CS2200
Systems and Networks
Spring 2024

Lecture 5: Datapath

Alexandros (Alex) Daglis
School of Computer Science
Georgia Institute of Technology
adaglis@gatech.edu

Lecture slides adapted from Bill Leahy of Georgia Tech

Announcements

- Lab 1 released
- Homework 1 released and is due tomorrow before lab
- Starting book's Chapter 3 today

Topics

- Logic design review
- Data paths
- Finite State Machines

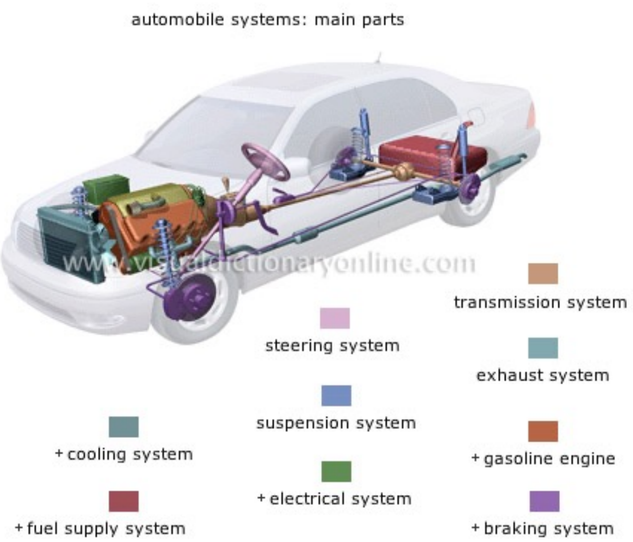
Architecture vs. Organization

- Architecture – the abstraction involving programmer-visible details of a computer system, such as
  - Instruction set
  - Memory layout

- Organization (aka, **micro-architecture**) – the details of the implementation of a particular architecture, involving many details that are not directly visible to a programmer, such as
  - Register and ALU implementation
  - Bus structure
  - Memory hierarchy and bank organization

Processor Implementation

- Implementation for a given instruction set
- Instruction-set is not a description of the implementation of the processor
  - Contract between hardware and software
  - Allows a compiler writer to generate code for different high-level languages to execute on a processor that implements this contract
- Can there be different implementations of the same instruction set?



IBM System/360: One architecture, many implementations

Model	Announced	Shipped	Scientific perform-ance (kIPS)	Commercial perform-ance (kIPS)	Memory band-width (MB/sec)	Memory size (in KB)
30	Apr 1964	Jun 1965	10.2	29	0.7	8-64
40	Apr 1964	Apr 1965	40	75	0.8	16-256
50	Apr 1964	Aug 1965	133	169	2.0	64-512
20	Nov 1964	Mar 1966	2.0	2.6		4-32
91	Jan 1966	Oct 1967	1,900	1,800	164	1,024-4,096

Architecture versus Implementation

Why?

- Market demands (different price points)
- Parallel hardware and software development
- Maintain compatibility for legacy software

What is involved in Processor Implementation?

- Organization of the electronic components (ALUs, buses, registers, etc.) commensurate with the expected price/performance characteristic of the processor.

- Thermal and mechanical aspects including cooling and physical geometry for placement on chip/motherboard

Primary objectives for:

Supercomputers	Servers	Desktops & PCs	Embedded
High performance	Intermediate performance and cost	Low cost	Small size, low cost, low power consumption

Circuits

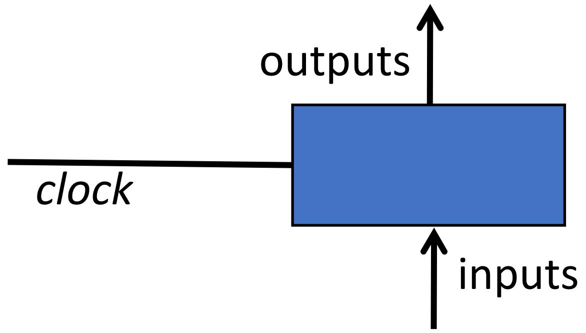
- Combinational logic
  - For a given set of inputs there is one unique output

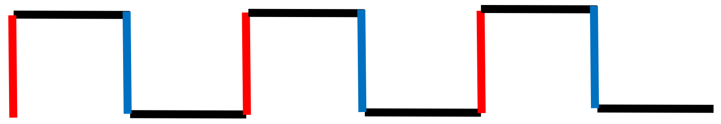
- Sequential logic
  - Circuits contain elements that remember *state*
  - Outputs depends on combinational logic that considers circuit inputs **and** previous *state*

Hardware resources of the datapath

- Memory
- ALU
- Register file
- Program Counter
- Instruction Register

Logic triggering





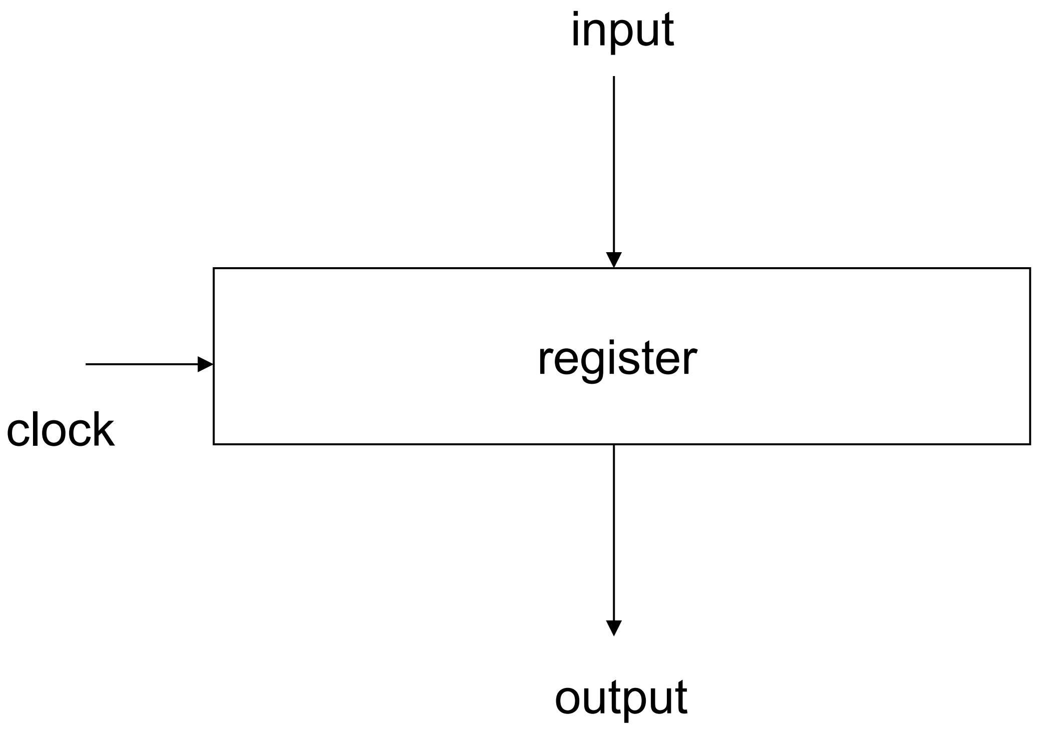
Level Triggering

- Outputs change based on inputs whenever *clock* is high
- Memory will be considered to be level triggered

Edge Triggering

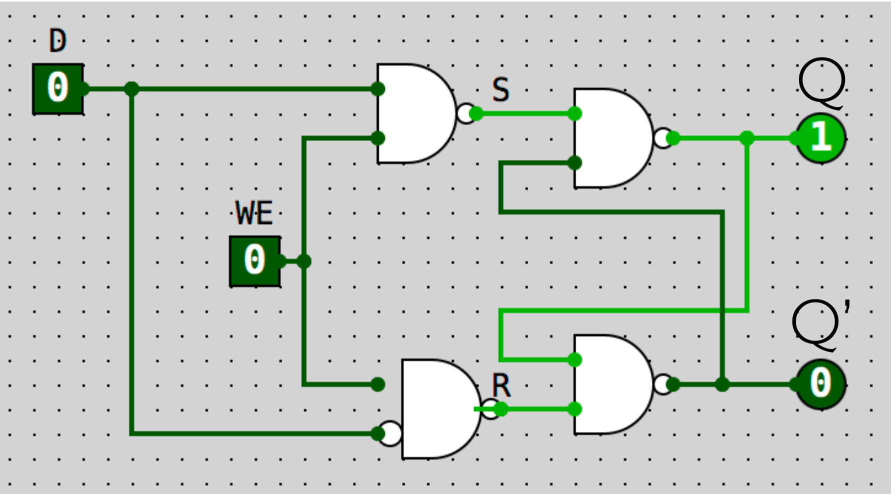
- Outputs change based on inputs only when clock transitions
- Positive edge-triggered logic when rising edge cause triggering
- Negative edge-triggered when falling edge causes triggering

Registers



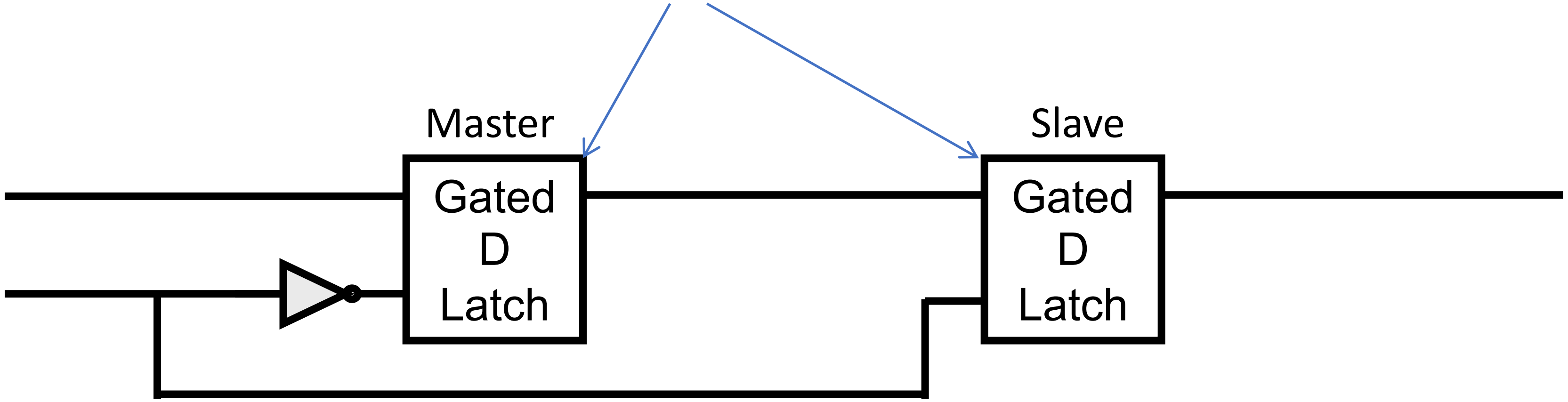
Master-Slave D Flip-Flop

Remember our Gated D Latch?

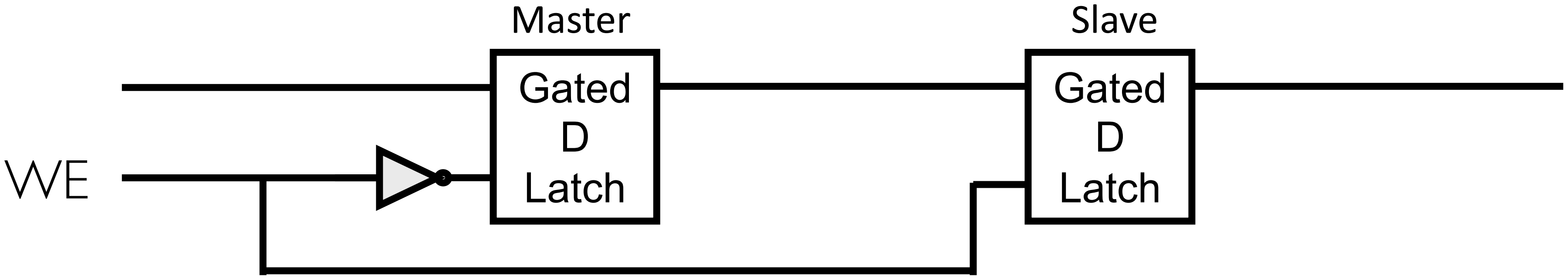


Truth Table

WE	D	Q	Q’
1	0	0	1
1	1	1	0
0	0	Q	Q’
0	1	Q	Q’

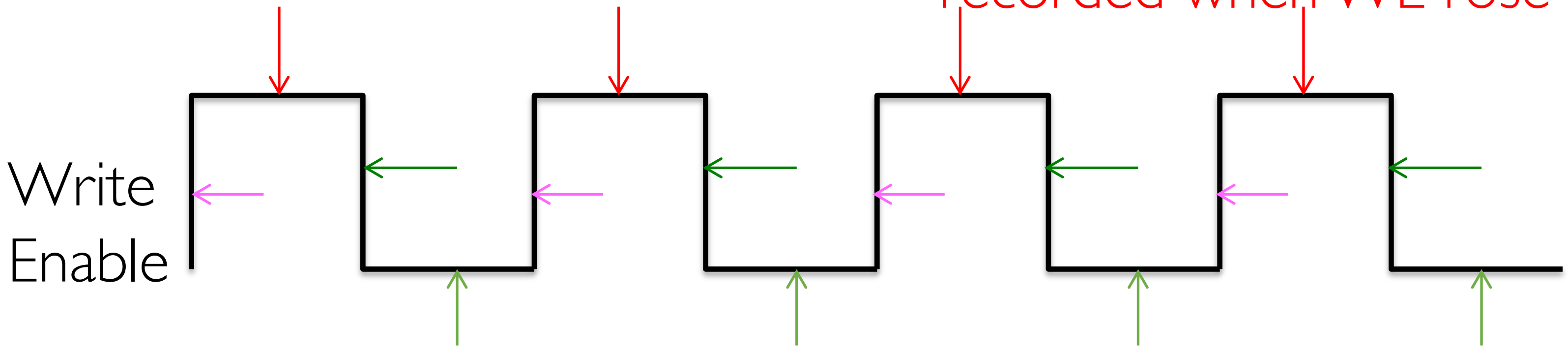


Clock Edges and Master-Slave



Master latch records what is presented on its input.

Slave latch outputs its inputs. Master latch outputs what was recorded when WE rose to 1.

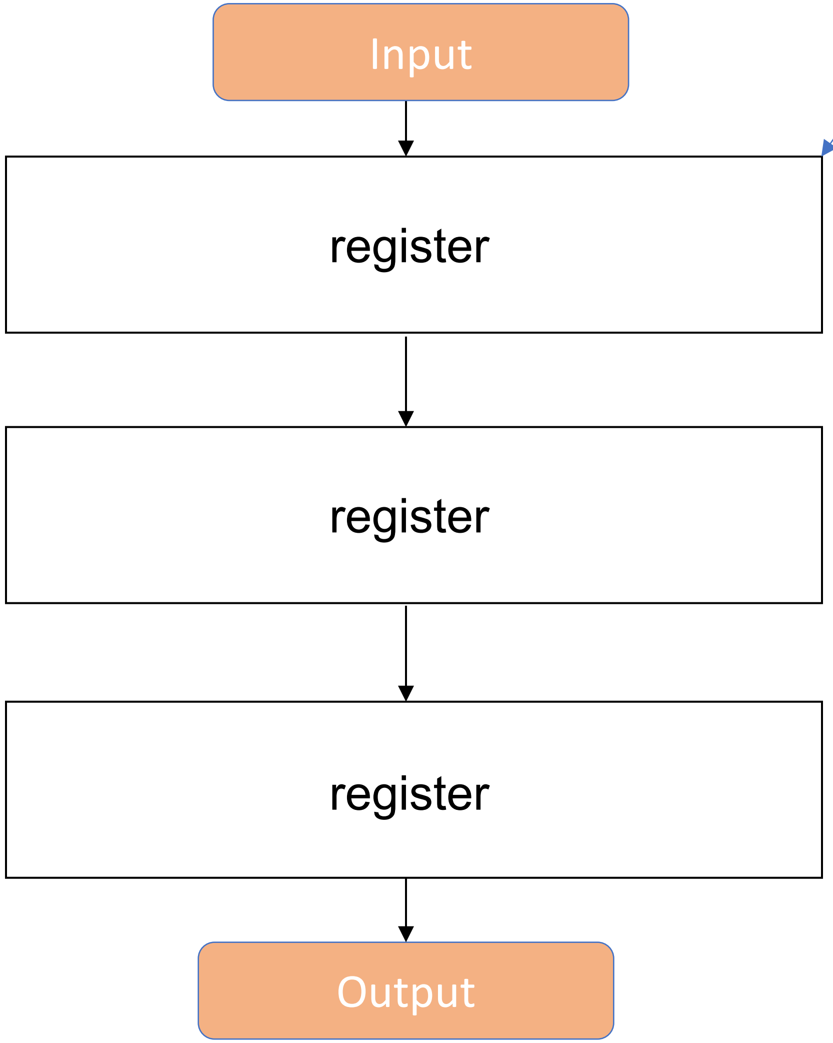


Slave latch records what is presented on its input.

Slave latch outputs what was recorded when WE fell to 0. Master latch outputs its inputs.

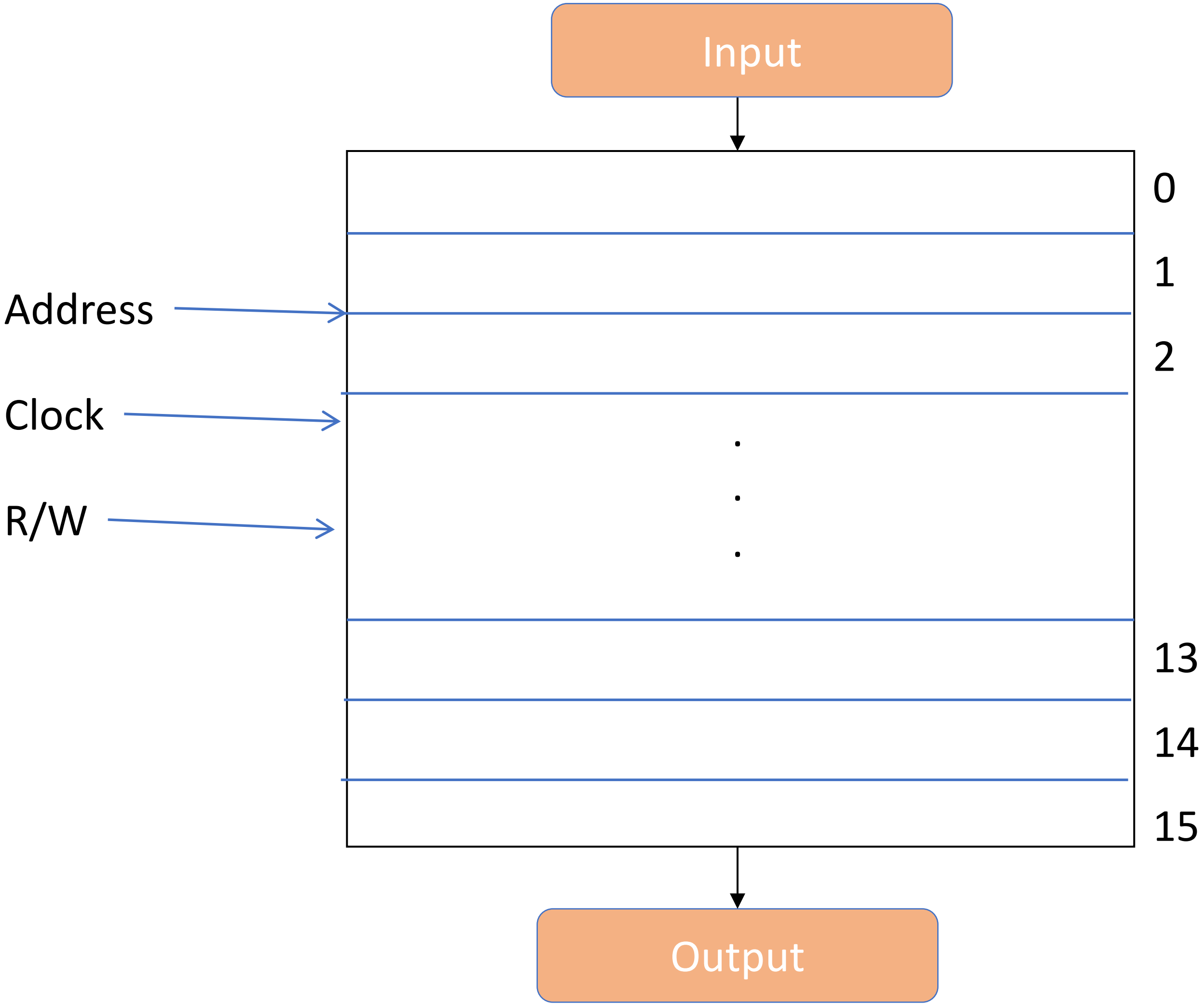
Now we just need to connect Write Enable to the clock....

How Many Clock Cycles from Input to Output?

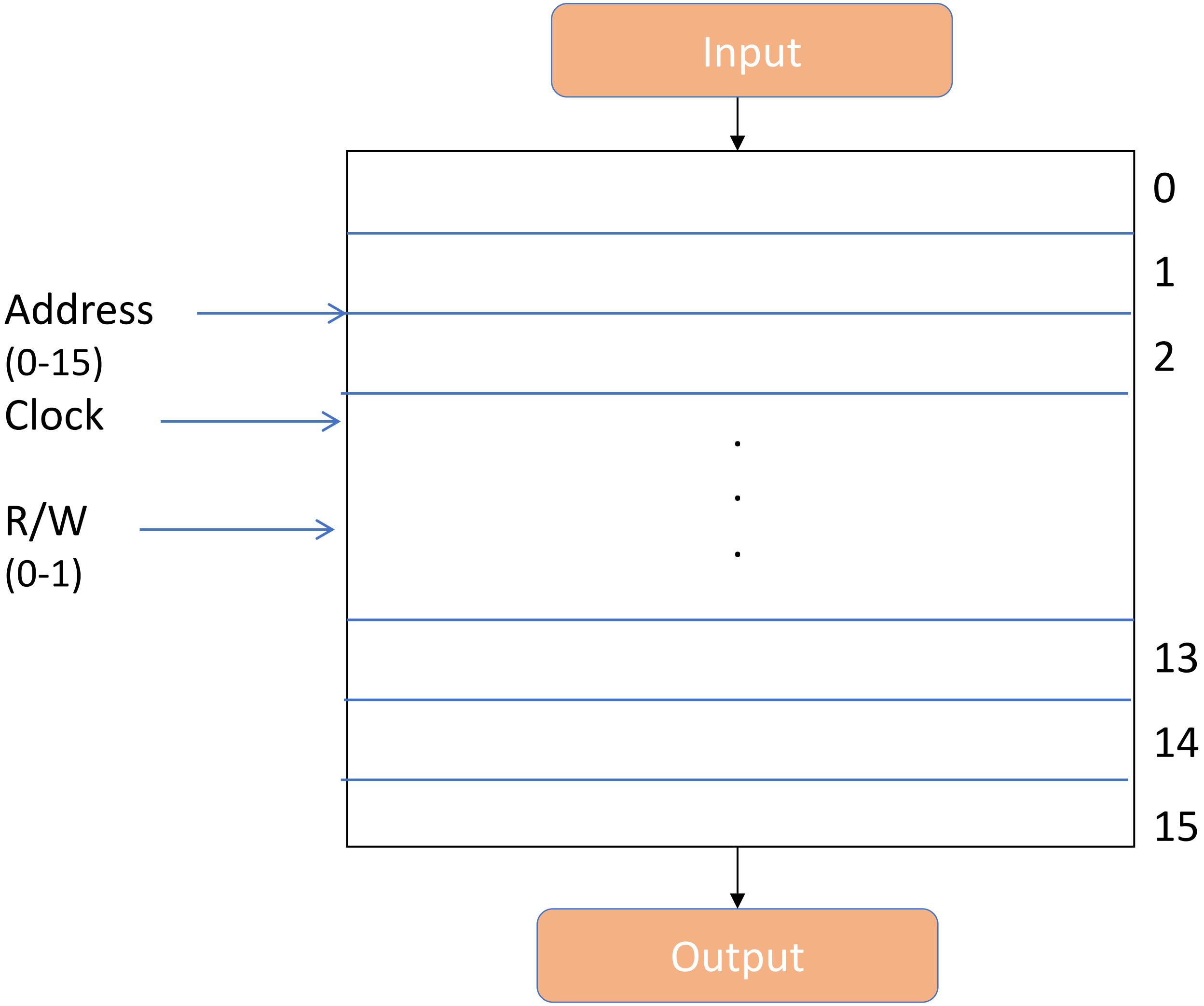


An array of flip-flops (one per bit)

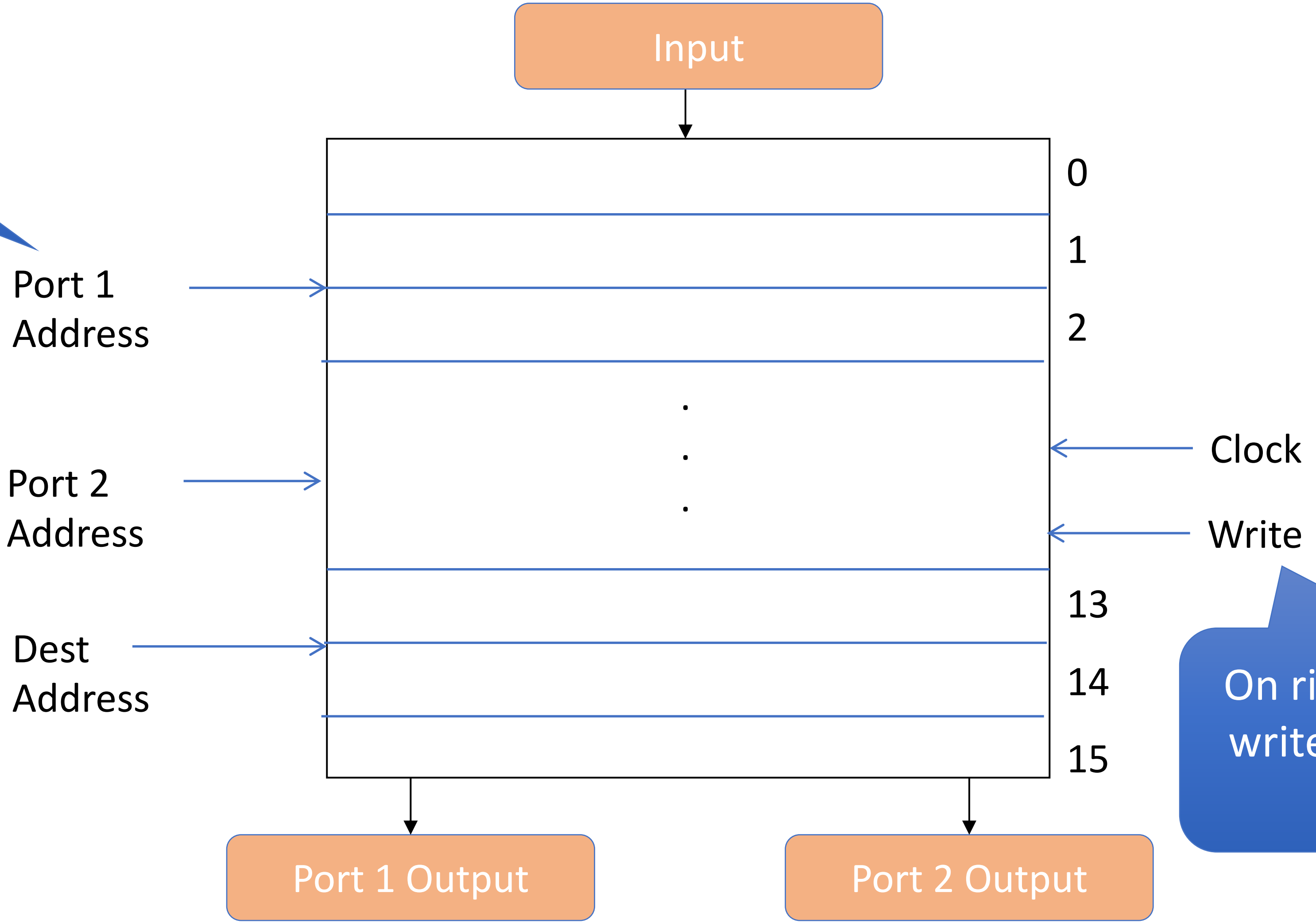
Register File



Register File



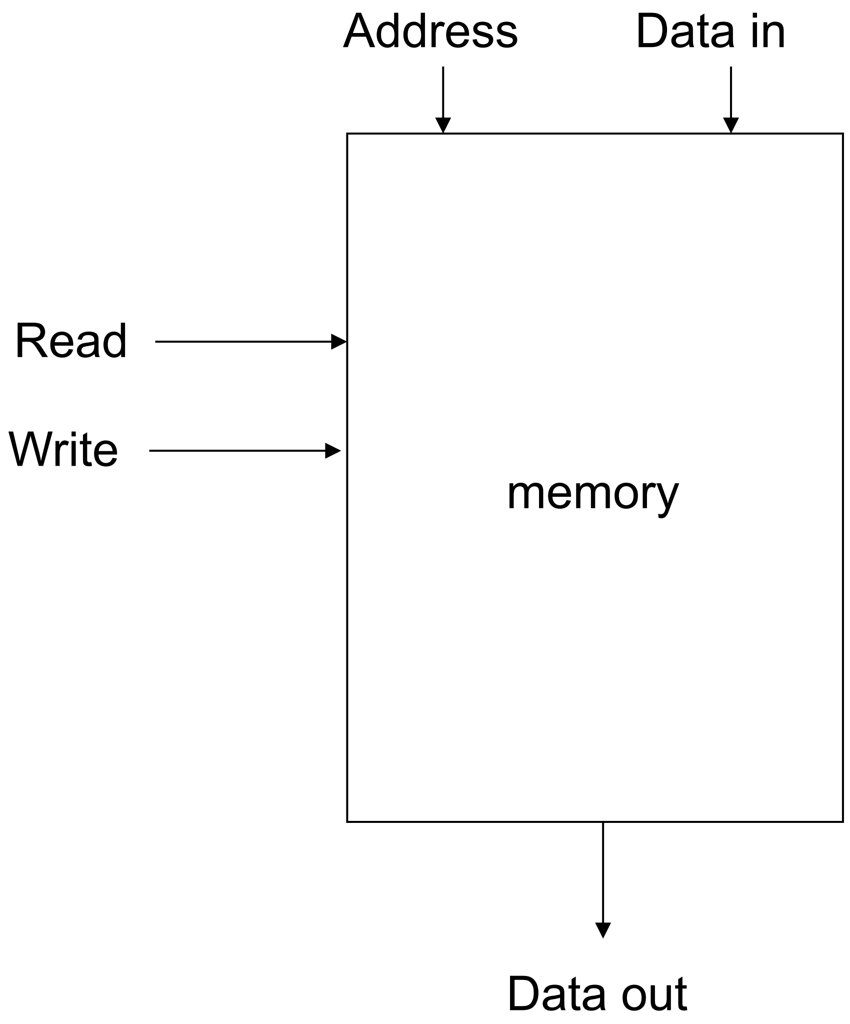
Dual Ported Register File



Lets us read two registers in one clock cycle

On rising edge of clock, write input to the Dest address

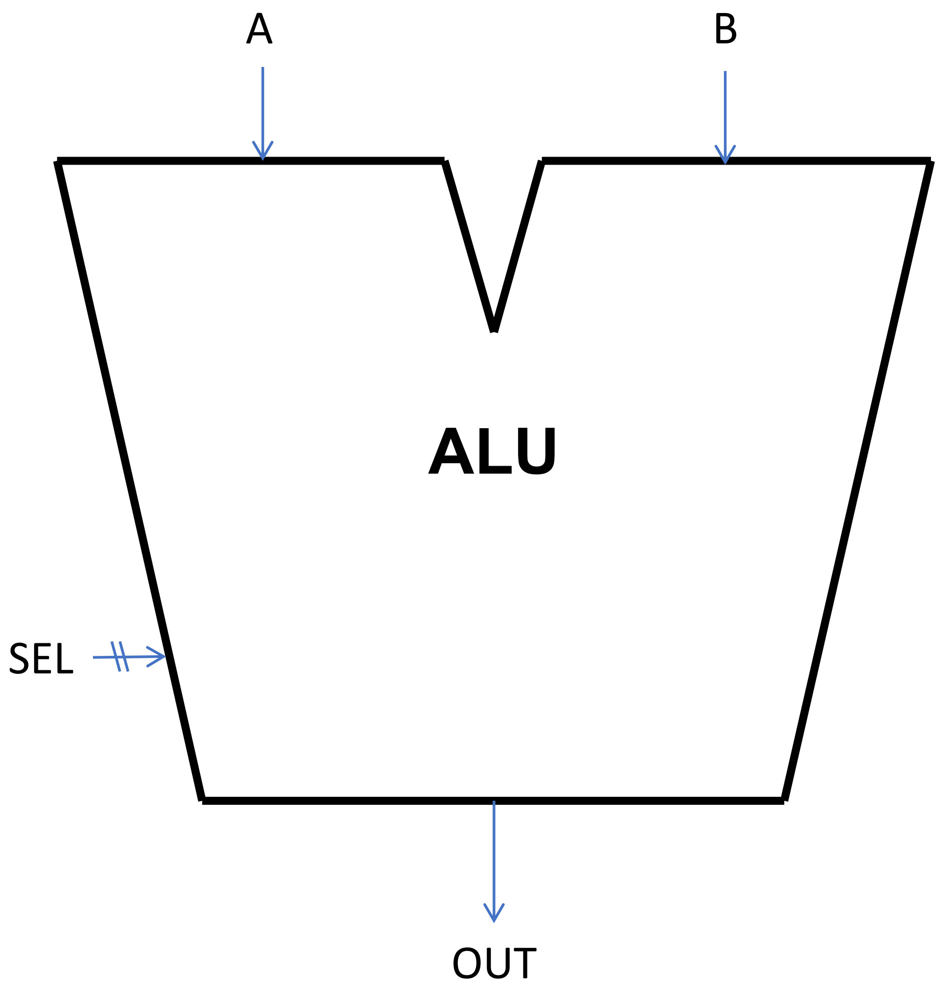
Memory (DRAM)



- Works differently from a register
  - Not clocked (for the purposes of cs2200)
  - Level-triggered logic
- Each DRAM cell is a single transistor and capacitor → higher density but slower

A Typical ALU

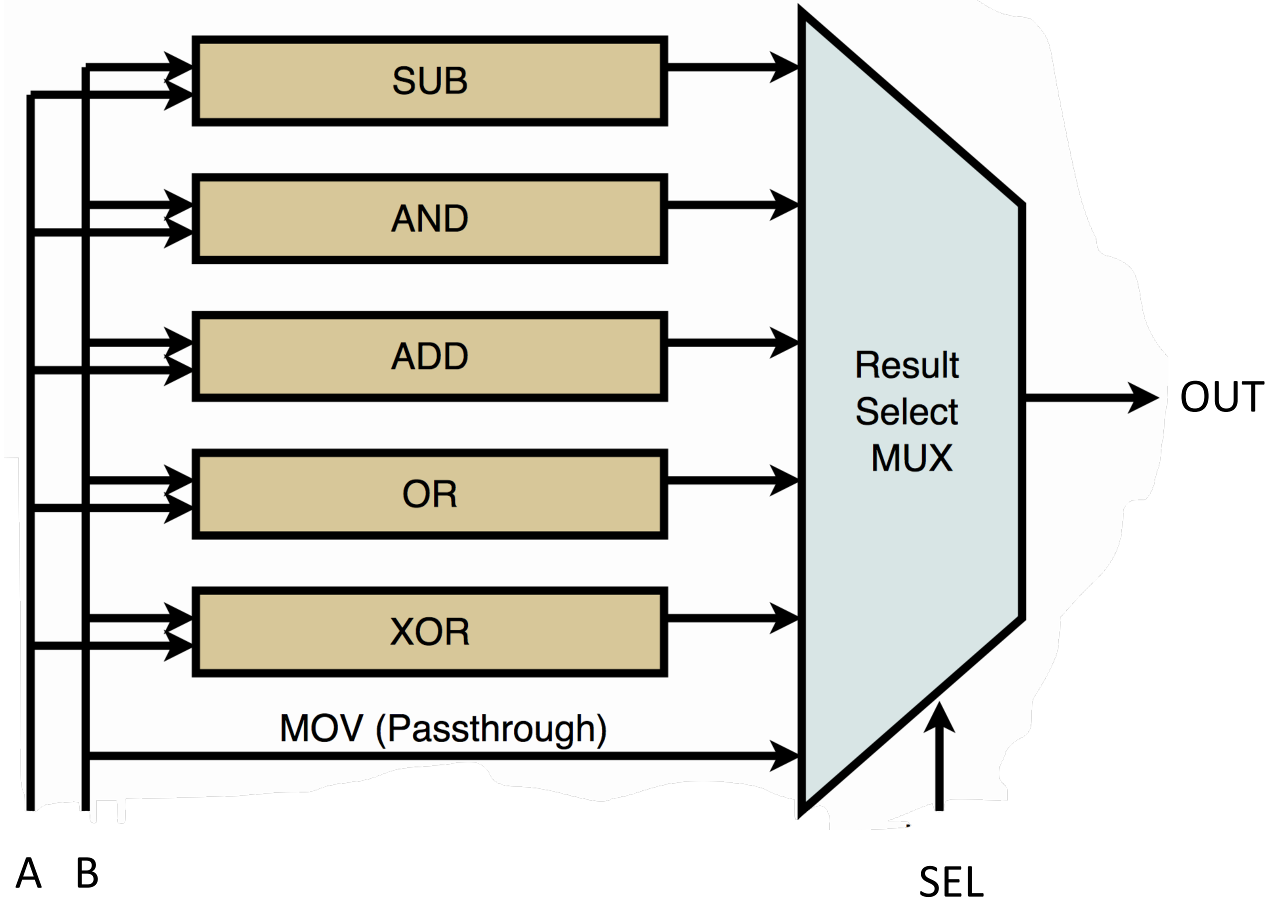
- Combinational circuit
- Two inputs
- One output
- "Function select" selects which functional unit is presented on the output



What's Inside an ALU?

How many bits needed for SEL?

How many bits wide are the data paths?

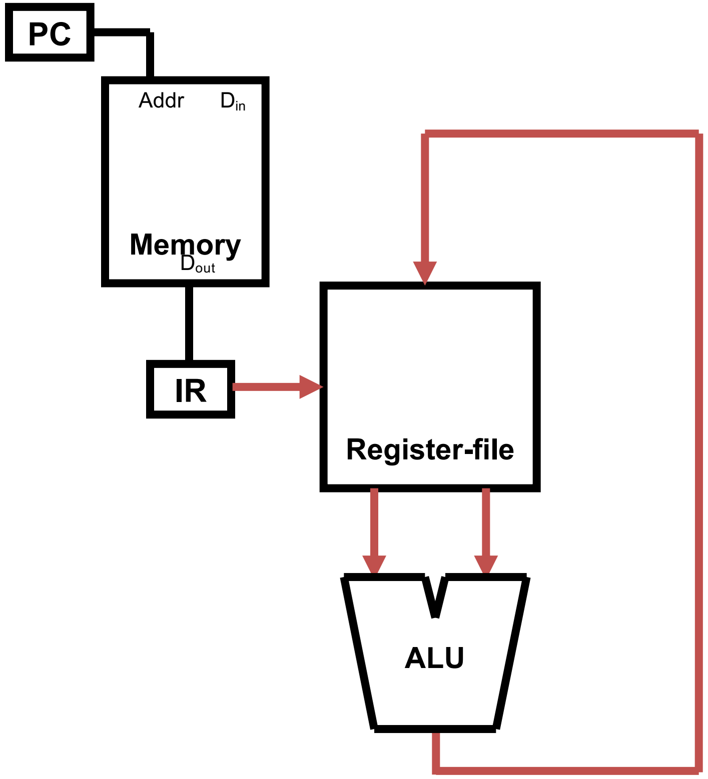


Gisselquist Technology, Inc. https://zipcpu.com/zipcpu/2017/08/11/simple-alu.html

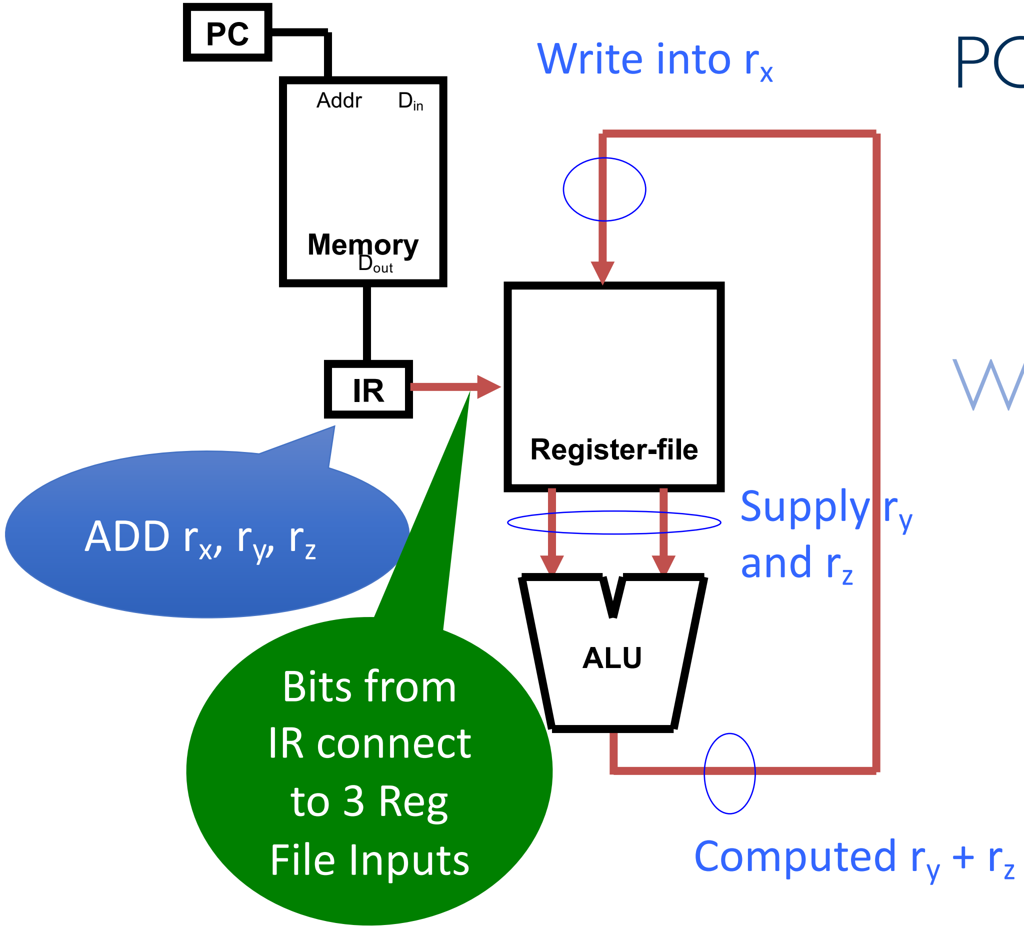
Review Components

- Clock signal
- Register file – read: level triggered
  write: edge triggered
- Memory – read: level triggered (in LC-2200)
  write: edge triggered
- ALU – Level triggered (combinational)

Connecting the Datapath Elements



Connecting the Datapath Elements



PC → ADD r_x, r_y, r_z

↓

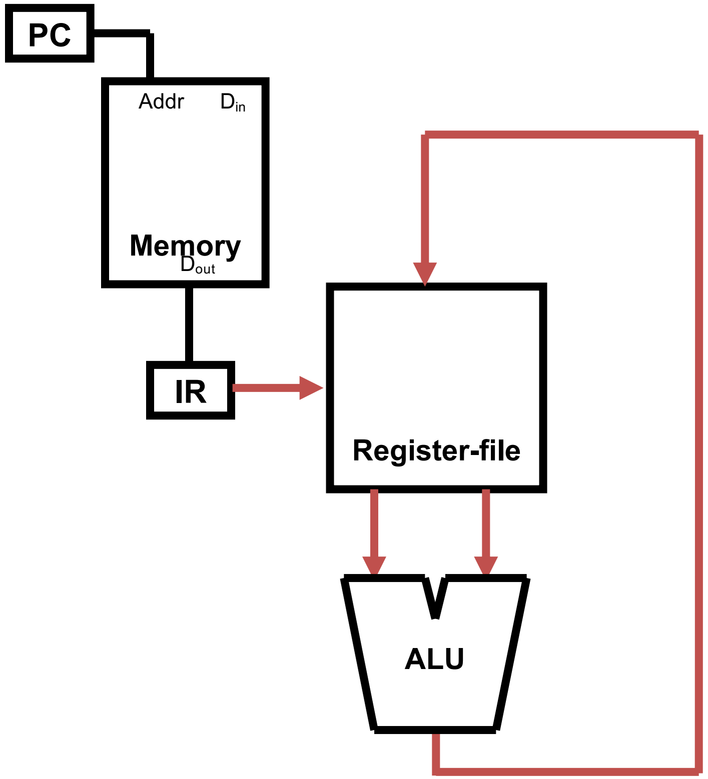
$r_x <= r_y + r_z$

Work done:

PC → Mem → IR →

Reg File → ALU → Reg File

How Can We Tell?

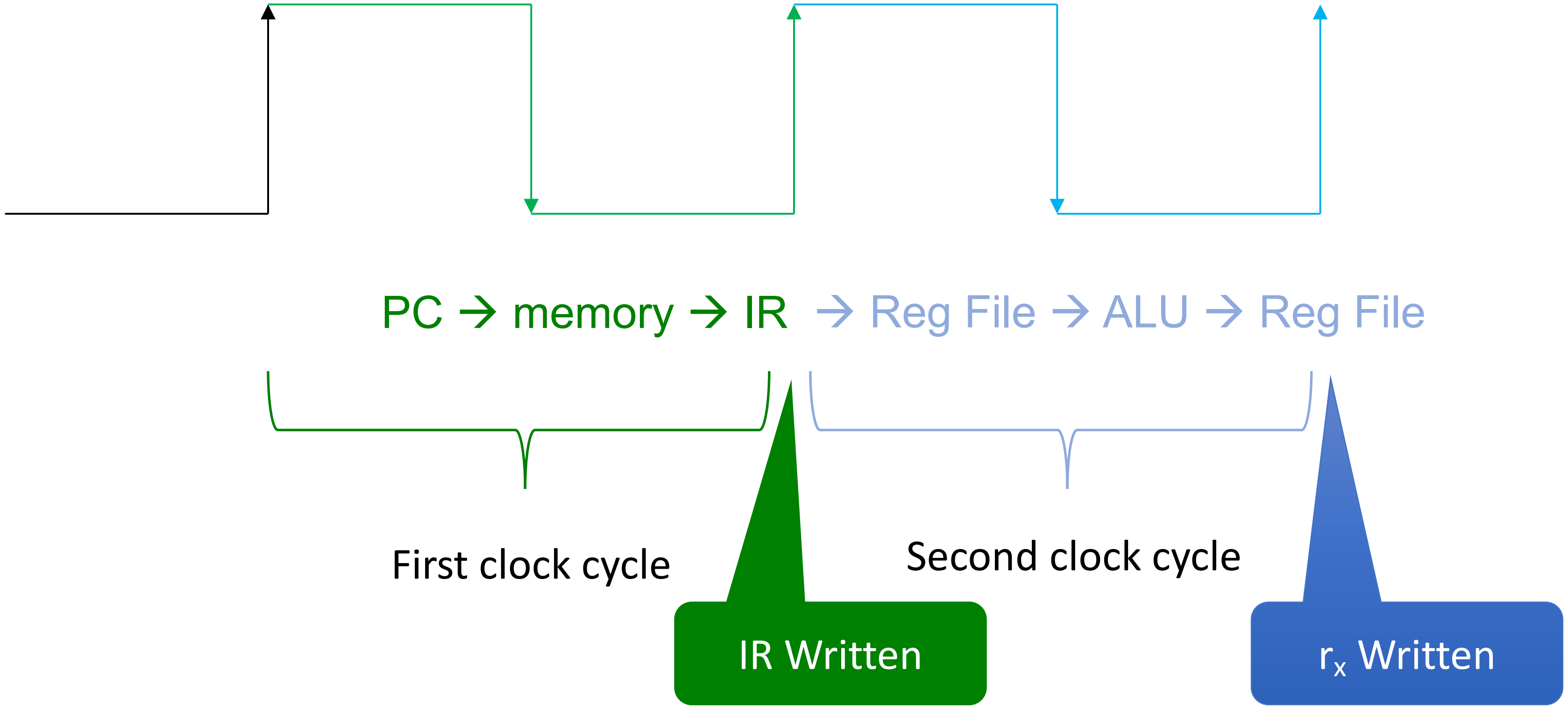


$\underbrace{\text{PC} \rightarrow \text{Mem}}_{\text{Comb + Level}} \rightarrow \text{IR} \rightarrow$

Rising Edge

$\underbrace{\text{Reg File} \rightarrow \text{ALU}}_{\text{Comb + Level}} \rightarrow \text{Reg File}$

Two Clock Cycles



PC → memory → IR → Reg File → ALU → Reg File

First clock cycle

Second clock cycle

IR Written

r_x Written

Session ID: dag2200

How Many Clock Cycles?

- How many clock cycles did it take to execute

  PC → Mem → IR → Reg File → ALU → Reg File

A. One

B. Two

C. Three

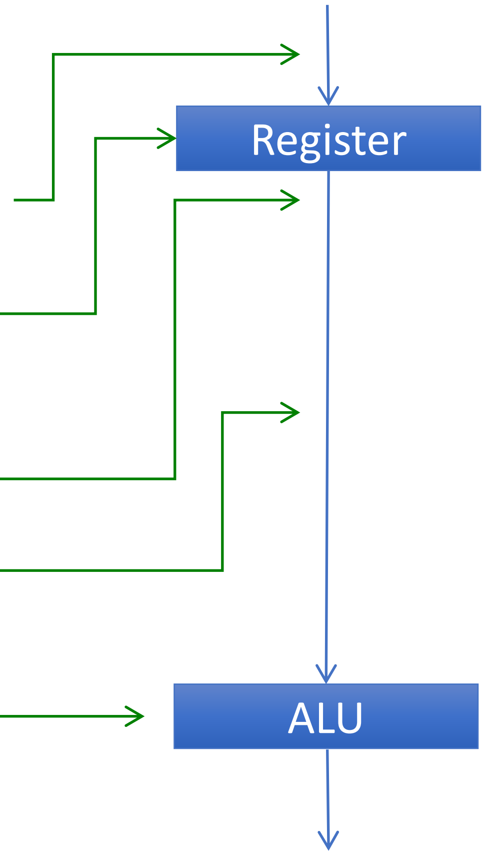
D. Four

E. Five

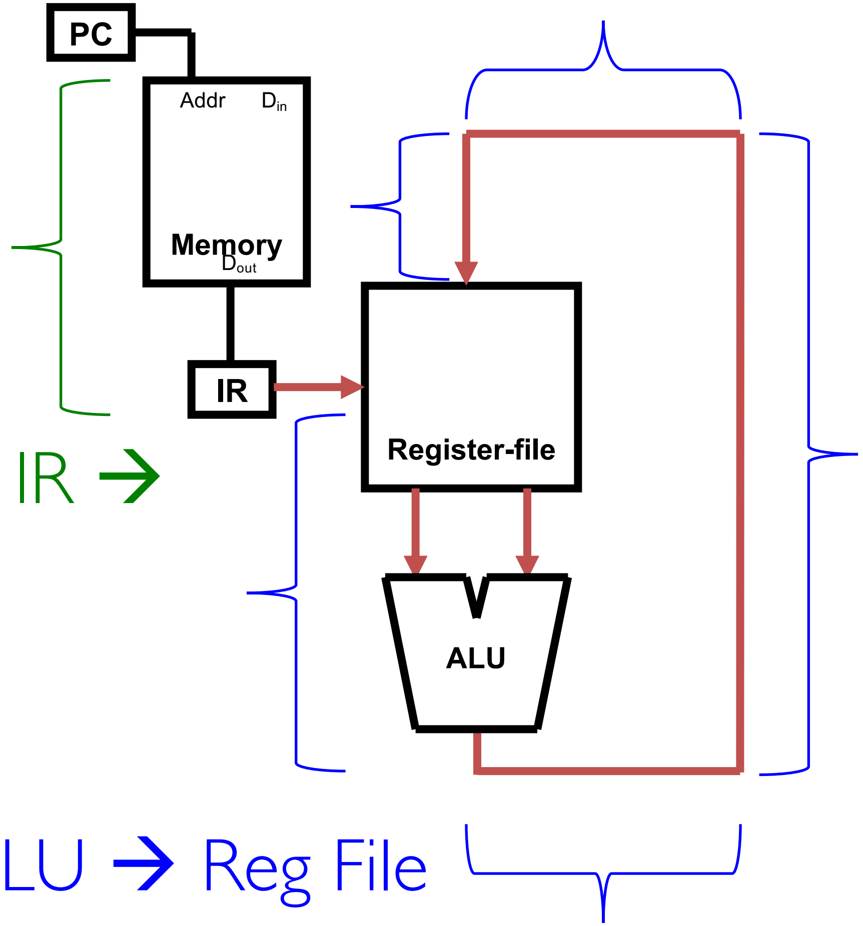
F. Six

Delays

- Register Input
  - Setup (before clock edge)
  - Hold (after clock edge)
- Output stable
  - (before it can be used)
- Propagation
  - (wire)
- ALU op
  - (combinational logic)



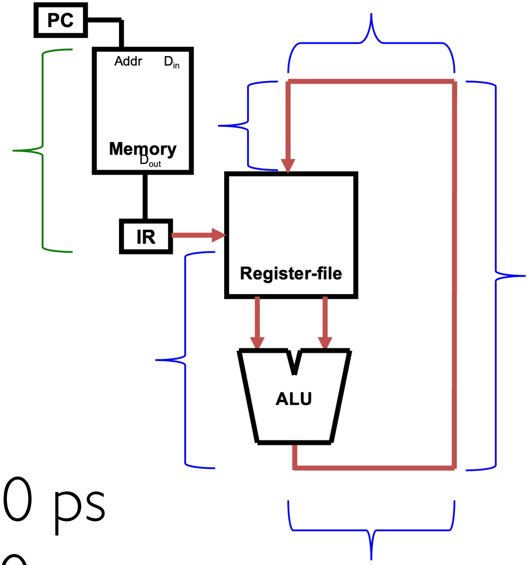
Clock Duration



PC → memory → IR →

Reg File → ALU → Reg File

Example Delay Parameters



Given the following parameters (all in picoseconds), determine the minimum clock width of the system.

$D_{\text{r-ouput-stable}}$	(PC output stable)	-	20 ps
$D_{\text{wire-PC-Addr}}$	(wire delay from PC to Addr of Memory)	-	250 ps
$D_{\text{mem-read}}$	(Memory read)	-	1500 ps
$D_{\text{wire-Dout-IR}}$	(wire delay from Dout of Memory to IR)	-	250 ps
$D_{\text{r-setup}}$	(setup time for IR)	-	20 ps
$D_{\text{r-hold}}$	(hold time for IR)	-	20 ps
$D_{\text{wire-IR-regfile}}$	(wire delay from IR to Register file)	-	250 ps
$D_{\text{regfile-read}}$	(Register file read)	-	500 ps
$D_{\text{wire-regfile-ALU}}$	(wire delay from Register file to input of ALU)	-	250 ps
$D_{\text{ALU-OP}}$	(time to perform ALU operation)	-	100 ps
$D_{\text{wire-ALU-regfile}}$	(wire delay from ALU output to Register file)	-	250 ps
$D_{\text{regfile-write}}$	(time for writing into a Register file)	-	500 ps

Session ID: dag2200

What Should the Duration of Clock Cycle Be?

A. Ask Intel

B. Hmm maybe AMD knows better

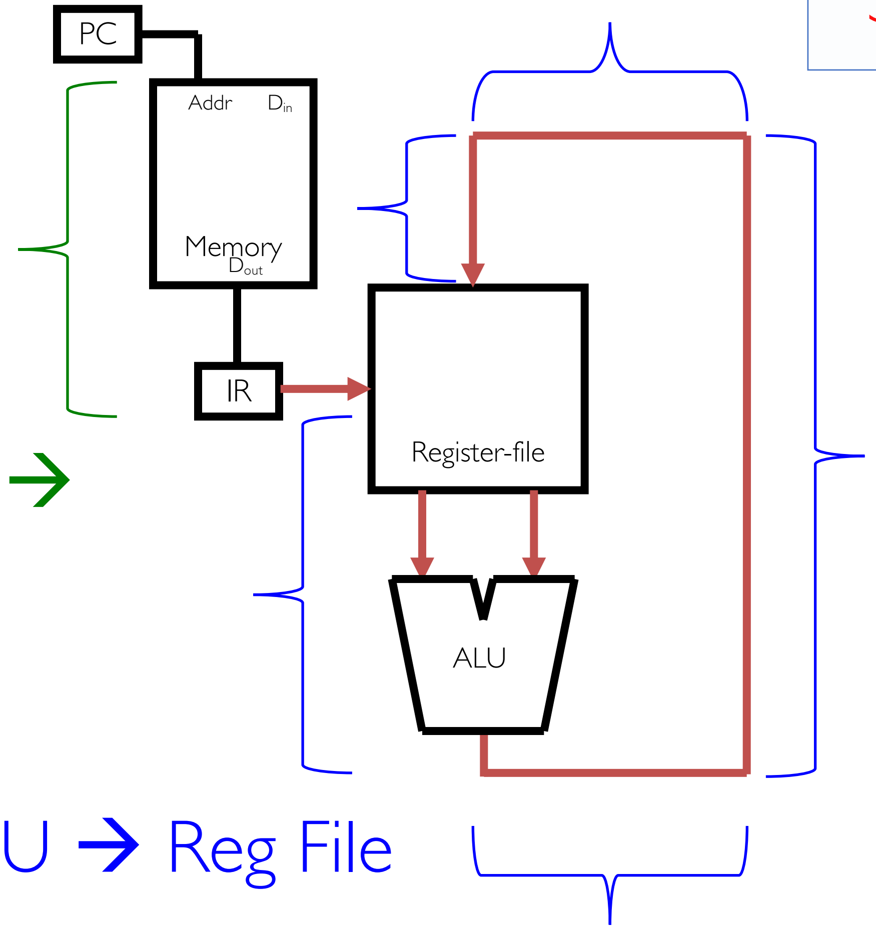
C. Add all the data path delays

D. Set it arbitrarily and hope it works
(42 ns sounds like a good number)

(that must be traversed in a single cycle)

Calculating Clock Duration

Split circuits at edge-triggered devices

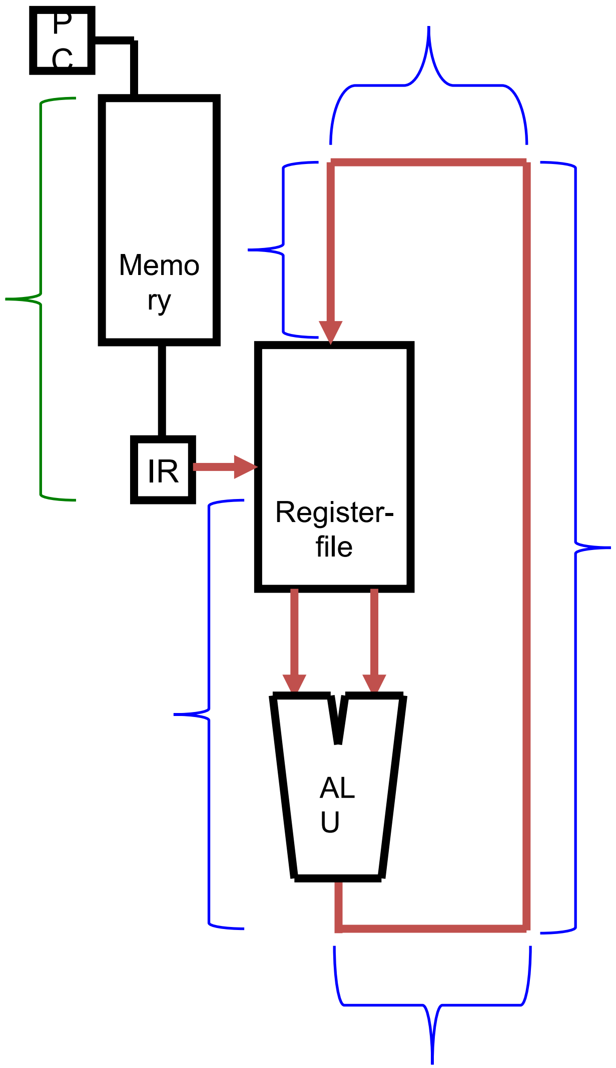


PC → memory → IR →

Reg File → ALU → Reg File

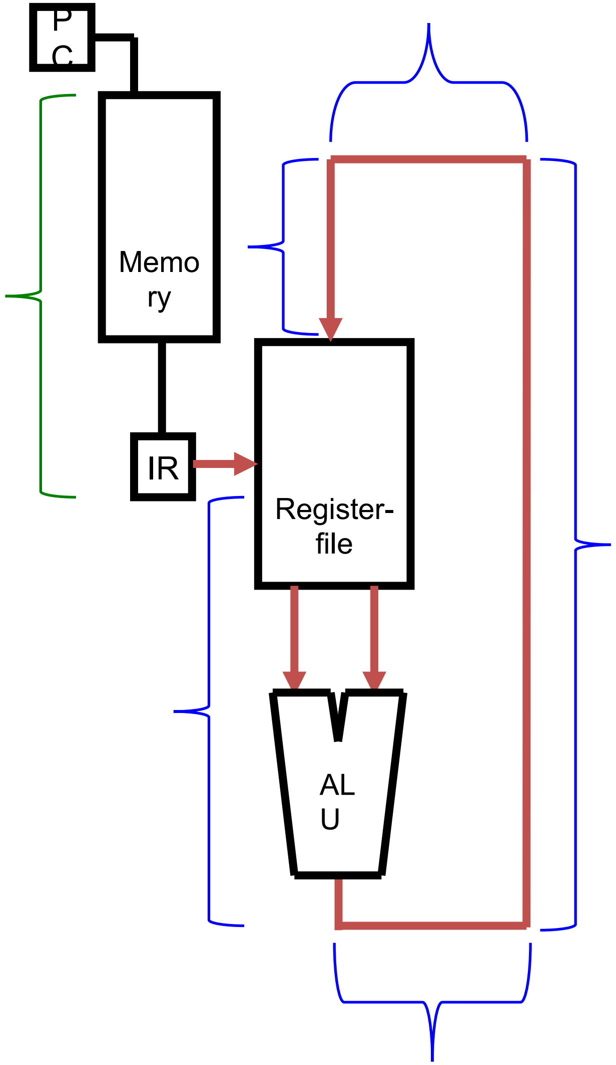
max(green time, blue time)

Calculating Clock Minimums



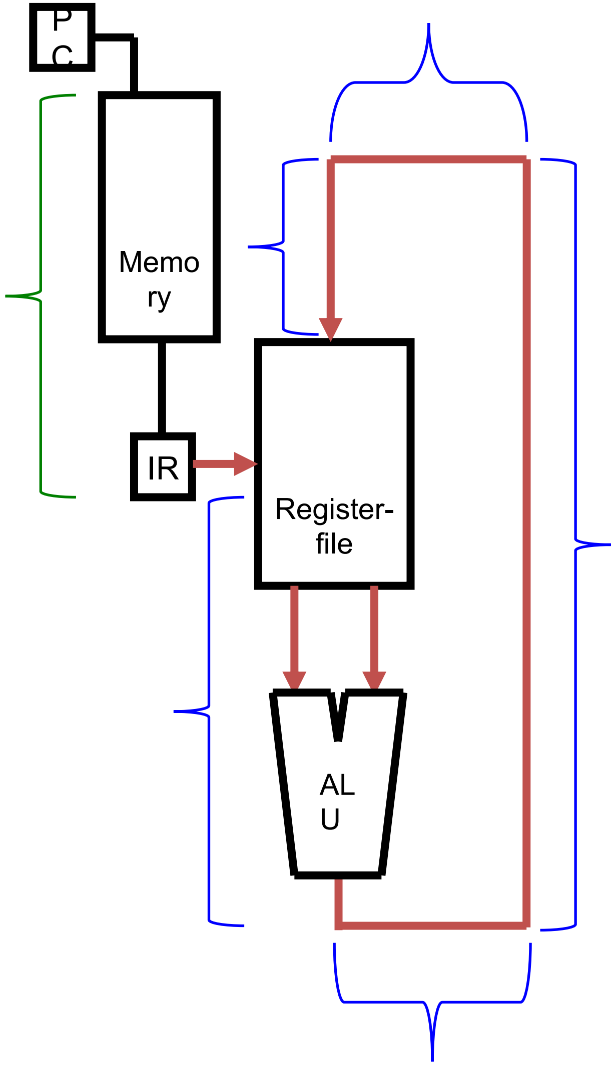
	Delay (ps)	Green bracket	Blue bracket 1	Blue brackets 2-5	
$D_{\text{r-ouput-stable}}$	20	20			(PC output stable)
$D_{\text{wire-PC-Addr}}$	250	250			(wire delay from PC to Addr of Memory)
$D_{\text{mem-read}}$	1500	1500			(Memory read)
$D_{\text{wire-Dout-IR}}$	250	250			(wire delay from Dout of Memory to IR)
$D_{\text{r-setup}}$	20	20			(setup time for IR)
$D_{\text{r-hold}}$	20	20			(hold time for IR)
$D_{\text{wire-IR-regfile}}$	250		250		(wire delay from IR to Register file)
$D_{\text{regfile-read}}$	500		500		(Register file read)
$D_{\text{wire-regfile-ALU}}$	250		250		(wire delay from Register file to input of ALU)
$D_{\text{ALU-OP}}$	100		100		(time to perform ALU operation)
$D_{\text{wire-ALU-regfile}}$	250			250	(wire delay from ALU output to Register file)
$D_{\text{regfile-write}}$	500			500	(time for writing into a Register file)

Calculating Clock Minimums



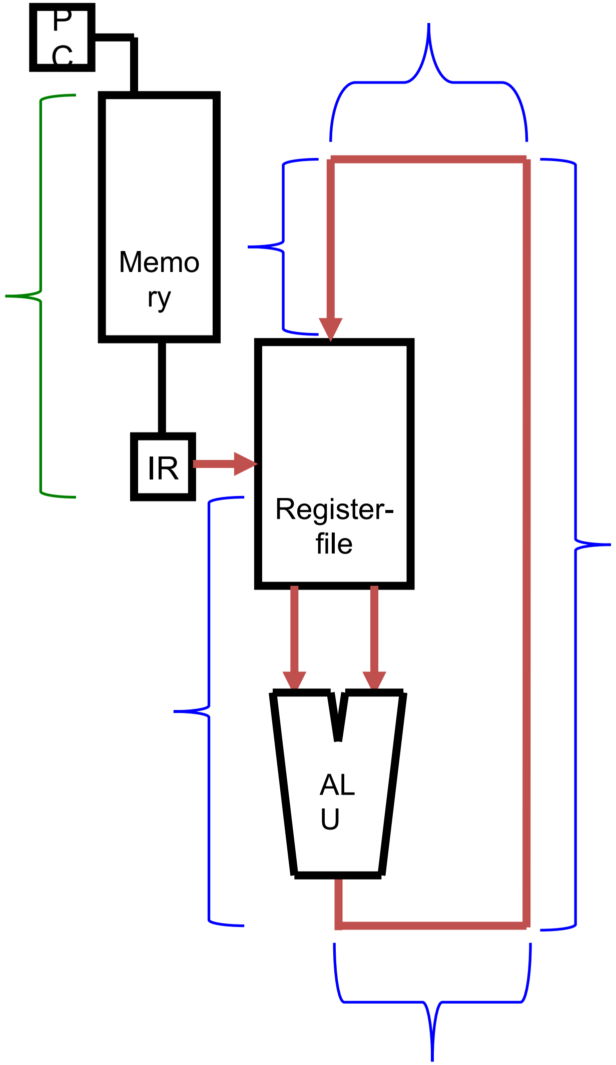
	Delay (ps)	Green bracket	Blue bracket 1	Blue brackets 2-5	
$D_{\text{r-ouput-stable}}$	20	20			(PC output stable)
$D_{\text{wire-PC-Addr}}$	250	250			(wire delay from PC to Addr of Memory)
$D_{\text{mem-read}}$	1500	1500			(Memory read)
$D_{\text{wire-Dout-IR}}$	250	250			(wire delay from Dout of Memory to IR)
$D_{\text{r-setup}}$	20	20			(setup time for IR)
$D_{\text{r-hold}}$	20	20			(hold time for IR)
$D_{\text{wire-IR-regfile}}$	250		250		(wire delay from IR to Register file)
$D_{\text{regfile-read}}$	500		500		(Register file read)
$D_{\text{wire-regfile-ALU}}$	250		250		(wire delay from Register file to input of ALU)
$D_{\text{ALU-OP}}$	100		100		(time to perform ALU operation)
$D_{\text{wire-ALU-regfile}}$	250			250	(wire delay from ALU output to Register file)
$D_{\text{regfile-write}}$	500			500	(time for writing into a Register file)
		2060			

Calculating Clock Minimums



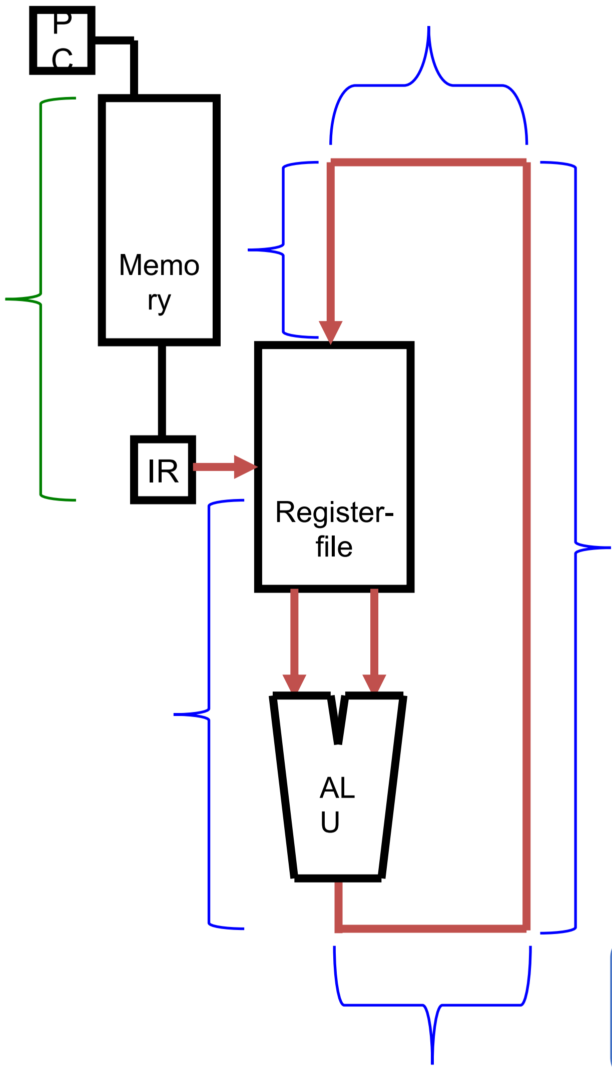
	Delay (ps)	Green bracket	Blue bracket 1	Blue brackets 2-5	
$D_{\text{r-ouput-stable}}$	20	20			(PC output stable)
$D_{\text{wire-PC-Addr}}$	250	250			(wire delay from PC to Addr of Memory)
$D_{\text{mem-read}}$	1500	1500			(Memory read)
$D_{\text{wire-Dout-IR}}$	250	250			(wire delay from Dout of Memory to IR)
$D_{\text{r-setup}}$	20	20			(setup time for IR)
$D_{\text{r-hold}}$	20	20			(hold time for IR)
$D_{\text{wire-IR-regfile}}$	250		250		(wire delay from IR to Register file)
$D_{\text{regfile-read}}$	500		500		(Register file read)
$D_{\text{wire-regfile-ALU}}$	250		250		(wire delay from Register file to input of ALU)
$D_{\text{ALU-OP}}$	100		100		(time to perform ALU operation)
$D_{\text{wire-ALU-regfile}}$	250			250	(wire delay from ALU output to Register file)
$D_{\text{regfile-write}}$	500			500	(time for writing into a Register file)
		2060	1100	750	

Calculating Clock Minimums



	Delay (ps)	Green bracket	Blue bracket 1	Blue brackets 2-5	
$D_{\text{r-ouput-stable}}$	20	20			(PC output stable)
$D_{\text{wire-PC-Addr}}$	250	250			(wire delay from PC to Addr of Memory)
$D_{\text{mem-read}}$	1500	1500			(Memory read)
$D_{\text{wire-Dout-IR}}$	250	250			(wire delay from Dout of Memory to IR)
$D_{\text{r-setup}}$	20	20			(setup time for IR)
$D_{\text{r-hold}}$	20	20			(hold time for IR)
$D_{\text{wire-IR-regfile}}$	250		250		(wire delay from IR to Register file)
$D_{\text{regfile-read}}$	500		500		(Register file read)
$D_{\text{wire-regfile-ALU}}$	250		250		(wire delay from Register file to input of ALU)
$D_{\text{ALU-OP}}$	100		100		(time to perform ALU operation)
$D_{\text{wire-ALU-regfile}}$	250			250	(wire delay from ALU output to Register file)
$D_{\text{regfile-write}}$	500			500	(time for writing into a Register file)
		2060	1100	750	
				1850	

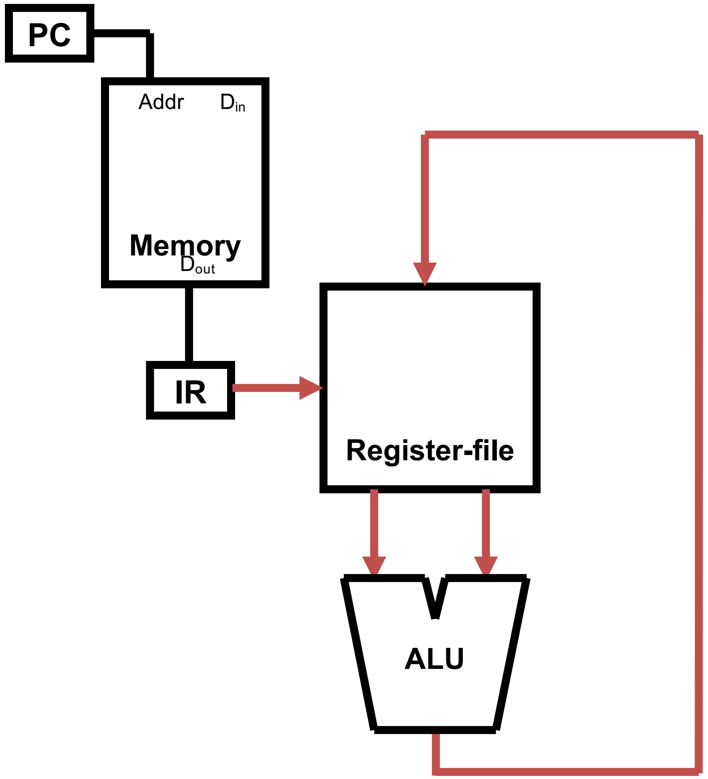
Calculating Clock Minimums



	Delay (ps)	Green bracket	Blue bracket 1	Blue brackets 2-5	
$D_{\text{r-ouput-stable}}$	20	20			(PC output stable)
$D_{\text{wire-PC-Addr}}$	250	250			(wire delay from PC to Addr of Memory)
$D_{\text{mem-read}}$	1500	1500			(Memory read)
$D_{\text{wire-Dout-IR}}$	250	250			(wire delay from Dout of Memory to IR)
$D_{\text{r-setup}}$	20	20			(setup time for IR)
$D_{\text{r-hold}}$	20	20			(hold time for IR)
$D_{\text{wire-IR-regfile}}$	250		250		(wire delay from IR to Register file)
$D_{\text{regfile-read}}$	500		500		(Register file read)
$D_{\text{wire-regfile-ALU}}$	250		250		(wire delay from Register file to input of ALU)
$D_{\text{ALU-OP}}$	100		100		(time to perform ALU operation)
$D_{\text{wire-ALU-regfile}}$	250			250	(wire delay from ALU output to Register file)
$D_{\text{regfile-write}}$	500			500	(time for writing into a Register file)
		2060	1100	750	
				1850	

Clock must be >= 2060 ps

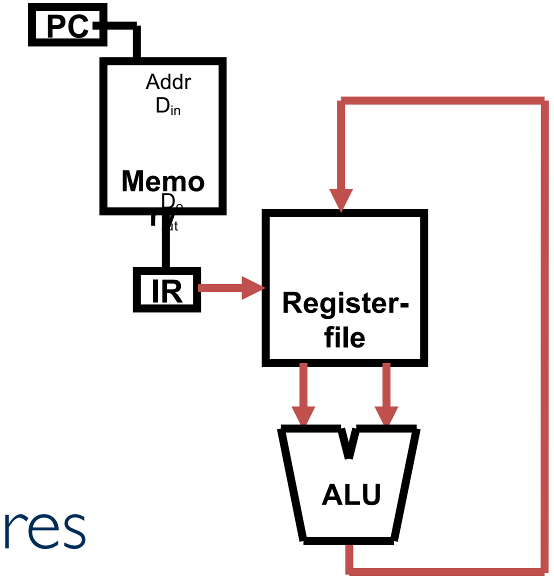
What Other Connections Do We Need?



- R-type instructions (add, nand):
  - Opcode, x, y, z
- I-type instructions (addi, lw, sw, beq):
  - Opcode, x, y, offset
- J-type instructions (jalr):
  - Opcode, x, y
- O-type instructions (halt):
  - Opcode

Adding Connections for Our LC-2200 ISA

- Memory to Reg File input for LW
- Reg File output to Memory for SW
- IR to ALU In to carry offset
- ALU Out to PC for JALR



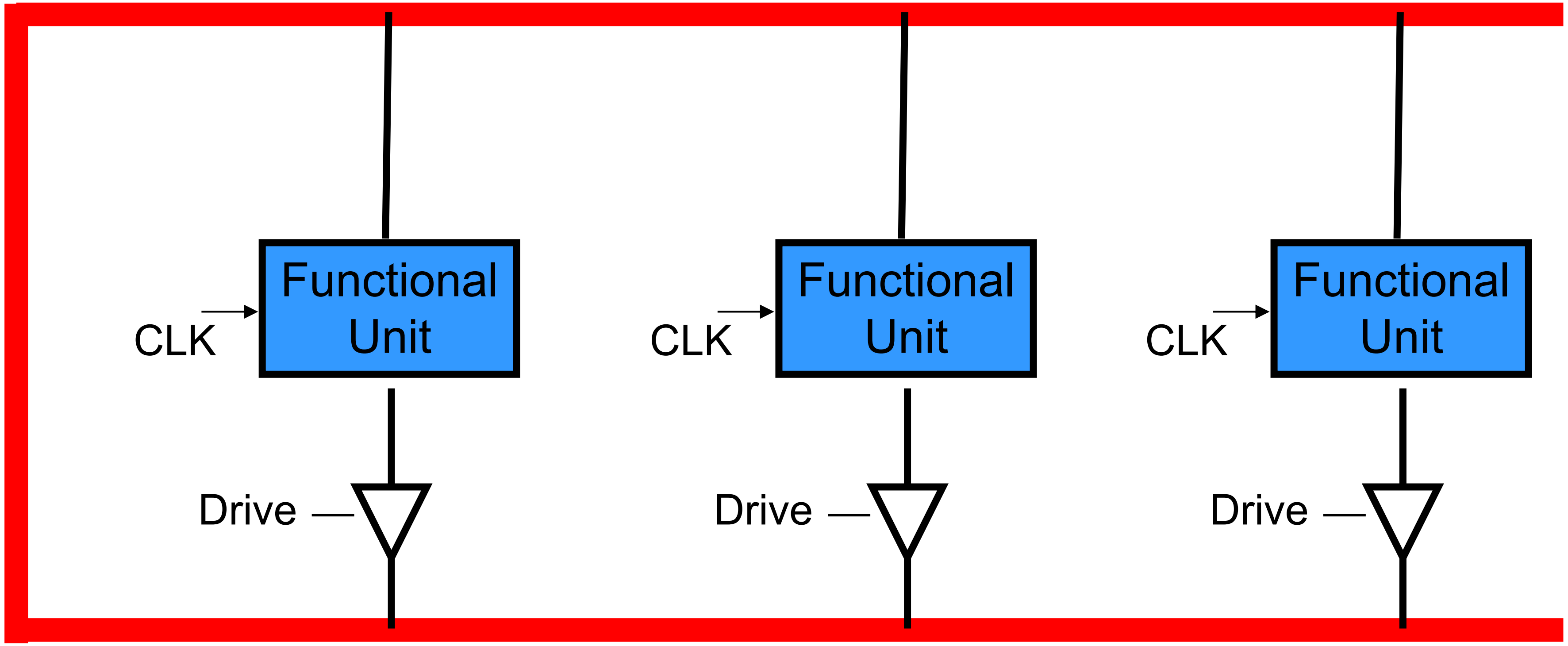
- In other words, lots more wires
- Is it possible to share some of them?

- R-type instructions (add, nand):
  - Opcode, x, y, z
- I-type instructions (addi, lw, sw, beq):
  - Opcode, x, y, offset
- J-type instructions (jalr):
  - Opcode, x, y
- O-type instructions (halt):
  - Opcode

Towards bus-based design

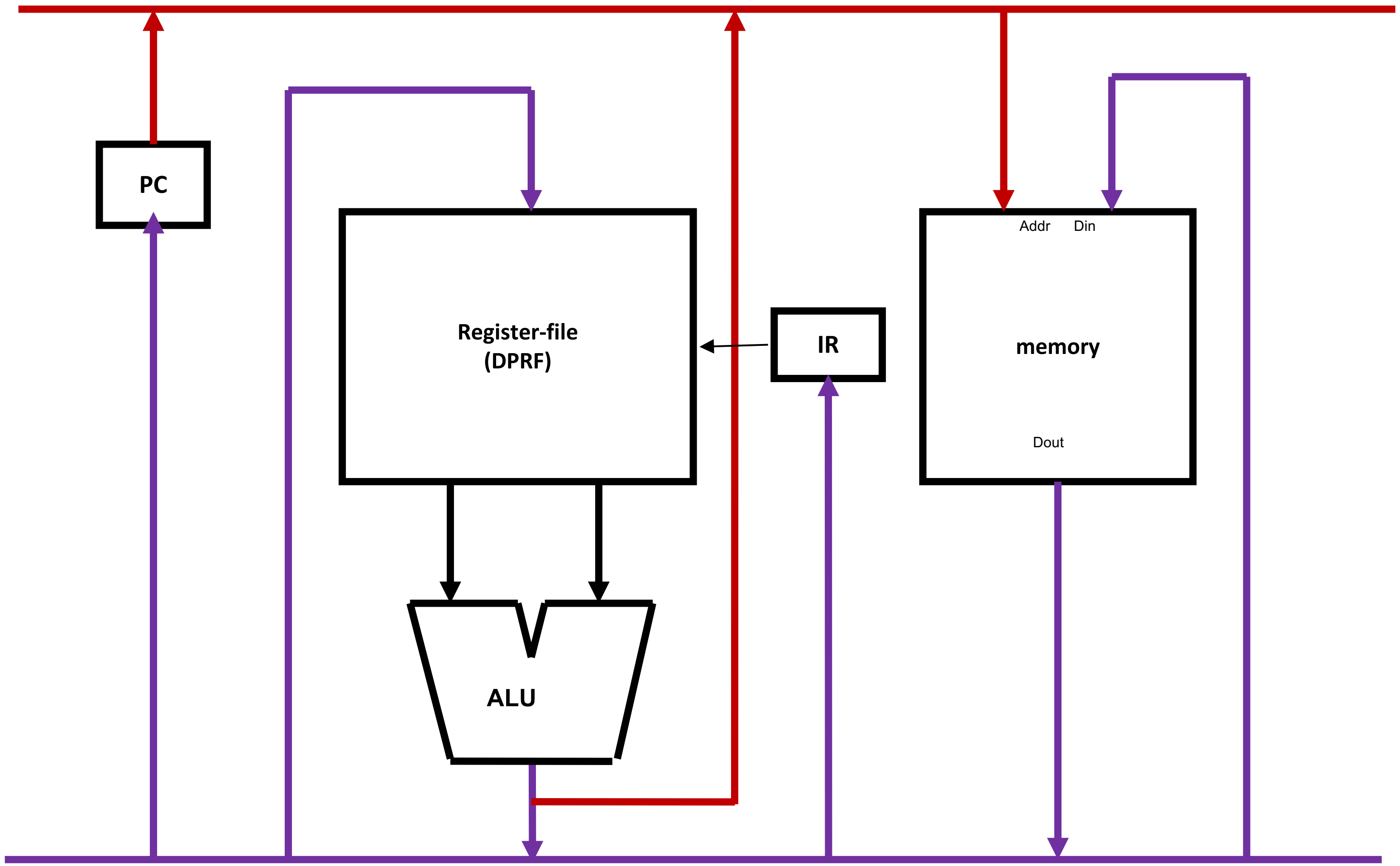
- In principle we must make connections between circuit elements for every instruction
- Numerous connections are expensive and take up valuable space
- Have a set of wires that all elements can connect to and share in order to transfer information

Concept of a Bus



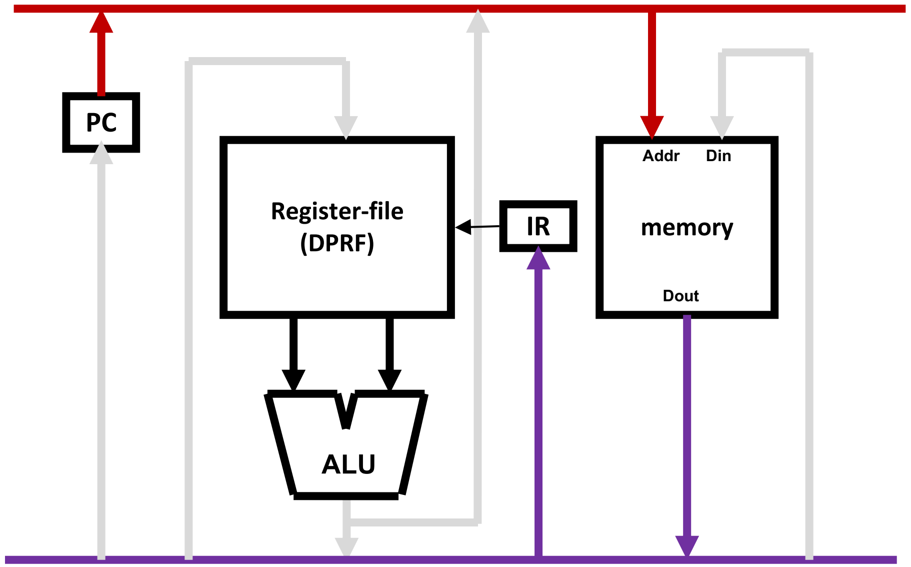
- So what's that "Drive" thing?
- It's another term for a Tri-state Buffer

A Two-Bus Design



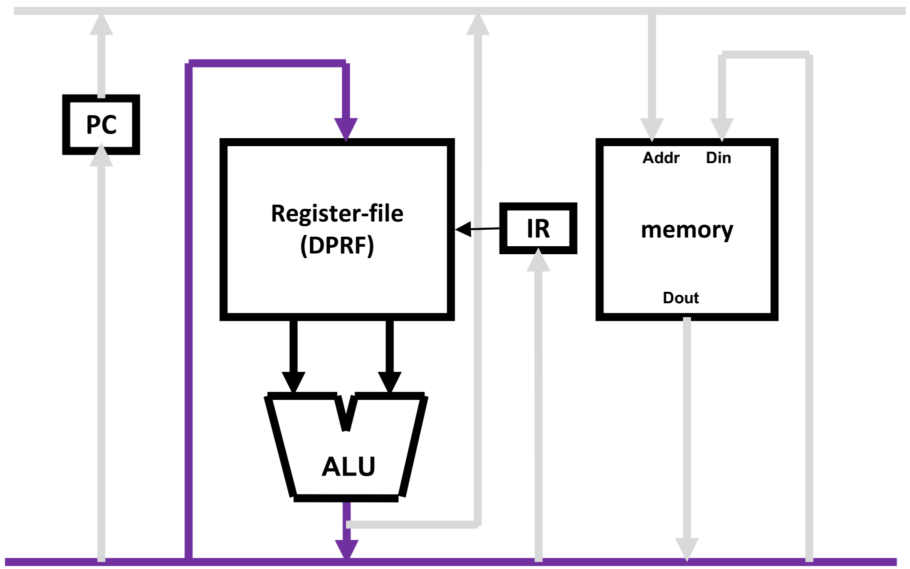
DPRF: Dual-Ported Register File

Clock Cycle



PC → MEM → IR

Clock Cycle



IR → Reg File → ALU → Reg File

Session ID: dag2200

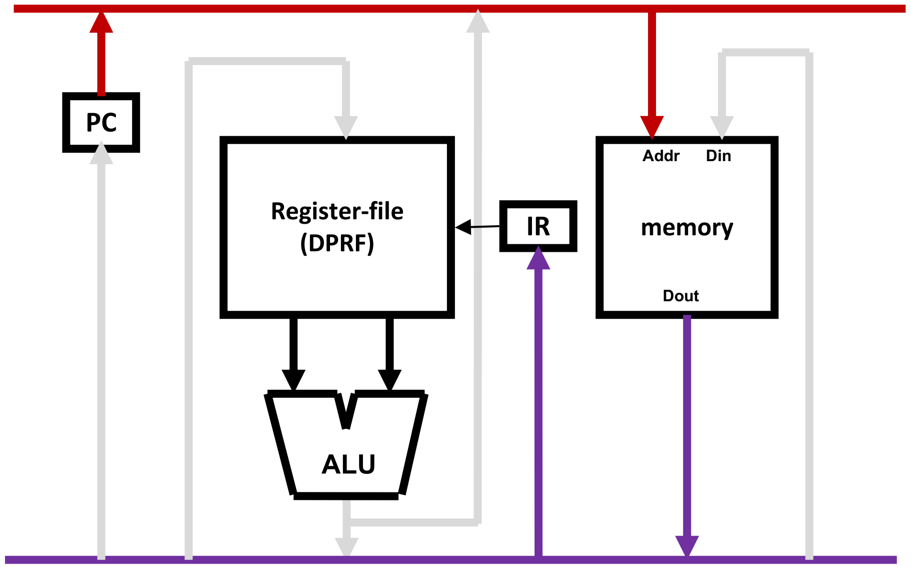
Two Busses: How many cycles?

Using just two busses, how many clock cycles does it take to execute

PC → Mem → IR → Reg File → ALU → Reg File

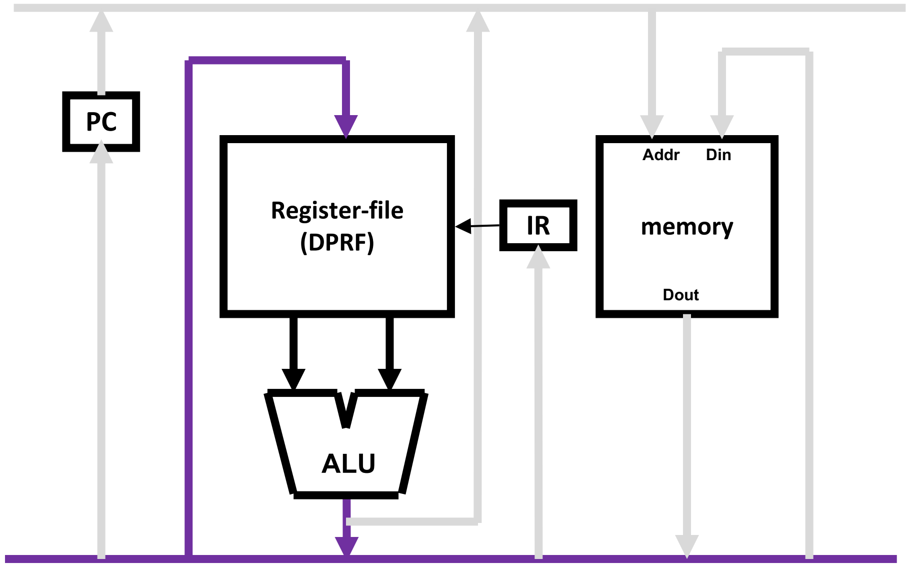
A. One
B. Two
C. Three
D. Four
E. Five
F. Six

First Clock Cycle



First: PC → MEM → IR

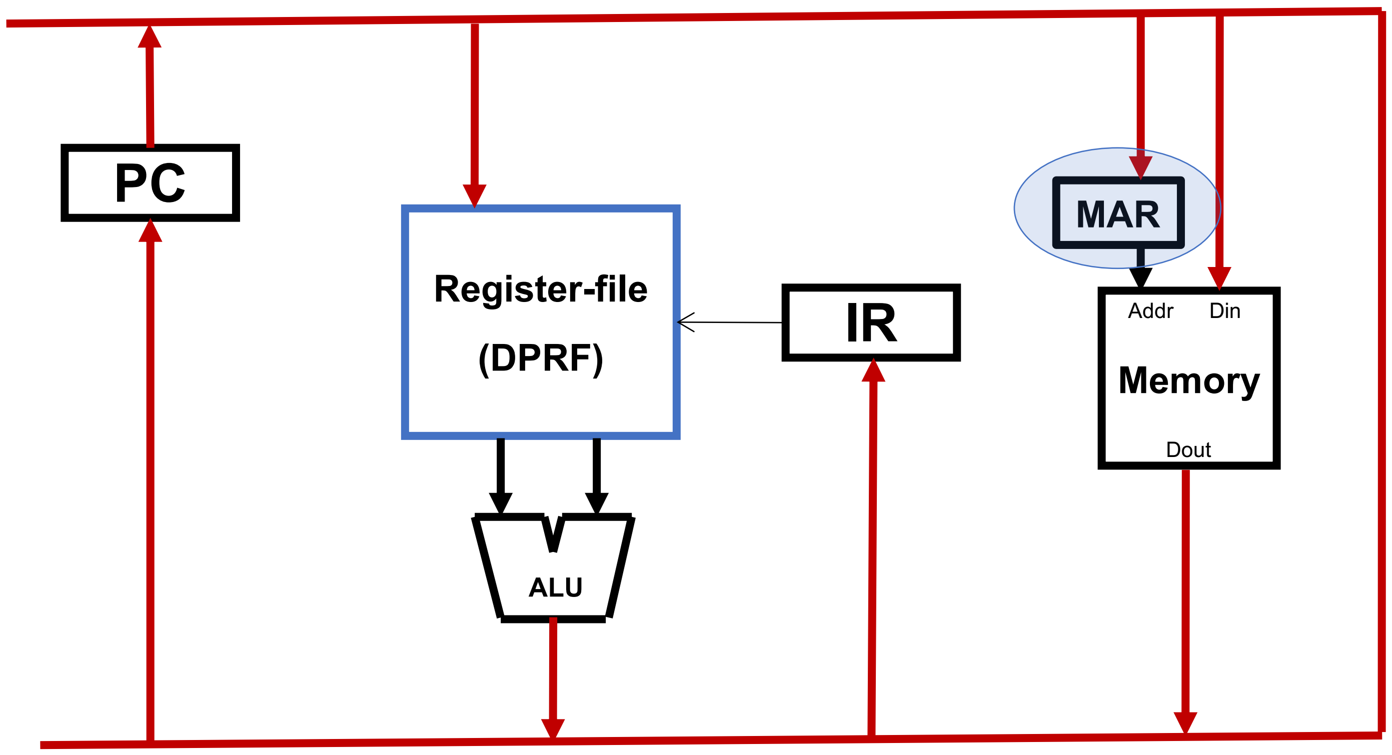
Second Clock Cycle



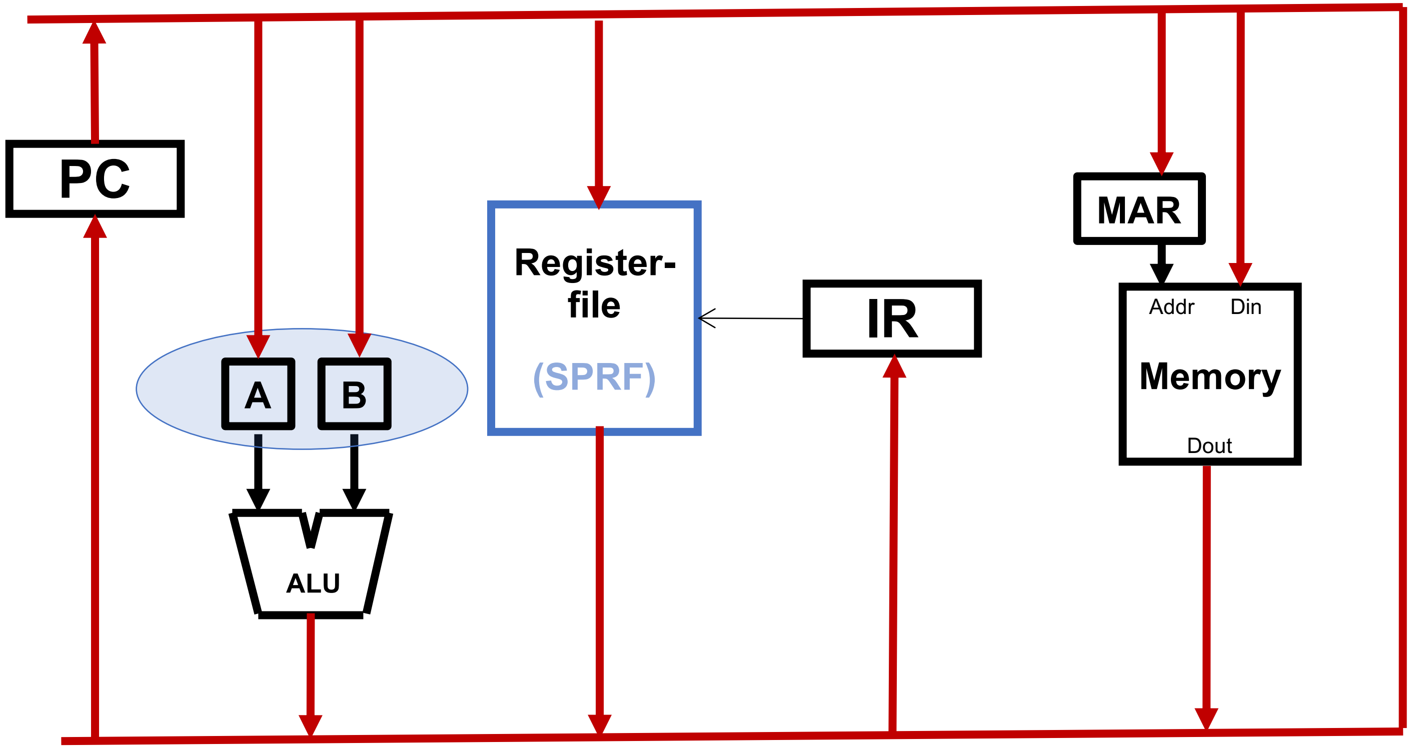
Second: IR → Reg File → ALU → Reg File

Single Bus Design

(…or, "let's spice things up, pt 1")



Single Bus Design w/o DPRF
(…or, "let's spice things up, pt 2")

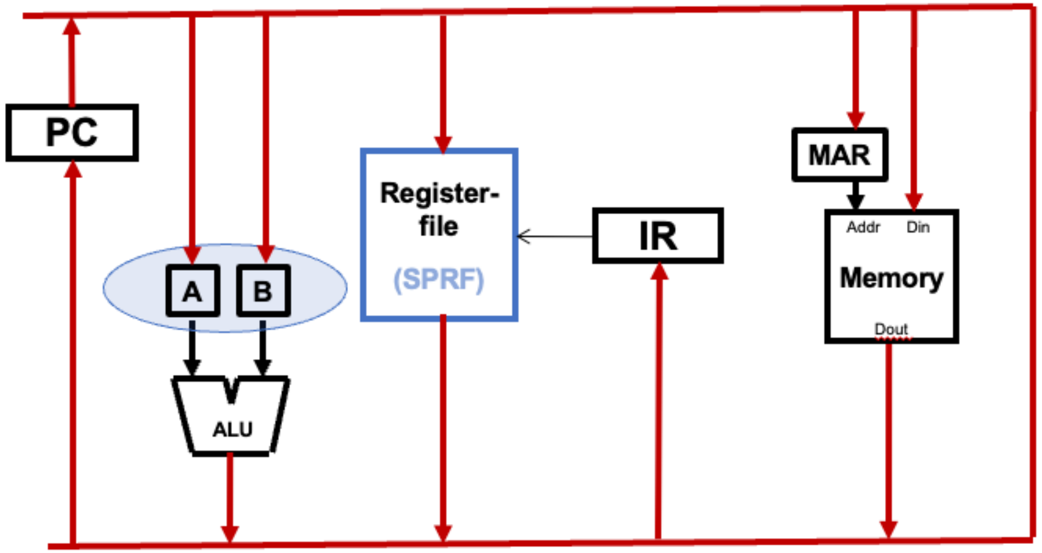


This is pretty close to the LC-2200 data path we'll be using

SPRF: Single-Ported Register File

Question…

Using a single-ported register file, what is the minimum number of cycles it will take to prepare the ALU for a two-operand calculation?

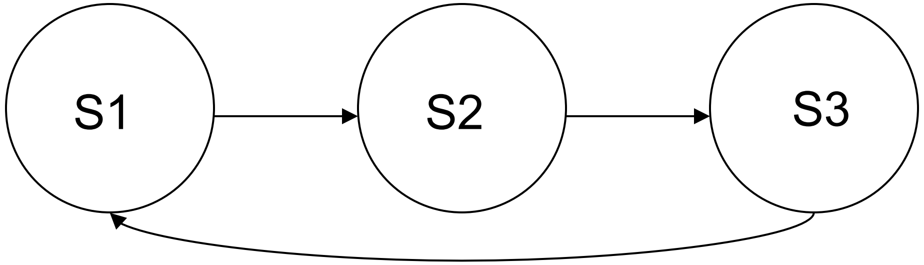


Rank **Responses**

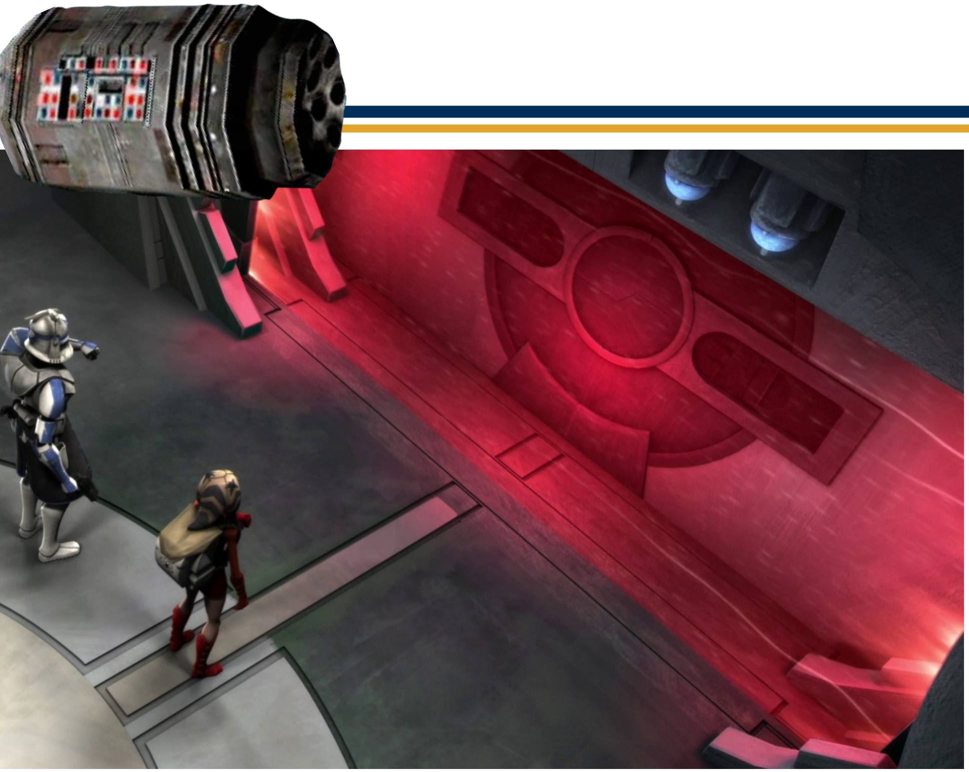
Topics

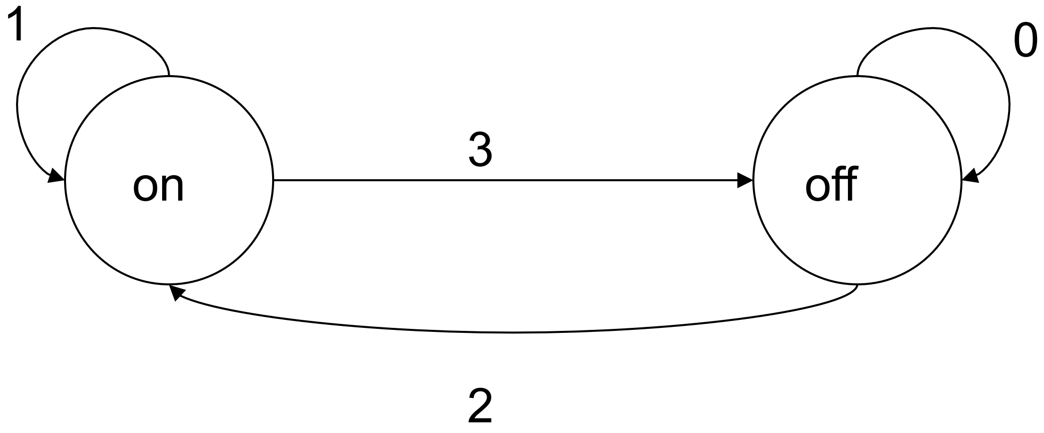
- ~~Logic design review~~
- ~~Data paths~~
- Finite State Machines

Simple FSM Example



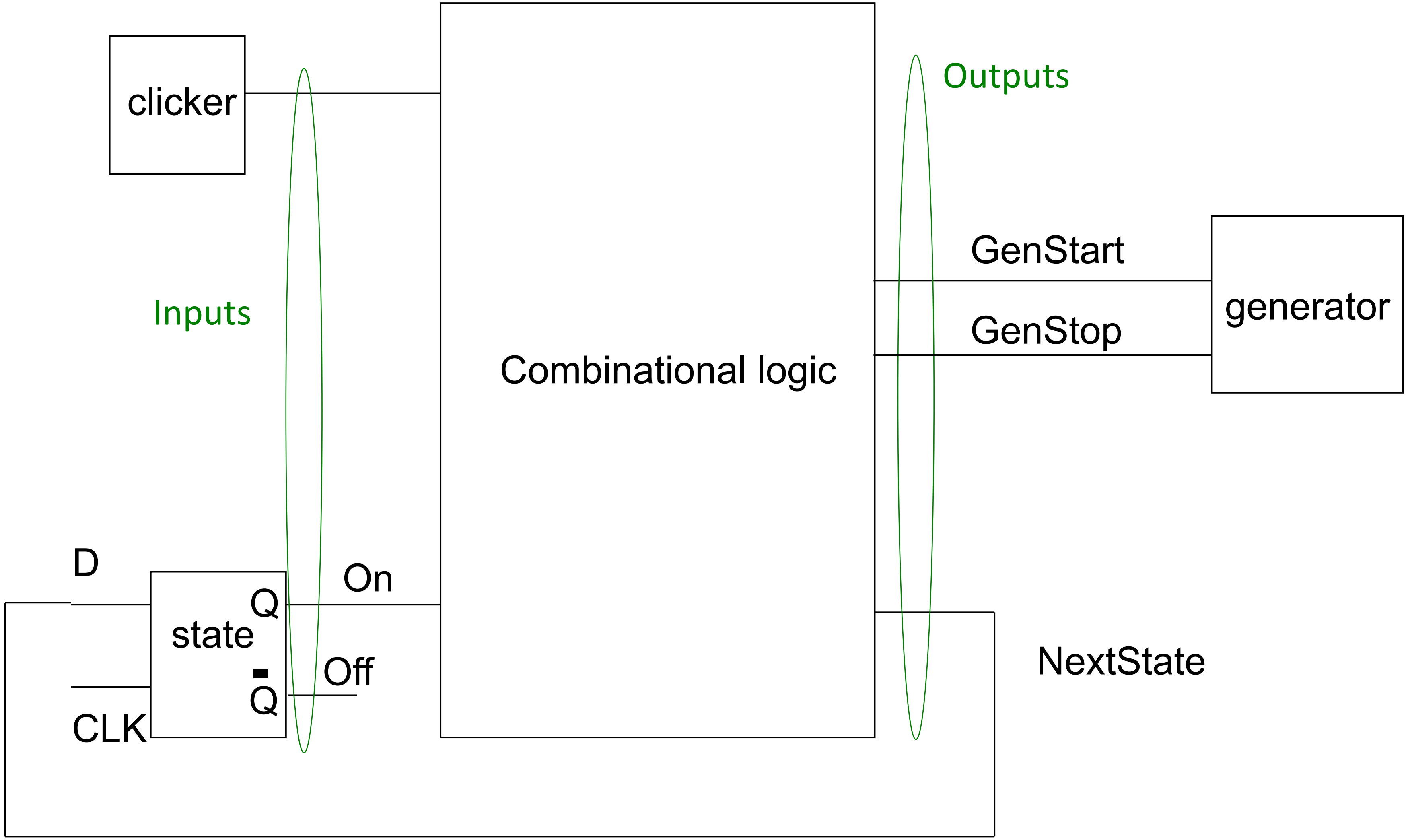
Ray Shield





Transition No	Current State	Clicker	GenStart	GenStop	Next State
0	0	0	0	0	0
1	1	0	0	0	1
2	0	1	1	0	1
3	1	1	0	1	0

Ray Shield Controller



Combinational Logic

- GenStart = CurrentState’&Clicker
- GenStop = CurrentState&Clicker
- NextState = (CurrentState&Clicker’) | (CurrentState’&Clicker)

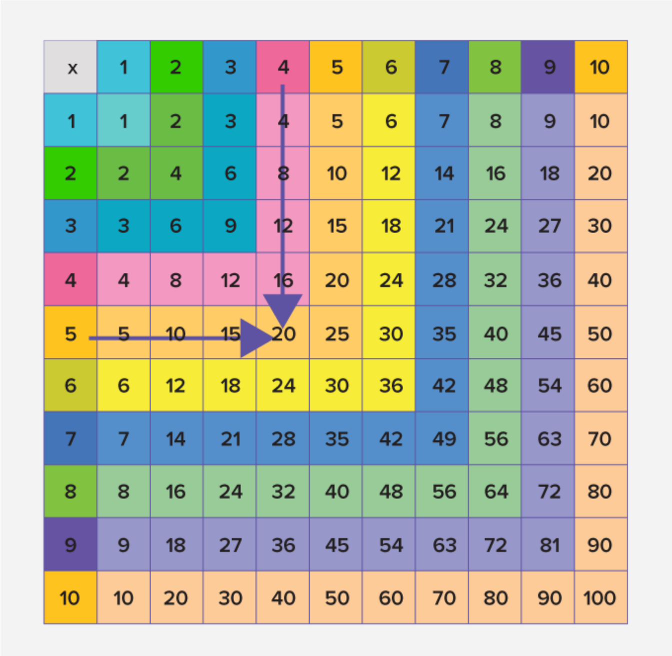
Transition No	Current State	Clicker	GenStart	GenStop	Next State
0	0	0	0	0	0
1	1	0	0	0	1
2	0	1	1	0	1
3	1	1	0	1	0

Can We Replace Combinational Logic with a ROM?

- Since we can describe combinational logic as boolean expressions, what if we could replace a combinational circuit with a hardware truth table?

- Think of a properly programmed ROM as a literal truth table describing an FSM
  - The **address** represents the input bits (including state)
  - The **contents** of the ROM produce the output bits (including the next state)

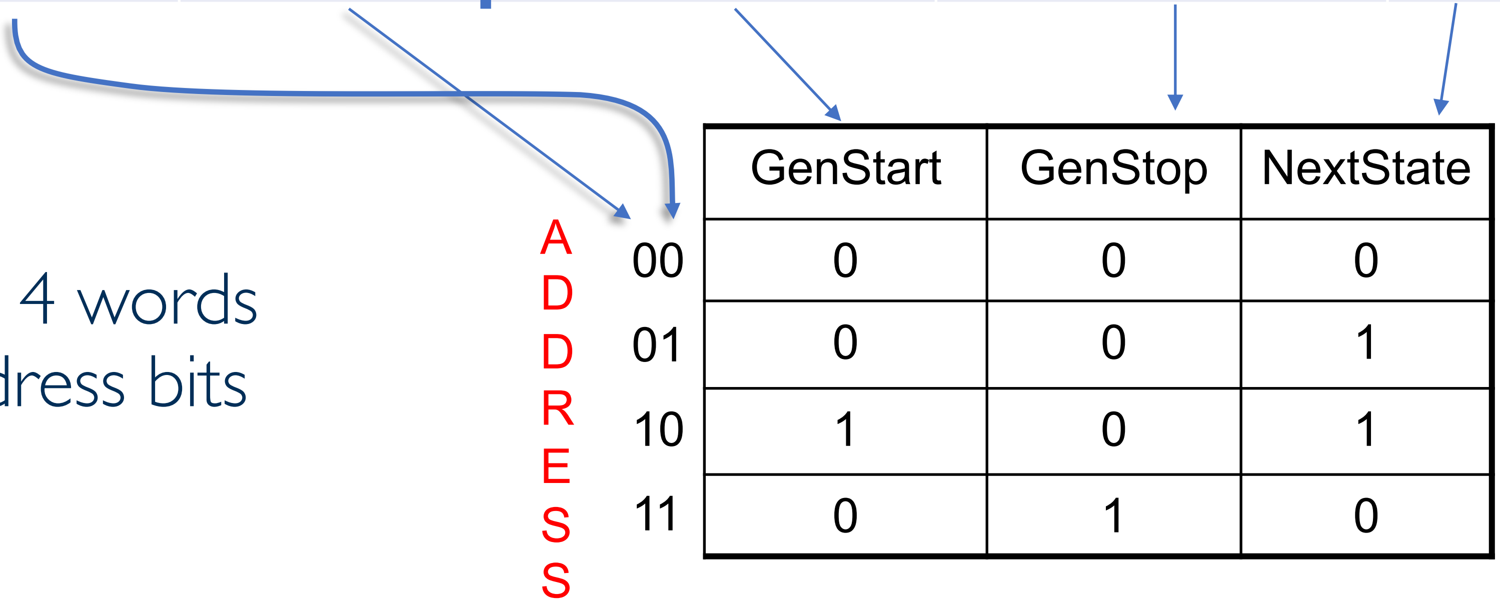
- Have you seen this somewhere before?



x	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

From FSM to ROM

Transition No	Current State	Clicker	GenStart	GenStop	Next State
0	0	0	0	0	0
1	1	0	0	0	1
2	0	1	1	0	1
3	1	1	0	1	0

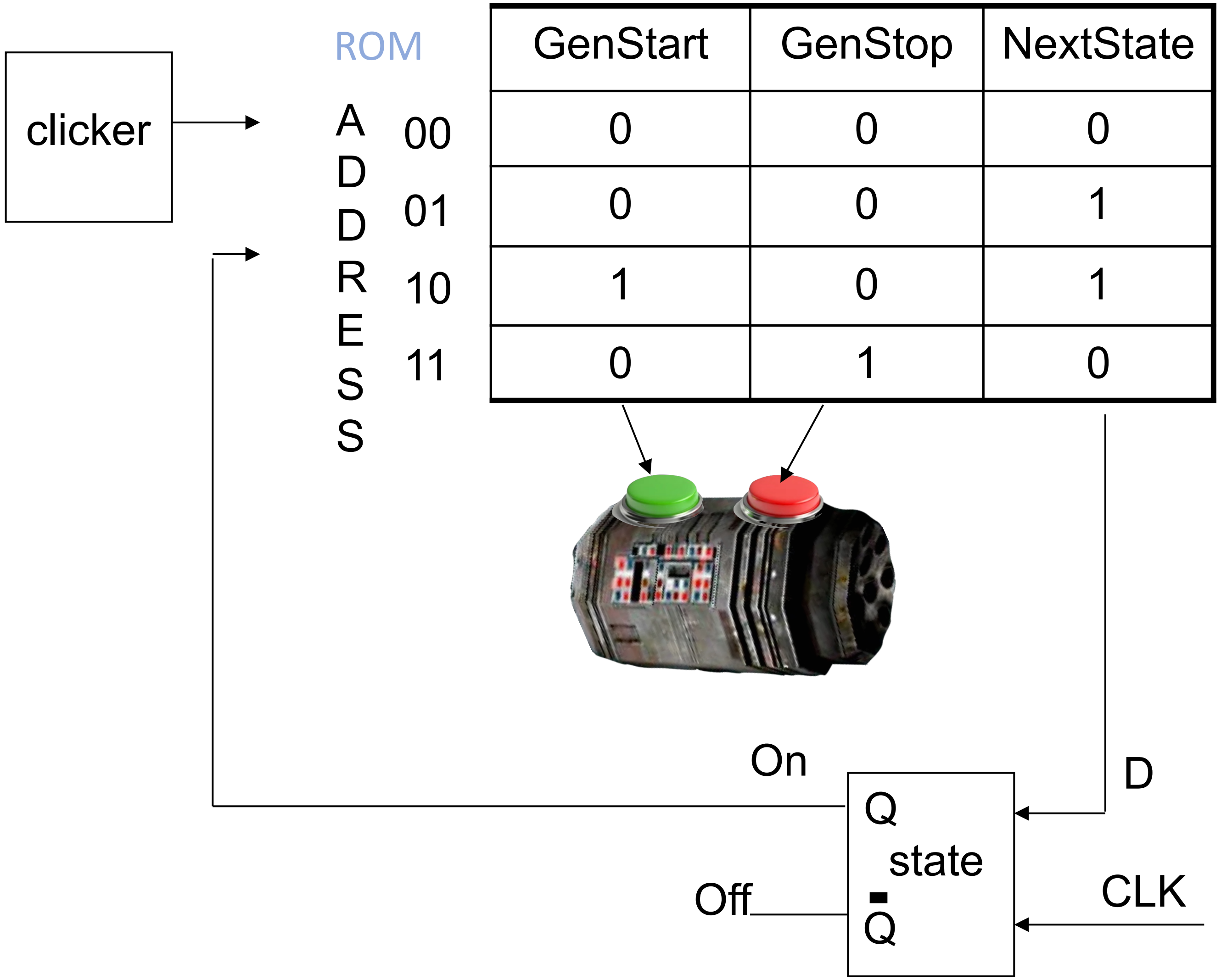


	GenStart	GenStop	NextState
00	0	0	0
01	0	0	1
10	1	0	1
11	0	1	0

ROM: 3-bit word x 4 words

→ 3 data bits, 2 address bits

Replacing Discrete Logic with a ROM



ADDRESS	GenStart	GenStop	NextState
00	0	0	0
01	0	0	1
10	1	0	1
11	0	1	0

How large a ROM?

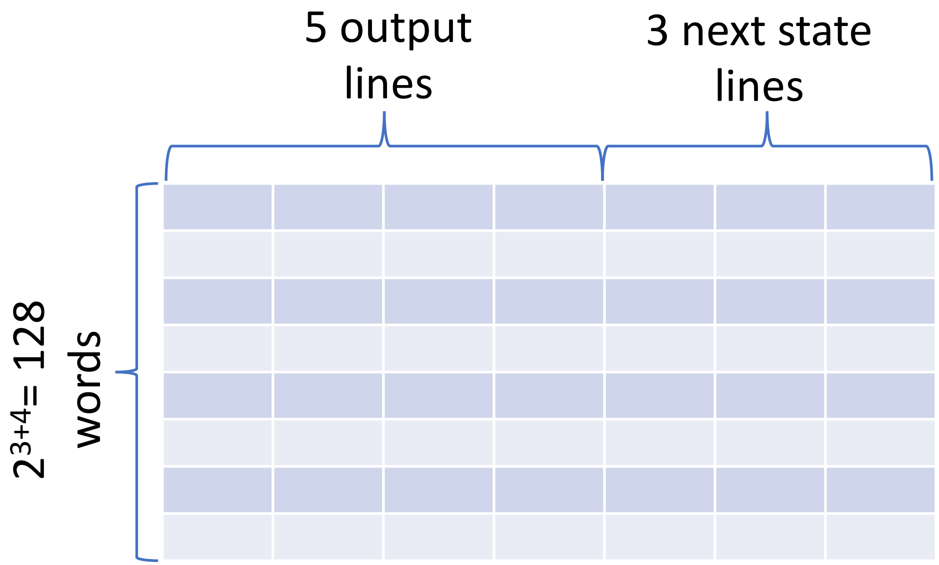
If you have a truth table with a 4-bit input, 8 states (i.e., 3 state bits), and 5 outputs, what size ROM should you use to encode it?

A. 2^7 words of 8 bits

B. 2^4 words of 5 bits

C. 2^8 words of 7 bits

D. 2^4 words of 8 bits



Checkpoint

- Basics of logic design
  - Combinational
  - Sequential
- Elements of the datapath
  - Registers & register file
  - ALU
  - Mux
  - Decoders
  - Clock & clock width
  - Finite state machine (combinational and truth table)