Hi! We'd love to meet you

- Please send me welcome slides!
- I'd also love to hear about cool clubs or activities on campus
  - Opportunity to promote!
    - *first-come, first-served; presented with my discretion; etc etc
- Or just your favorite music ☺
  - Today: *Songs we Sing*, by Matt Costa

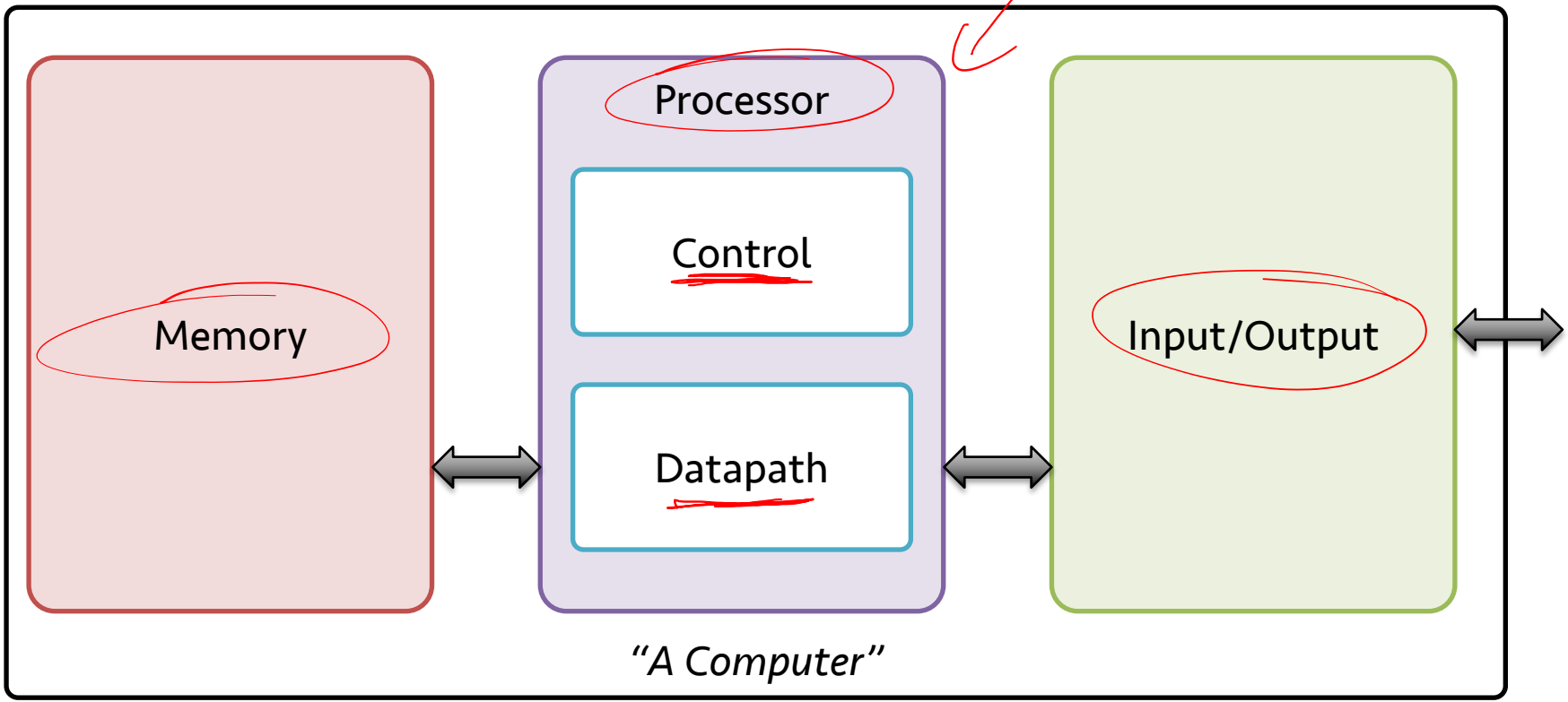
CSE 141: Introduction to Computer Architecture

The Single Cycle Machine

Announcements

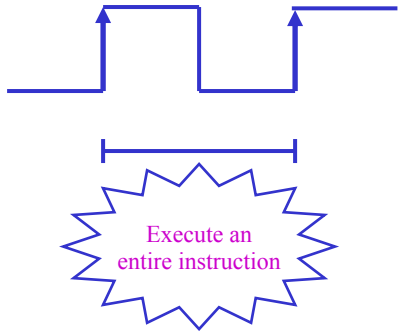
- Midterm Logistics
  - Tuesday, November 10th
  - Given via Canvas
  - 80 minute timed exam [set aside 90 minutes!]
  - 12 hour window (8am – 8pm Pacific Standard Time [UTC -8])
    - **Note:** This is after Daylight Savings Time (i.e. currently we are UTC -7)
  - Class on Monday, November 9th will be review session (for both sections)
- Homework 1 due last night; Homework 2 out now
- Piazza is not a public homework forum
  - *General* questions can be public, *specific* homework questions should not be
  - Use **private** questions for "asynchronous, digital office hours"

Zooming out for a moment…
The major building blocks of a computer



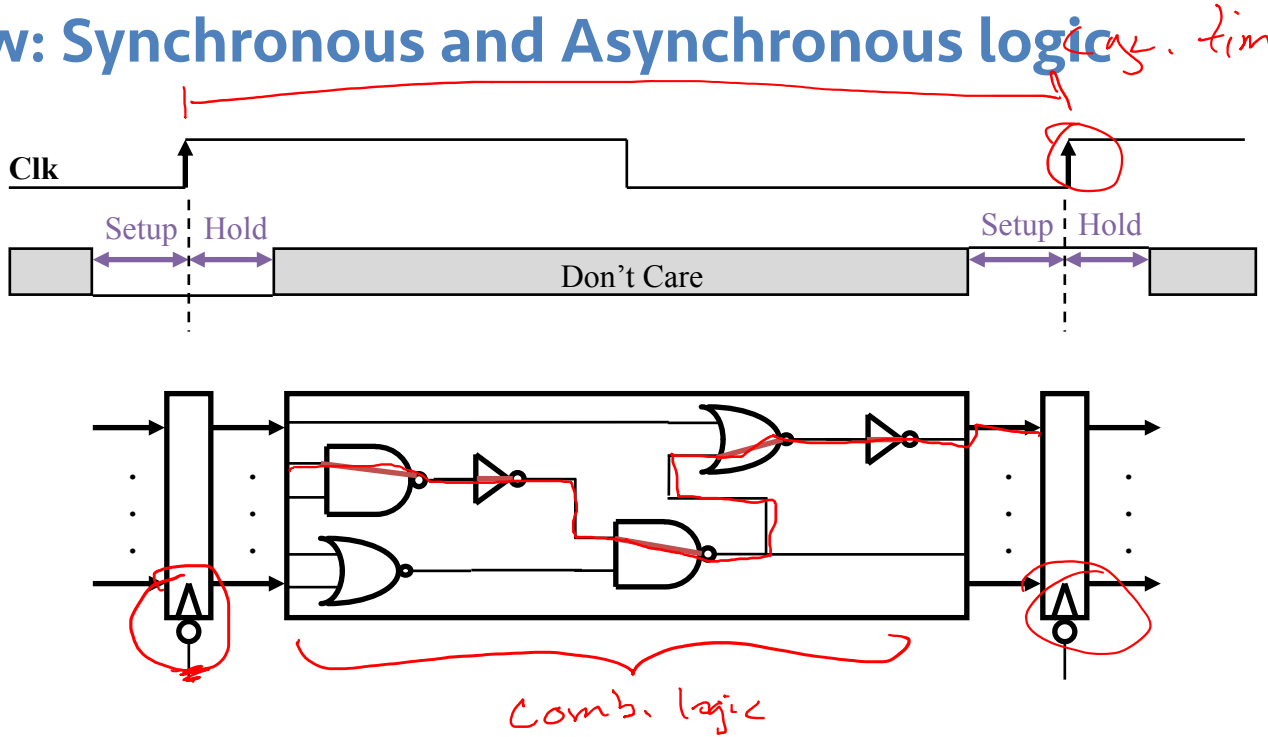
The Big Picture: The Performance Perspective

- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction
- Starting today:
  - Single cycle processor:
    - Advantage: One clock cycle per instruction
    - Disadvantage: long cycle time



- ET = Insts * CPI * Cycle Time

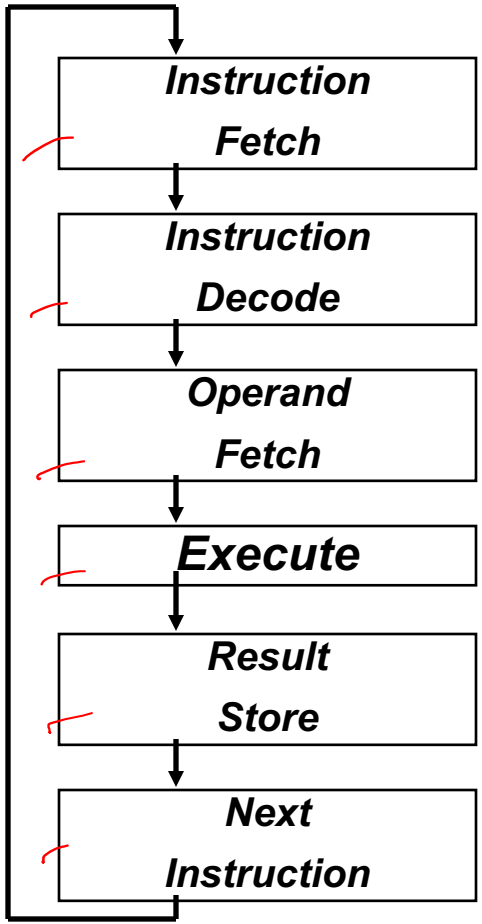
Review: Synchronous and Asynchronous logic



- All storage elements are clocked by the same clock edge

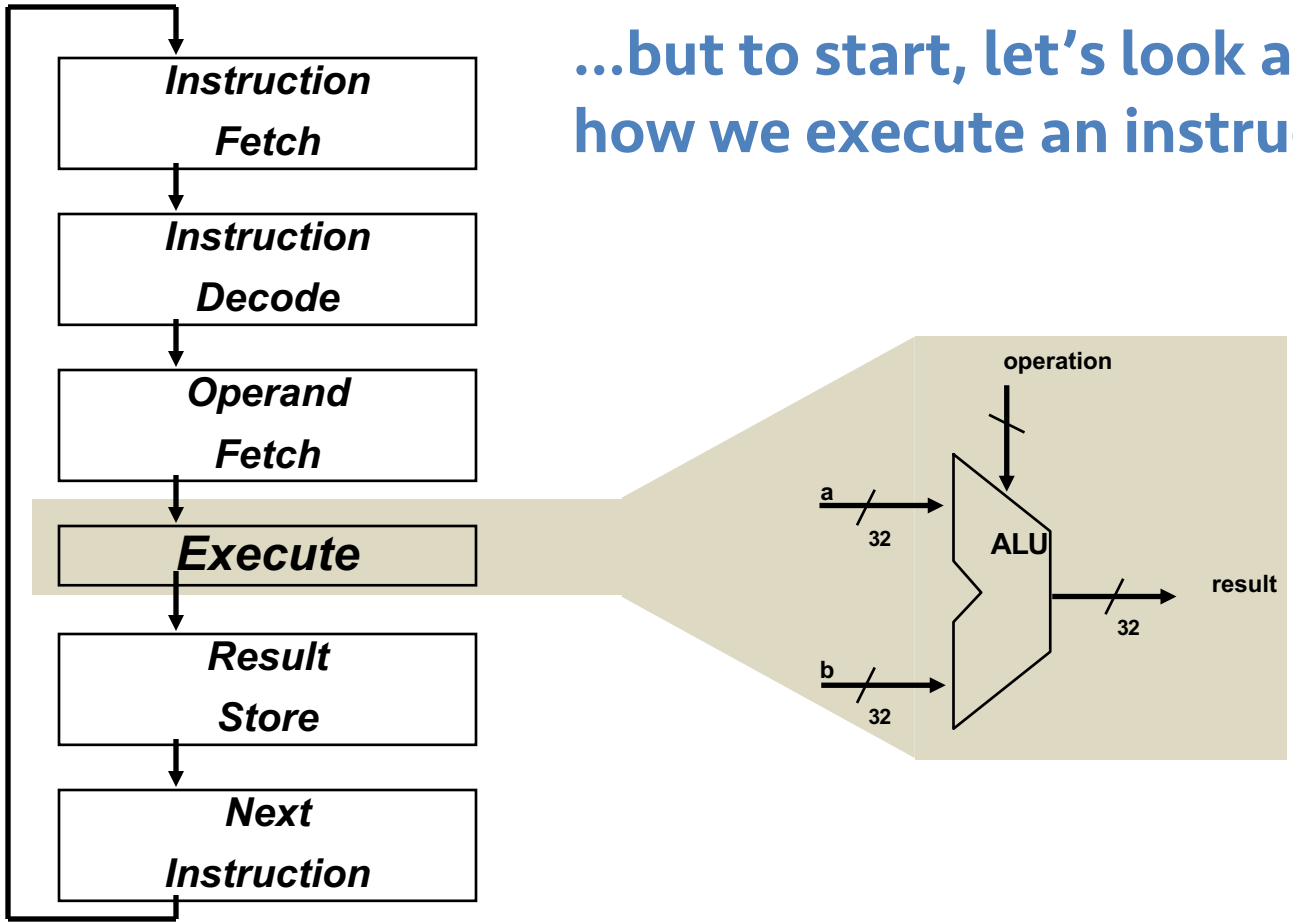
The Processor: Datapath & Control

- We're ready to look at a simplified MIPS with only:
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq`
- Generic Implementation:
  - use the program counter (PC) to supply instruction address
  - get the instruction from memory
  - read registers
  - use the instruction to decide exactly what to do



Recall…

Computing is much more than just executing instructions!



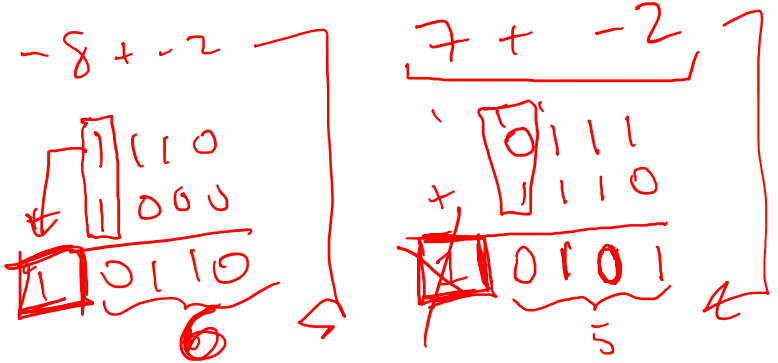
...but to start, let's look at how we execute an instruction

Recall: 2's complement

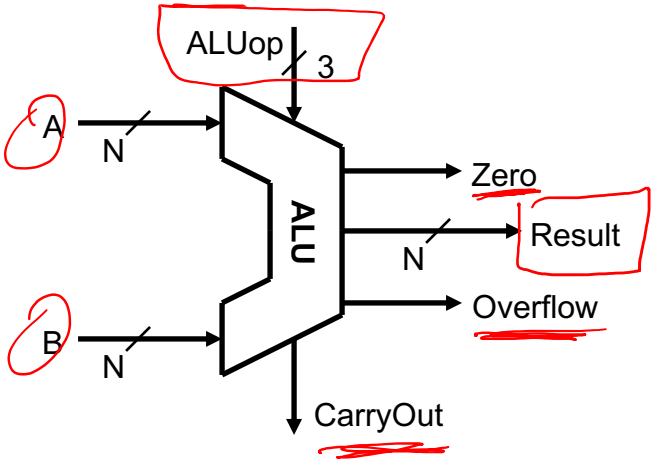
- Need a ***number system*** that provides
  - obvious representation of 0,1,2...
  - uses an adder for both unsigned and signed addition
  - single value of 0
  - equal coverage of positive and negative numbers
  - easy detection of sign
  - easy negation

-8 → 7

binary	unsigned	signed
0001	1	1
1110	14	-2
1000	8	-8



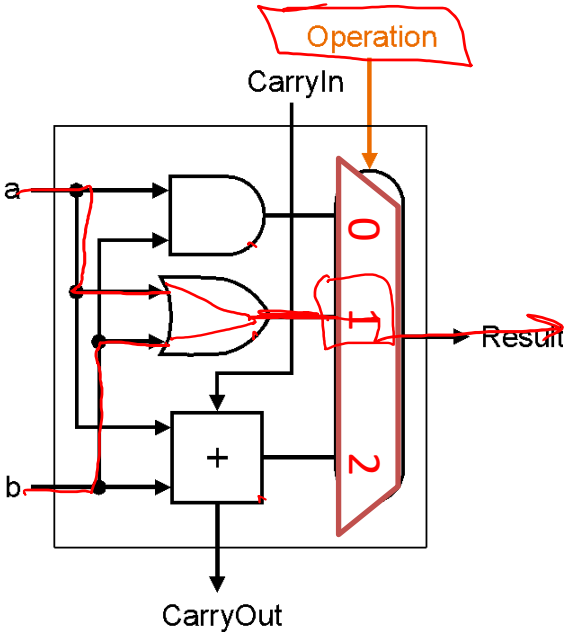
“Arithmetic Logic Units” are the computing part of computers — how do they work?



ALU Control Lines (ALUop)	Function
000	And
001	Or
010	Add
110	Subtract
111	Set-on-less-than

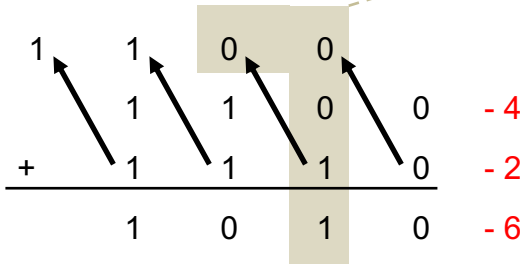
Start small: A one-bit ALU

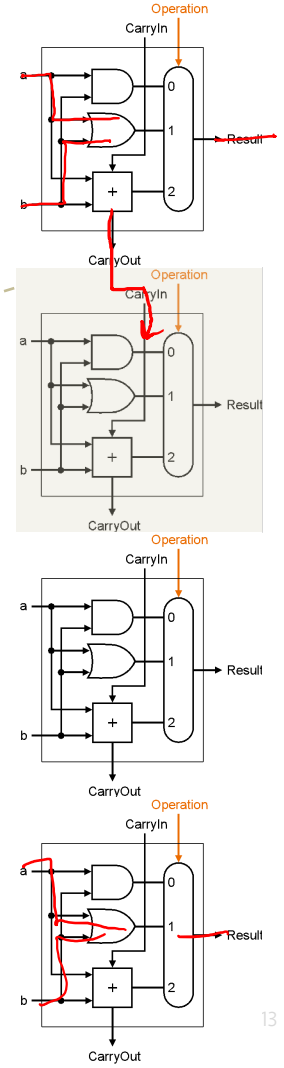
- This 1-bit ALU will perform AND, OR, and ADD



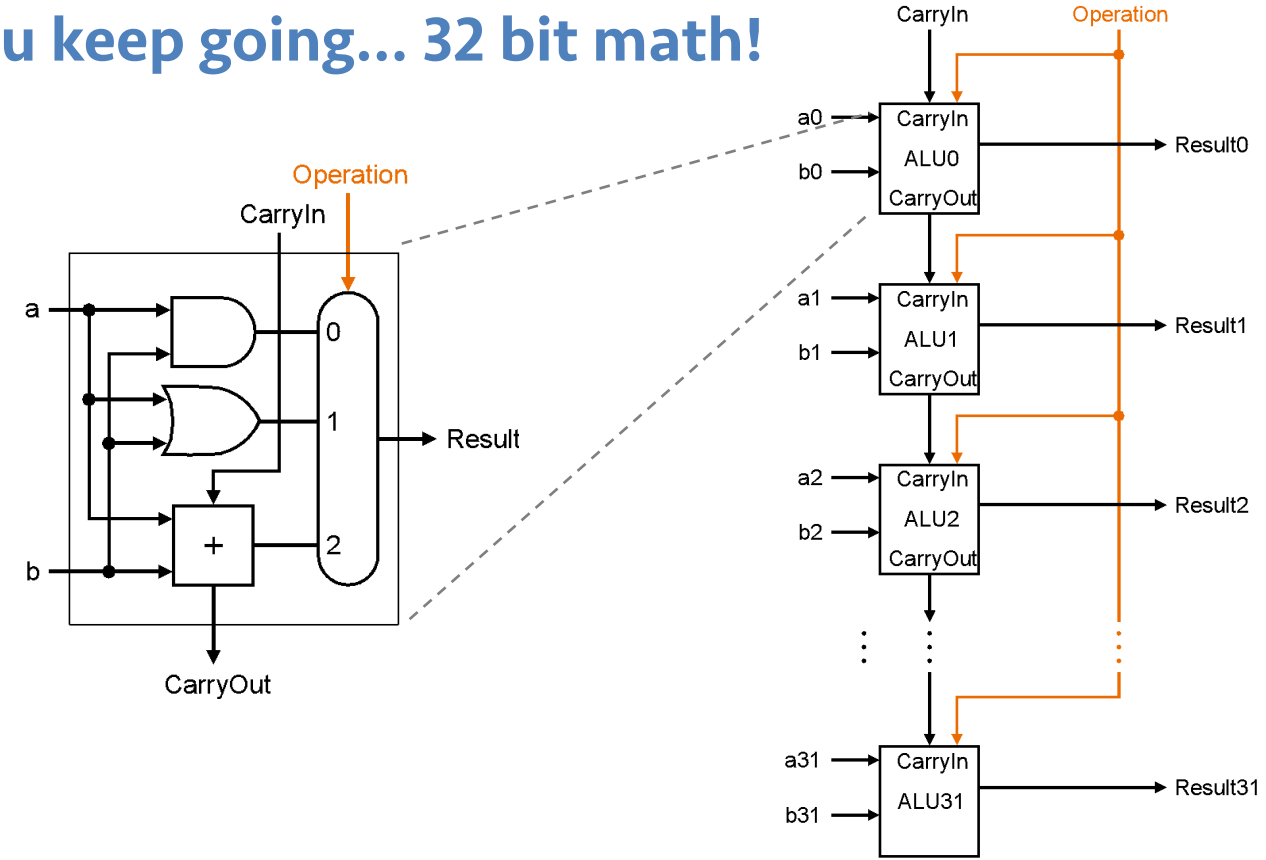
Recall: Binary addition works just like “normal” (base 10), but you end up “carrying” more often

- A 4-bit ALU can be made from four 1-bit ALUs





And if you keep going... 32 bit math!

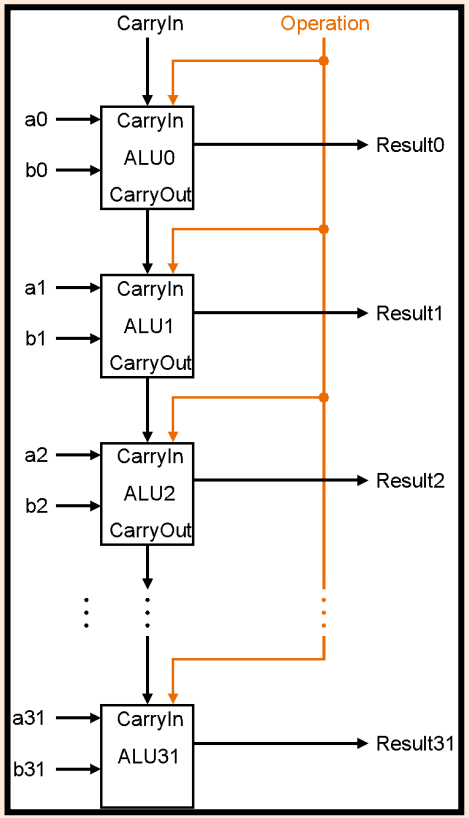


Hint: A-B is the same as A + (-B)

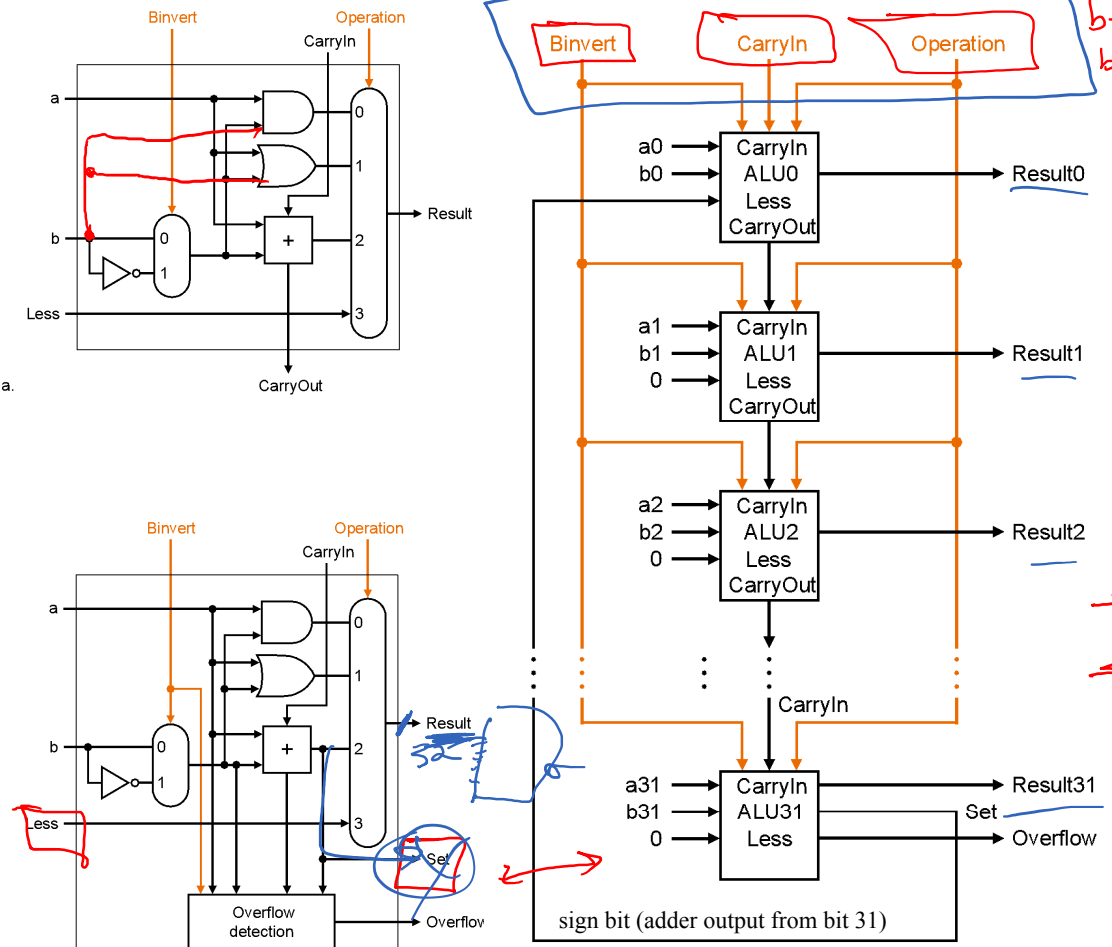
Poll Q: We'd like to implement a means of doing A-B (subtract) but with only minor changes to our hardware. How?

1. Provide an option to use bitwise NOT A
2. Provide an option to use bitwise NOT B
3. Provide an option to use bitwise A XOR B
4. Provide an option to use 0 instead of the first $Carry_{In}$
5. Provide an option to use 1 instead of the first $Carry_{In}$

Selection	Choices
A	1 alone
B	Both 1 and 2
C	Both 3 and 4
D	Both 2 and 5
E	None of the above



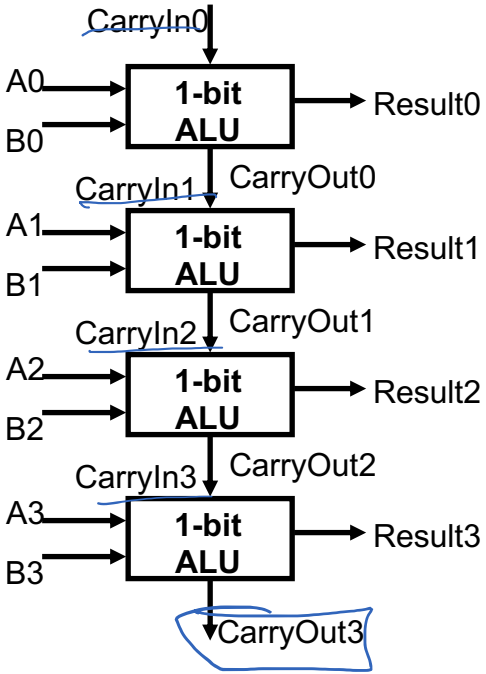
The full ALU

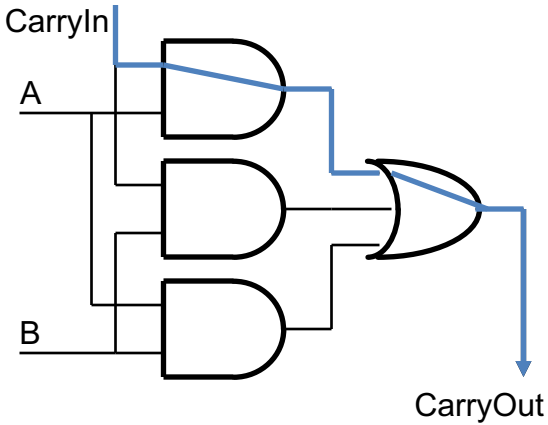


	B_{invert}	$Carry_{In}$	Oper-ation
and	0	X	0
or			
add			
sub	1	1	2
beq	1	1	2
slt	1	1	3

The Disadvantage of Ripple Carry

- The adder we just built is called a "Ripple Carry Adder"
  - The carry bit may have to propagate from LSB to MSB
  - Worst case delay for an N-bit RC adder: 2N-gate delay





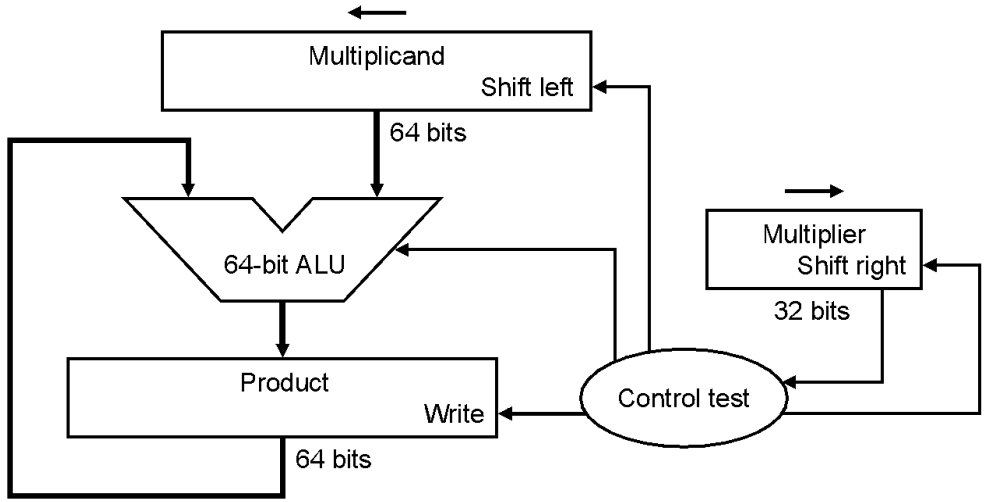
The point: ripple carry adders are slow. Faster addition schemes are possible that *accelerate* the movement of the carry from one end to the other. Optimizing this is *digital logic* (CSE 140).

Why doesn't our (simplified) single-cycle machine support multiplication or division?

We're ready to look at a simplified MIPS with only:
- memory-reference instructions: `lw, sw`
- arithmetic-logical instructions: `add, sub, and, or, slt`
- control flow instructions: `beq`

- How does a computer multiply?
  - How do you multiply?

```
   123
 x 321
   123
  246
 369
 39483
```

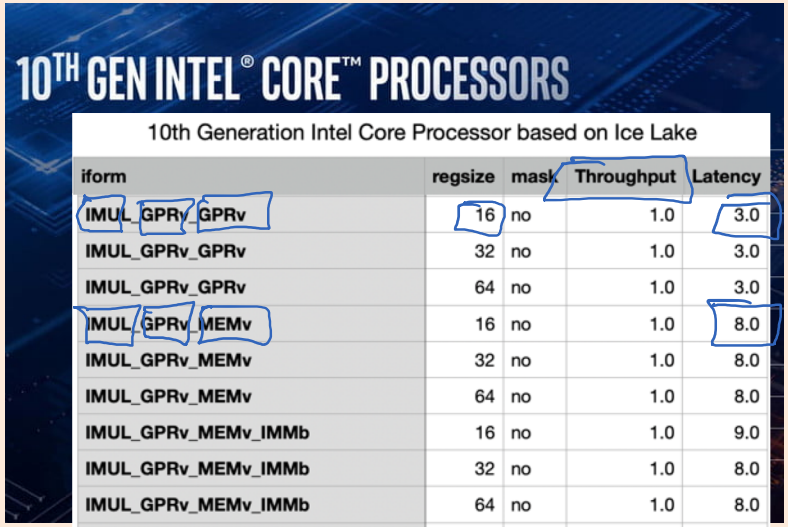


The point: Multiplication (and division) is a lot of work to try to do in a single cycle

Poll Q: Which of these real-world processors supports single-cycle multiply?

A) "Biggest, best" Intel [core i7]
   - ~$500

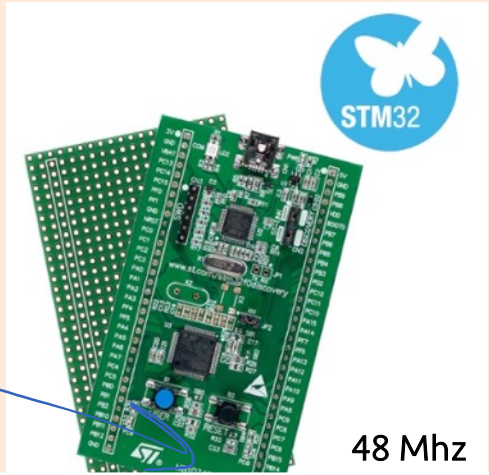
B) "Smallest" ARM [Cortex M0]
   - ~$0.50



10th Generation Intel Core Processor based on Ice Lake

iform	regsize	mask	Throughput	Latency
IMUL_GPRv_GPRv	16	no	1.0	3.0
IMUL_GPRv_GPRv	32	no	1.0	3.0
IMUL_GPRv_GPRv	64	no	1.0	3.0
IMUL_GPRv_MEMv	16	no	1.0	8.0
IMUL_GPRv_MEMv	32	no	1.0	8.0
IMUL_GPRv_MEMv	64	no	1.0	8.0
IMUL_GPRv_MEMv_IMMb	16	no	1.0	9.0
IMUL_GPRv_MEMv_IMMb	32	no	1.0	8.0
IMUL_GPRv_MEMv_IMMb	64	no	1.0	8.0
IMUL_GPRv_MEMv_IMMz	16	no	1.0	9.0
IMUL_GPRv_MEMv_IMMz	32	no	1.0	8.0
IMUL_GPRv_MEMv_IMMz	64	no	1.0	8.0

100x



48 Mhz

The Cortex-M0 processor is built on a highly area and power optimized 32-bit processor core, with a 3-stage pipeline von Neumann architecture. The processor delivers exceptional energy efficiency through a small but powerful instruction set and extensively optimized design, providing high-end processing hardware including a single-cycle multiplier.

Modern Concerns about Execute

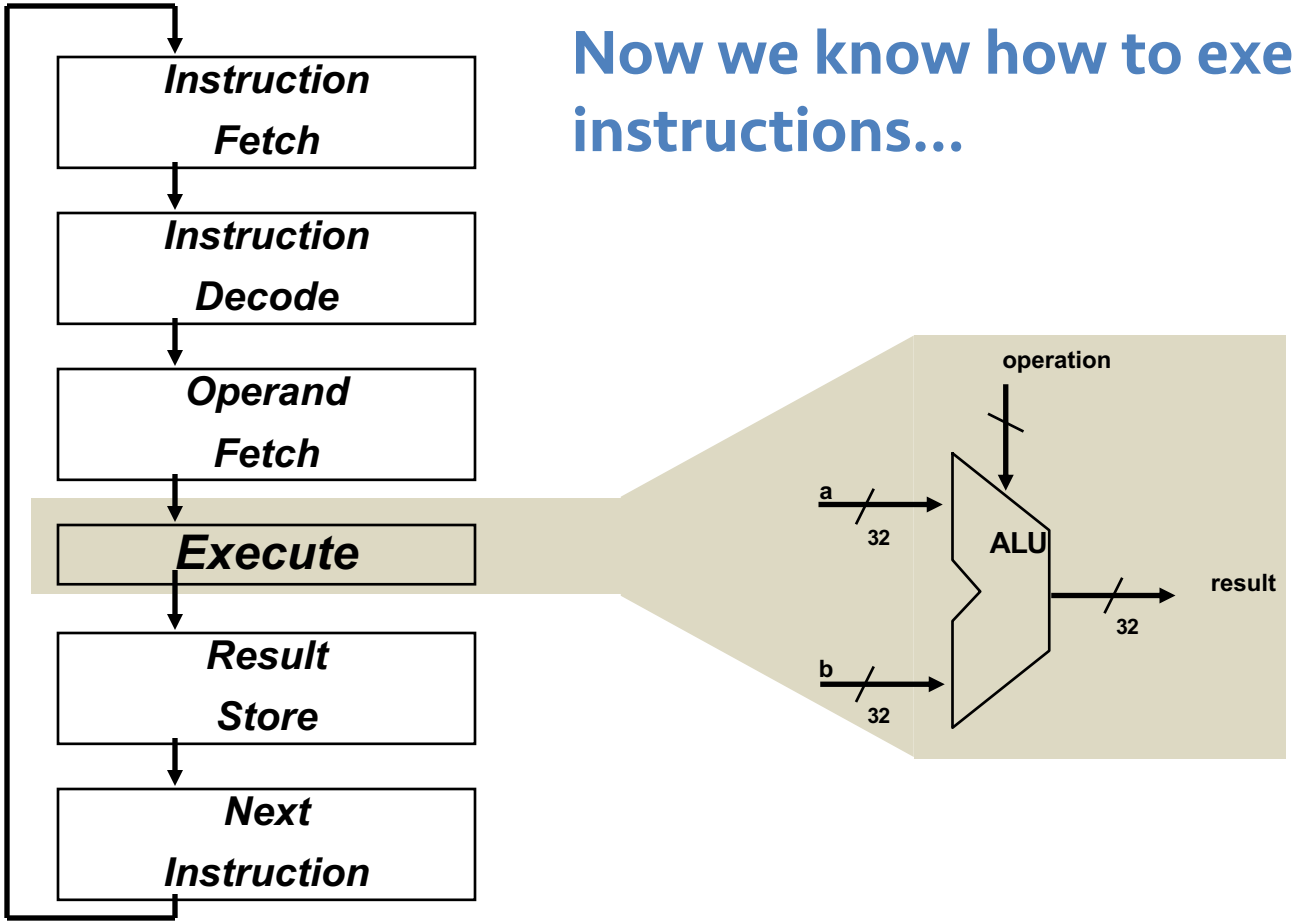
(aka, why is no one angry that an i7 can't do single-cycle multiply?)

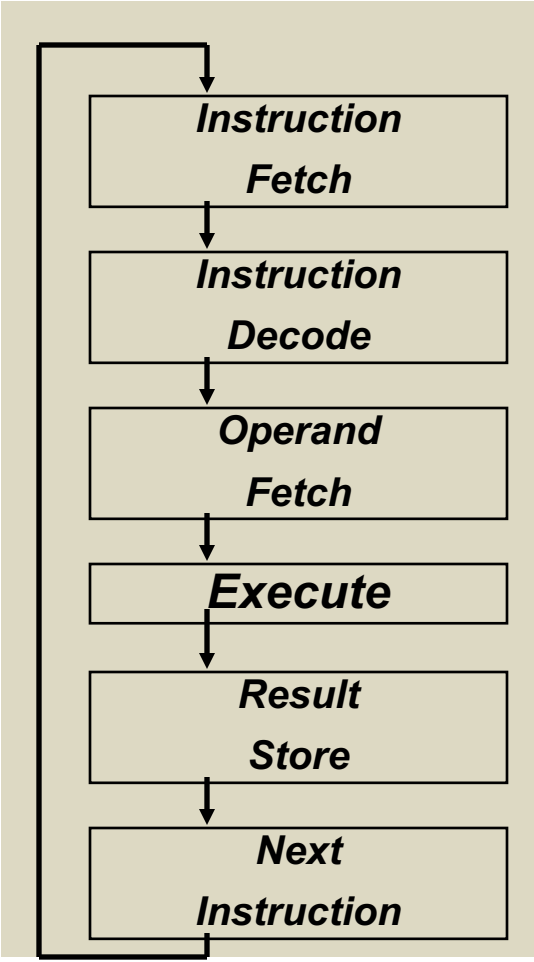
- Hardware designers have done an excellent job optimizing multiply/FP hardware, but additions are still faster, than, say multiply. Divides are even slower and have other problems.

- More complex topics in later lectures will show how multiply/FP/divide may not be on the "critical path" and hence may not hurt performance as much as expected.

- More recent years have taught us that even "slow" multiply is not nearly as important as cache/memory issues we'll discuss in later lessons.

Now we know how to execute instructions…





…so let's look at the rest of the machine!

Our previous view of a computer had no organization

- From Part I…