CSC 252: Computer Organization
Spring 2025: Lecture 12

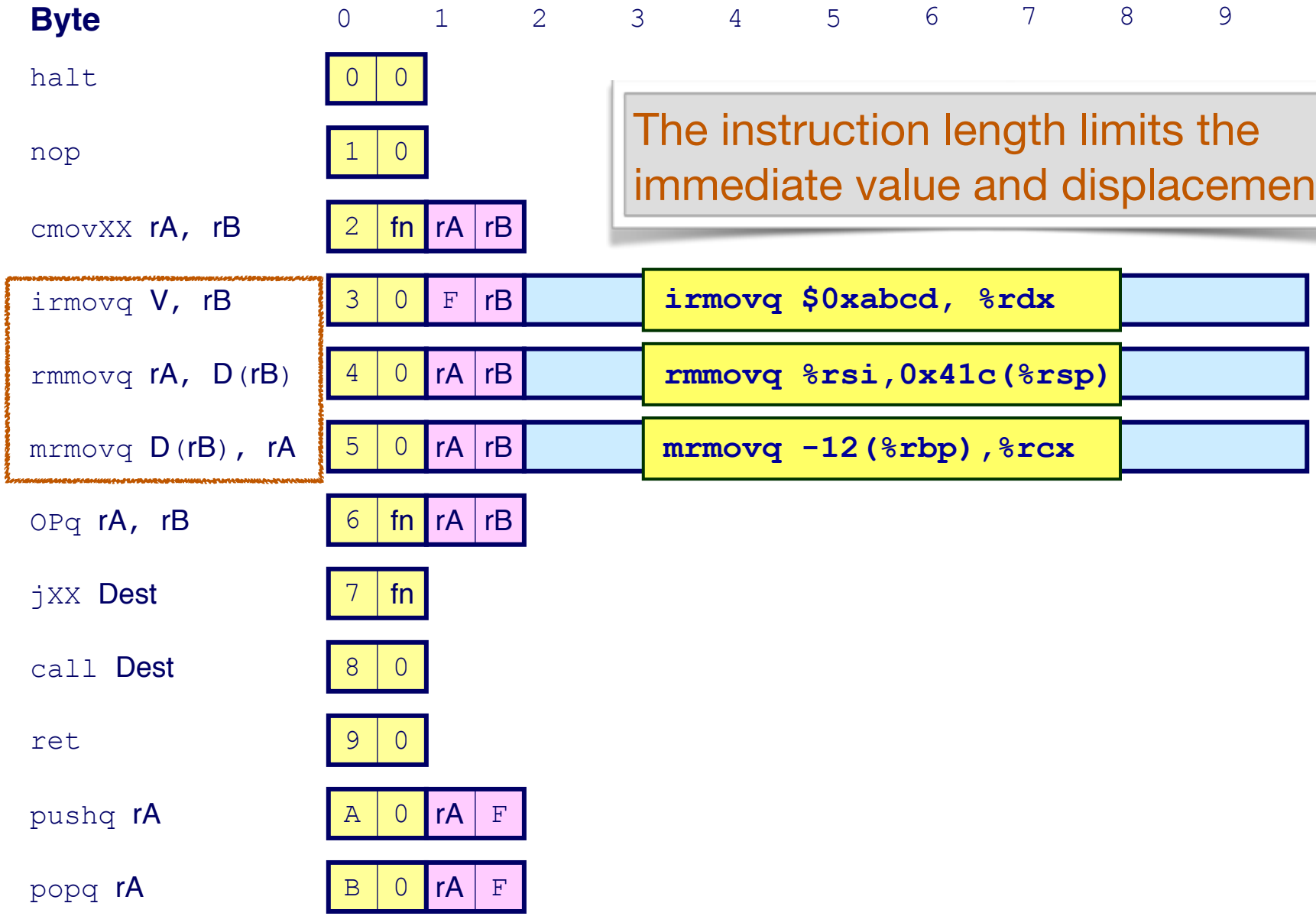
Instructor: Yuhao Zhu

Department of Computer Science
University of Rochester

Announcement

- Programming assignment 3 out.
- See blackboard announcement for PA 1 grades
- Open-book mid-term on Friday at this time
  - Anything on paper is fair, nothing electronically.
- "I don't know" is given 15% partial credit
  - You need to decide if guessing is worthwhile
  - Saves grading time
  - You have to write "I don't know" and cross out /erase anything else to get credit: A blank answer doesn't count

Move Instructions



Move Instruction Examples

Y86-64

```
irmovq $0xabcd, %rdx
```

Encoding: `30 82 cd ab 00 00 00 00 00 00`

```
rrmovq %rsp, %rbx
```

Encoding: `20 43`

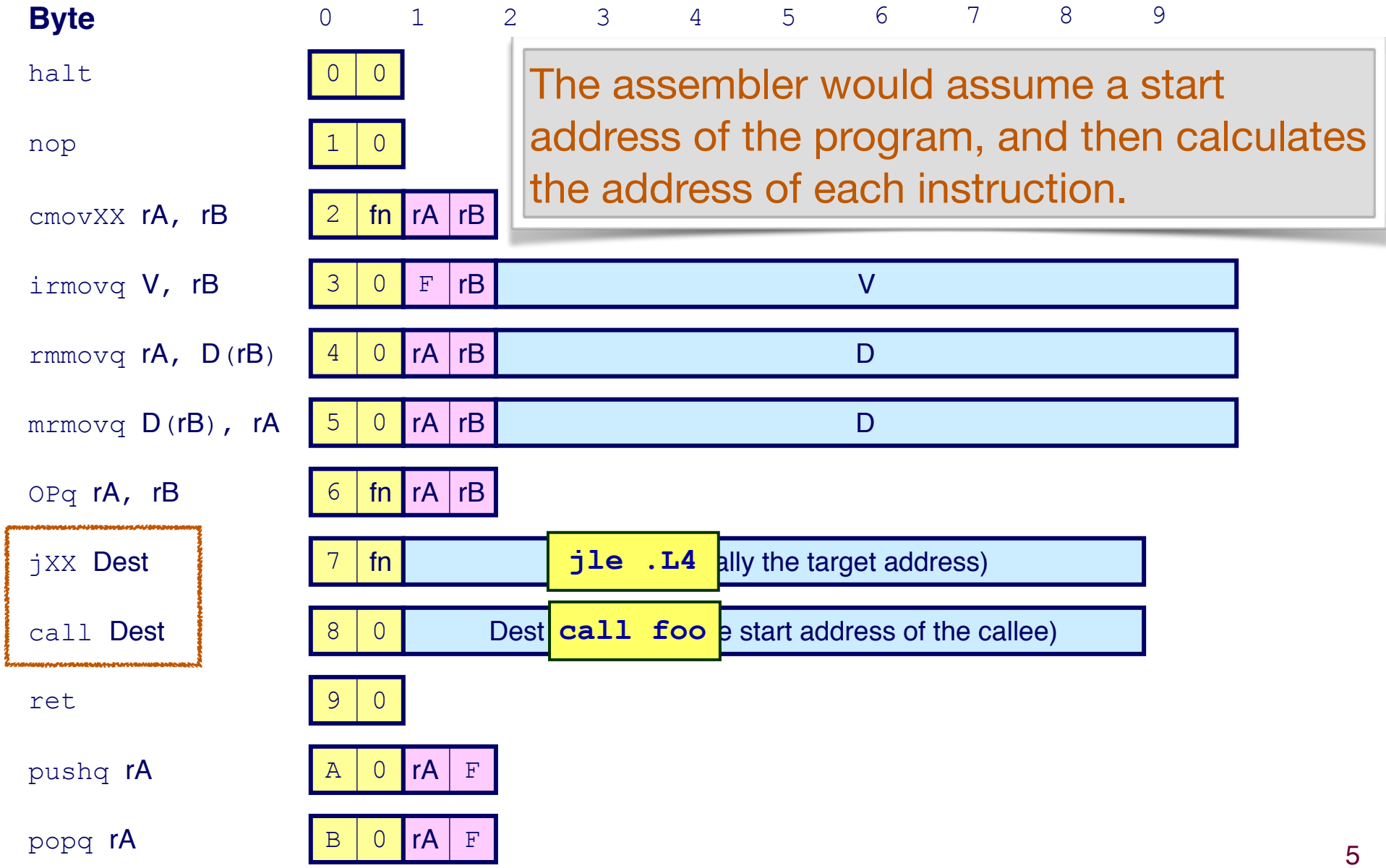
```
mrmovq -12(%rbp),%rcx
```

Encoding: `50 15 f4 ff ff ff ff ff ff ff`

```
rmmovq %rsi,0x41c(%rsp)
```

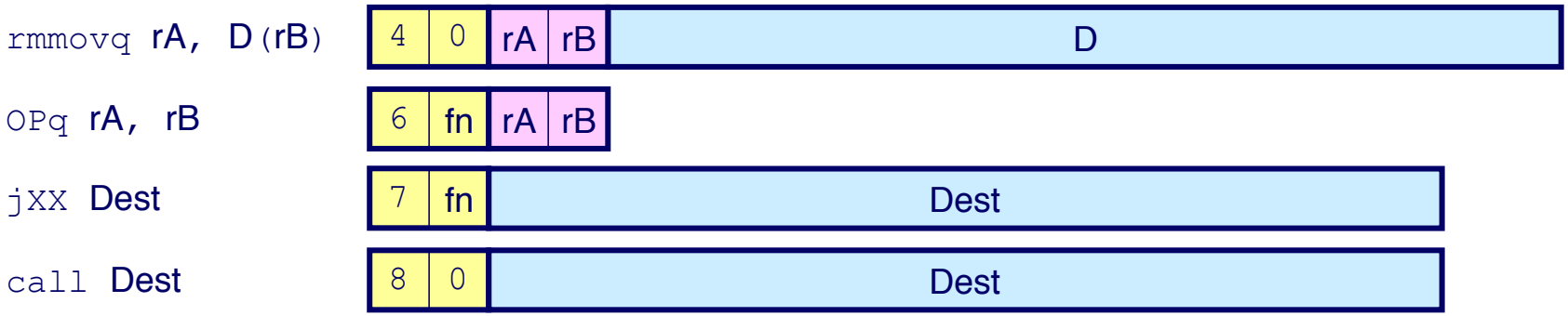
Encoding: `40 64 1c 04 00 00 00 00 00 00`

Jump/Call Instructions



The assembler would assume a start address of the program, and then calculates the address of each instruction.

How Does An Assembler Work?

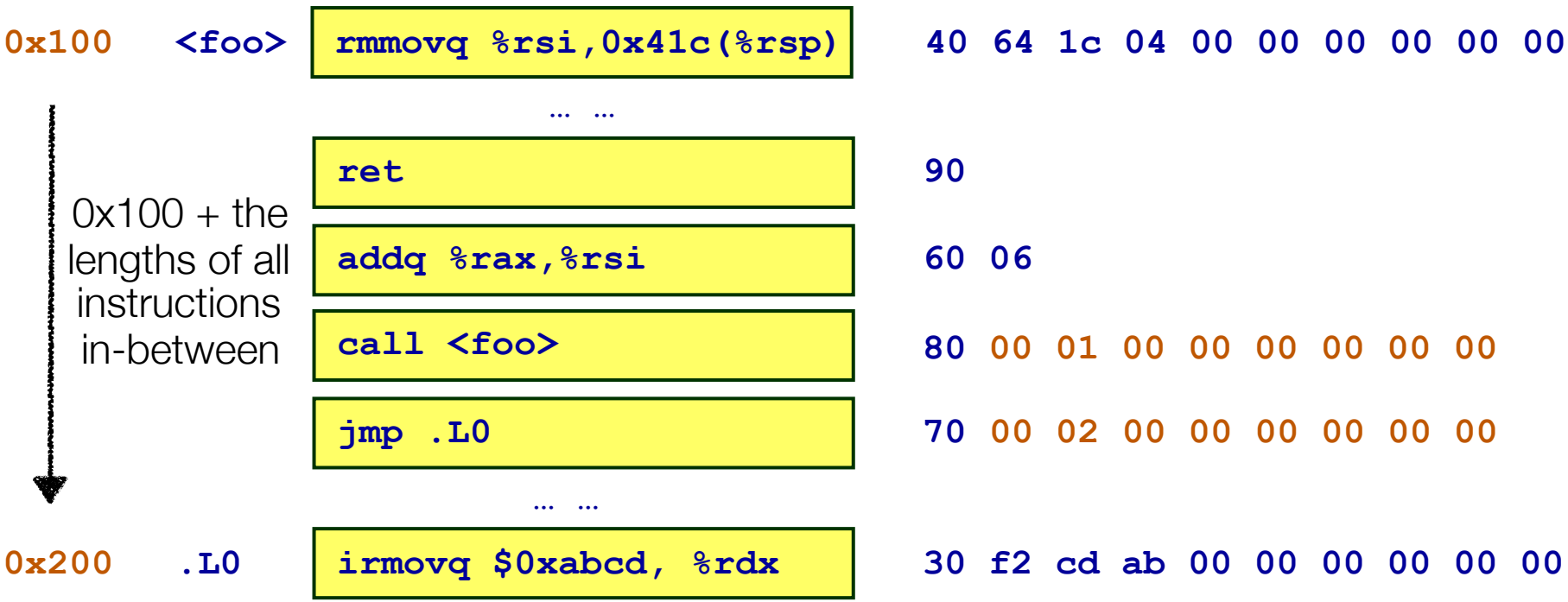



```
0x100    <foo>   rmmovq %rsi,0x41c(%rsp)     40 64 1c 04 00 00 00 00 00 00
                 ... ...
                 ret                        90
                 addq %rax,%rsi             60 06
                 call <foo>                 80 00 01 00 00 00 00 00 00
                 jmp .L0                    70 00 02 00 00 00 00 00 00
                 ... ...
0x200    .L0     irmovq $0xabcd, %rdx       30 f2 cd ab 00 00 00 00 00 00
```

0x100 + the lengths of all instructions in-between

How Does An Assembler Work?

- The assembler is a program that translates assembly code to binary code
- The OS tells the assembler the start address of the code (sort of…)
- Translate the assembly program line by line
- Need to build a “label map” that maps each label to its address



Jump Instructions

Jump Unconditionally

Jump When Less or Equal

Jump When Less

Jump When Equal

Jump When Not Equal

Jump When Greater or Equal

Jump When Greater

Subroutine Call and Return

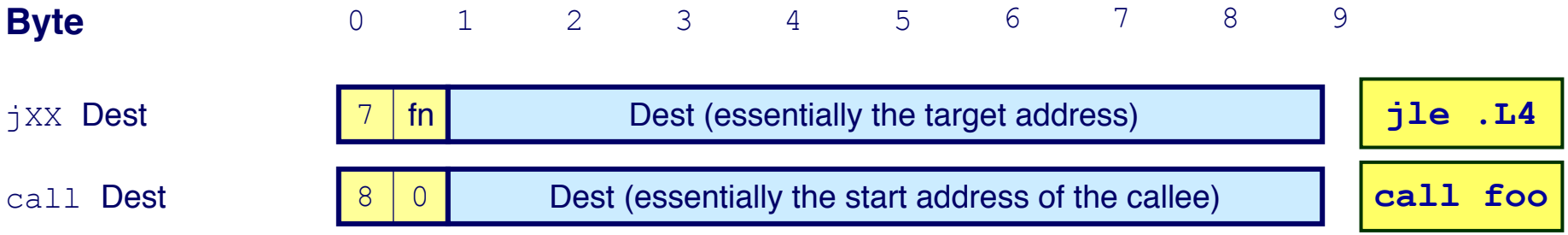
`call` Dest

- Push address of next instruction onto stack
- Start executing instructions at Dest
- Like x86-64

`ret`

- Pop value from stack
- Use as address for next instruction
- Like x86-64

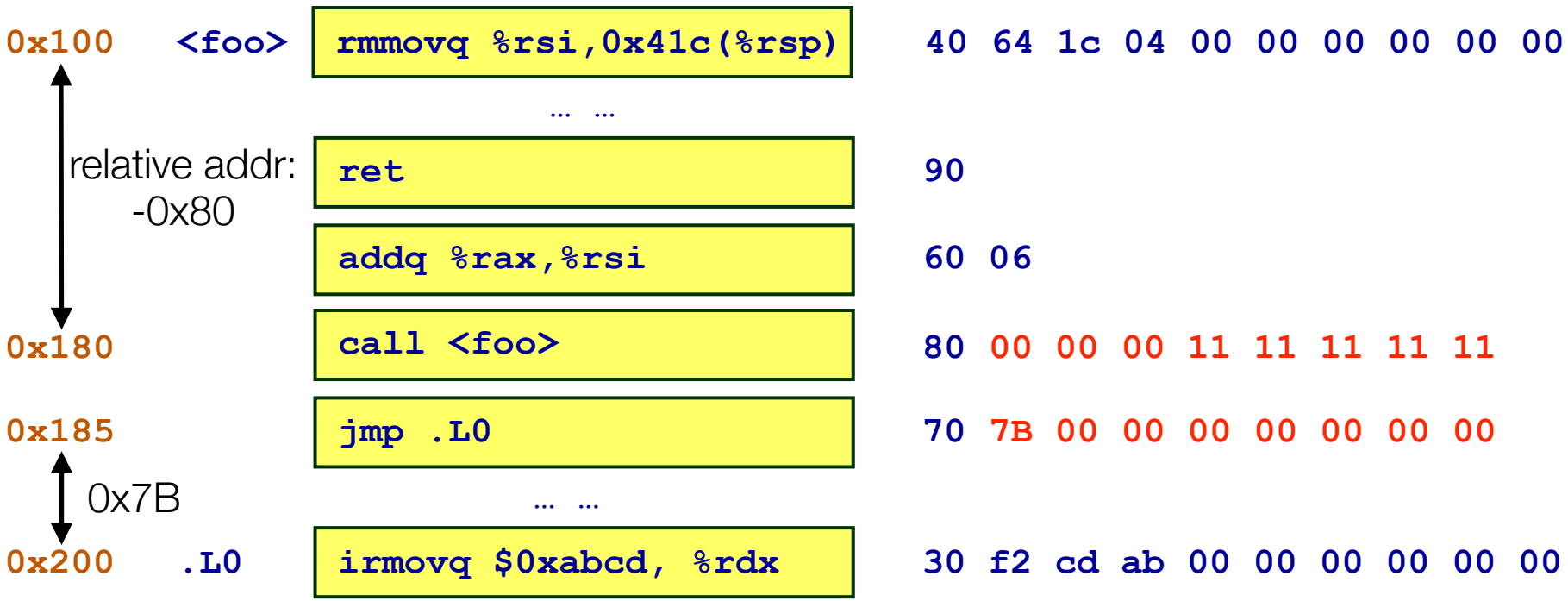
One More Complication…



- The instruction length limits how far you can jump/call functions. What if the jump target has a very long address that can't fit in 8 bytes?
  - Or if we can use only say 4 bytes for the target address?
- One alternative: use a super long instruction encoding format.
  - Simple to encode, but space inefficient (waste bits for jumps to short addr.)
- Another alternative: encode the relative address, not the absolute address
  - E.g., encode (.L4 - current address) in Dest

Using Relative Addresses for Jumps

- What if the ISA encoding uses relative address for jump and call?
- If we use relative address, the exact start address of the code doesn't matter. Why?
- This code is called Position-Independent Code (PIC)



Miscellaneous Instructions

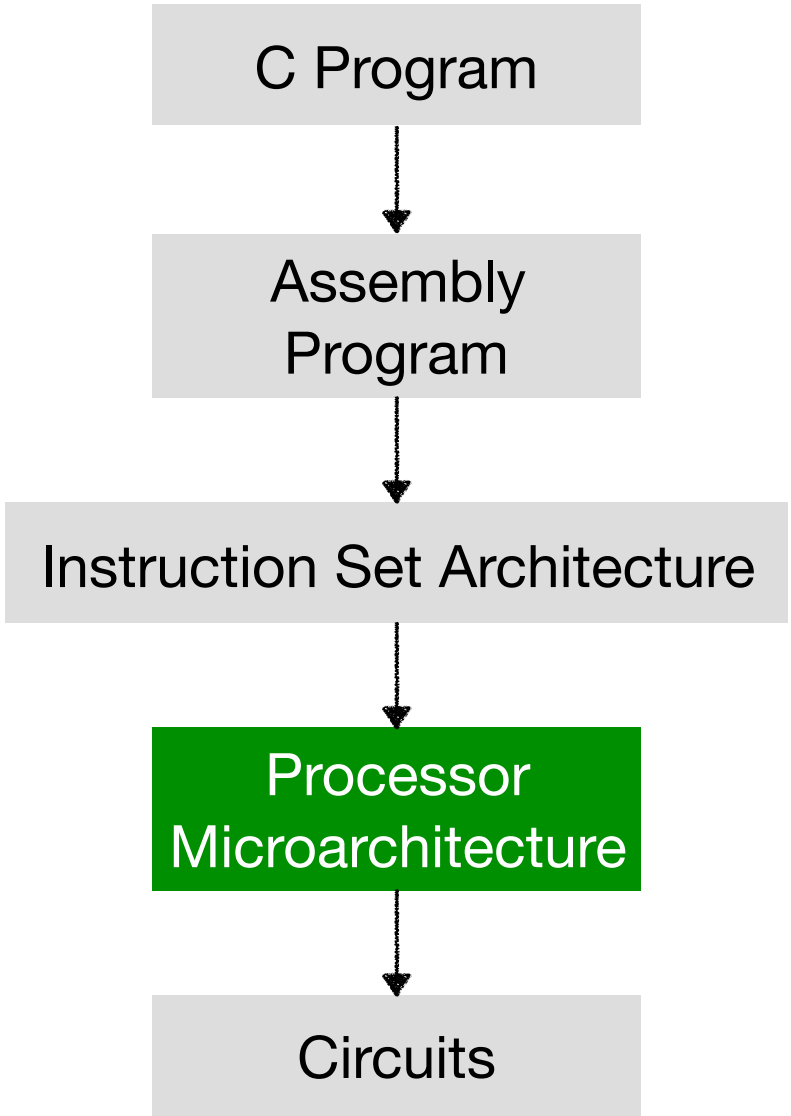
- Don’t do anything

- Stop executing instructions
- Usually can’t be executed in the user mode, only by the OS
- Encoding ensures that program hitting memory initialized to zero will halt

Variable Length Instructions

- X86 (and Y86) is a variable length ISA (1 to 15 bytes), where different instructions have different lengths.
- There are fixed length ISAs: all instructions have the same length
  - ARM's ISA for micro-controllers have a 4-bit ISA. Very Long Instruction Word (VLIW) ISAs have instructions that are hundreds of bytes long.
  - Or you can have a combination of both: e.g., 16-bit ISA with 32-bit extensions (e..g, ARM Thumb-extension).
- Advantages of variable length ISAs
  - More compact. Some instructions do not need that many bits. (Actually what's the optimal way of encoding instructions in a variable length ISA?)
  - Can have arbitrary number of instructions: easy to add new inst.
- What is the down side?
  - Fetch and decode are harder to implement. More on this later.
- A good writeup showing some of the complexity involved: http://www.c-jump.com/CIS77/CPU/x86/lecture.html

So far in 252…

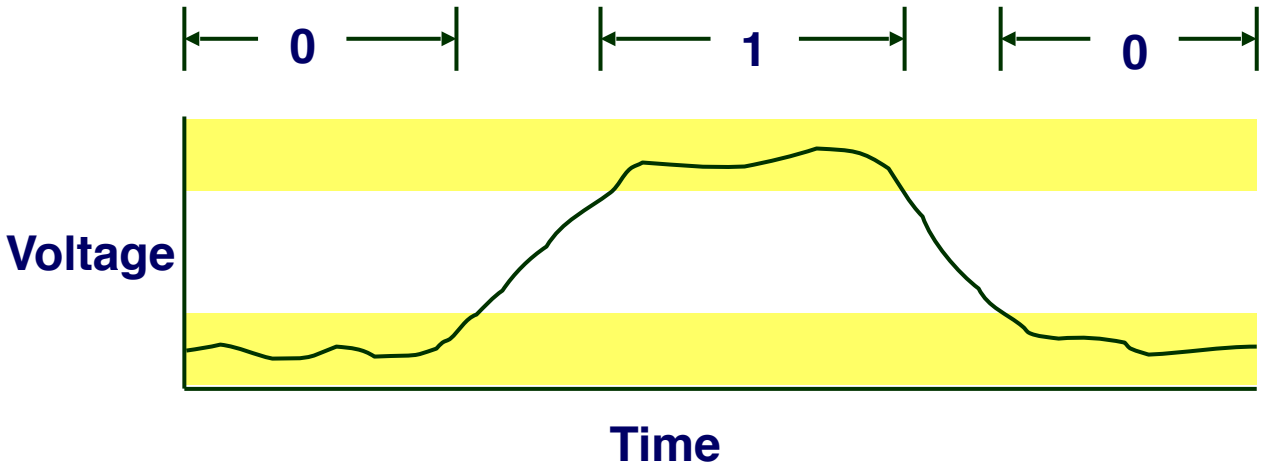

C Program
Assembly Program
Instruction Set Architecture
Processor Microarchitecture
Circuits

Today: Circuits Basics

- Basics
- Circuits for computations
- Circuits for storing data

Overview of Circuit-Level Design

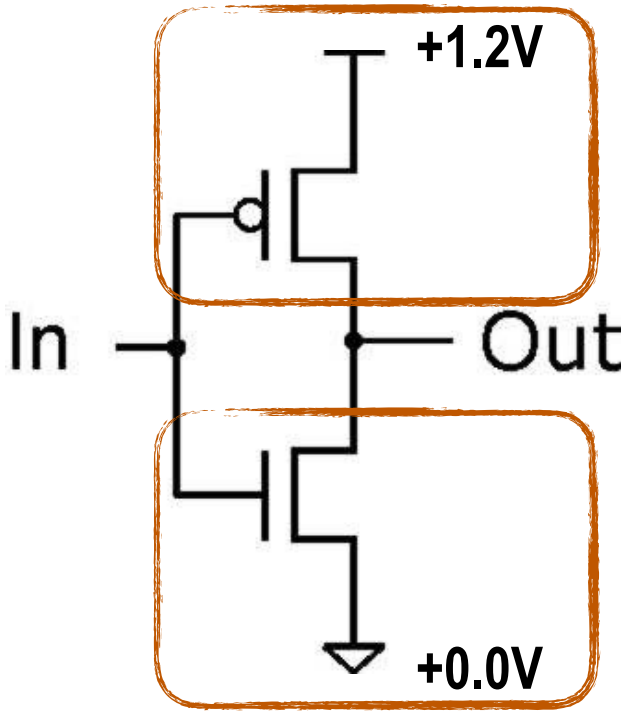
- Fundamental Hardware Requirements
  - Communication: How to get values from one place to another. Mainly three electrical **wires**.
  - Computation: **transistors**. Combinational logic.
  - Storage: **transistors**. Sequential logic.
- Circuit design is often abstracted as logic design



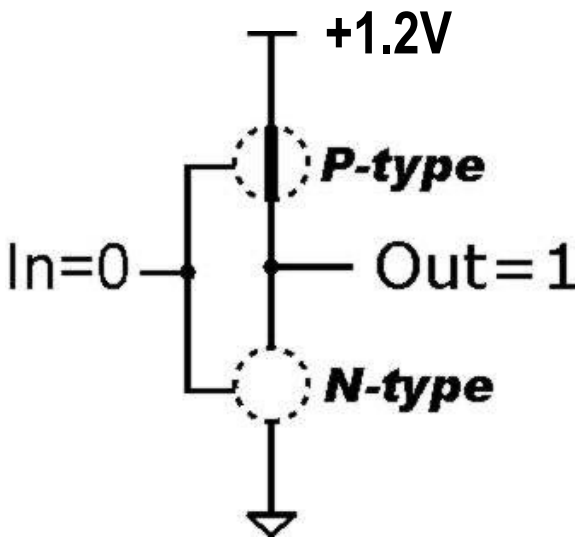
Today: Circuits Basics

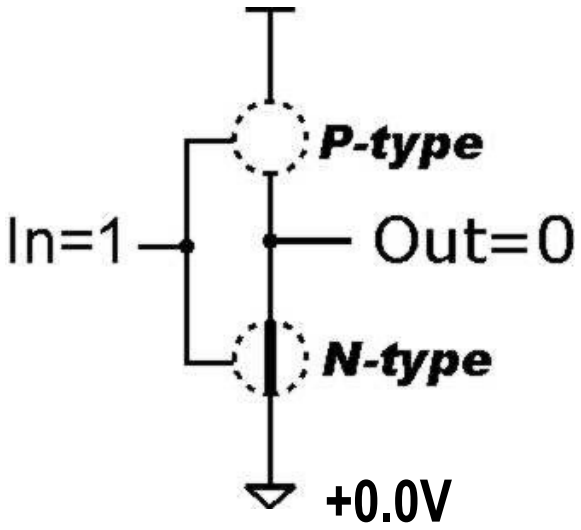
- Transistors
- Circuits for computations
- Circuits for storing data

Inverter (NOT Gate)

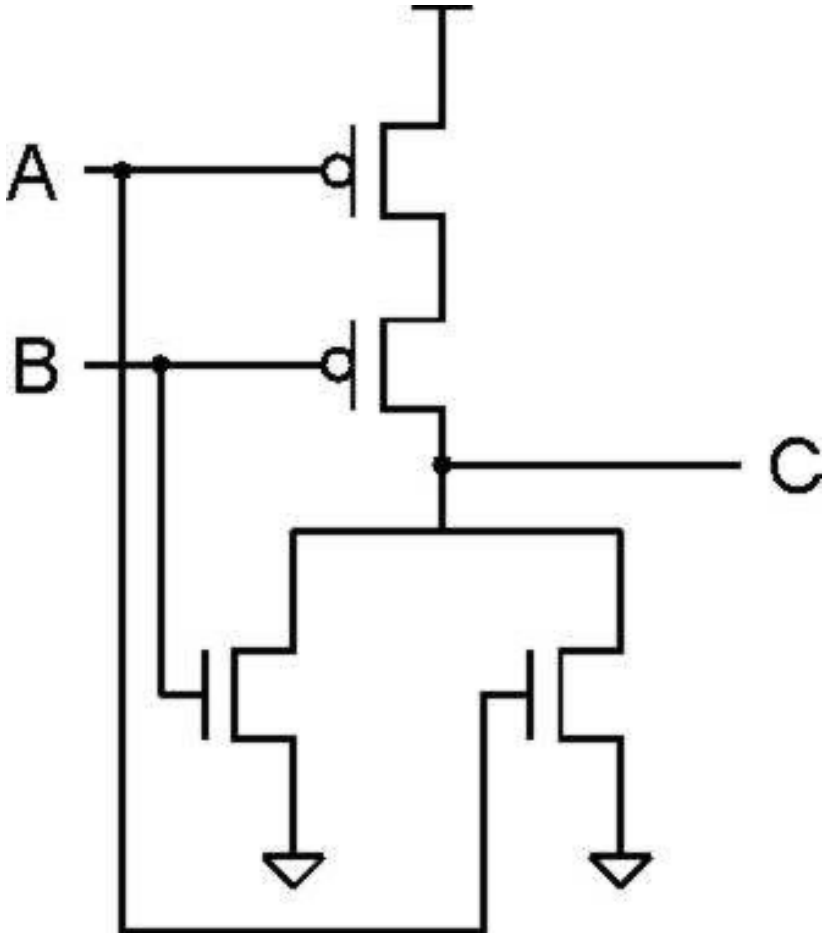


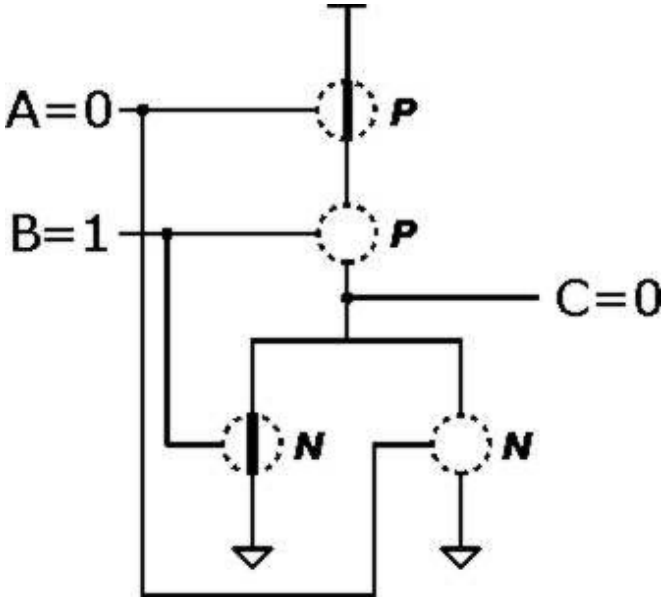
In	Out
0	1
1	0





NOR Gate (NOT + OR)

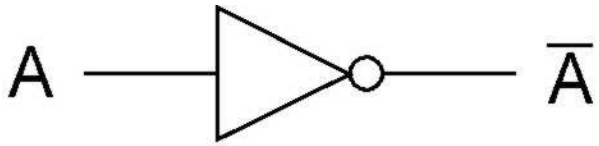




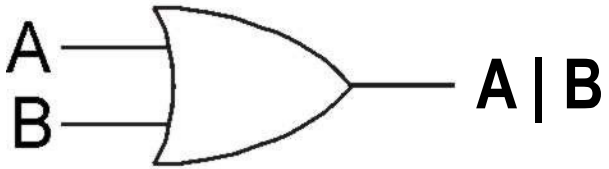
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Note: Serial structure on top, parallel on bottom.

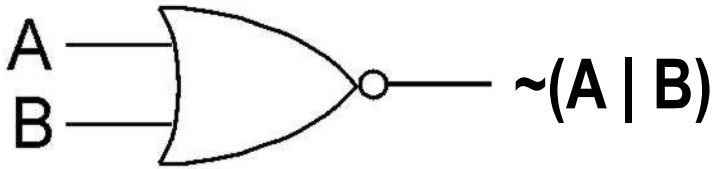
Basic Logic Gates



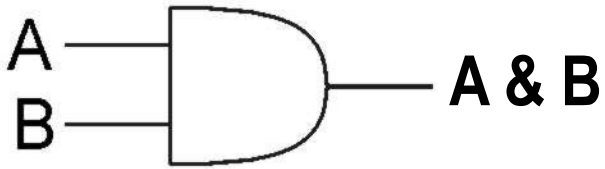
NOT



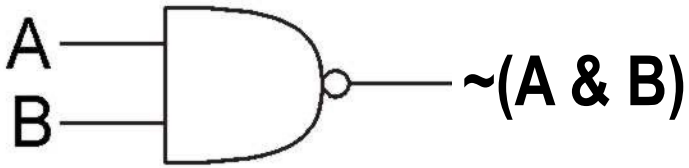
OR



NOR

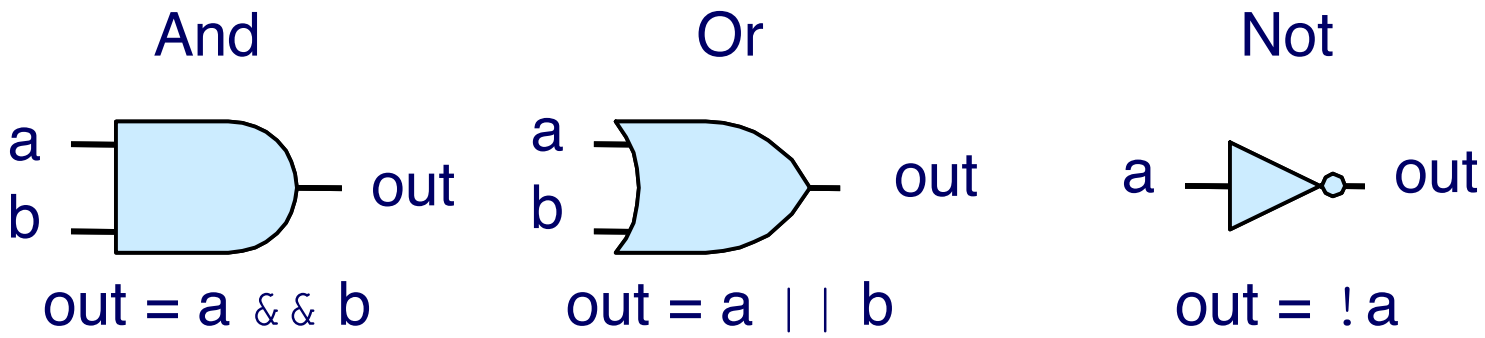


AND

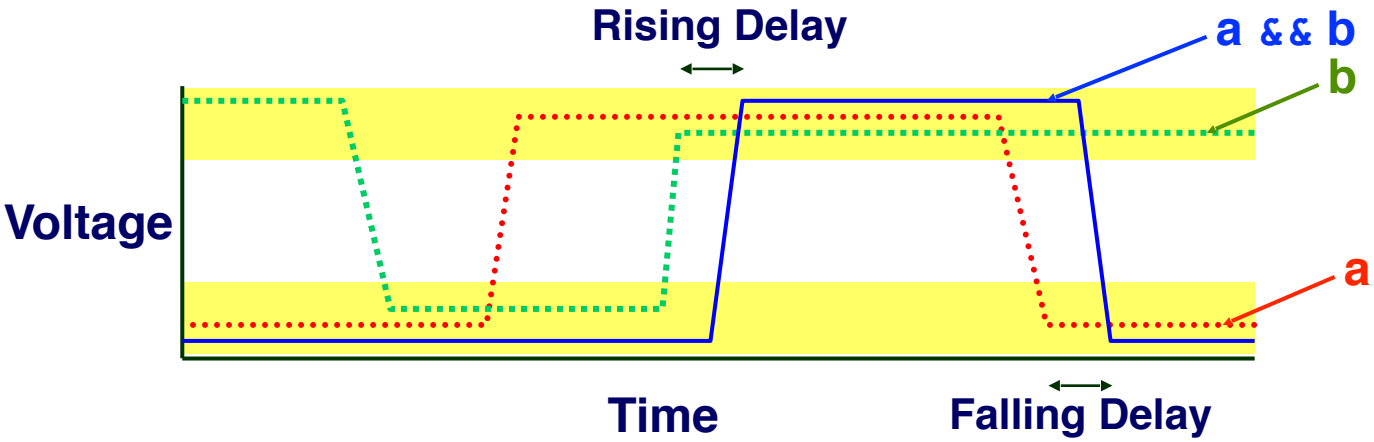


NAND

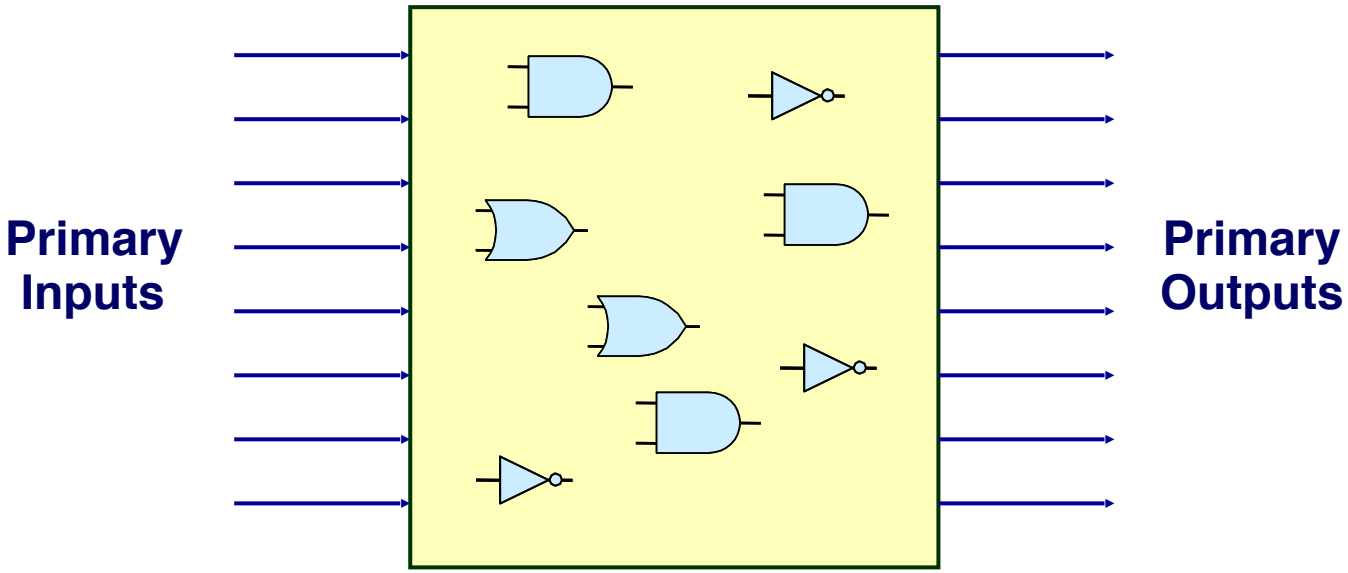
Computing with Logic Gates



- Outputs are Boolean functions of inputs
- Respond continuously to changes in inputs with some small delay
- Different gates have different delays (b/c different transistor combinations)

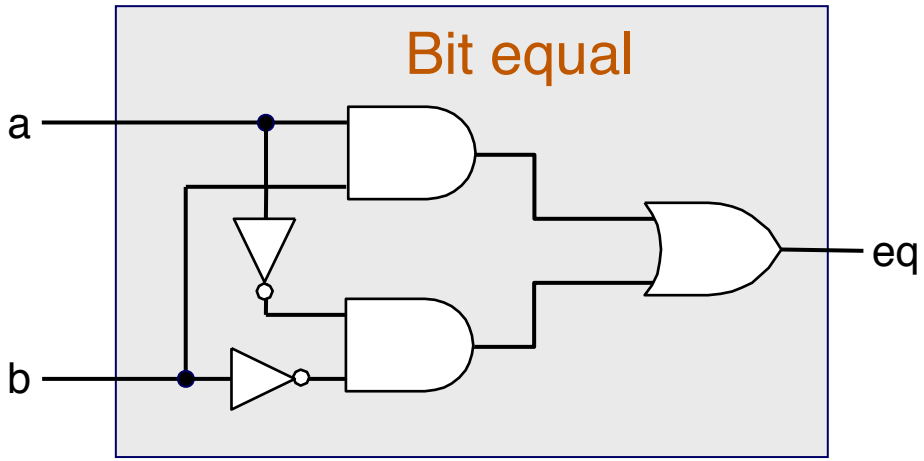


Combinational Circuits

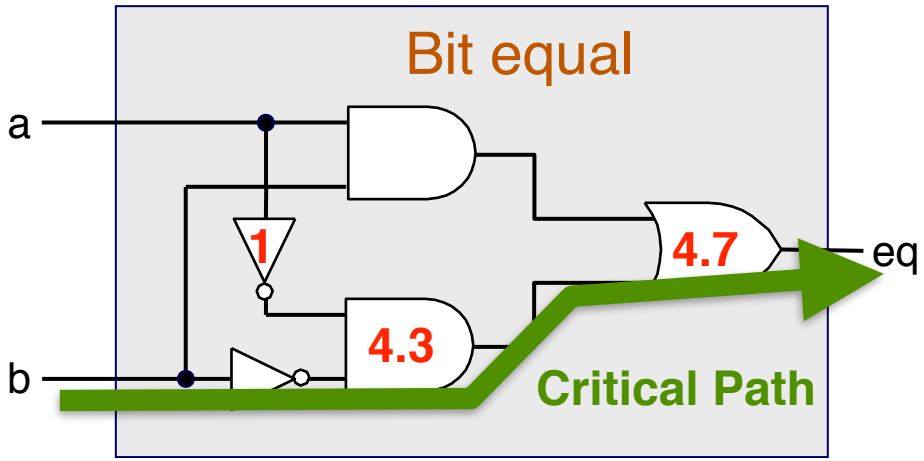


- A Network of Logic Gates
  - Continuously responds to changes on primary inputs
  - Primary outputs become (**after some delay**) Boolean functions of primary inputs

Bit Equality

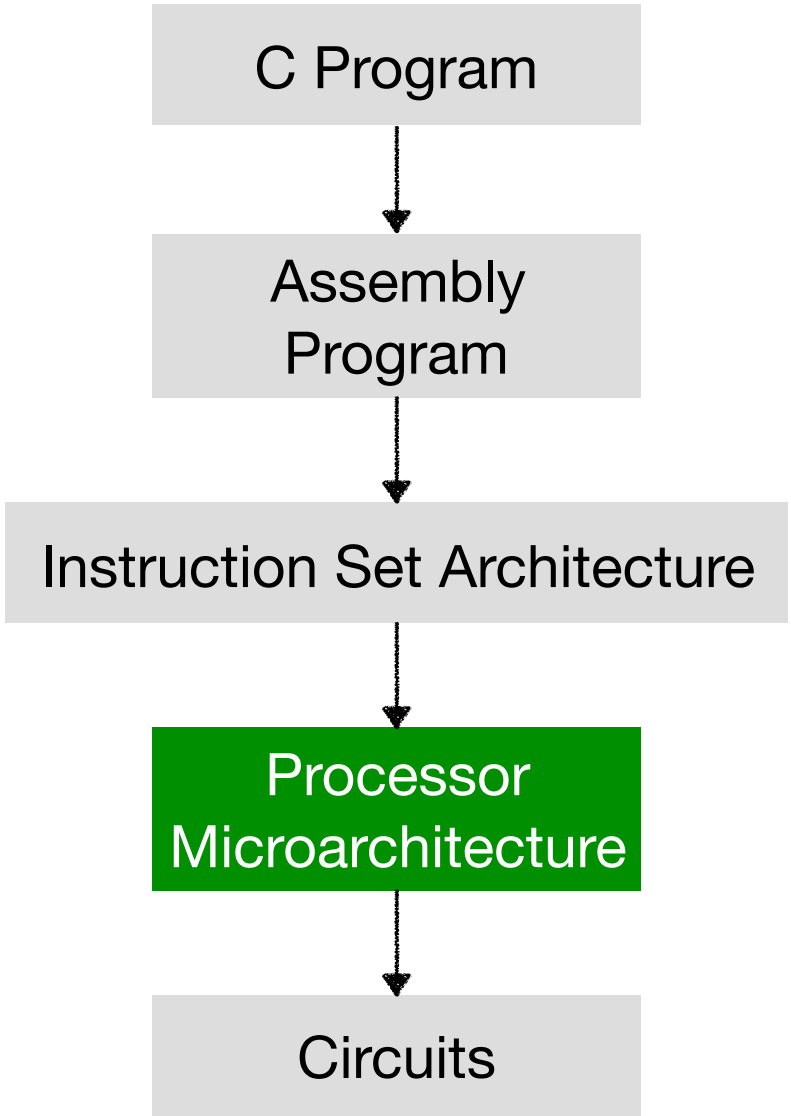


Delay of Bit Equal Circuit



- What's the delay of this bit equal circuit?
  - Assuming 1-input NOT takes 1 unit of time, 2-input AND takes 4.3, and 2-input OR takes 4.7
- The delay of a circuit is determined by its "critical path"
  - The path between an input and the output that the maximum delay
  - Estimating the critical path delay is called static timing analysis

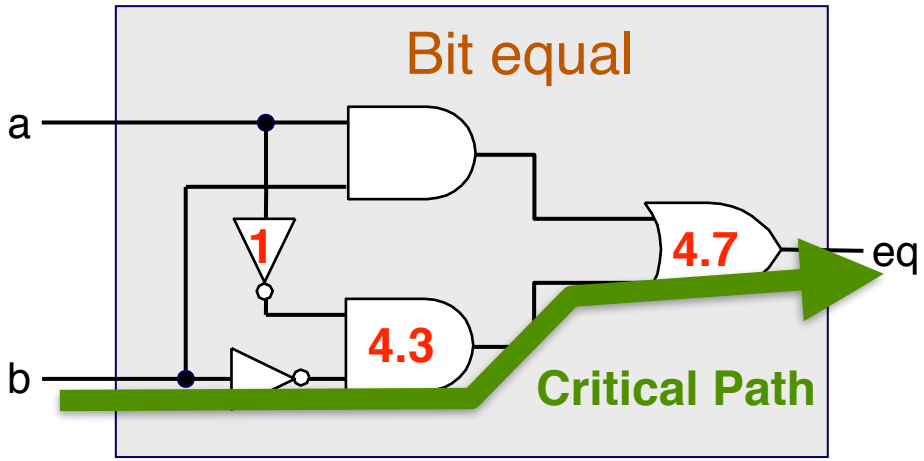
So far in 252…


C Program
Assembly Program
Instruction Set Architecture
Processor Microarchitecture
Circuits

Today: Circuits Basics

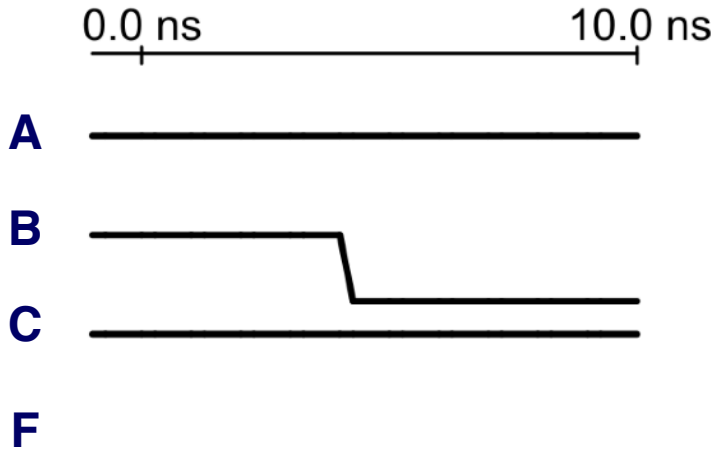
- Basics
- Circuits for computations
- Circuits for storing data

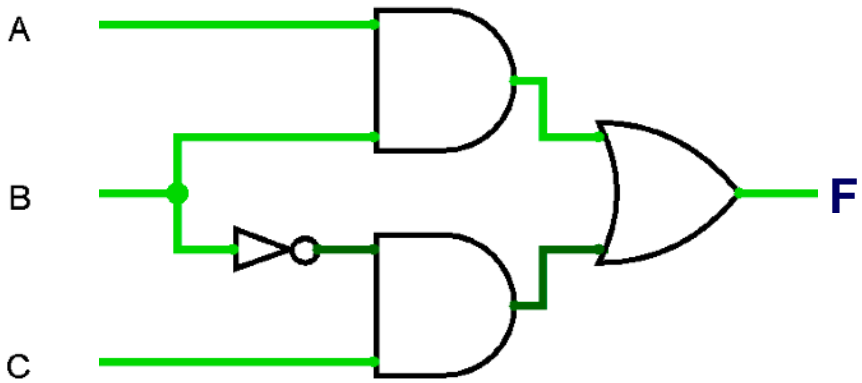
Delay of Bit Equal Circuit



- What's the delay of this bit equal circuit?
  - Assuming 1-input NOT takes 1 unit of time, 2-input AND takes 4.3, and 2-input OR takes 4.7
- The delay of a circuit is determined by its "critical path"
  - The path between an input and the output that the maximum delay
  - Estimating the critical path delay is called static timing analysis

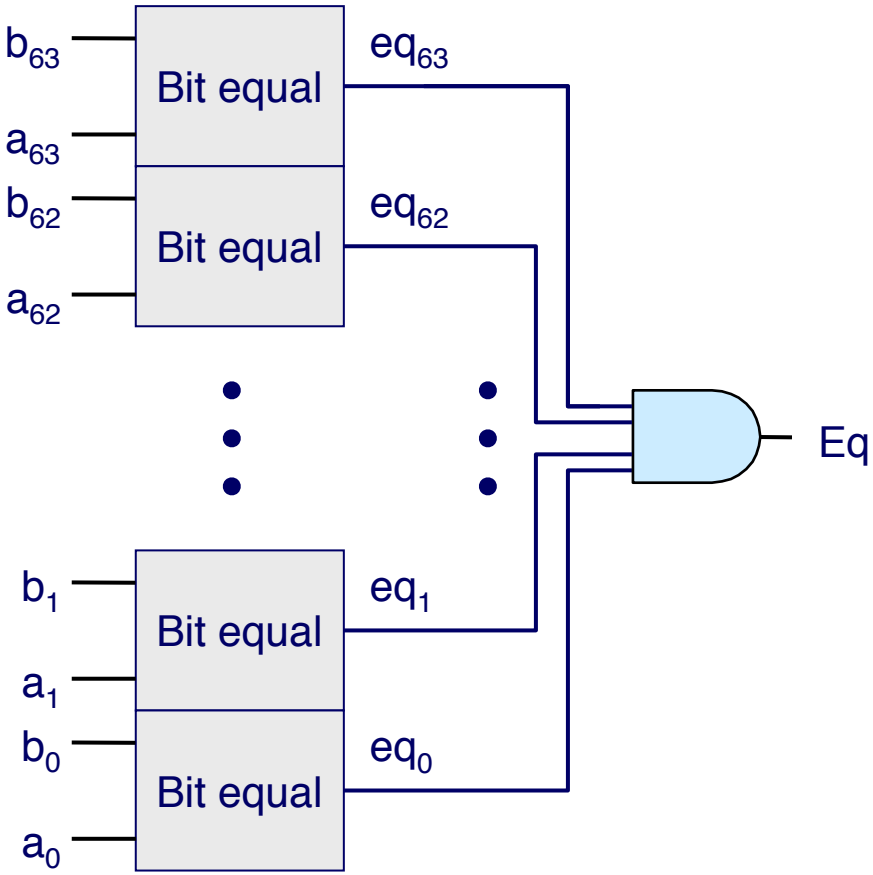
Glitch/Hazard

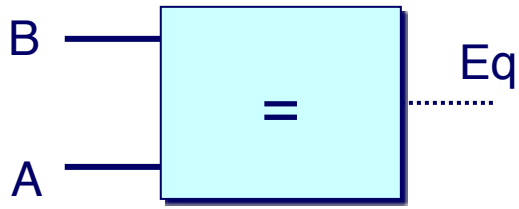




- A glitch is an unnecessary signal transition without functionality.
- Why is it bad? When transistors switch they consume power, but the power consumed during a glitch is a waste.
- Without care, glitch power dissipation is 20%-70% of total power dissipation.

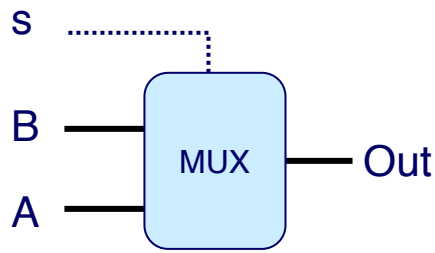
64-bit Equality



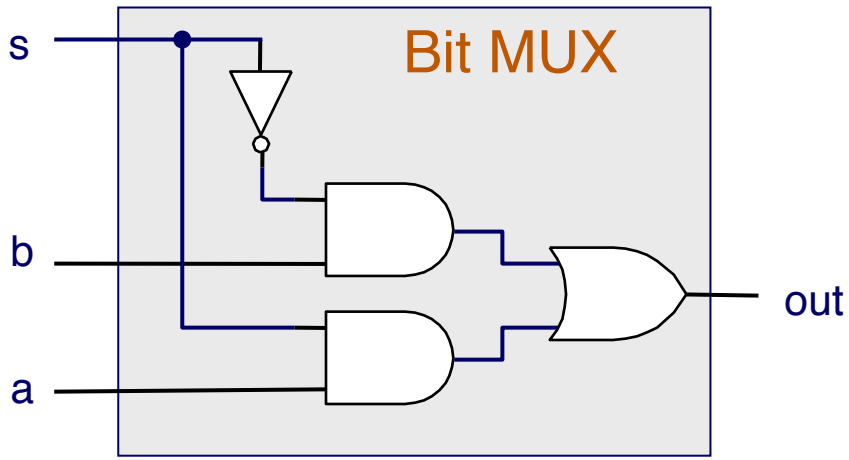


Bit-Level Multiplexor (MUX)

- Control signal s
- Data signals A and B
- Output A when s=1, B when s=0

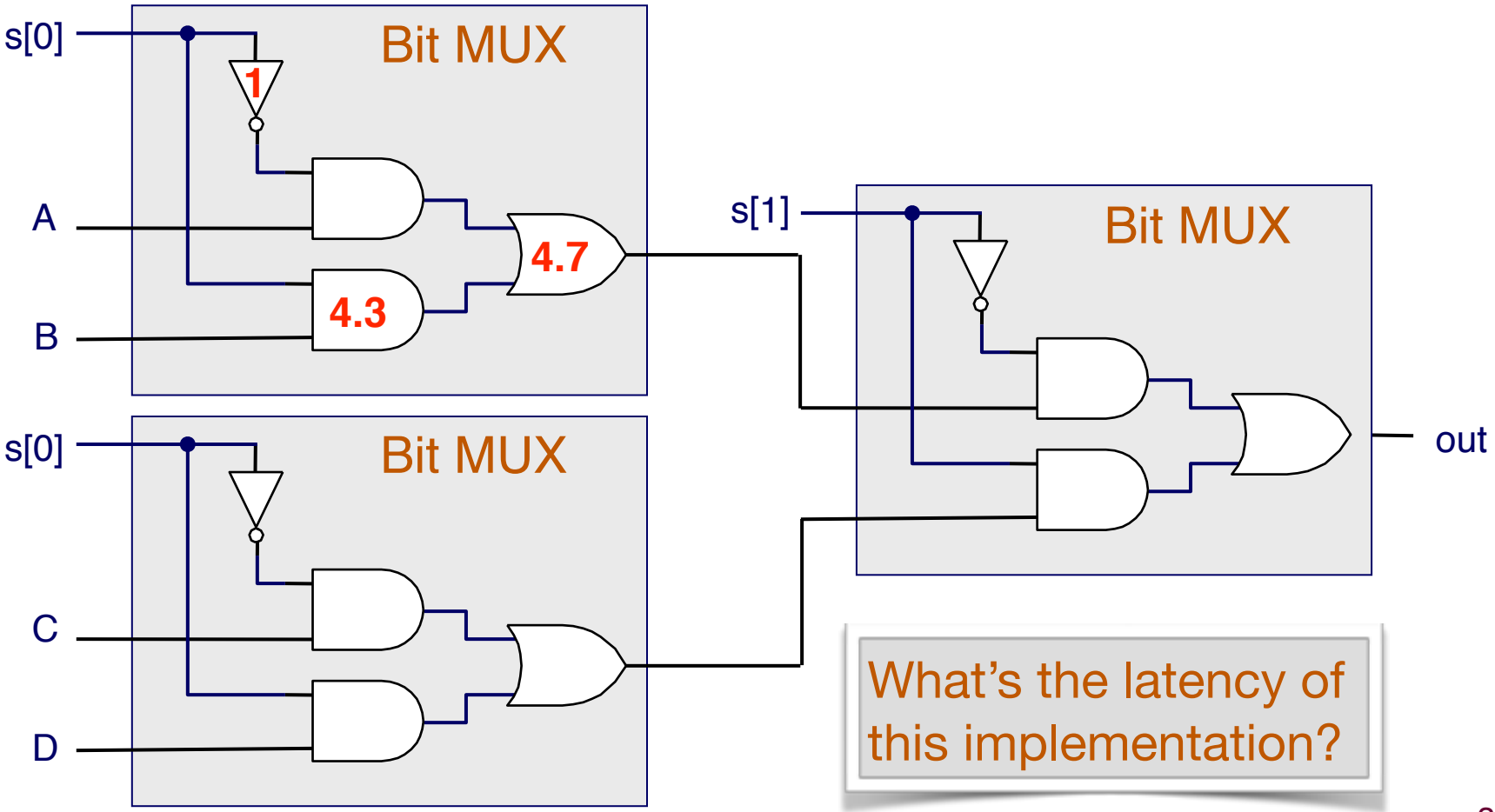



```
bool out = (s&&a)||(!s&&b)
```



4-Input Multiplexor

- Control signal s; Data signals A, B, C, and D
- Output: A when s = 00, B when s = 01, C when s = 10, D when s = 11



Logic Design and VLSI

- The number of inputs of a gate (fan-in) and the number of outputs of a gate (fan-out) will affect the gate delay.
- Think of logic gates as LEGO chips, using which you generate the gate level circuit design for complex functionalities.
- A standard cell library is a collection of well defined and appropriately characterized logic gates (delay, operating voltage, etc.) that can be used to implement a digital design.
- The *logic synthesis tool* will automatically generate the “best” gate-level implementation of a piece of logic.
- Take a Logic Design or Very Large Scale Integrated-Circuit (VLSI) course if you want to know more about circuit design.
  - Logic design uses the gate-level abstractions
  - VLSI tells you how the gates are implemented at transistor-level

Recall: Full (1-bit) Adder

Add two bits and carry-in,
produce one-bit sum and carry-out.

$$S = (\sim A \,\&\, \sim B \,\&\, C_{in}) \mid (\sim A \,\&\, B \,\&\, \sim C_{in}) \mid (A \,\&\, \sim B \,\&\, \sim C_{in}) \mid (A \,\&\, B \,\&\, C_{in})$$

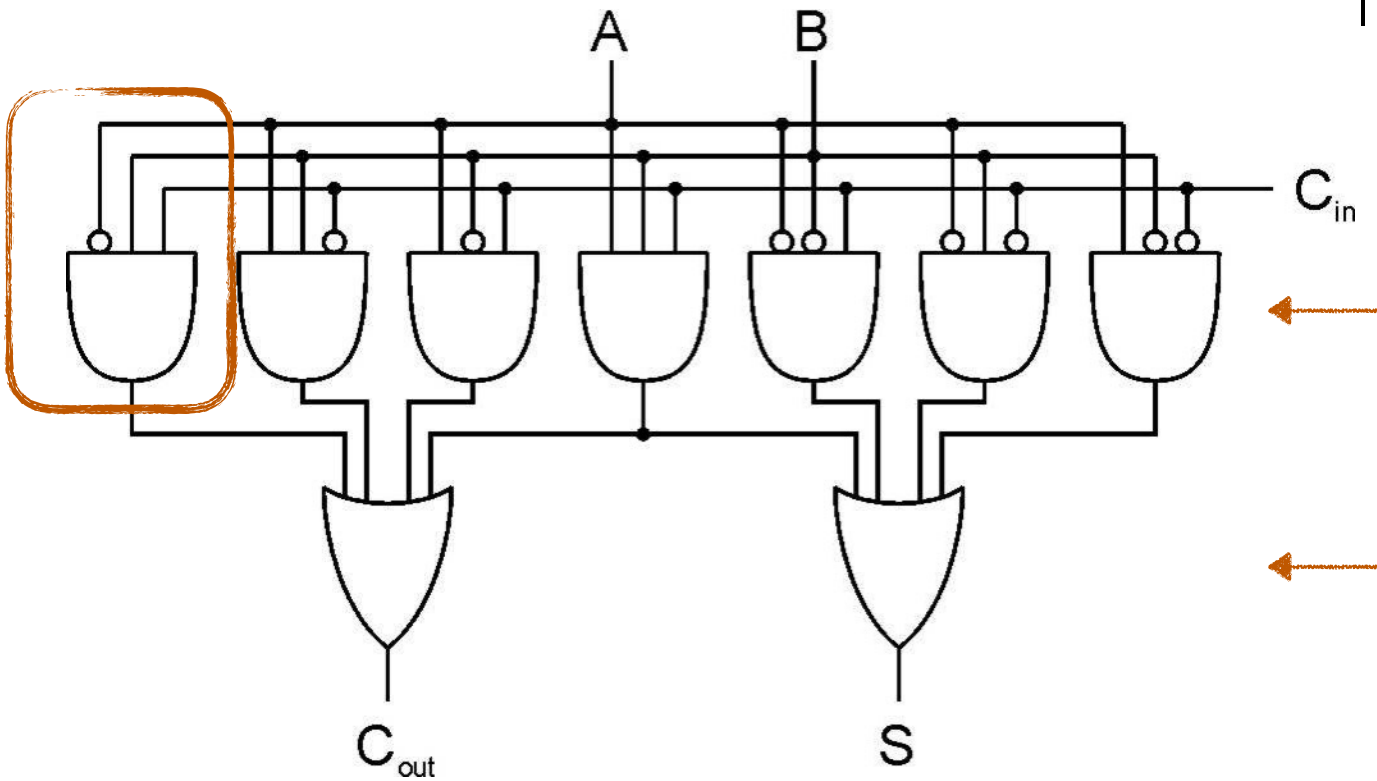
$$C_{ou} = (\sim A \,\&\, B \,\&\, C_{in}) \mid (A \,\&\, \sim B \,\&\, C_{in}) \mid (A \,\&\, B \,\&\, \sim C_{in}) \mid (A \,\&\, B \,\&\, C_{in})$$

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Recall: 1-bit Full Adder

$C_{ou} = (\sim A \,\&\, B \,\&\, C_{in})$
$\mid (A \,\&\, \sim B \,\&\, C_{in})$
$\mid (A \,\&\, B \,\&\, \sim C_{in})$
$\mid (A \,\&\, B \,\&\, C_{in})$

Add two bits and carry-in,
produce one-bit sum and carry-out.

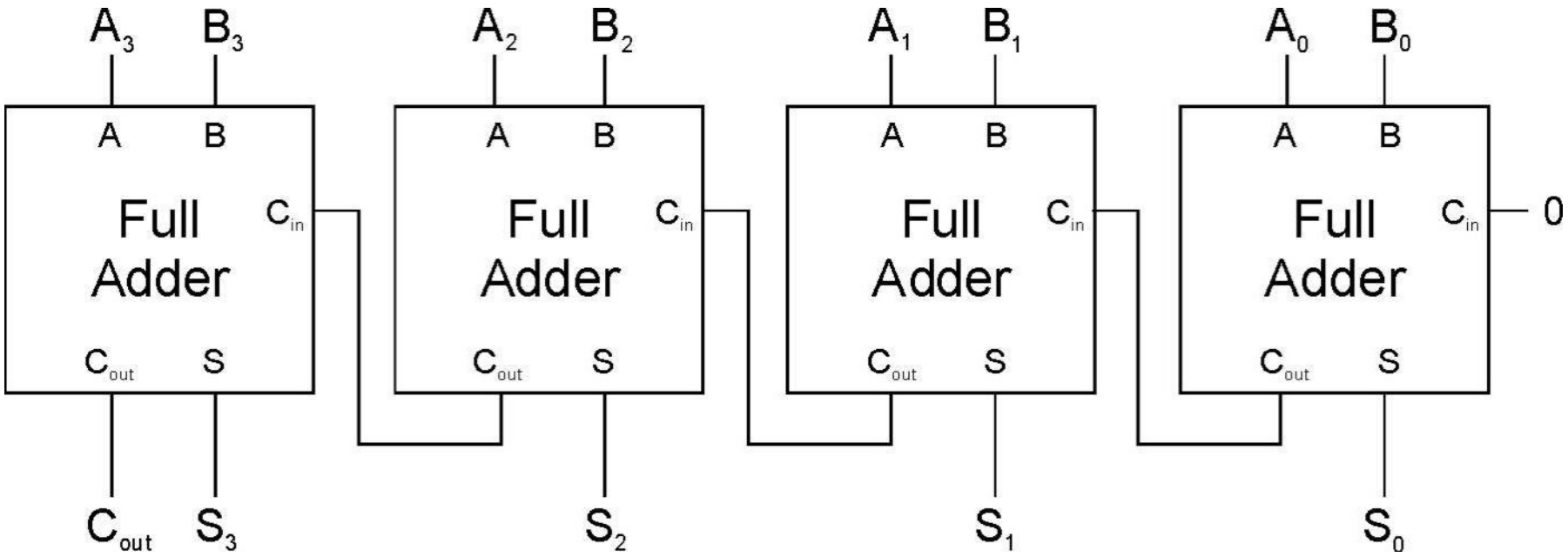


AND Gates

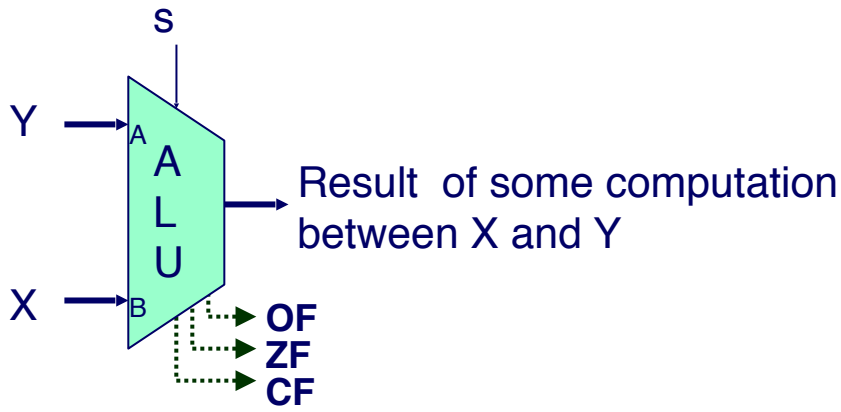
OR Gates

Recall: Four-bit Adder

- Ripple-carry Adder
  - Simple, but performance linear to bit width
- Carry look-ahead adder (CLA)
  - Generate all carriers simultaneously



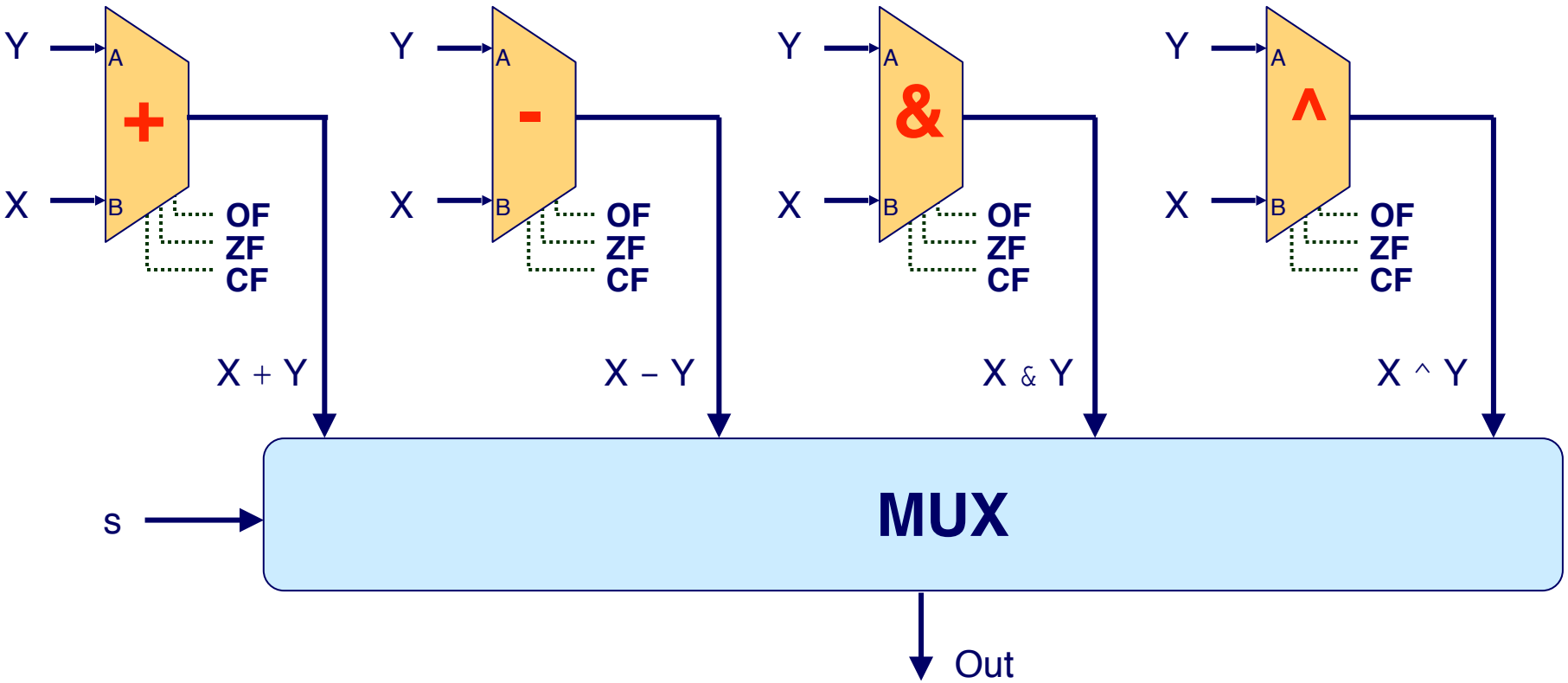
Arithmetic Logic Unit



- An ALU performs multiple kinds of computations.
- The actual computation depends on the selection signal s.
- Also sets the condition codes (status flags)
- For instance:
    - X + Y when s == 00
    - X - Y when s == 01
    - X & Y when s == 10
    - X ^ Y when s == 11
- How can this ALU be implemented?

Arithmetic Logic Unit

- Implement 4 different circuits, one for each operation.
- Then use a MUX to select the results



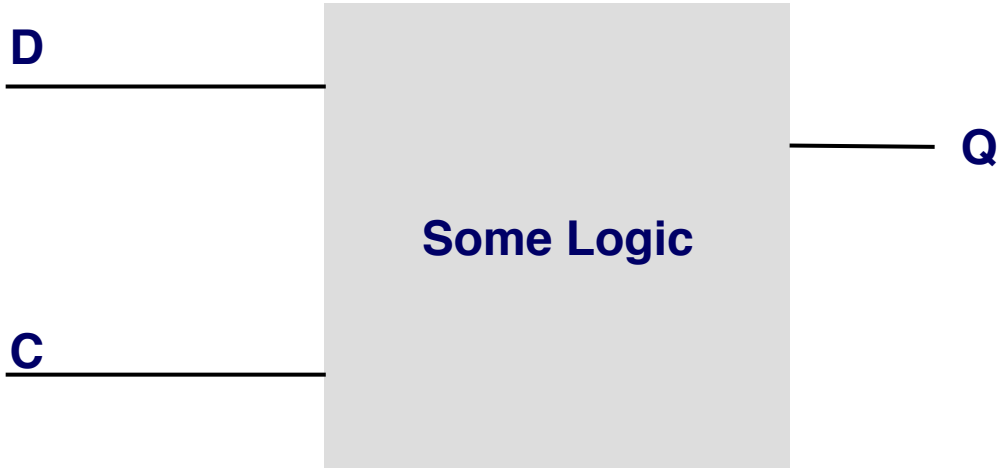
Today: Circuits Basics

- Transistors
- Circuits for computations
- Circuits for storing data

The Need for Storing Bits

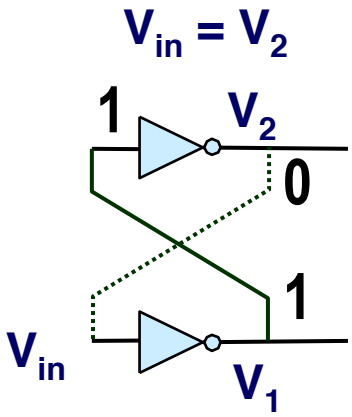
- Assembly programs set architecture (processor) states.
  - Register File
  - Status Flags
  - Memory
  - Program Counter
- Every state is essentially some bits that are stored/loaded.
- Think of the program execution as an FSM.
- The hardware must provide mechanisms to load and store bits.
- There are many different ways to store bits. They have trade-offs.

Build a 1-Bit Storage

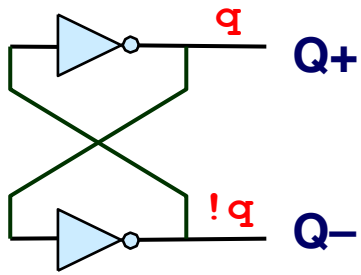


- What we would like:
  - D is the data we want to store (0 or 1)
  - C is the control signal
    - When C is 1, Q becomes D (i.e., storing the data)
    - When C is 0, Q doesn't change with D (data stored)

Bitstable Element



Bistable Element

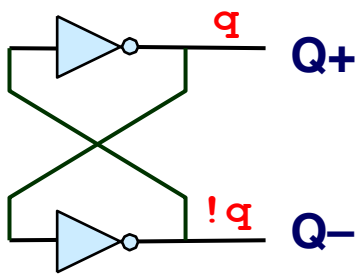


q = 0 or 1

Q+ ***continuously*** outputs q.

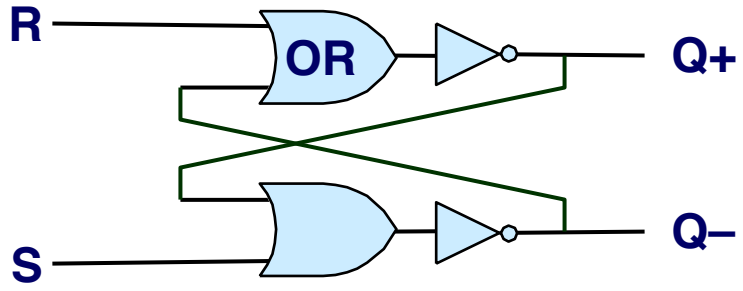
Storing and Accessing 1 Bit

Bistable Element

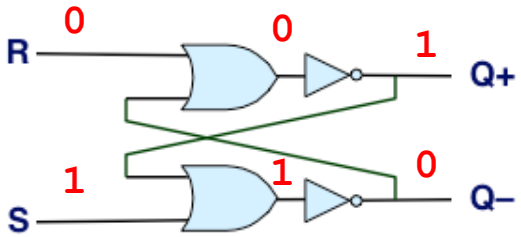


q = 0 or 1

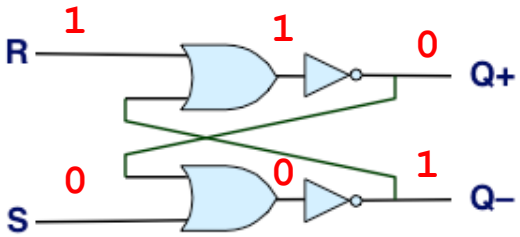
R-S Latch



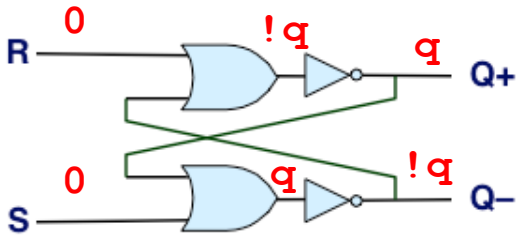
Setting Q+ to 1



Setting Q+ to 0



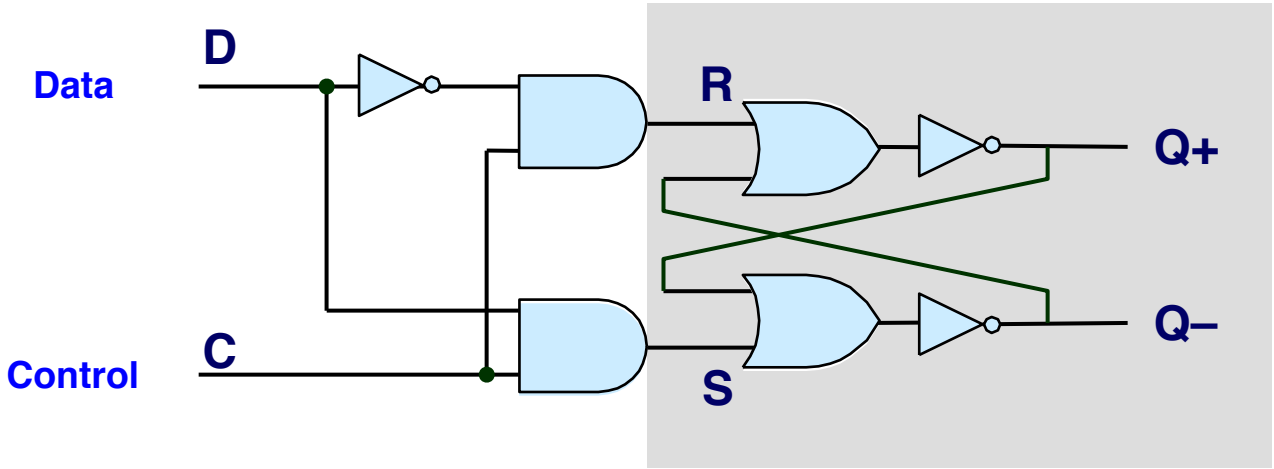
Q+ value unchanged i.e., stored!



If R and S are different, Q+ is the same as S

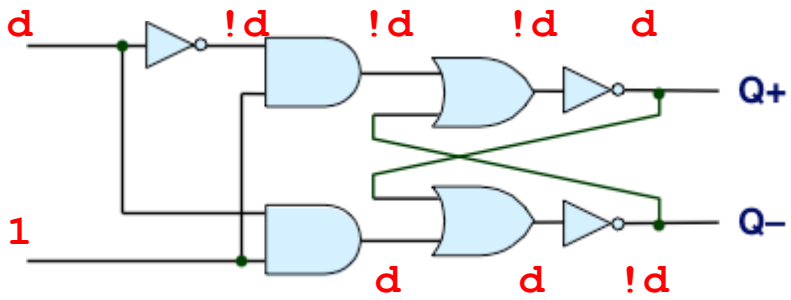
Building on top of R-S Latch

D Latch



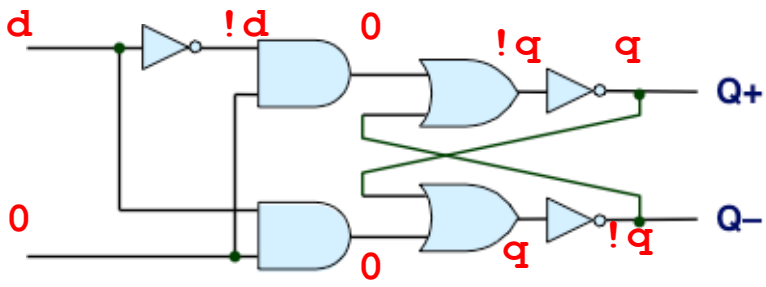
If R and S are different, Q+ is the same as S

Storing Data (Latching)



Q+ will continuously change as d changes

Holding Data



Q+ doesn't change with d