CMPE110 Lecture 09
Processor Architecture

Heiner Litz

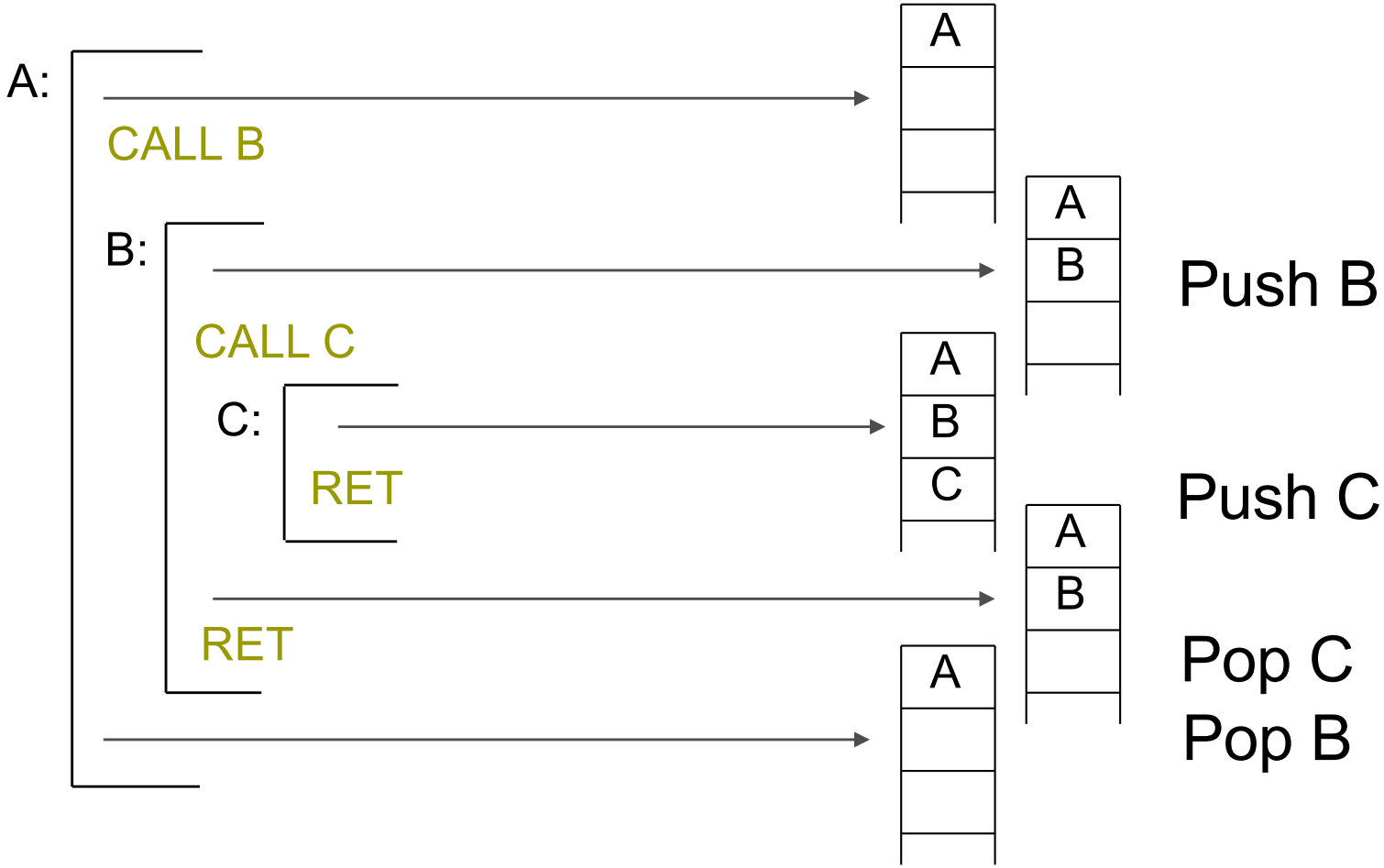
https://canvas.ucsc.edu/courses/12652

Announcements

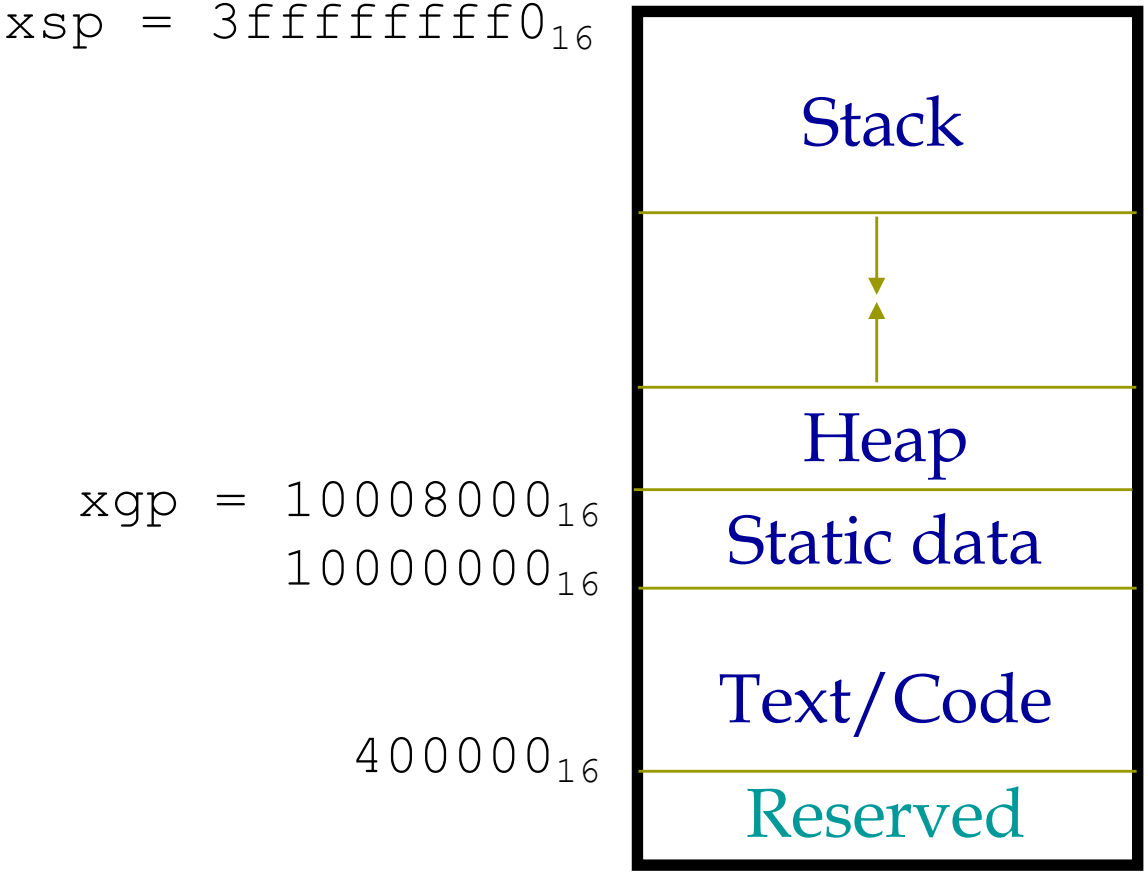
Review

Nested Stacks

- The stack grows downward and shrinks upward



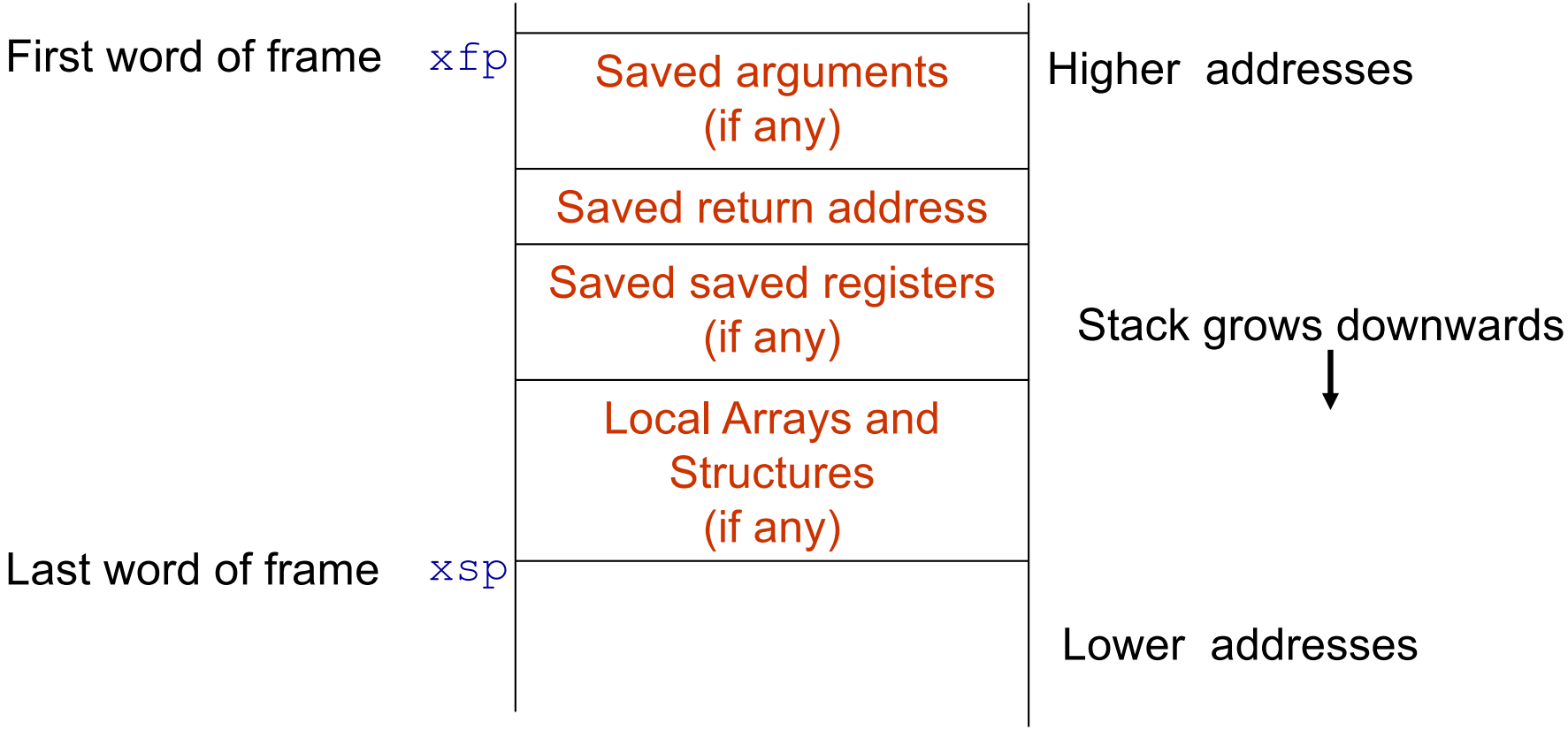
RISC-V Storage Layout (Software Convention)



Stack and heap grow towards one another to maximize storage use before collision

Procedure Activation Record or Stackframe

- Each procedure creates an activation record on stack



RISC-V Calling Convention

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5–7	t0–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

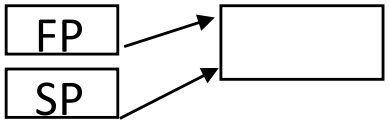
Table 20.2: RISC-V calling convention register usage.

Call and Return: the Details

- Caller
  - Save caller-saved registers `(ra, temporaries)` as needed
  - Load arguments in `x10`–`x17`, rest passed on stack
  - Execute `jal`
- Callee setup
  1. Allocate memory for new frame (`xsp` = `xsp` - frame)
  2. Save callee-saved registers `non-temporal`, `xfp`, `xra` **as needed**
  3. Set frame pointer (`xfp` = `xsp` + frame size - 4)
- Callee return
  - Place return value in `x10` and `x11`
  - Restore any callee-saved registers
  - Pop stack (`xsp` = `xsp` + frame size)
  - Return by `jr xra`
- Caller
  - Restore any caller-saved registers as needed

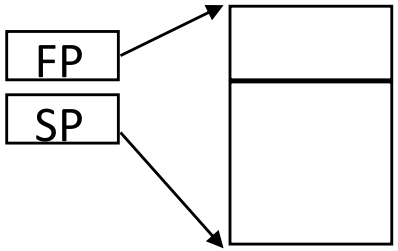
Calling Convention Steps

Before call:



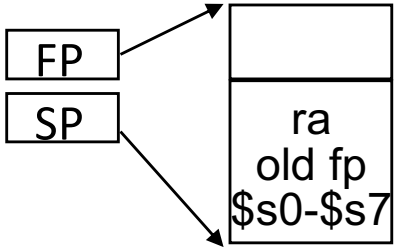
First four arguments passed in registers

Callee setup; step 1



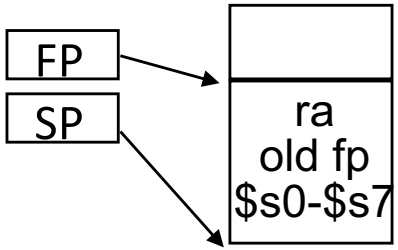
Adjust SP

Callee setup; step 2



Save registers as needed

Callee setup; step 3



Adjust FP

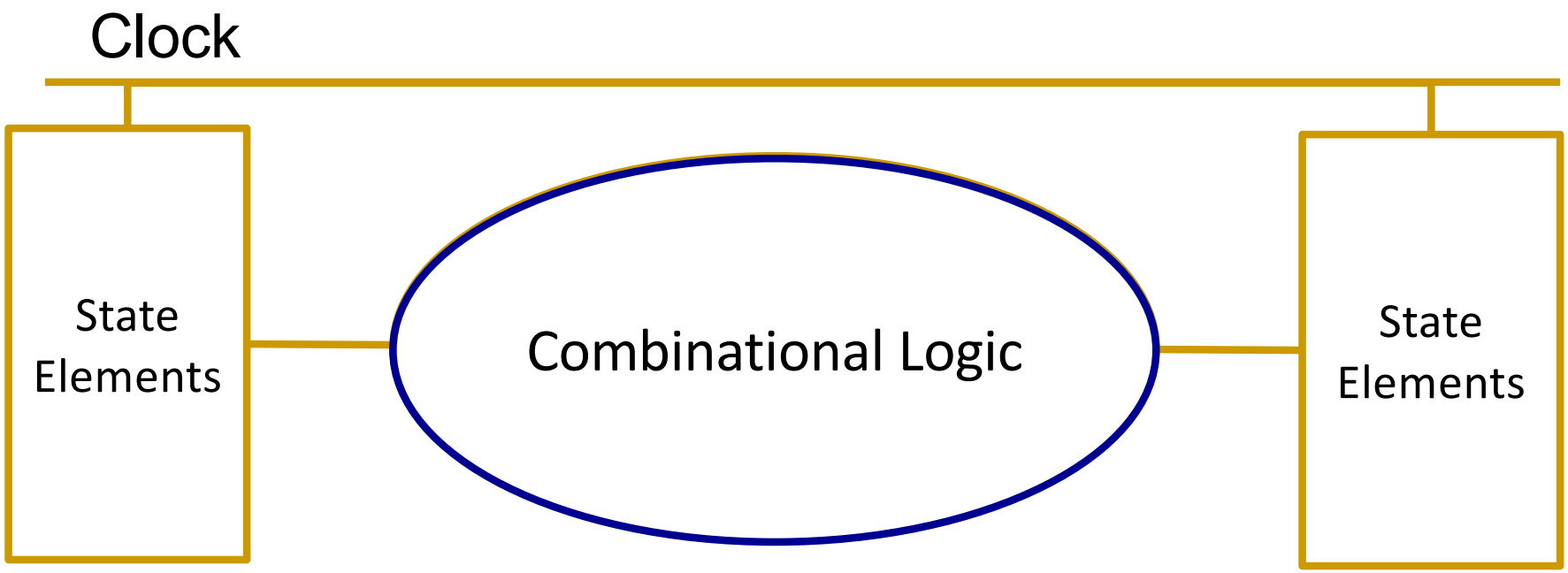
Simple Example


```
int foo(int num)
{
   return(bar(num + 1));
}

int bar(int num)
{
   return(num + 1);
}
```

```
foo:
        addiu   xsp, xsp, -32 # push frame
        sw      xra, 28(xsp)  # Save xra
        sw      xfp, 24(xsp)  # Save xfp
        addiu   xfp, xsp, 28  # Set new xfp
        addiu   x10, x10, 1   # num + 1
        jal     xra, bar      # call bar
        lw      xfp, 24(xsp)  # Restore xfp
        lw      xra, 28(xsp)  # Restore xra
        addiu   xsp, xsp, 32  # pop frame
        jr      xra           # return

bar:
        addiu   x10, x10, 1   # leaf procedure
        jr      xra           # with no frame
```

Review: Logic Design

Digital Systems



- State elements, combinational logic, and clock

Boolean Algebra

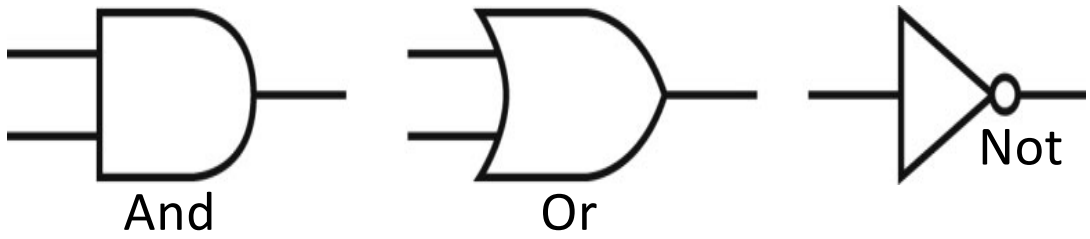
- Basic functions:
  - AND: true if both A and B are true (A*B, AB)
  - OR: true if either or both A or B is true (A+B, A|B)
  - NOT: true if A is false (!A, ~A, $\tilde{A}$, A')
  - Sufficient to implement all complex functions

- Laws of Boolean algebra
  - Identity: A+0=A; A*1=A
  - 0 and 1: A+1=1; A*0=0
  - Inverse: A+!A=1; A*!A=0

Basic Logic Gates

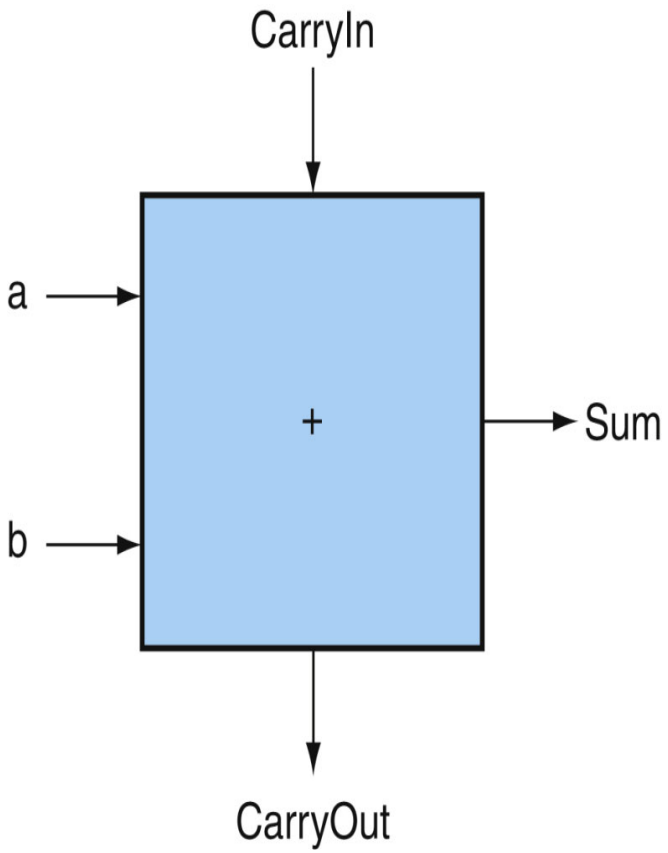
- Boolean operators map to hardware gates

  Represented in schematics by these symbols

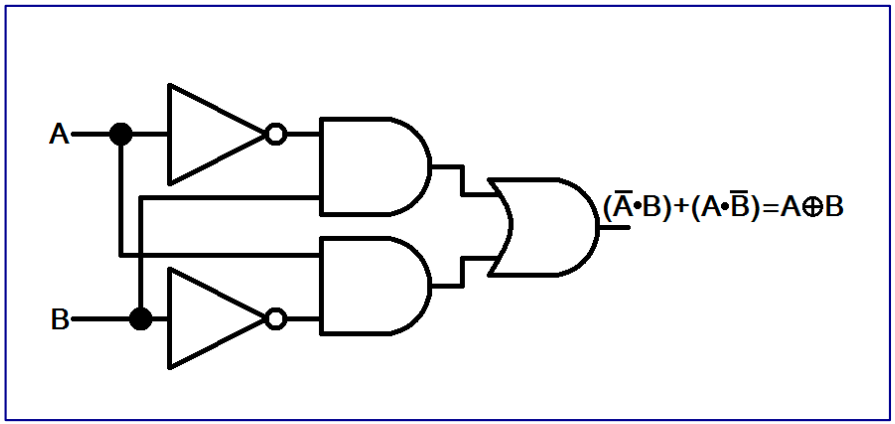


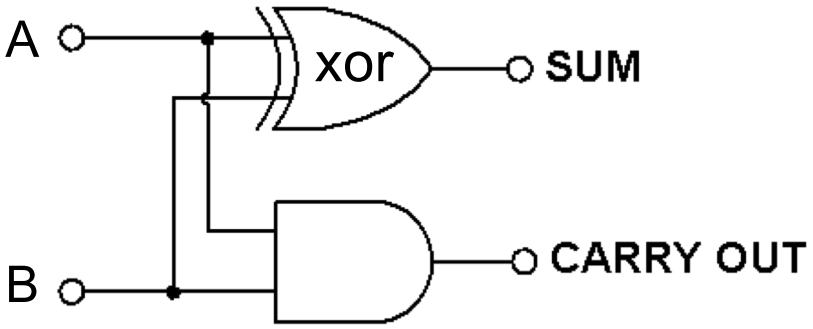
- Hardware Gates are made from transistors
  - Four transistors to build And, Or
  - Two transistors to build Not

1-Bit Adder

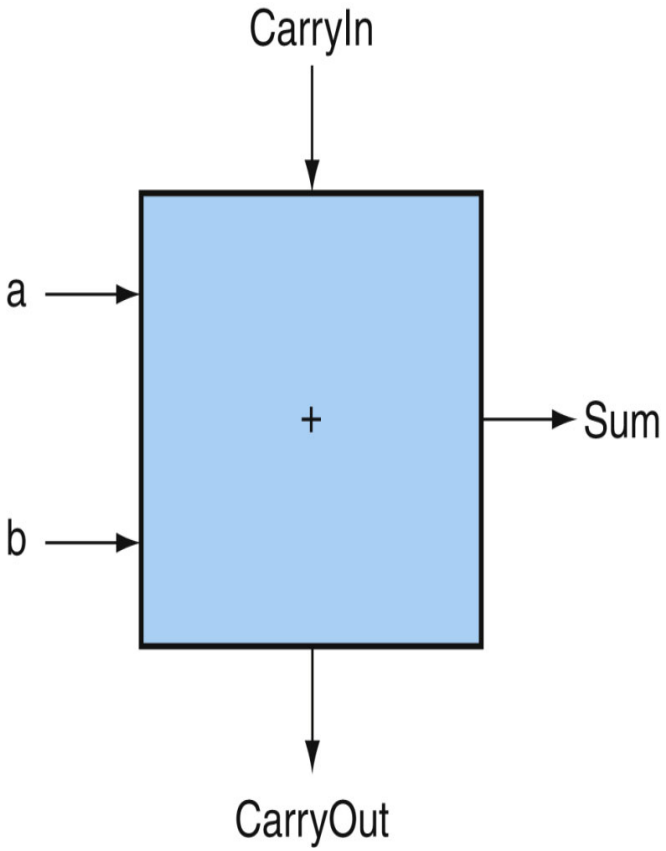


XOR-Gate





1-Bit Adder



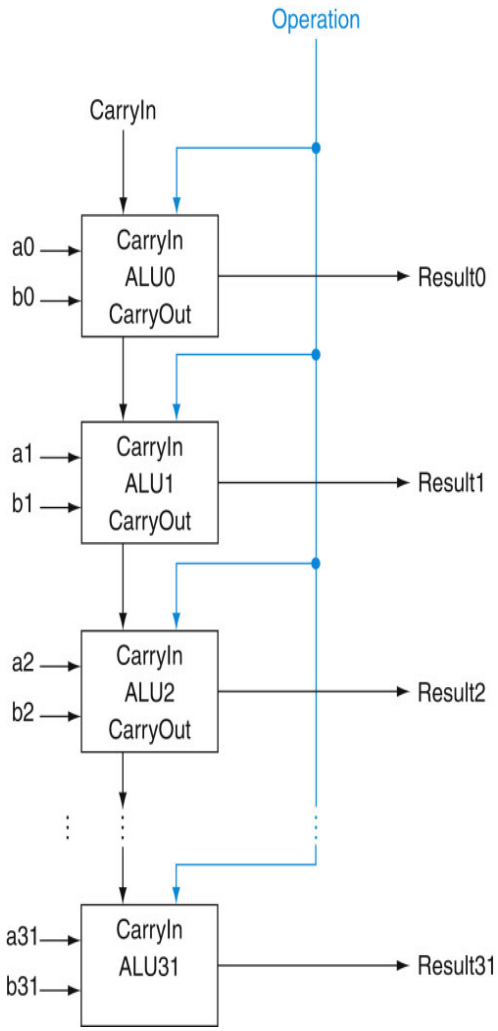
Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1 + 1 = 11_{two}$

32-bit ALU

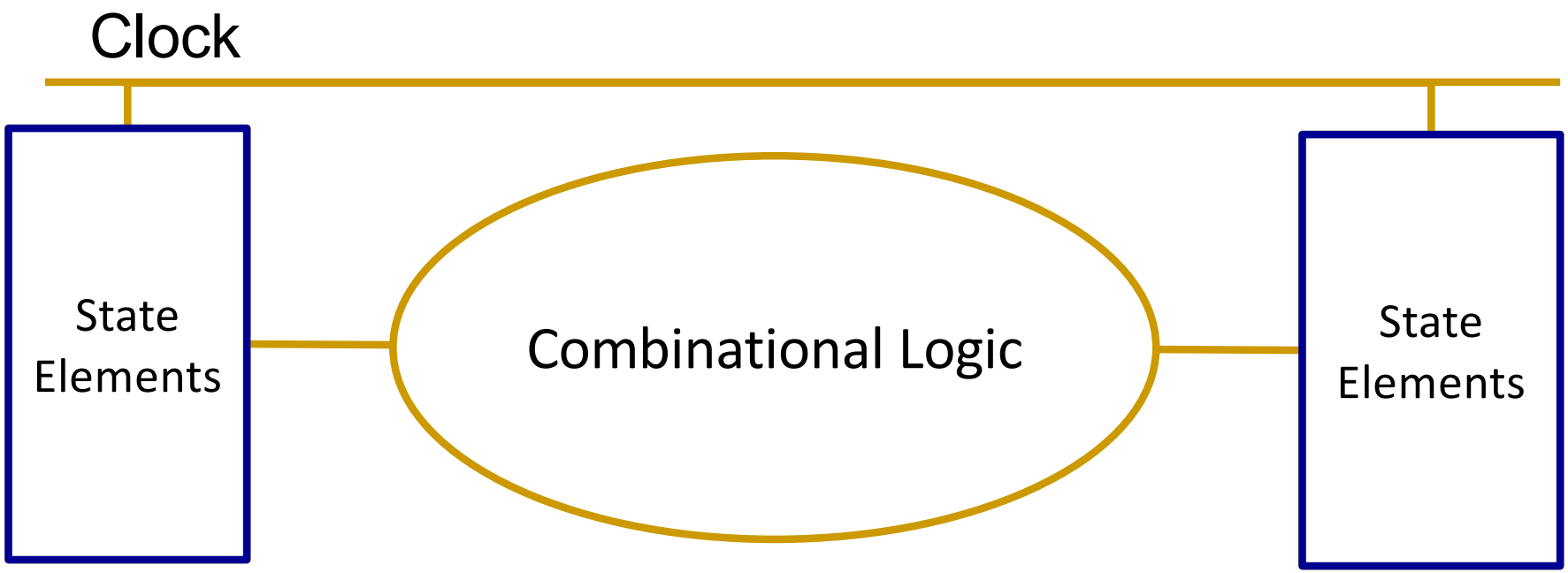
- We must compute 32-bit ops
- How do we make ALU 32 bits?

- Can also do the same for 32-bit muxes, ands, and ors

Cascade 32 1-bit ALUs



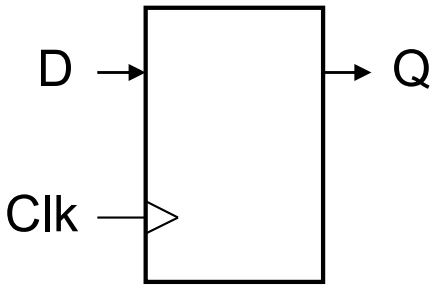
Digital Systems

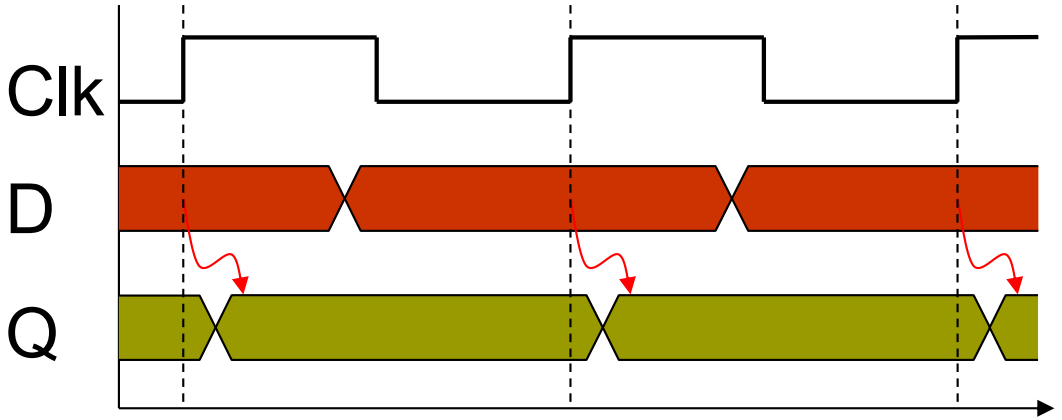


- State elements, combinational logic, and clock

State or Sequential Elements

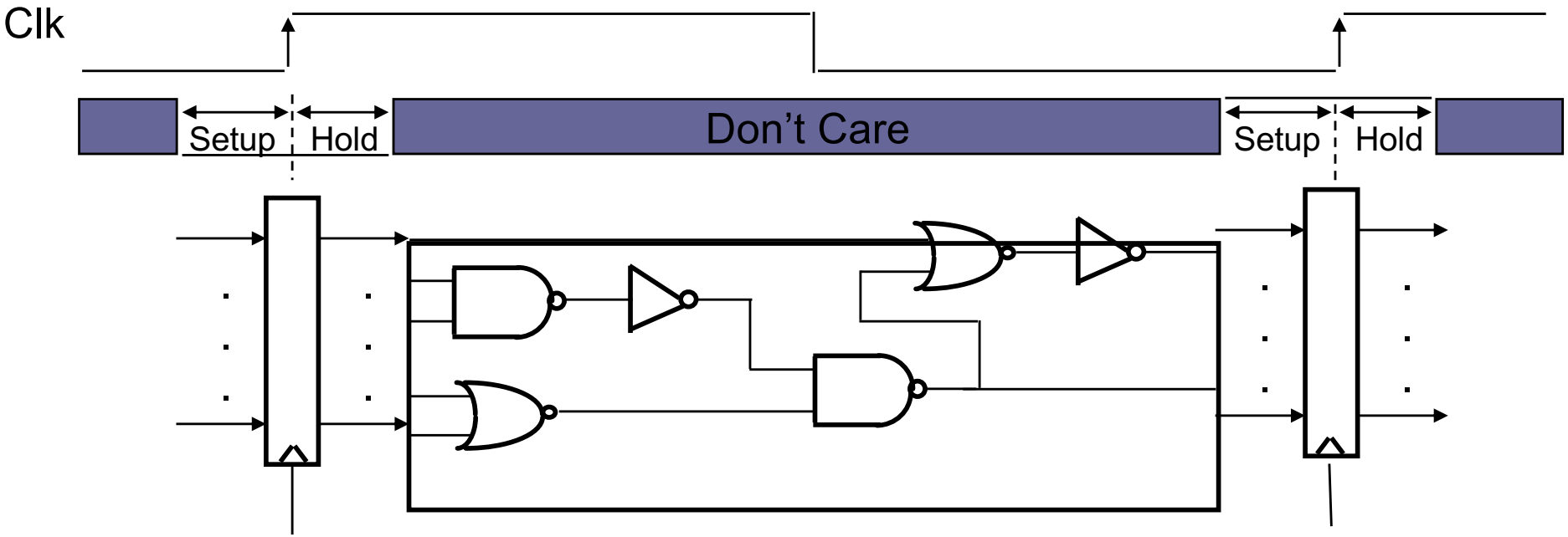
- Flip-flops (or registers): stores data in a circuit
  - Uses a clock signal to determine when to update stored value
  - Edge-triggered: update when Clk changes from 0 to 1





Critical Timing Issues

- Flops work great if the input is stable around clock edges
  - Called setup and hold windows
  - Setup time: $t_{ck} > t_{pd} + t_s + t_{skew}$
  - Hold time: min delay must be longer than hold time



Building our first Processor

Building a Processor

- Generally hardware consists of two parts
  - **Datapath**: the hardware that processes and stores data
    - Combinational circuits and state elements
  - **Control**: the hardware that manages the datapath
- **Break instruction execution into steps**
  - Find simple + efficient datapath & control for each step
  - Some are obvious, others can be more subtle
- Start simple, optimize performance/energy later

Subset of Instructions

- We will focus on a subset of the RISC-V instructions
  - Memory: `lw` and `sw`
  - Arithmetic: `addu`, `subu`, `and`, `ori`, and `slt`
  - Branch: `beq` and `j`

- Similar implementations for remaining instructions

Simple Processor

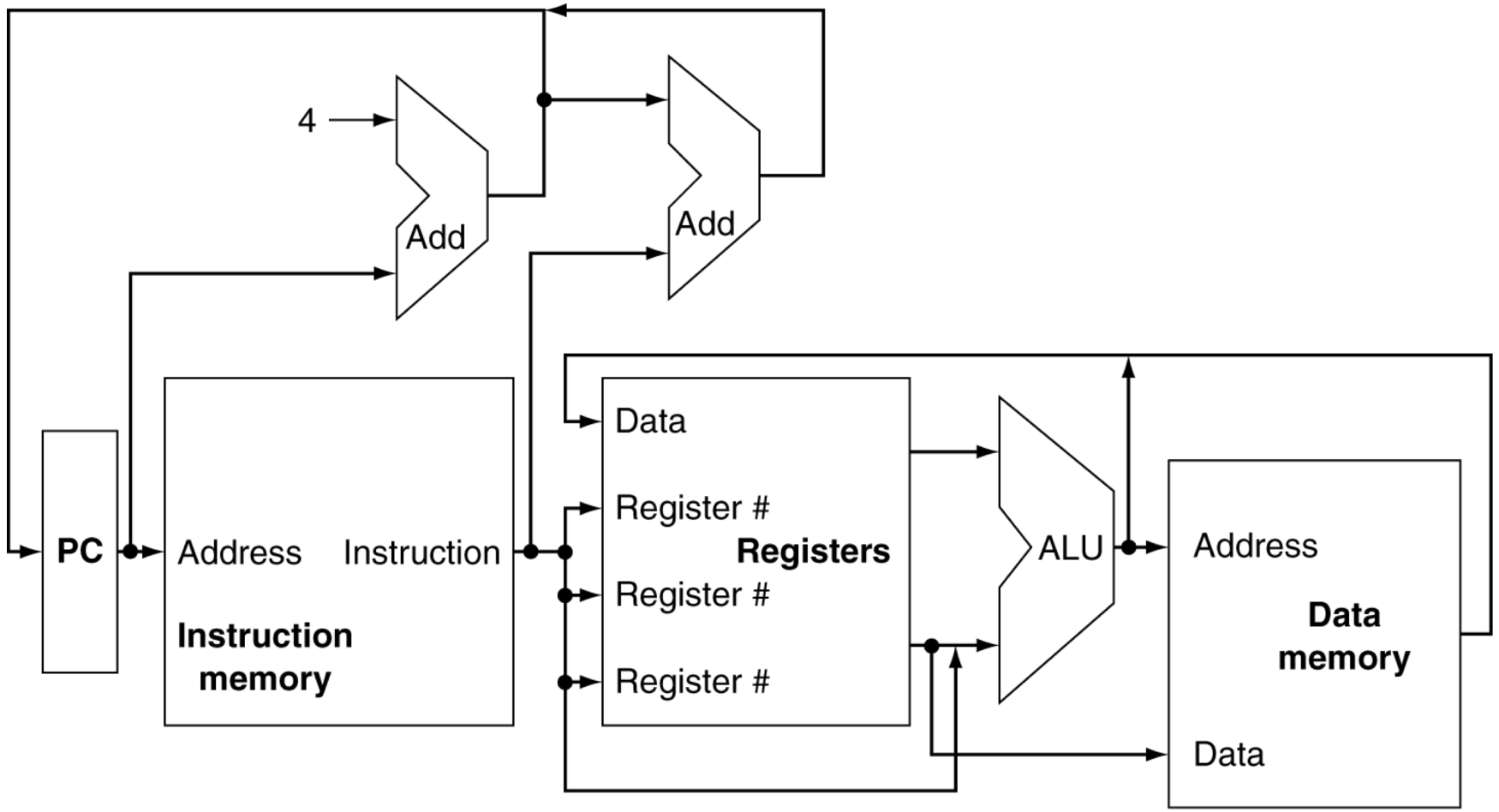
- Follow the ISA to the letter
  - Execute one instruction to completion before moving the next one
- Execute each instruction within 1 clock cycle
  - CPI = 1
- Needed hardware components
  - State: PC, 32-entry register file, memory
  - ALU, multiplexors

Instruction Steps

1. Fetch instruction from memory
   - Address is specified by PC
2. Read one or two registers
3. Do add/sub/.... using an ALU block
4. Fetch a value from memory
5. Store results to register-file/memory
6. Update the PC as needed

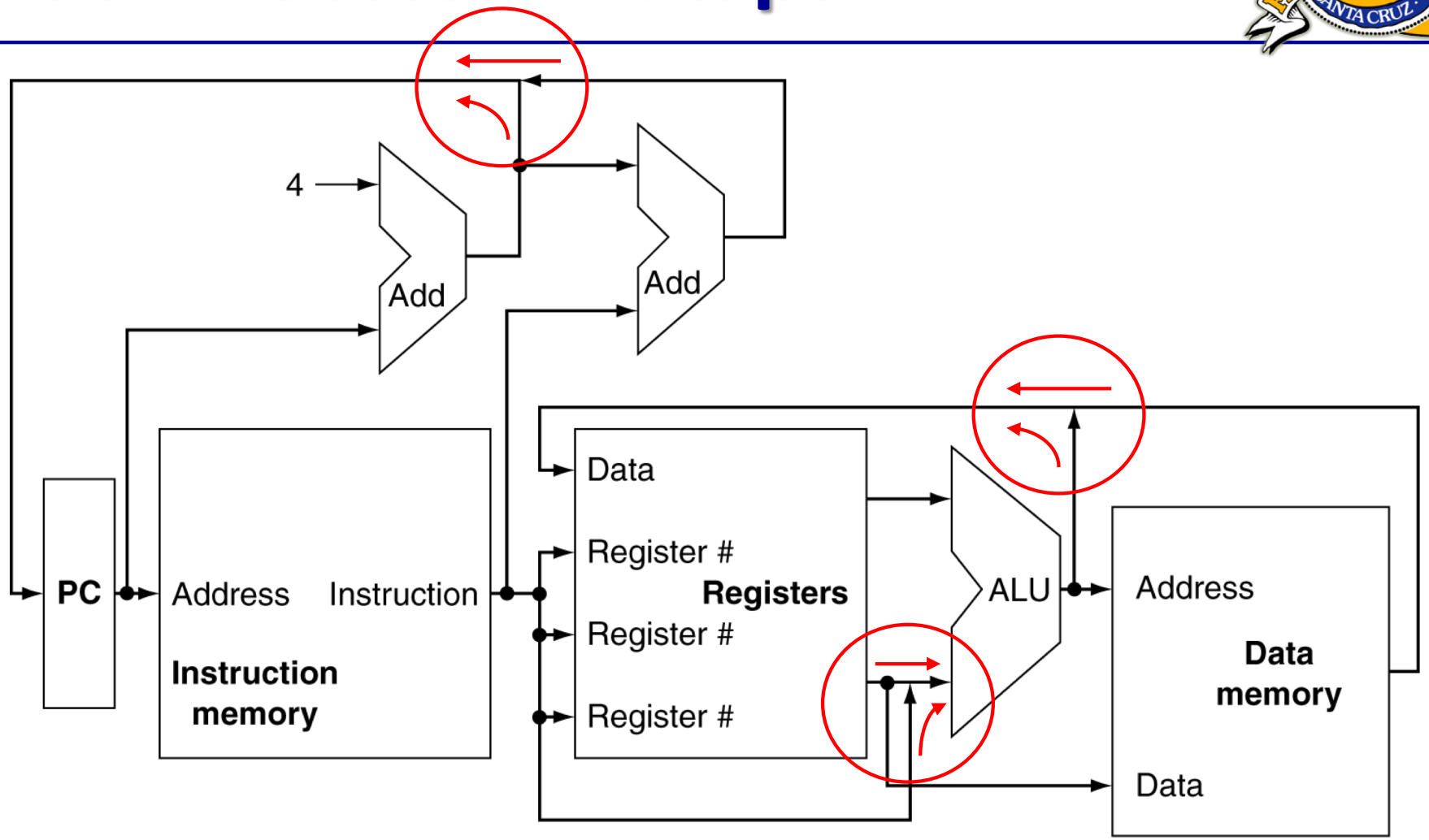
Note: Some are optional, depends on instruction

Initial Processor Datapath



- Major functional units & major connections
  - Can you spot the major inconsistency on this diagram?

Initial Processor Datapath



- Cannot just join wires together
  - These connections will actually require multiplexors