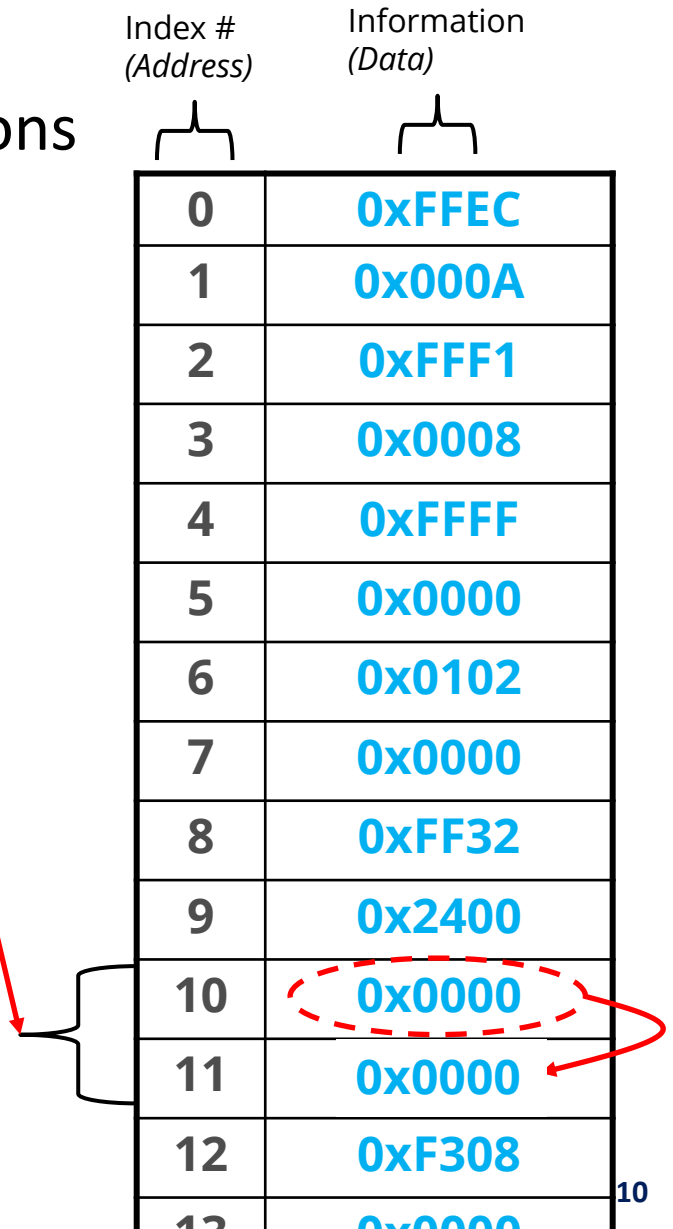
Index # *(Address)*	Information *(Data)*
0	0xFFEC
1	0x000A
2	0xFFF1
3	0x0008
4	0xFFFF
5	0x0000
6	0x0102
7	0x0000
8	0xFF32
9	0x2400
10	0x0000
11	0x0009
12	0xF308

Basic Memory Usage

- There are two basic memory operations
  - Selecting a location to **read** from
  - Selecting a location to **write** to
- Consider our example from before

```
int a = 0;
int b = 9;
```

- What if we did `b = a;`
- Done in two steps:
  - Read the value stored in a
  - Store the value in b



Poll Everywhere

pollev.com/tqm

- If we wanted to use memory that contains 128 different locations, how many bits do we need at minimum to represent an address?

A. 5

B. 7

C. 8

D. 6

E. I'm not sure

Poll Everywhere

pollev.com/tqm

- If we wanted to use memory that contains 128 different locations, how many bits do we need at minimum to represent an address?

A. 5

B. 7

C. 8

D. 6

E. I'm not sure

7 bits is = 2^7 different values

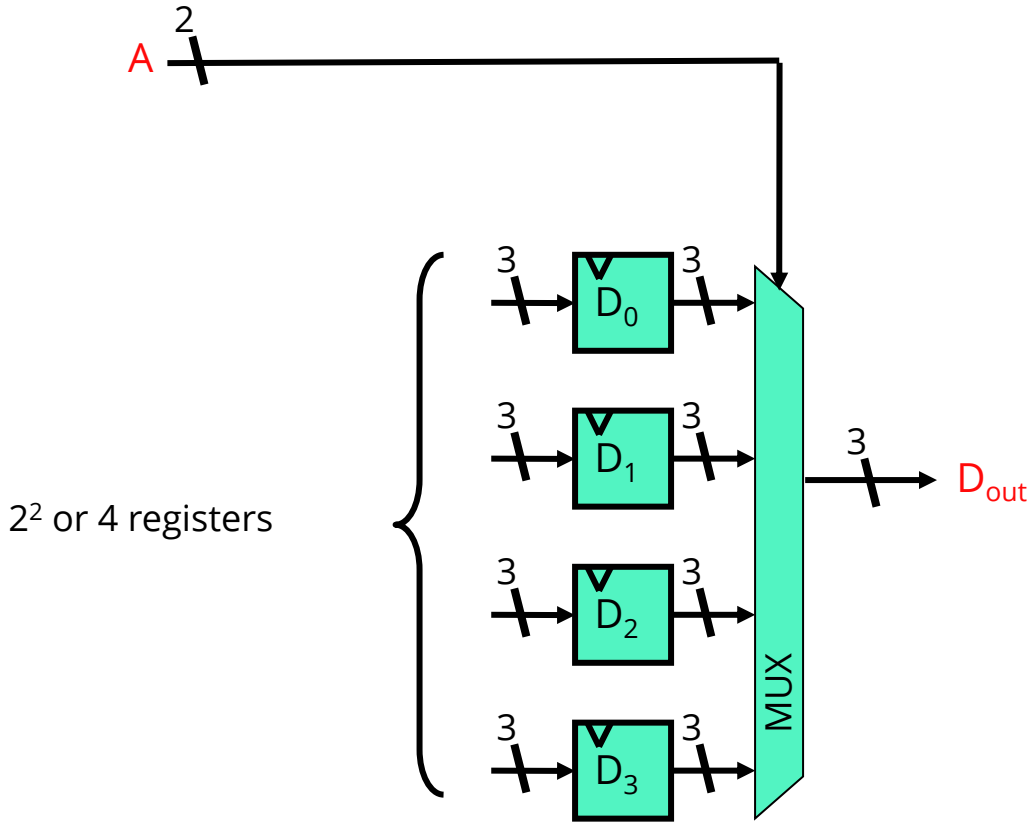
2^7 is 128

Lecture Outline

- D Flip Flops & Registers
- Memory at a high level
- **Memory using flip flops**
- Memory Hierarchy
- LC4 (start)

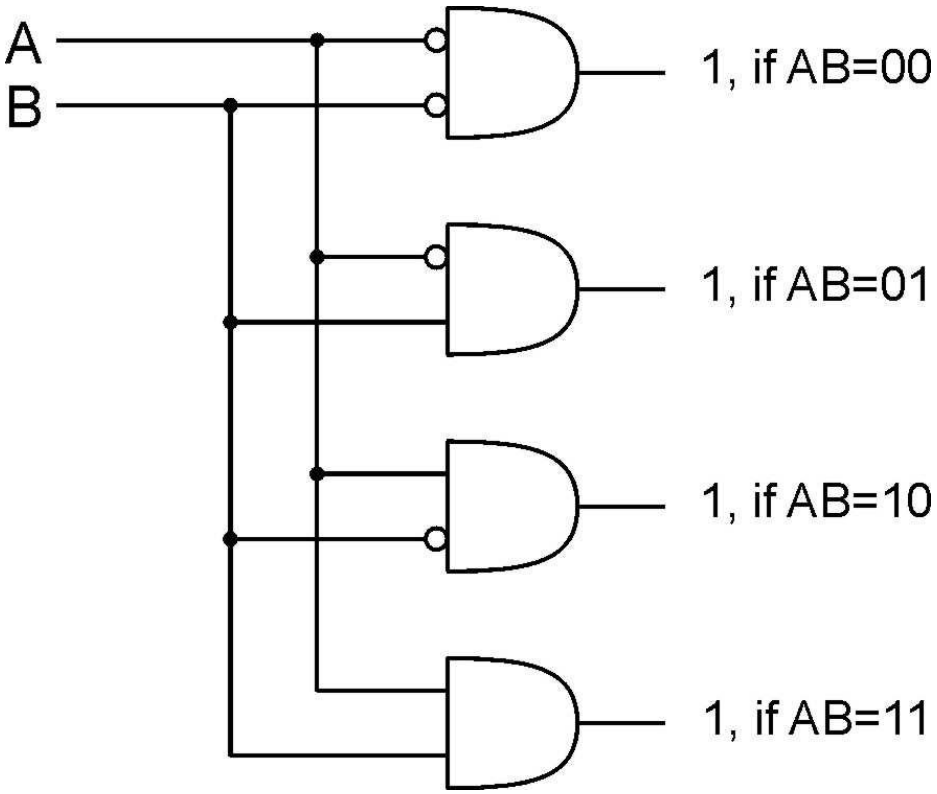
Let's build a simple 2^2 by 3-bit memory

- We can implement memory as a collection of registers
- Read operation:



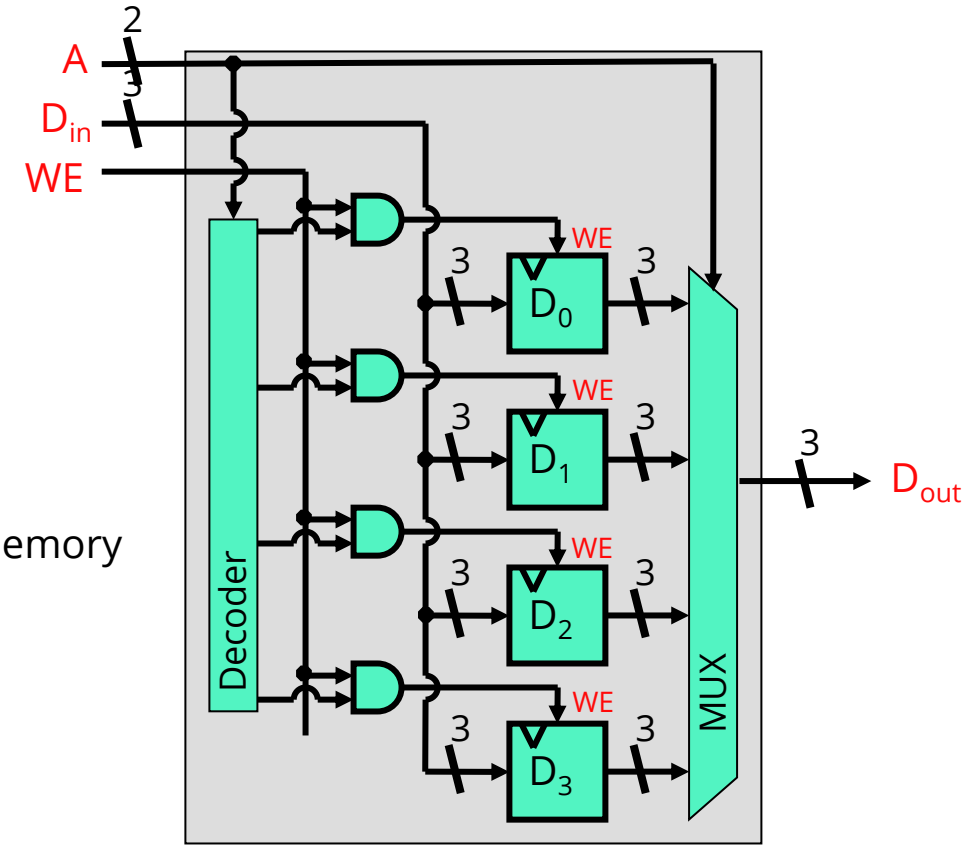
Aside: Decoder

- n inputs, 2^n outputs
  - n = 2 for this example inputs are A and B
- A single output will be 1, the rest will be 0
  - Putting in a binary number will have the corresponding output wire "turn on"
- Sort of a "reverse MUX"
  - Instead of 4-to-1 we are sort of doing 1-to-4



2^2 by 3-bit writeable memory

Write operation



Limitation:
You can only read or write at any given time

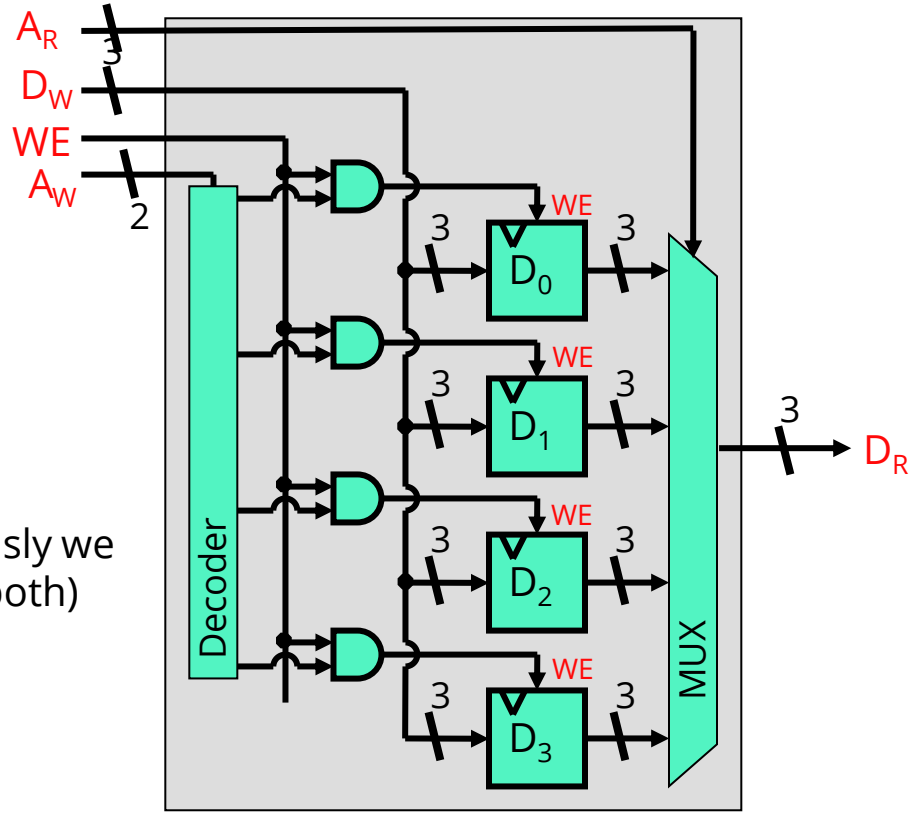
This is called "single port" memory

2^2 by 3-bit memory – independent R/W

- Can have independent read/write operations if we take in separate addresses for each.

You can read from one address and write to another with this arrangement

(notice 1 address line for R
1 address line for W. Previously we used the same address for both)

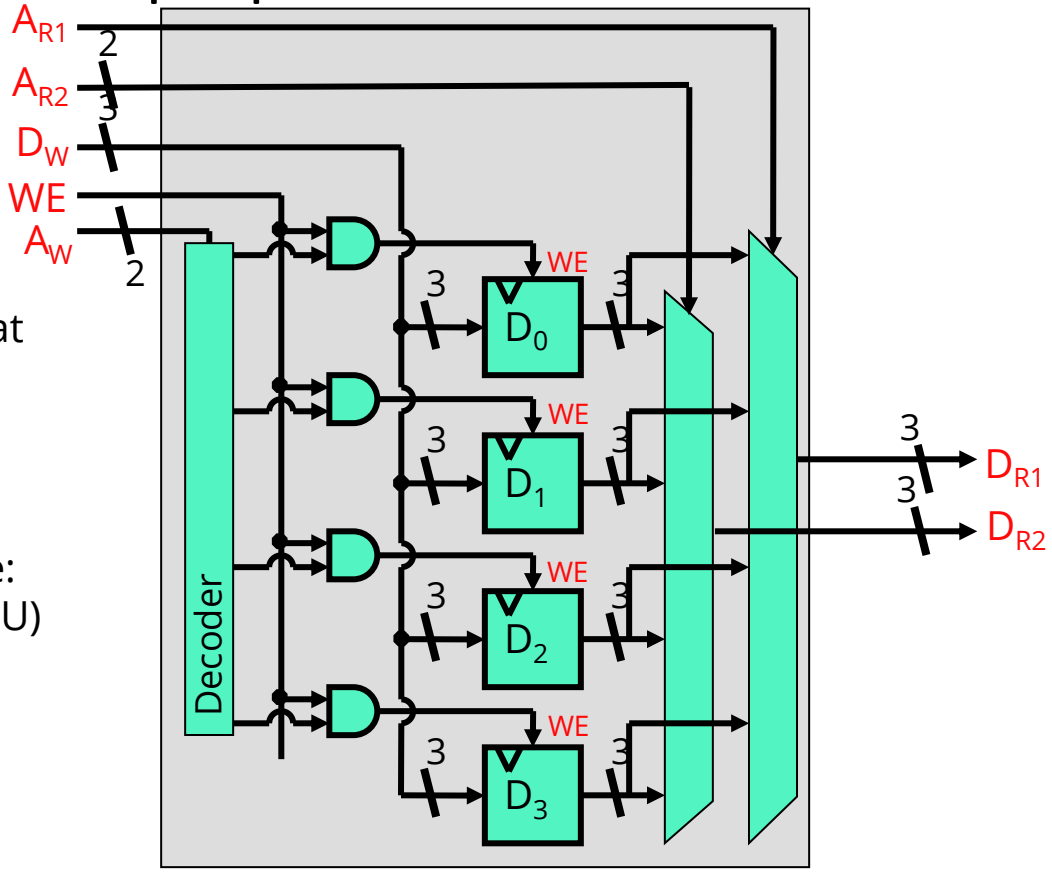


2^2 by 3-bit memory – Multiple Reads

- Can read from multiple places at once!

Read from 2 locations at once, write to a third! (notice 3 address lines)

(We will use this later In something called the: "register file" for the CPU)



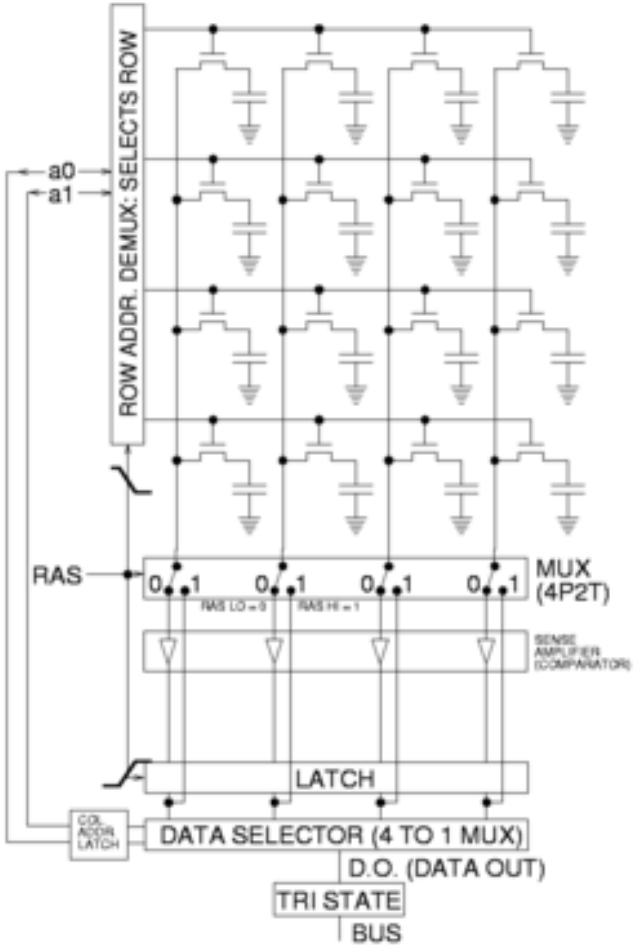
More Memory Details

- The Memory We've Created Would need many transistors, but is a good starting place to understand how it works!
  - Real memory: fewer transistors, denser, relies on analog properties
- The logical structure of all memory is similar
  - Address decoder
  - "Word select line", word write enable
  - "Bit line"
- Two basic kinds of RAM (Random Access Memory)
- Static RAM (SRAM) - 6 transistors per bit
  - We've created a type of SRAM in this presentation, we can do better (6 transistors!)
  - Fast, maintains data as long as power applied
- Dynamic RAM (DRAM) - 1 transistor per bit
  - Denser but slower, relies on "capacitance" to store data, needs constant "refreshing" of data to hold charge on capacitor

Also, non-volatile memories: ROM, PROM, flash, …

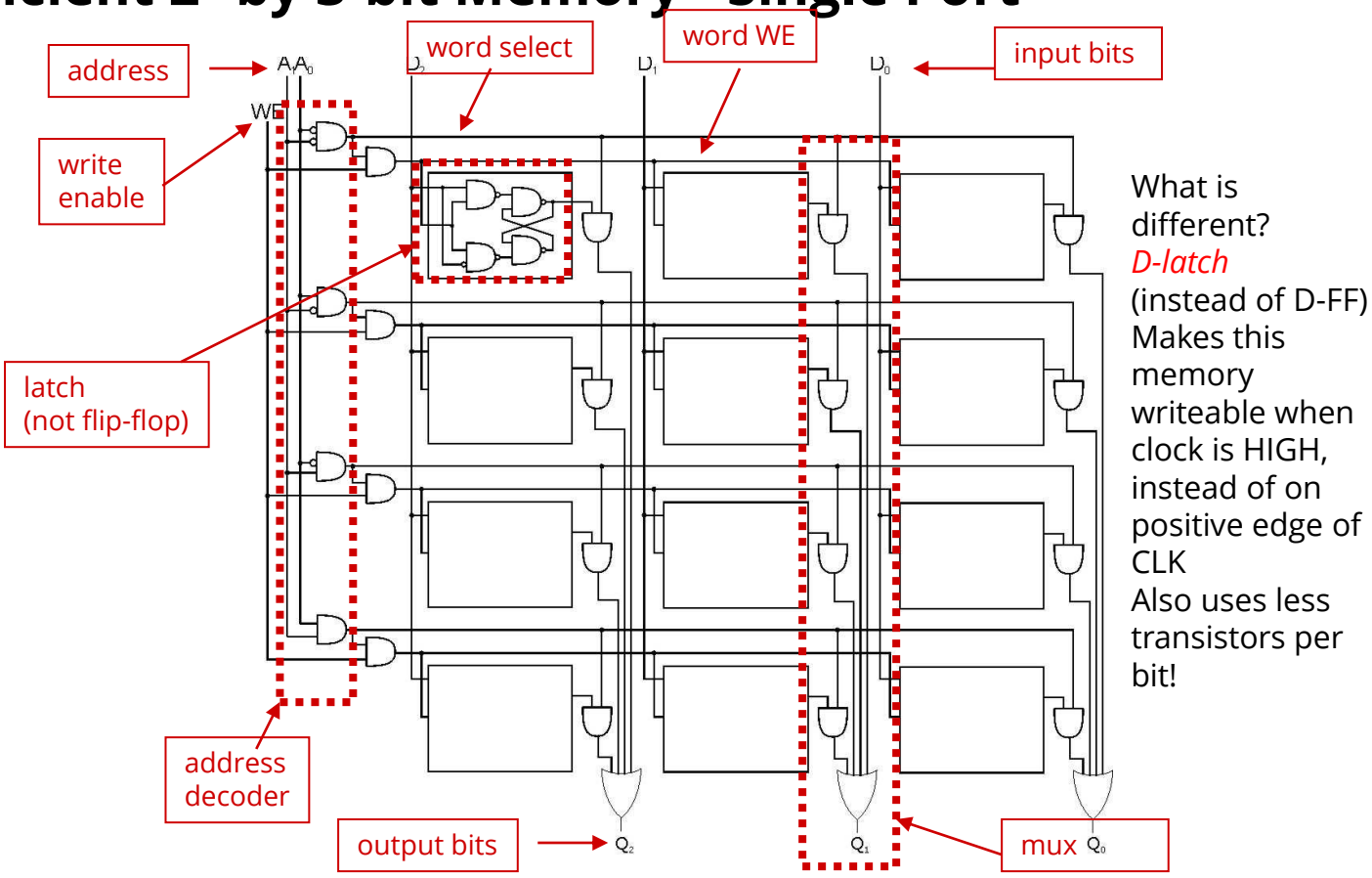
Dynamic RAM

- Information stored as charge on capacitors
- Capacitors *leak* so values have to be 'refreshed' continually
- As memory chips get larger, access times tend to increase. The processor spends more time waiting for data.
  - This is a major issue limiting computer systems performance



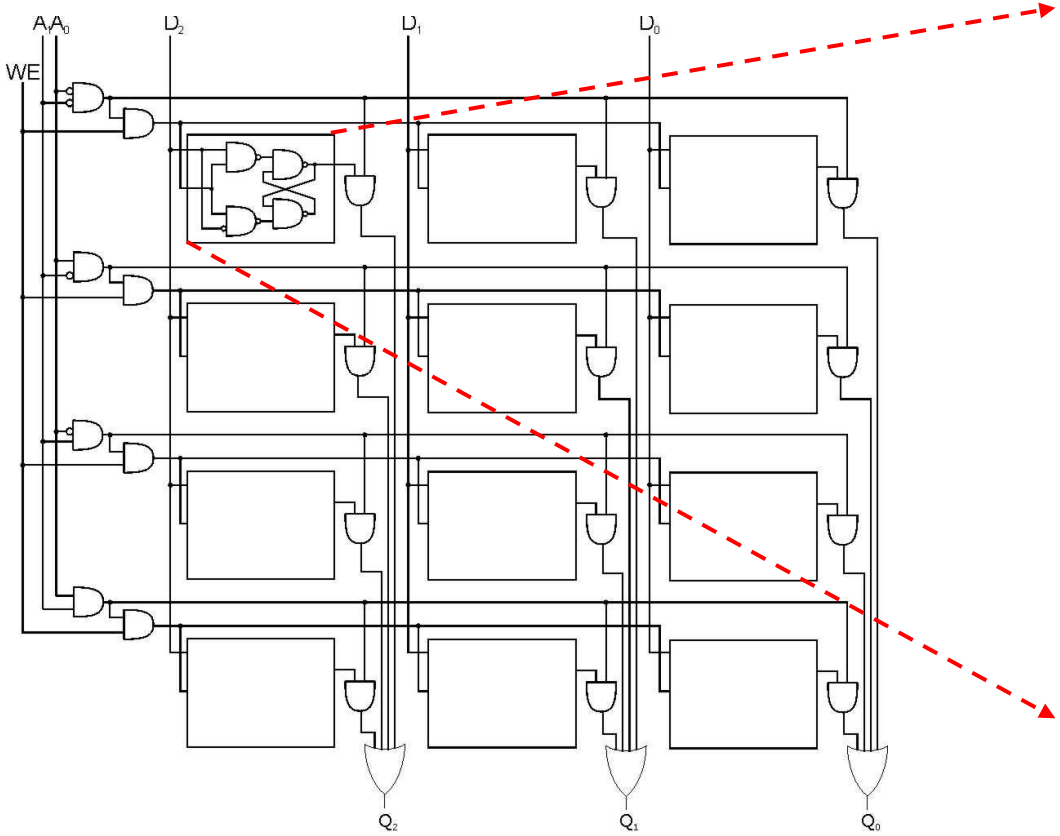
Dynamic RAM

An Efficient 2^2 by 3-bit Memory - Single Port

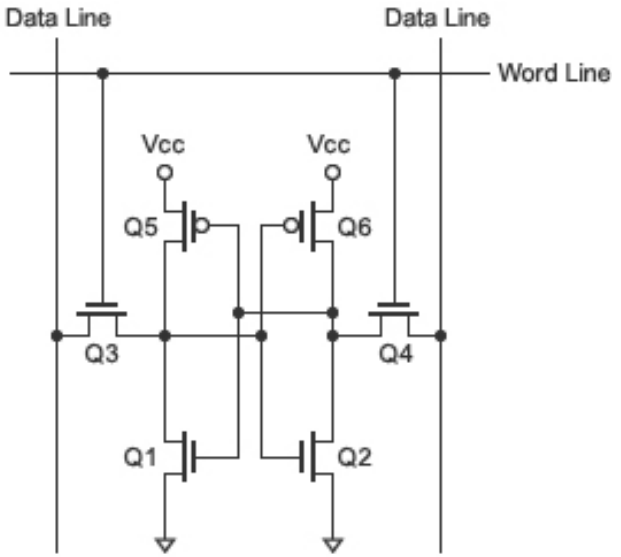


What is different?
D-latch (instead of D-FF)
Makes this memory writeable when clock is HIGH, instead of on positive edge of CLK
Also uses less transistors per bit!

Efficient 2^2 by 3-bit Memory – Single Port – SRAM Cell



A 6 Transistor D-Latch!



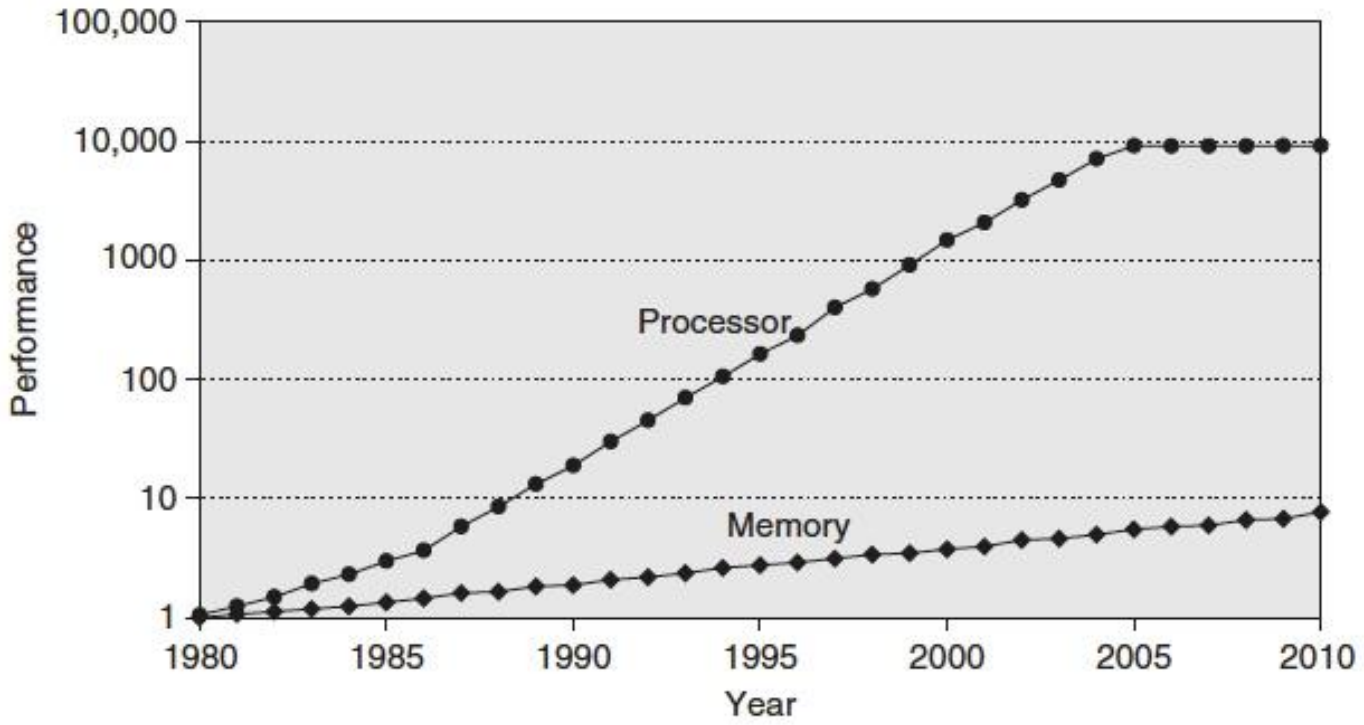
Lecture Outline

- D Flip Flops & Registers
- Memory at a high level
- Memory using flip flops
- **Memory Hierarchy**
- LC4 (start)

Data Access Time

- Data is stored on a physical piece of hardware
- The distance data must travel on hardware affects how long it takes for that data to be processed
- Example: data stored closer to the CPU is quicker to access
  - We will see this as we discuss memory vs registers in LC4 programming
  - As we go further from the CPU, storage space goes up, but access times increase

Processor-Memory Gap



- Processor speed kept growing ~55% per year
- Time to access memory didn't grow as fast ~7% per year
- Memory access would create a bottleneck on performance

Cache

- Pronounced “cash”
- English: A hidden storage space for equipment, weapons, valuables, supplies, etc.
- Computer: Memory with shorter access time used for the storage of data for increased performance. Data is usually either something frequently and/or recently used.

Principle of Locality

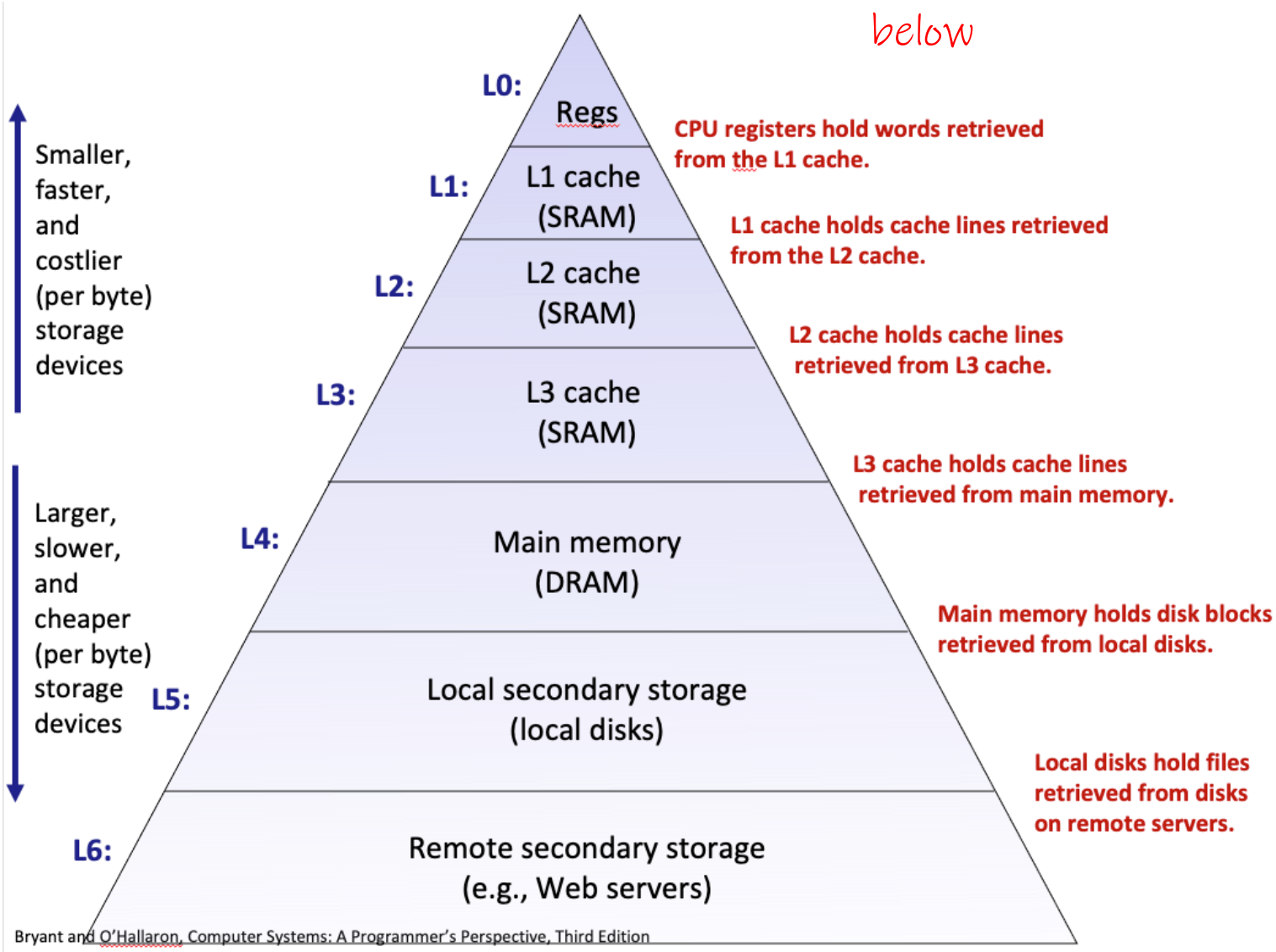
- The tendency for the CPU to access the same set of memory locations over a short period of time

- Two main types:
  - **Temporal Locality**: If we access a portion of memory, we will likely reference it again soon
  - **Spatial Locality**: If we access a portion of memory, we will likely reference memory close to it in the near future.

- Caches take advantage of these tendencies with the cache policies to decide what data is stored in the cache.

Memory Hierarchy

Each layer can be thought of as a "cache" of the layer below



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Lecture Outline

- D Flip Flops & Registers
- Memory at a high level
- Memory using flip flops
- Memory Hierarchy
- **LC4 (start)**

LC4 start

- Now that we have an idea of memory, we need to have an idea of how programs interact with memory to see how we develop more hardware.

- We are now looking at “code”,
  but not in the way you may think…

- We are looking at Assembly! (ASM)
  - Note: our assembly language is LC4 and only exists at UPenn

LC4 ISA

- What is an ISA?
  - **I**nstruction **S**et **A**rchitecture
  - Defines everything needed to create a program that runs directly on a processor
  - Serves as the "contract" between software and hardware

- Basic Components of any ISA: (more in later lectures)
  - Memory
    - Address space and addressability
  - Instructions
    - What operations are available, how instructions are encoded
  - Registers
    - How many registers, what size they are, and how they are used.

LC4 Instruction Set

- Code broken up so that one line is the most basic of operational "instructions" that can run directly on a CPU

- Instead of operating on variables, we are operating on processor registers.
  - Each register holds some 16-bit value
  - We have 8 of these: (R0, R1, R2 … R7)
  - (Program variables aren't just processor registers in reality, but we will treat them like that for now)

- Control structures and functions don't exist normally
  - Do not have: If/else/while/for
  - We can create our own versions of these (more later)

LC4 Instruction Examples

- **`ADD Rd, Rs, Rt`**
  - Action: `Rd = Rs + Rt`

Rt = Second source or sometimes "Target" register

Rd = Destination Register

Rs = Source Register

- **`CONST Rd, IMM9`**
  - Action: `Rd = SEXT(IMM9)`
  - Store an integer constant in the specified register
  - IMM9 = 9-bit 2C integer immediate
  - SEXT stands for **S**ign **Ext**ension.
  - A register is 16 bits, but the value we are storing is only 9 bits

LC4 Instruction Examples

- **`MUL Rd, Rs, Rt`**
  - Action: **`Rd = Rs * Rt`**

- **`ADD Rd, Rs, IMM5`**
  - Action: **`Rd = Rs + SEXT(IMM5)`**
  - IMM5 = 5-bit 2C integer immediate
  - SEXT stands for **S**ign **Ext**ension.
  - A register is 16 bits, but the value we are adding is only 5 bits

LC4 ASM vs C (Learning Example)

- Instead of operating on variables, we are operating on processor registers.
  - We have 8 of these: (R0, R1, R2 … R7)
  - (Program variables aren't just processor registers in reality, but we will treat them like that for now)

- Example comparing C cod to ASM:

C code

```
int R0 = 0;
int R1 = 12;

R0 = (R1 + 5)
R0 = R0 * R1;
```

LC4

```
CONST R0, #0
CONST R1, #12

ADD R0, R1, #5
MUL R0, R1, R0
```

C doesn't translate into assembly this way; this is just a comparison for learning

All Arithmetic Instructions in LC4

- All arithmetic operations in LC4:

`ADD Rd Rs Rt`	`Rd = Rs + Rt`
`MUL Rd Rs Rt`	`Rd = Rs * Rt`
`SUB Rd Rs Rt`	`Rd = Rs - Rt`
`DIV Rd Rs Rt`	`Rd = Rs / Rt`
`ADD Rd Rs IMM5`	`Rd = Rs + sext(IMM5)`
`MOD Rd Rs Rt`	`Rd = Rs % Rt`

  - Note the order of registers matter for some operations
    - (DIV, SUB, MOD)
  - Note that DIV does integer division

Poll Everywhere

pollev.com/tqm

- What is the final value of R0 after the following instructions are executed:

A. 32

B. -32

C. 28

D. -28

E. I'm not sure

```
CONST R0, #32
CONST R1, #16
CONST R2, #64

DIV R3, R2, R1
ADD R3, R3, R0
SUB R0, R2, R3
```

Poll Everywhere

pollev.com/tqm

- What is the final value of R0 after the following instructions are executed:

A. 32

B. -32

C. 28

D. -28

E. I'm not sure

```
CONST R0, #32
CONST R1, #16
CONST R2, #64

DIV R3, R2, R1
ADD R3, R3, R0
SUB R0, R2, R3
```

Next instruction to execute

Registers	Value
R0	32
R1	16
R2	64
R3	???

Poll Everywhere

pollev.com/tqm

- What is the final value of R0 after the following instructions are executed:

A. 32

B. -32

C. 28

D. -28

E. I'm not sure

```
CONST R0, #32
CONST R1, #16
CONST R2, #64

DIV R3, R2, R1
ADD R3, R3, R0
SUB R0, R2, R3
```

Next instruction to execute

Registers	Value
R0	32
R1	16
R2	64
R3	4

Poll Everywhere

pollev.com/tqm

- What is the final value of R0 after the following instructions are executed:

A. 32

B. -32

C. 28

D. -28

E. I'm not sure

```
CONST R0, #32
CONST R1, #16
CONST R2, #64

DIV R3, R2, R1
ADD R3, R3, R0
SUB R0, R2, R3
```

Next instruction to execute

Registers	Value
R0	32
R1	16
R2	64
R3	36

Poll Everywhere

pollev.com/tqm

- What is the final value of R0 after the following instructions are executed:

A. 32

B. -32

C. 28

D. -28

E. I'm not sure

```
CONST R0, #32
CONST R1, #16
CONST R2, #64

DIV R3, R2, R1
ADD R3, R3, R0
SUB R0, R2, R3
```

Registers	Value
R0	28
R1	16
R2	64
R3	36

Bitwise Instructions in LC4

- Bitwise operations in LC4:

`AND Rd Rs Rt`	`Rd = Rs & Rt`
`NOT Rd Rs`	`Rd = ~Rs`
`OR Rd Rs Rt`	`Rd = Rs | Rt`
`XOR Rd Rs Rt`	`Rd = Rs ^ Rt`
`AND Rd Rs IMM5`	`Rd = Rs & sext(IMM5)`

  - Very similar layout to arithmetic operations, just performing bitwise operations instead
  - Shifting also exists and will be discussed later

LC4 "Cheat Sheet"

- Contains every LC4 instruction, its behaviour, and other information we will discuss later
  - On the website under "references"
  - HIGHLY recommend you print a copy
  - Will be provided on exams if needed

LC4 Instruction Set Reference v. 2017-01

Mnemonic	Semantics	Encoding
NOP	PC = PC + 1	0000 000x xxxx xxxx
BRp <Label>	(P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 001i iiii iiii
BRz <Label>	(Z) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 010i iiii iiii
BRzp <Label>	(Z|P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 011i iiii iiii
BRn <Label>	(N) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 100i iiii iiii
BRnp <Label>	(N | P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 101i iiii iiii
BRnz <Label>	(N|Z) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 110i iiii iiii
BRnzp <Label>	(N|Z|P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 111i iiii iiii
ADD Rd Rs Rt	Rd = Rs + Rt	0001 ddds ss00 0ttt
MUL Rd Rs Rt	Rd = Rs * Rt	0001 ddds ss00 1ttt
SUB Rd Rs Rt	Rd = Rs - Rt	0001 ddds ss01 0ttt
DIV Rd Rs Rt	Rd = Rs / Rt	0001 ddds ss01 1ttt
ADD Rd Rs IMM5	Rd = Rs + sext(IMM5)	0001 ddds ss1i iiii
MOD Rd Rs Rt	Rd = Rs % Rt	1010 ddds ss11 xttt
AND Rd Rs Rt	Rd = Rs & Rt	0101 ddds ss00 0ttt

Next Lectures

- Other LC4 ISA components
  - Program Counter
  - Program State Register & NZP
  - Memory
- Instructions as Data
- Tour through the rest of LC4 instructions