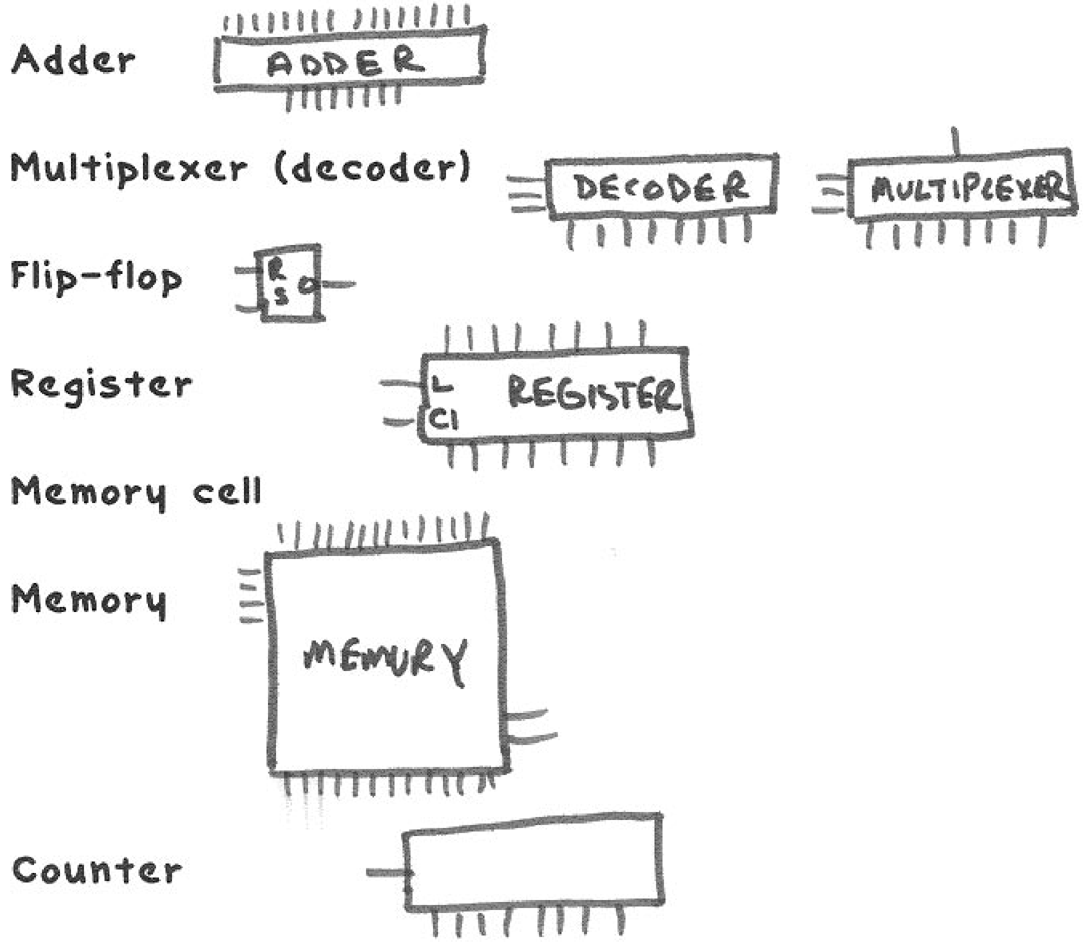
CS 126 Lecture A5: Computer Architecture

Outline

- **Introduction**
- Some basics
- Single-cycle TOY design
- Multicycle TOY design
- Conclusions

What We Have


Adder
ADDER
Multiplexer (decoder)
DECODER
MULTIPLEXER
Flip-flop
R
S
Q
Register
L
CI
REGISTER
Memory cell
Memory
MEMURY
Counter

What We Want to Do

```
repeat
    fetch instruction;
    update PC;
    decode instruction;
    execute instruction;
until halt signal
```

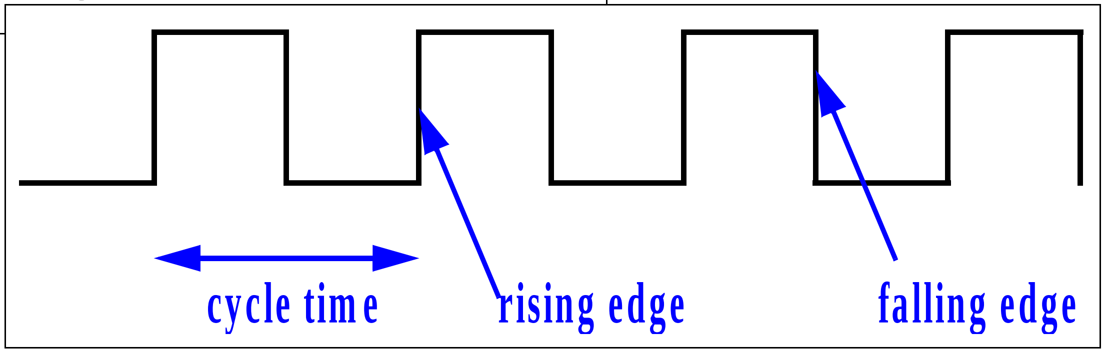
- Remember the TOY simulator written in C?
- Now it's time to use the components we have to implement this loop in **hardware**!

Outline

- ~~Introduction~~
- **Some basics**
- Single-cycle TOY design
- Multicycle TOY design
- Conclusions

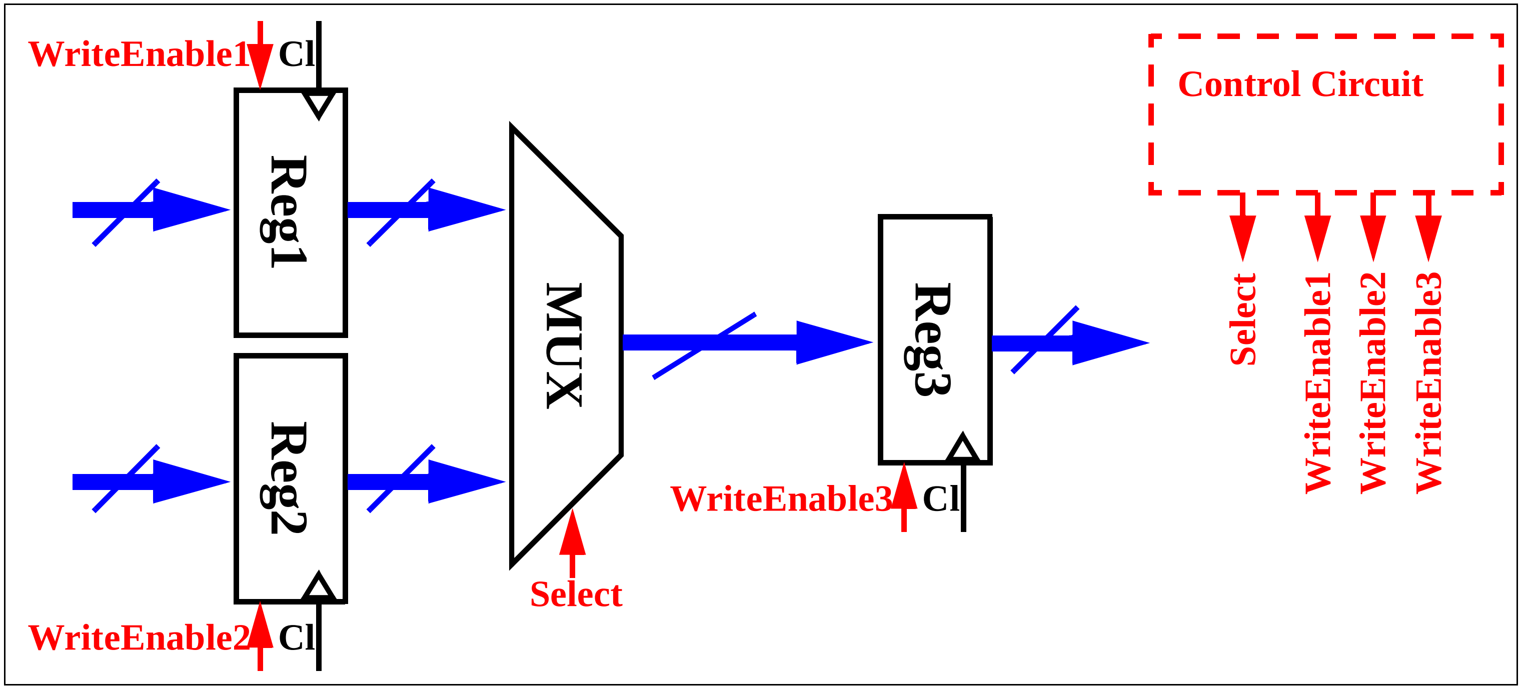
Single Cycle vs. Multicycle Design

```
repeat
    fetch instruction;
    update PC;
    decode instruction;
    execute instruction;
until halt signal
```



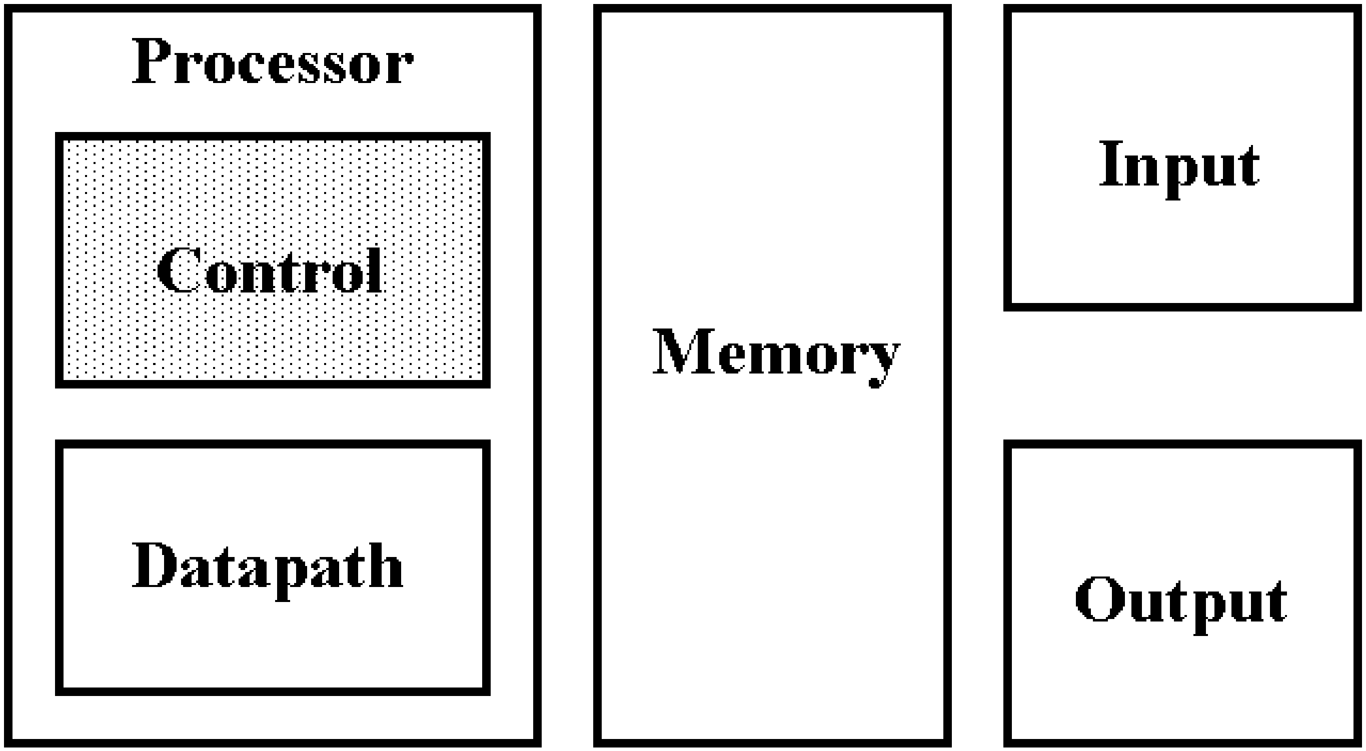
- Single cycle design: each iteration is completed within one clock cycle, long cycles, simple
- Multi-cycle design: each iteration is broken down into multiple clock cycles: short cycles, more complex
- More tradeoffs later

Datapath and Control: Definition by Example



- Blue: datapath, Red: control signals
- Control circuit decides how to set Select and whether to enable WriteEnable3
- When clock ticks
  - One of Reg1 or Reg2 gets copied to Reg3 if WriteEnable3 is on
  - Nothing gets copied to Reg3 if WriteEnable3 is off

The Big Picture

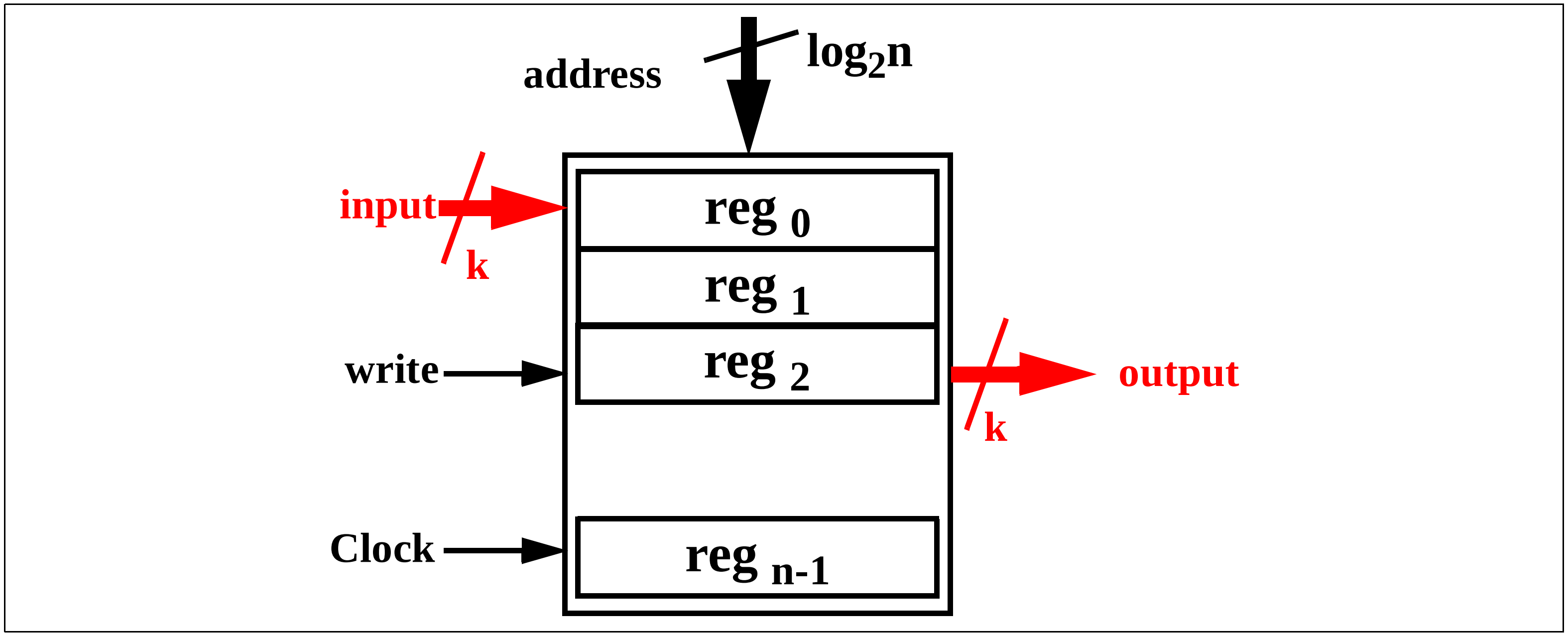


- The five classic components of a computer

Steps Towards Designing a Processor

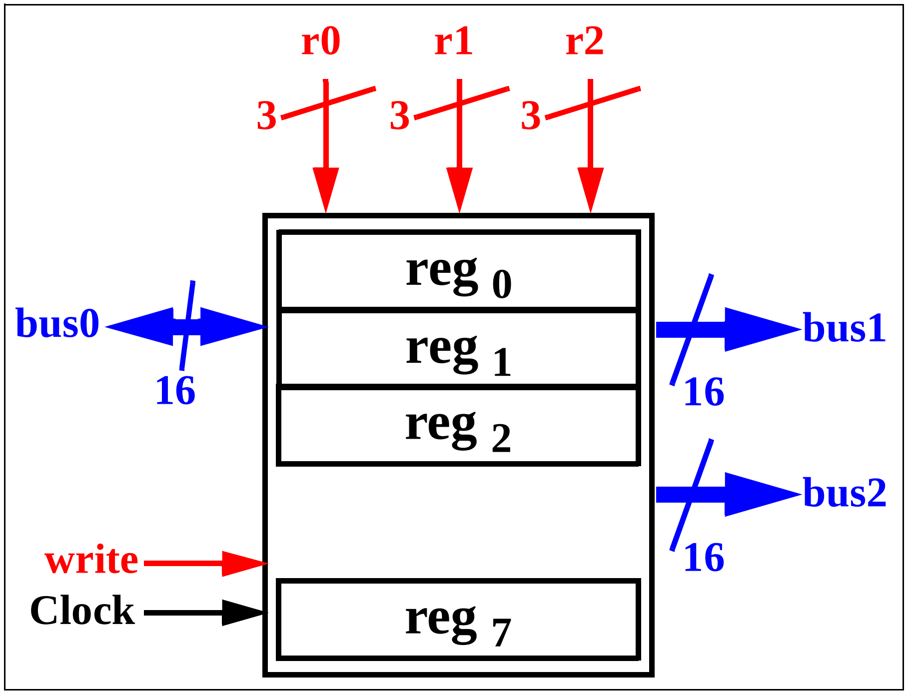
- Analyze instruction set architecture (ISA) and understand datapath requirements
- Select set of datapath components and establish clocking methodology
- Assemble datapath to meet ISA requirements
- Analyze how to implement each instruction to determine the setting of various control signals
- Assemble the control logic

Review: Register File (From Last Lecture)



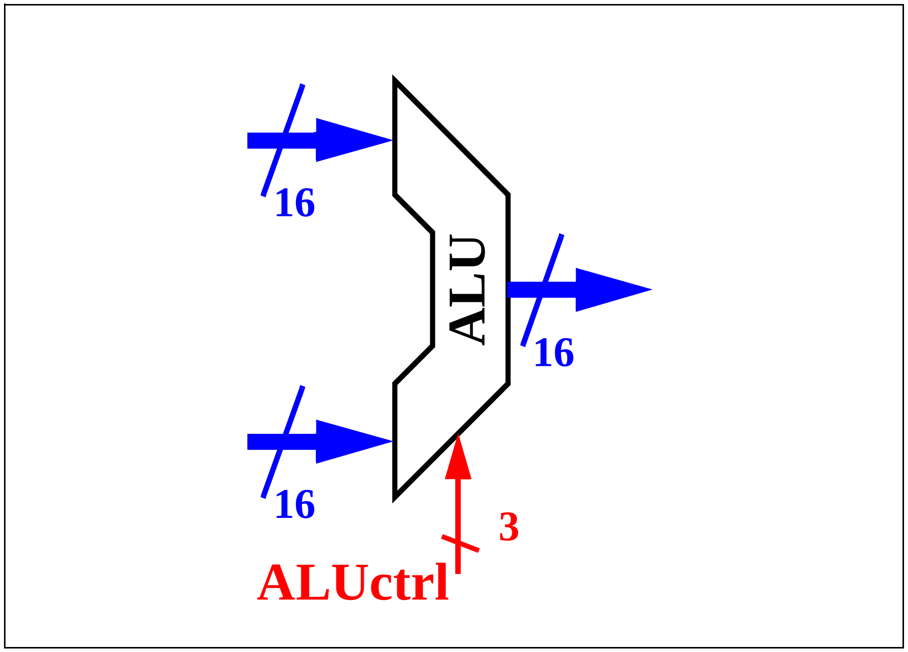
- Register file of k-bit words
- One address port, so can't read and write in the same clock cycle

What We Have (cont.): TOY Register File



- 8 general purpose registers
- 2 16-bit output busses, 1 16-bit input bus
- r1, r2 (3-bit numbers) specifies which registers go on bus1, 2
- r0 (3-bit) specifies which registers to receive input data when write enabled at clock pulse; when not write-enabled, the named register's value appears on bus 0

What We Have (cont.): TOY ALU



- We have learned about an adder. Generalize it to an ALU.
- Two 16-bit inputs, one 16-bit output
- A 3-bit control specifies which arithmetic or logic operation to perform (+ - * ^ & >> <<)

Outline

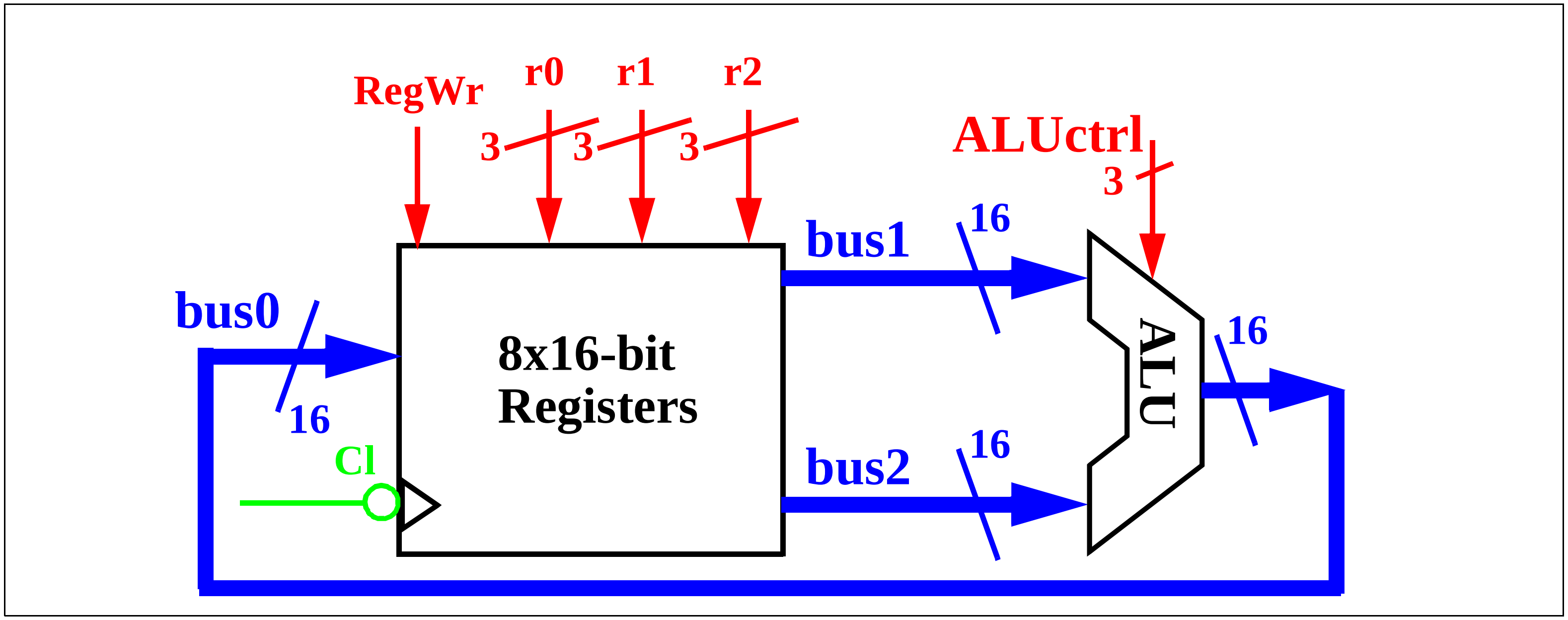
- ~~Introduction~~
- ~~Some basics~~
- **Single-cycle TOY design**
  - **Datapath design**
  - **Control design**
- Multicycle TOY design
- Conclusions

TOY Datapath Components

```
repeat
    fetch instruction;
    perform arithmetic operation;
    access memory if necessary;
    write back to register if necessary;
until halt signal
```

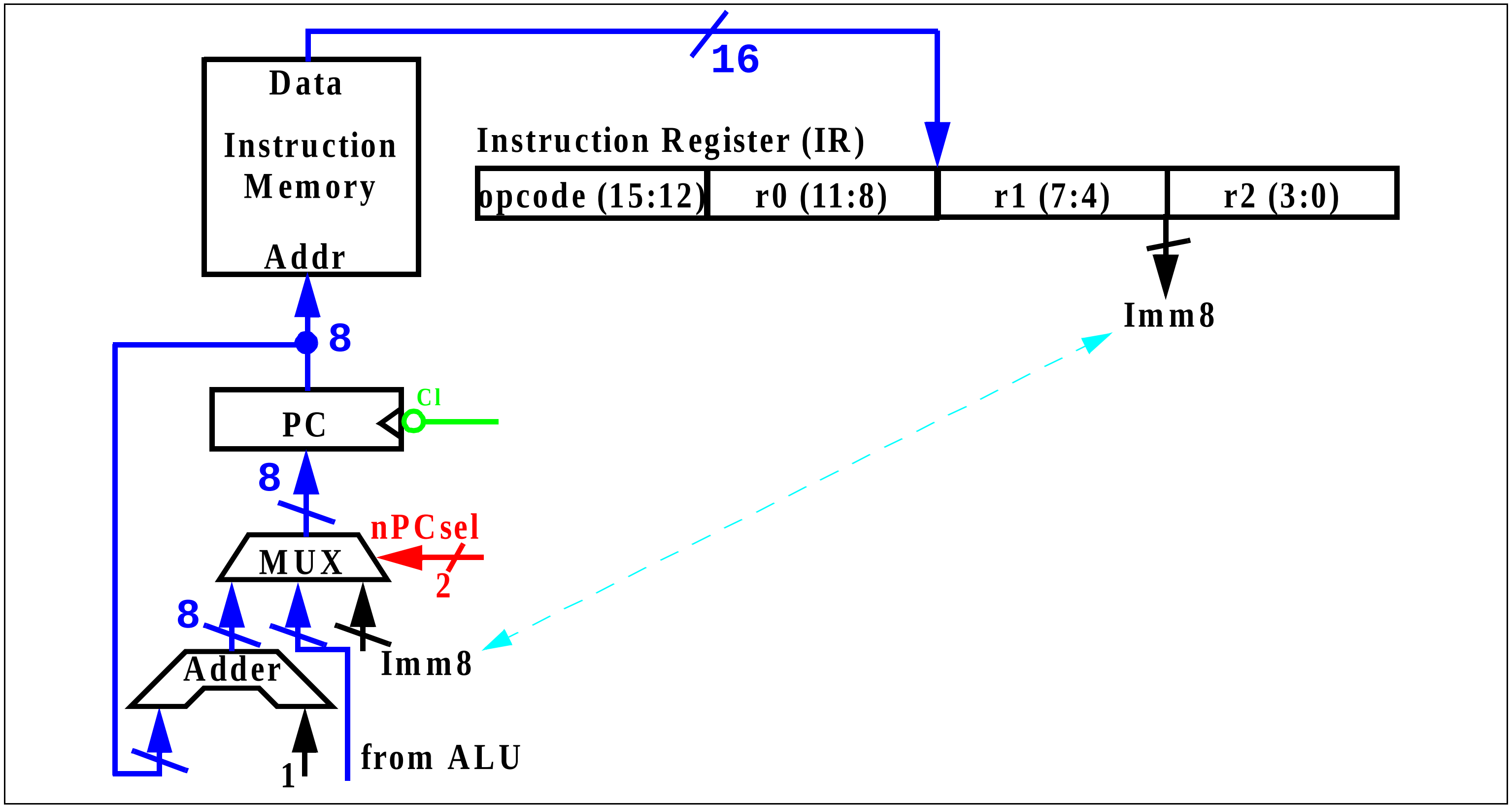
- Refine the simulator code to be more specific
- Each of these four lines will be handled by a piece of hardware
  - Instruction fetch
  - Arithmetic (execution)
  - Memory
  - Write back
- We will assemble them one at a time, and assemble all four together at the end
- <u>Caveat: I'm leaving out a few instructions as exercises</u>

TOY Arithmetic (Execution) Data Path



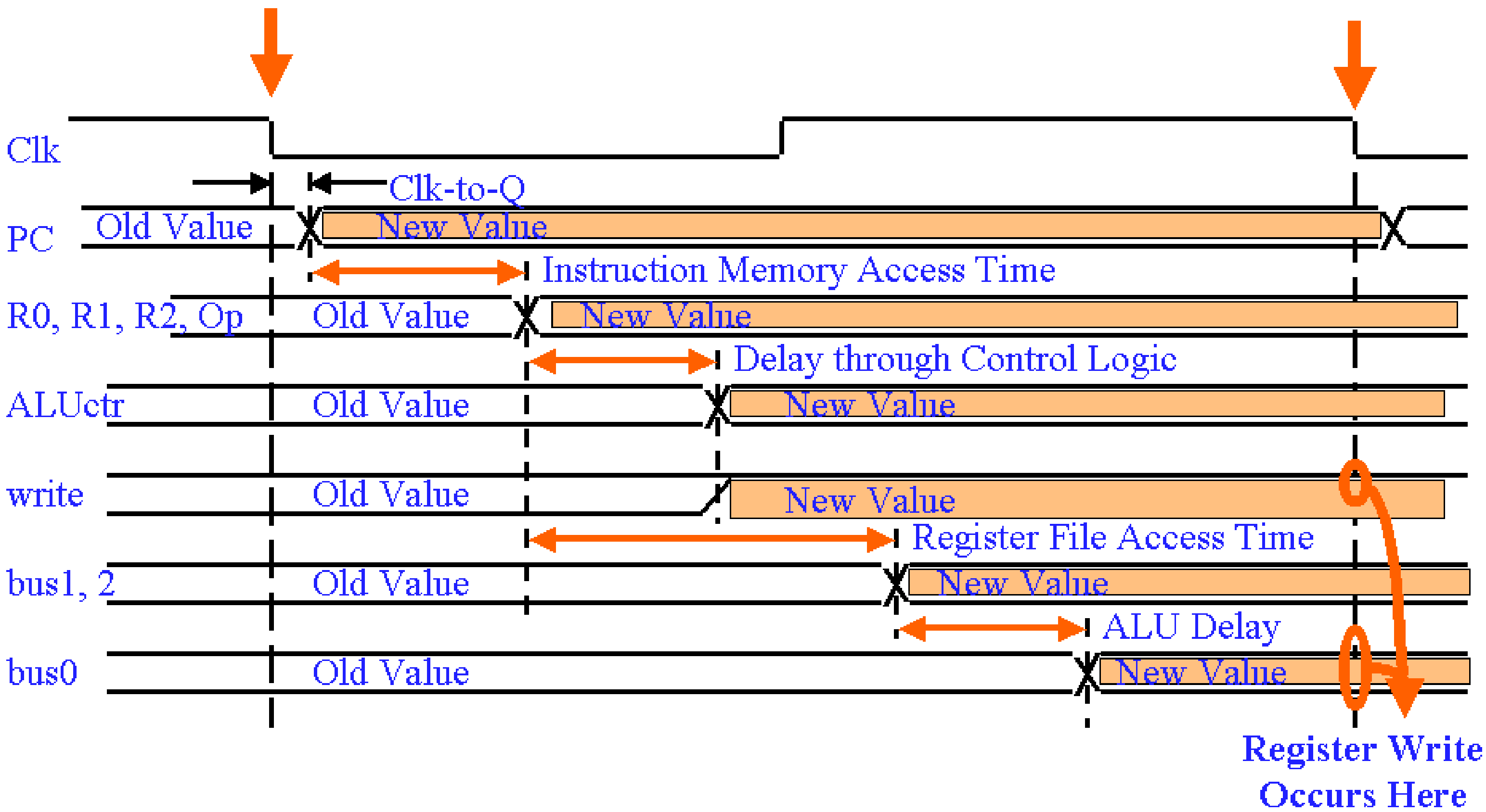
- Blue: datapath, Red: control signals
- (Part of) Implementation of TOY instruction:
  `r0 = r1 + r2`
- r0, r1, r2 control signals come straight from instruction, more on control later
- **Clock controls when write back occurs**
- **Reads behave as combinational logic: result valid after delay**

TOY Instruction Fetch Unit

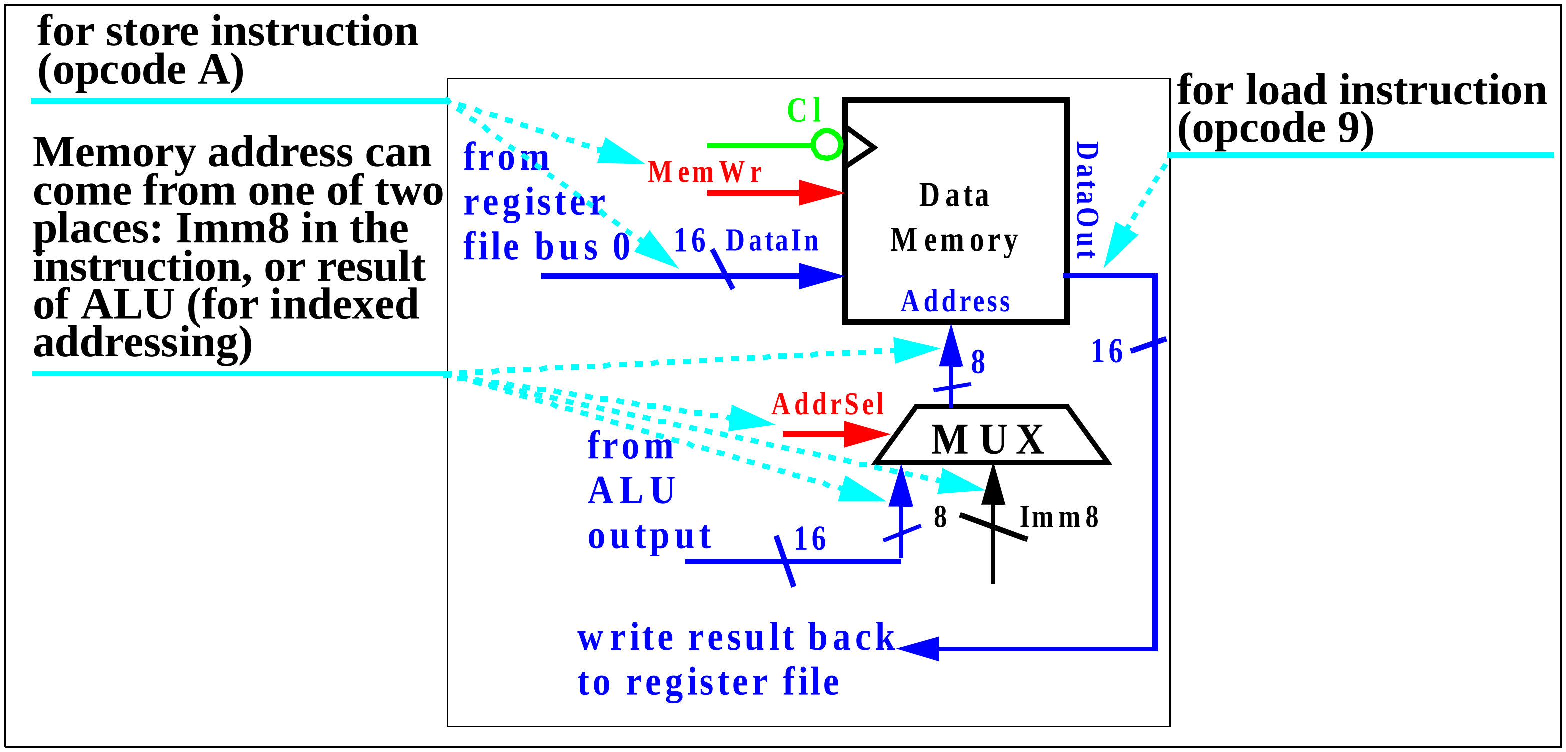


- Key question: which instruction to fetch
  - If jump, then fetch the jump target (which is in instruction itself)
  - Otherwise, fetch the next instruction

Timing Demo: Putting Instruction Fetch and Add Together

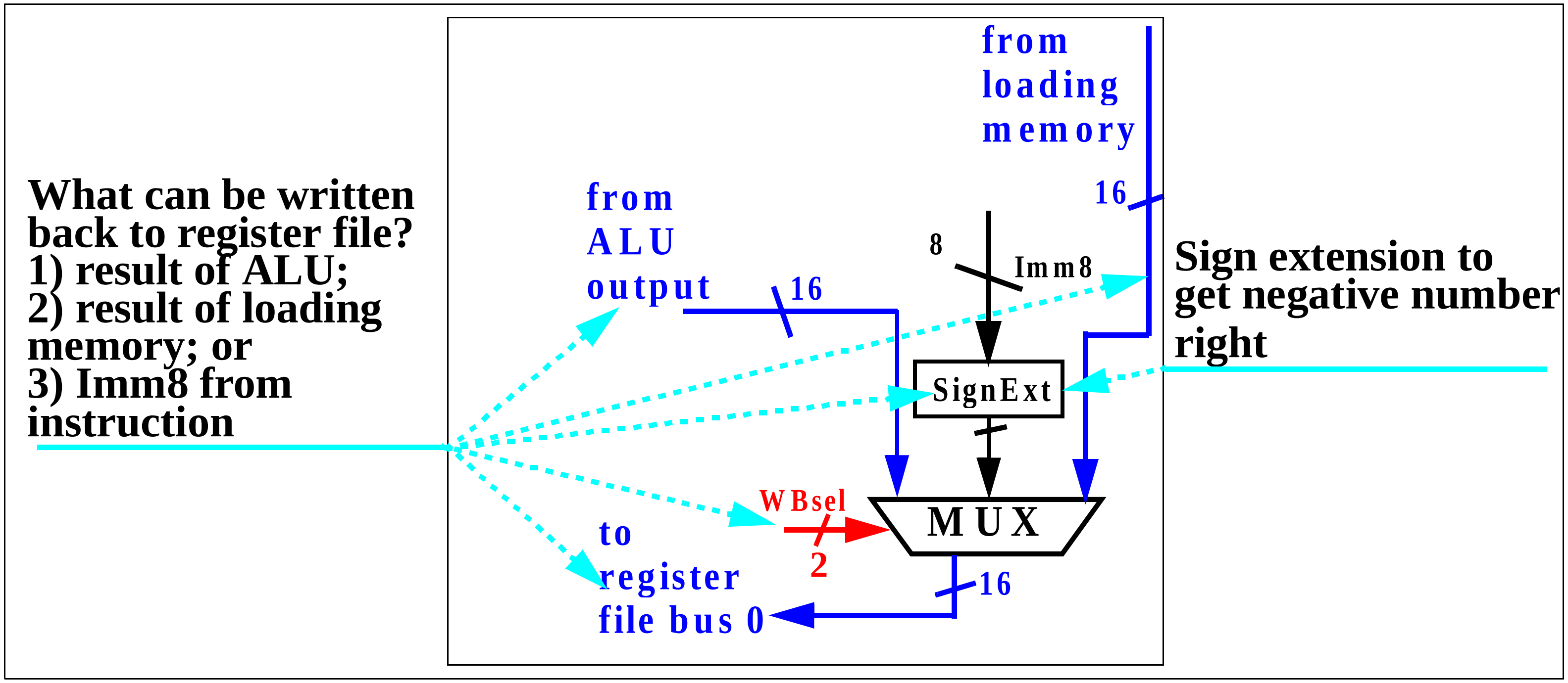


TOY Memory Datapath



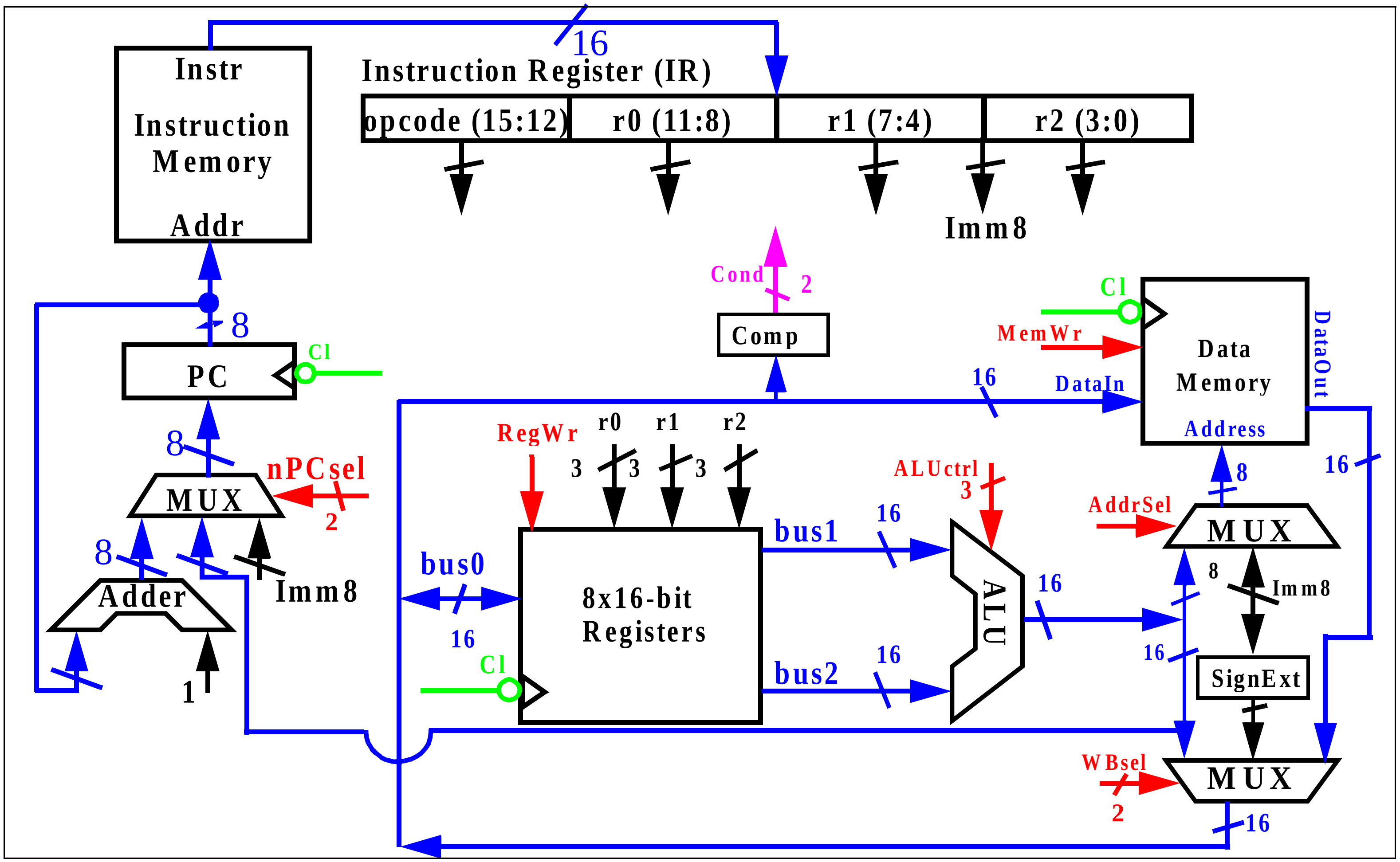
- For instructions that load from or write to memory
- Key question: where does address come from?
  - From instruction itself (example: **`r0 = mem[3D]`**)
  - From ALU (example: **`r0 = mem[r1+r2]`**)

TOY Write Back Datapath



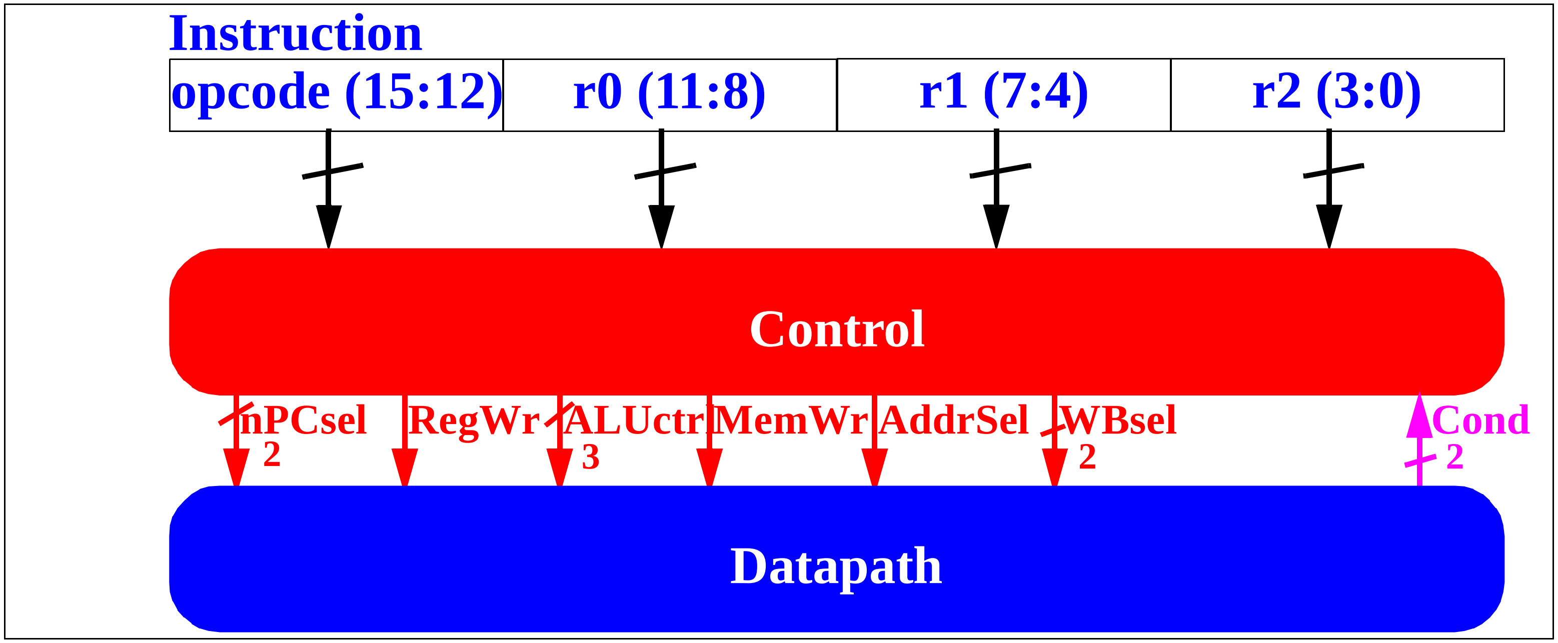
- Key question: what to write back to register file? One of three possibilities, examples:
  - **r0 = r1 + r2**
  - **r0 = mem[3D]**
  - **r0 = 3A**

Putting It All Together (Complete Single Cycle TOY Datapath)



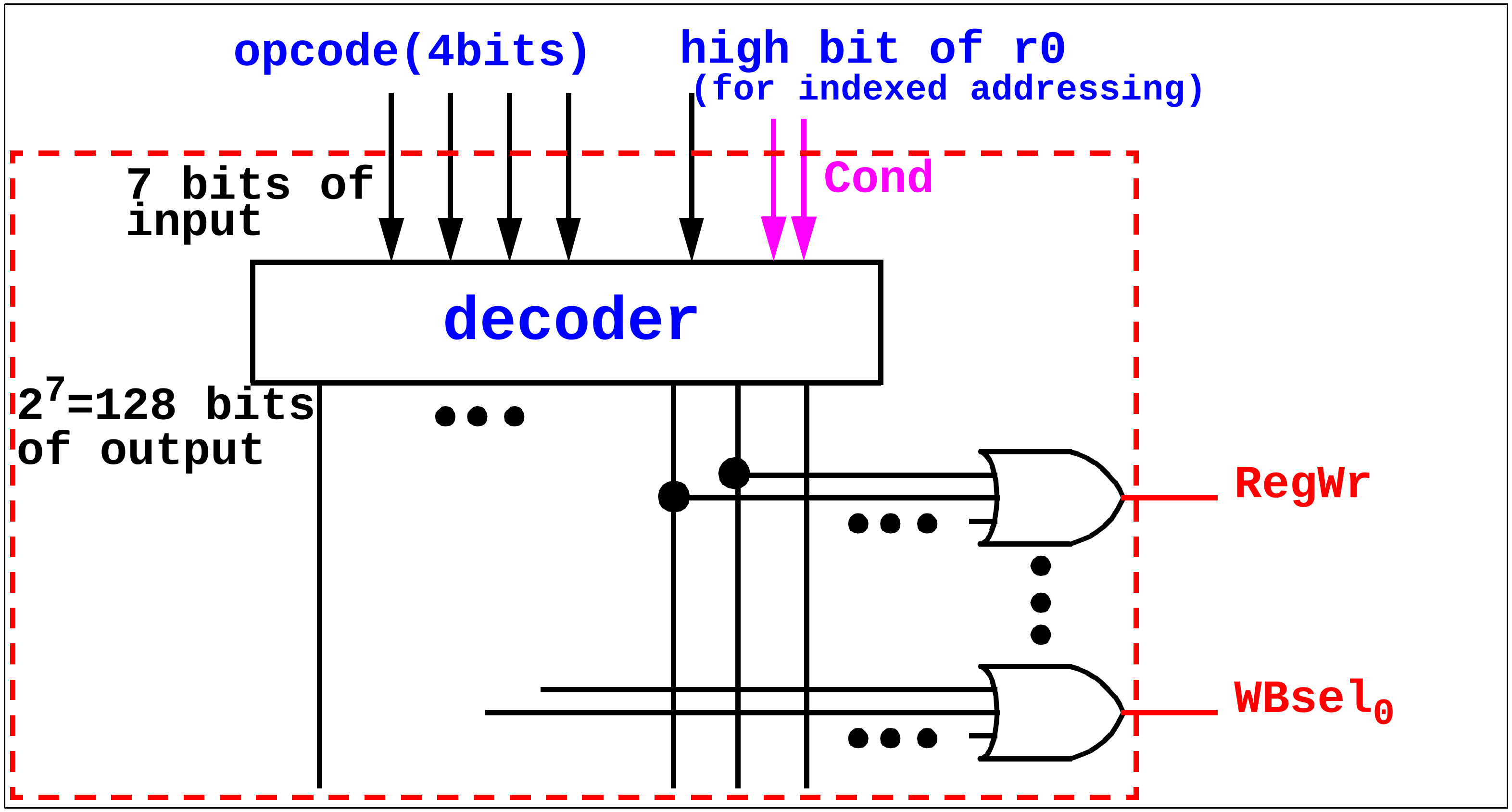
- Example TOY instruction 1A:9A45 (r2 = mem[r4+r5])
- Caveat: I'm leaving out a couple instructions as exercises

Abstract View of Relationship Between Single Cycle TOY Datapath and Control



- The flow of data in the datapath commanded by control signals
- Control signals issued by the control unit
- Control unit gets its input from the current instruction and condition codes from the datapath
- Control unit is nothing but a big combinational circuit

Implementing Single Cycle TOY Control



- Meaning of a decoder output that is 1: one particular instruction is executing **and** certain conditions are met
- Meaning of each OR-gate: turn on this control signal if any one of “these things” happen

Outline

- ~~Introduction~~
- ~~Some basics~~
- ~~Single-cycle TOY datapath design~~
- ~~Single-cycle TOY control design~~
- **Multicycle TOY design**
- Conclusions

Problems with Single-Cycle Implementation

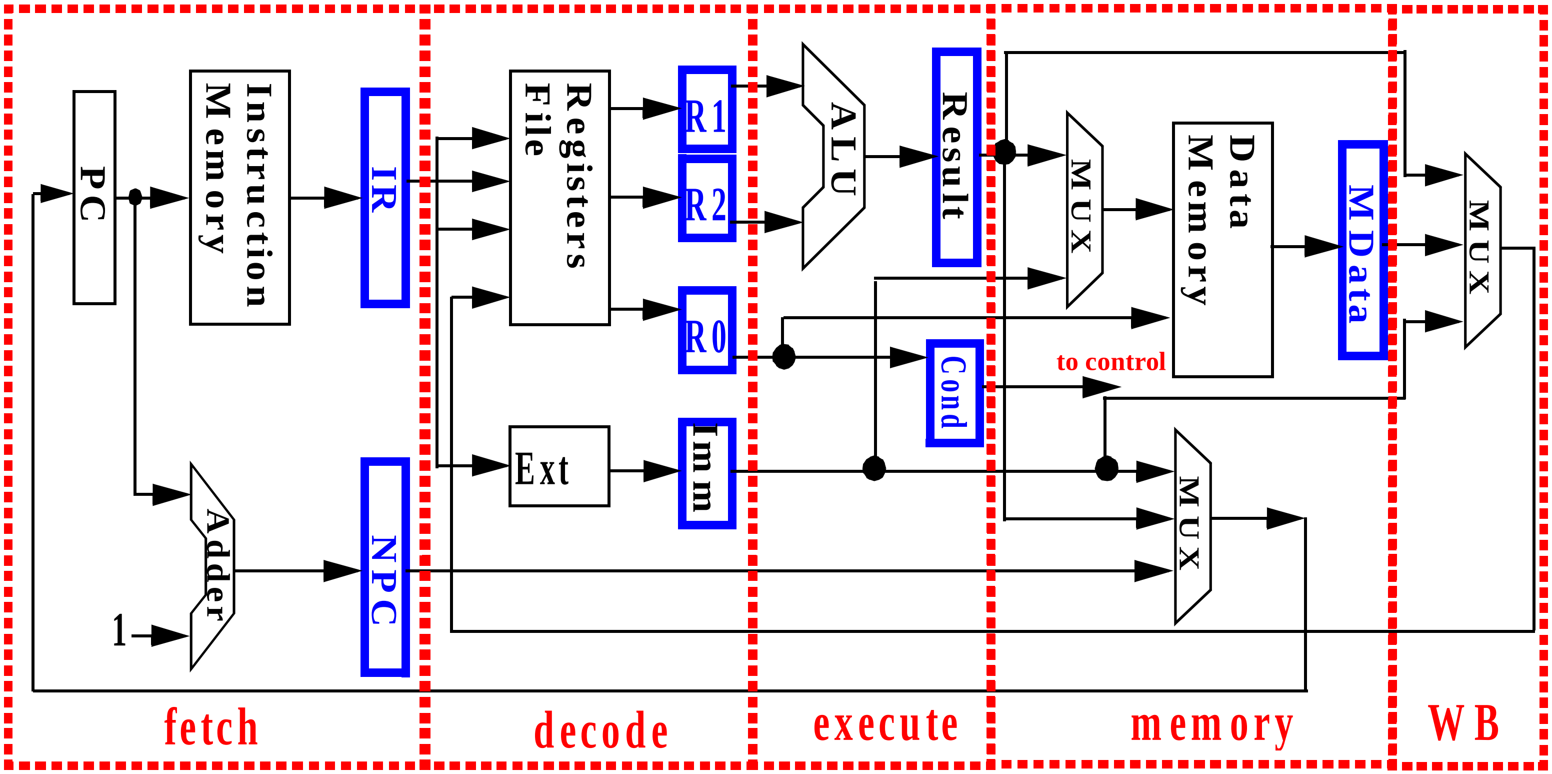
- Long cycle time
  - Not all instructions are equal, some longer, some shorter
  - Memory accesses can be a lot longer
  - The slowest instruction determines cycle time
  - The processor sits idle for faster instructions
- Waste of chip area, for example:
  - Need an adder to compute PC+=4 in addition to the ALU
  - Could in theory eliminate the adder and borrow ALU when it's not needed
  - But in a single cycle, we can't tell when ALU is done

Multicycle Design

```
repeat
    fetch instruction;
    decode instruction;
    execute instruction;
    access memory if necessary;
    write back to register if necessary;
until halt signal
```

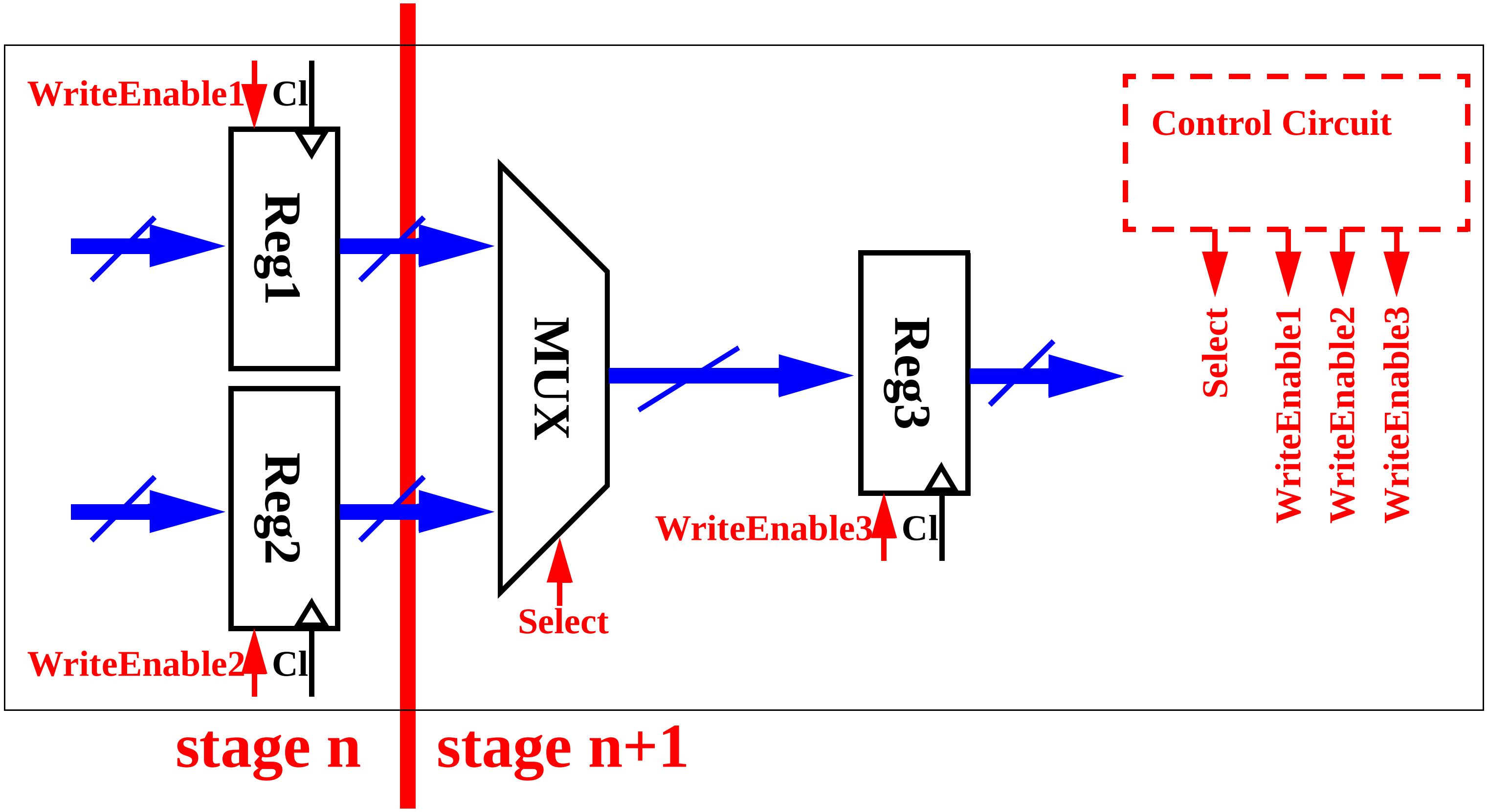
- Multicycle design
  - Look at our TOY simulator again
  - Carefully break down each instruction into these roughly equal **stages**
  - Use one (short) clock cycle to execute each stage
- Advantages
  - Shorter instructions can just skip unnecessary cycles, more efficient in time
  - Can borrow ALU to increment PC earlier: more efficient in chip area

Multicycle TOY Datapath



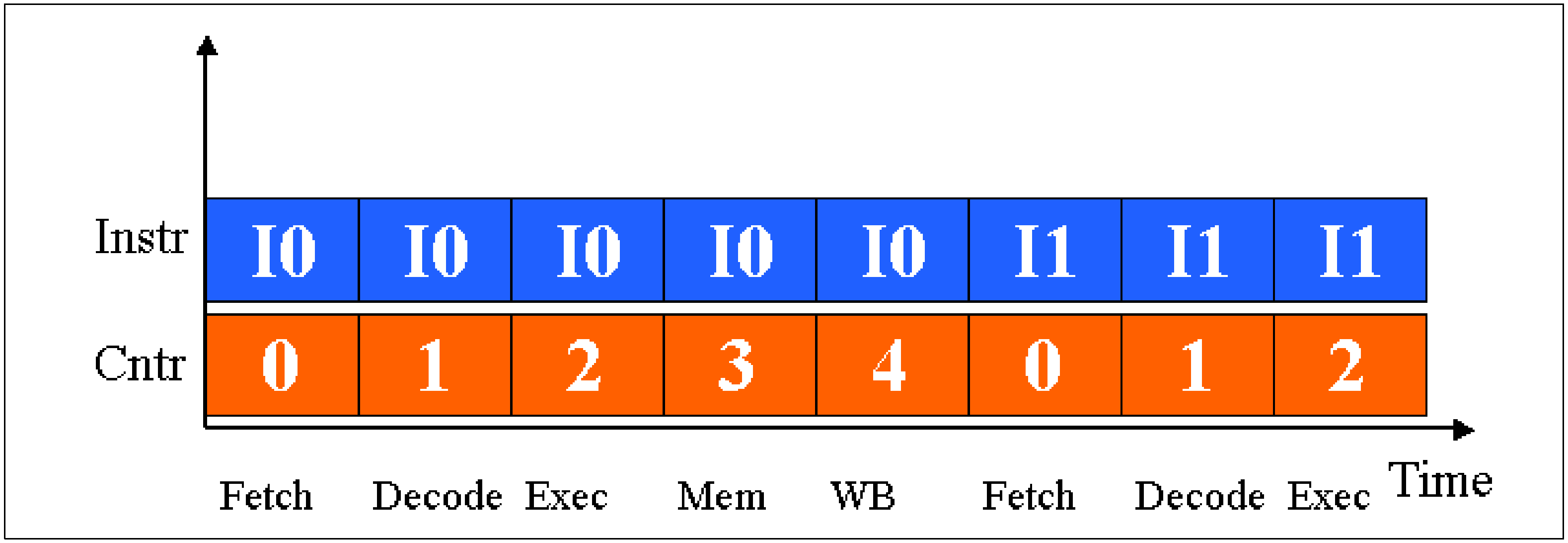
- Divide datapath up into 5 pieces (red boxes, analogous to the simulator code on previous slide: fetch, decode, execute, memory, write-back)
- Introduce temporary registers (blue boxes) to hold intermediate answers
- During each clock cycle, previous intermediate values are "clocked" into next stage, where the next intermeddiate value is calculated

“Clocking” Values from One Stage to Next



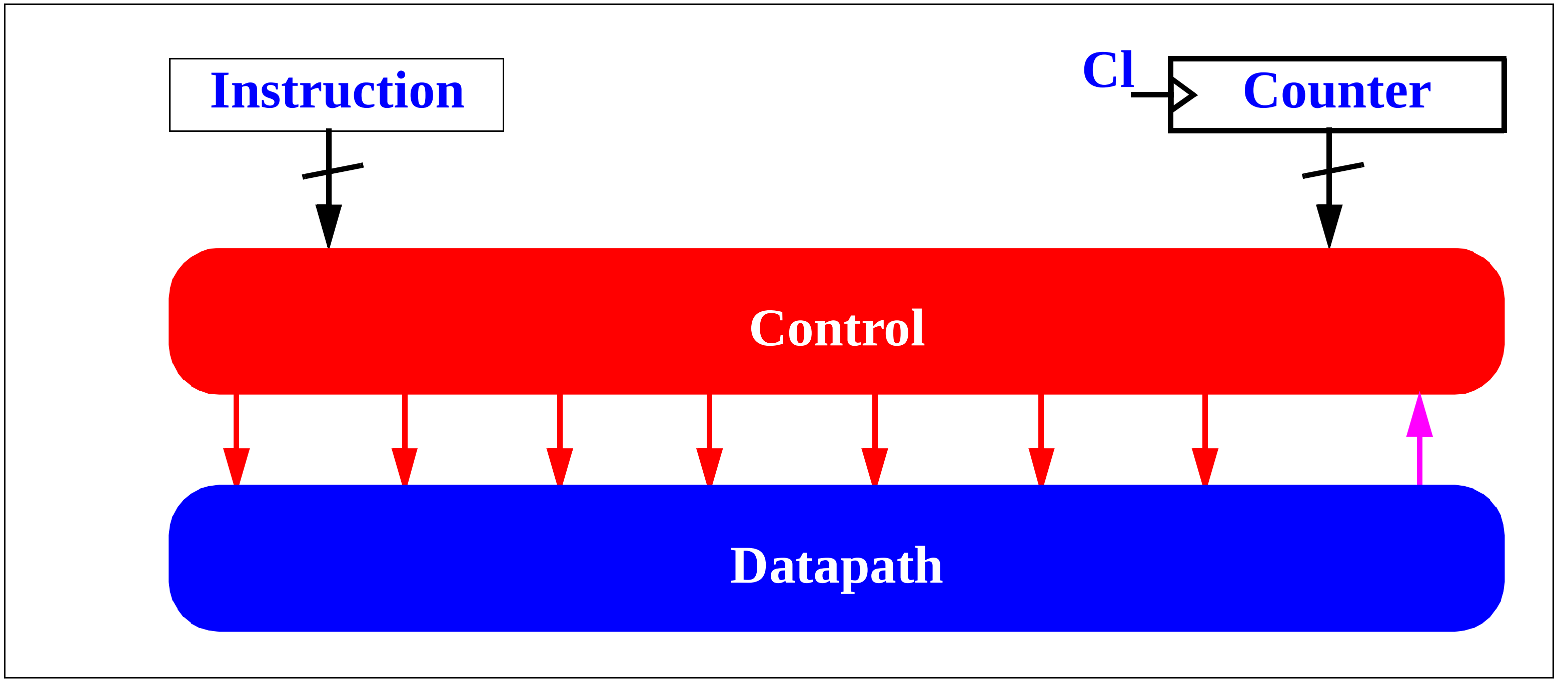
- (We have seen this slide before)
- The trick is to figure out how and when to set the control signals!

How to Modify Control



- Control depends on both instruction and time
- Use a counter to keep track of time (which stage the instruction is in)
- Will use counter to help determine control

What's New In This Picture?



- Counter output becomes part of control input

Outline

- ~~Introduction~~
- ~~Some basics~~
- ~~Single-cycle TOY datapath design~~
- ~~Single-cycle TOY control design~~
- ~~Multicycle TOY design~~
- **Conclusions**

Steps Towards Designing a Processor

- Analyze instruction set architecture (ISA) and understand datapath requirements
- Select set of datapath components and establish clocking methodology
- Assemble datapath to meet ISA requirements
- Analyze how to implement each instruction to determine the setting of various control signals
- Assemble the control logic

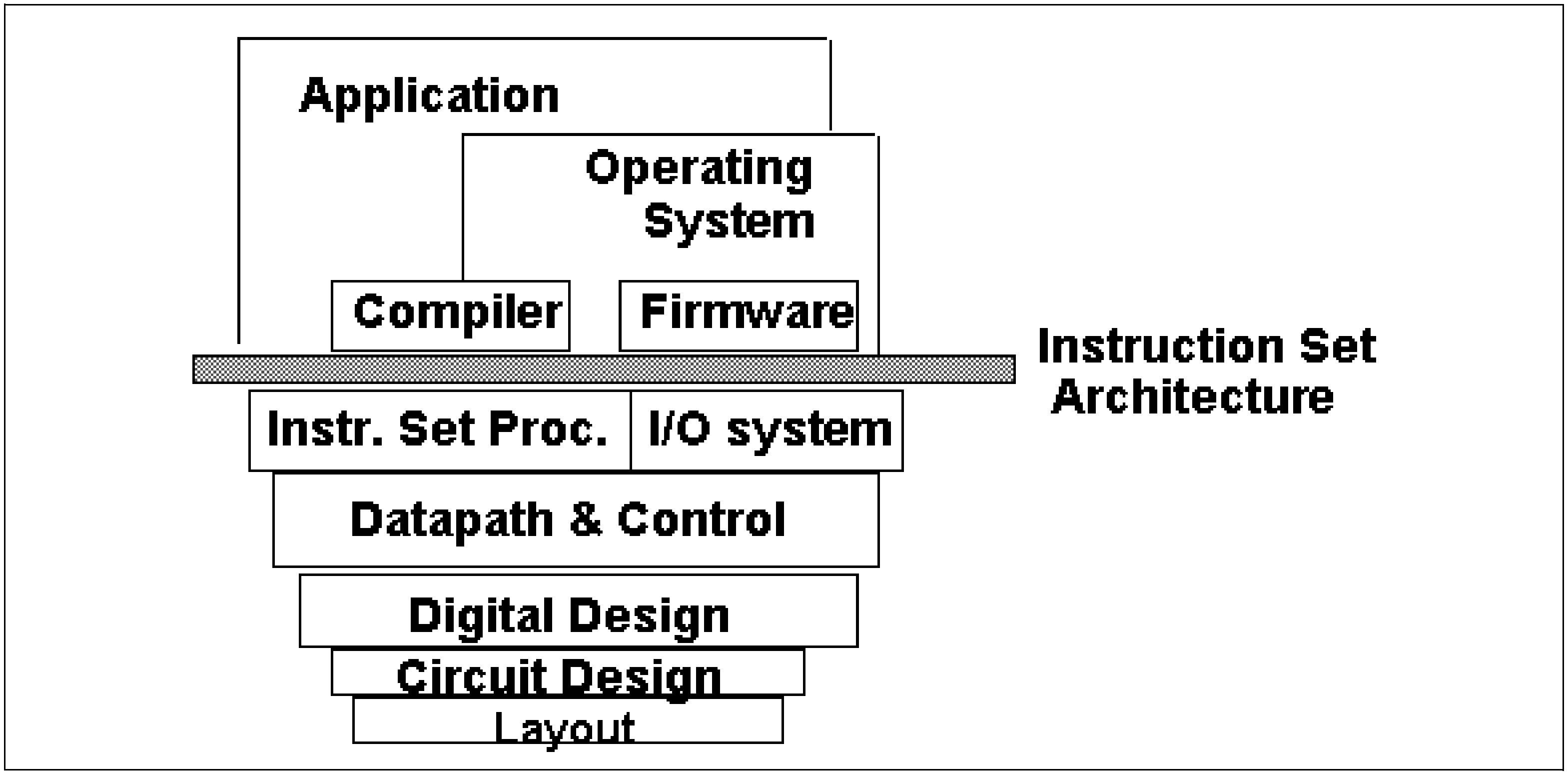
Where's the Science? Understanding Tradeoffs

- We saw a deceptively trivial tradeoff today: clocking methodology
  - Single cycle architecture vs. multicycle architecture
  - Multicycle sounds obviously superior, right?
  - Extra temporary registers and extra control logic of latter
    - + Introduce time overhead
    - + Introduce chip area overhead
    - + Introduce extra complexity, cost, time-to-market, ...
  - The question to a computer architect is whether this tradeoff is worth it
- More complex tradeoffs at each step of the prev. slide
- Nice to hide all this under the hood of an ISA

What We Have Learned Today

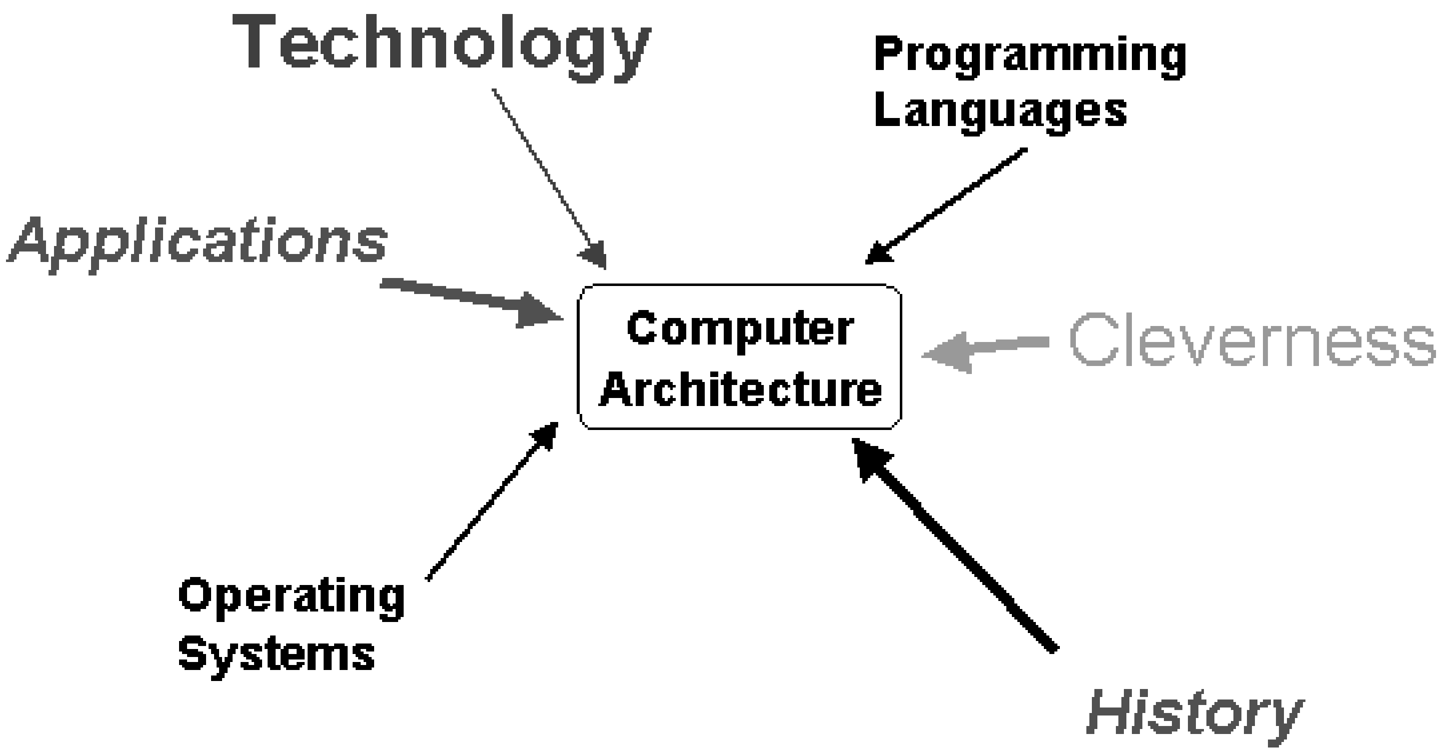
- Concepts:
  - Datapath vs. control
  - Single-cycle vs. multicycle designs
- More components: TOY register file and ALU
- Single-cycle design
  - How signals propagate in different parts of the datapath in general
  - How to implement control signals in general. Where do inputs come from?
- Multicycle design
  - Main general modifications made to datapath and control
- **I Don’t expect people to memorize all the details**

Computer Architecture



- Coordination of many levels of abstraction
- Under a rapidly changing set of forces
- Design, measurement, and evaluation

Forces Influencing Computer Architecture



Dramatic Technology Change

- Technology
  - **Processor** logic capacity: +30% / yr; clock rate: +20% / yr; overall performance: ~+60% / yr!
  - **Memory and disk** capacity: ~+60% / yr
- Numbers, though impressive, are boring. What's really exciting is revolutionary leaps in applications!
- Quantitative improvement and revolutionary leaps interleave as technology advances
  - ~1985: **Single-chip** (32-bit) **processors** and **single-board computers** emerged, led to revolutions in all aspects of computer science!
  - Conjecture: ~2002: Emergence of powerful **single-chip systems**, what will be its implication?!