Georgia Tech Computer Science

CS2200
Systems and Networks
Spring 2022

Lecture 5: Datapath

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

Lecture slides adapted from Bill Leahy and Charles Lively of Georgia Tech

Announcements

- Lab 1 released
- Homework 1 will be released tomorrow before lab
- Starting Chapter 3 today

Expect an announcement on labs

- Labs are a required component of the course
- Virtual section is only an accommodation for those unable to attend in person – must keep sections balanced

Topics

- Logic design review
- Data paths
- Finite State Machines

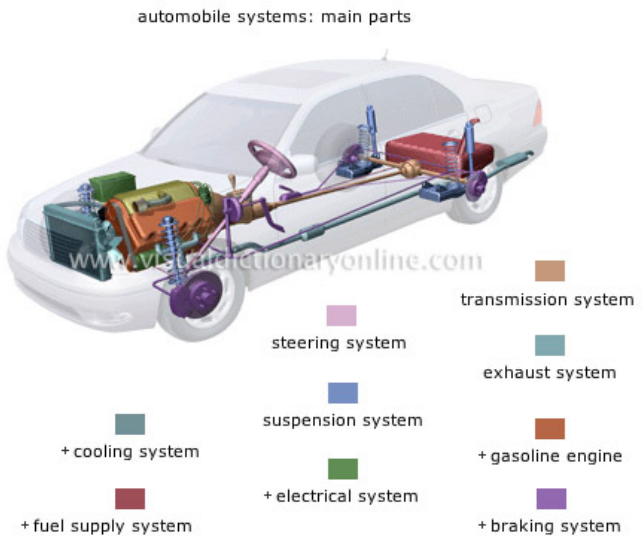
Architecture vs. Organization

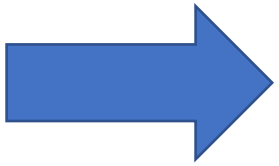
- Architecture – the abstraction involving programmer-visible details of a computer system, such as
  - Instruction set
  - Memory layout

- 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	Scien-tific perform-ance (kIPS)	Commer-cial 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 compatibility

What is involved in Processor Implementation?

- Organization of the electrical 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 in motherboards.

Super Computers	Servers	Desktops & PCs	Embedded
High performance primary objective	Intermediate performance and cost	Low cost primary objective	Small size, low cost, and low power consumption primary objectives

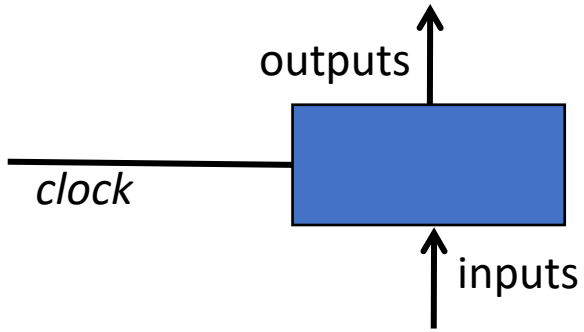
Circuits

- 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



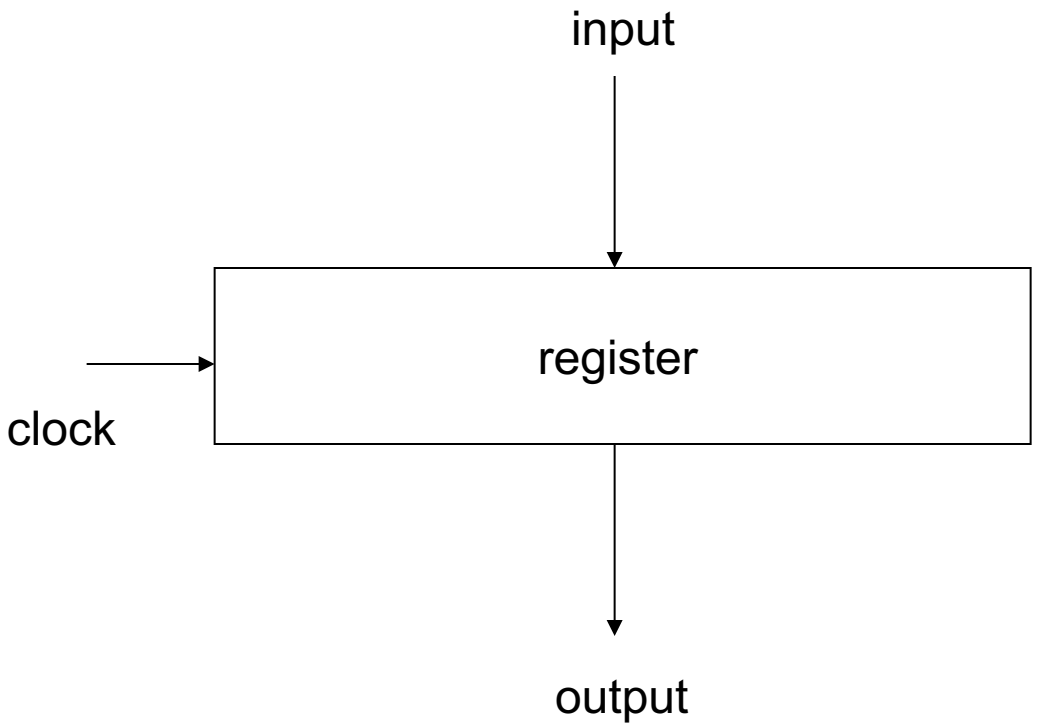
Level Triggering

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

Edge Triggering

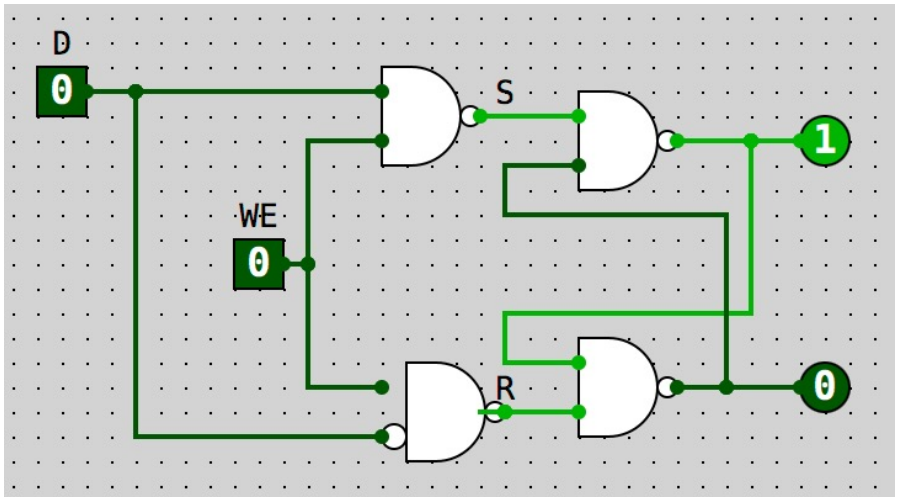
- Outputs change based on inputs only when clock transitions
- Positive edge-triggered logic when leading edge cause triggering
- Negative edge-triggered when trailing edge causes triggering

Registers



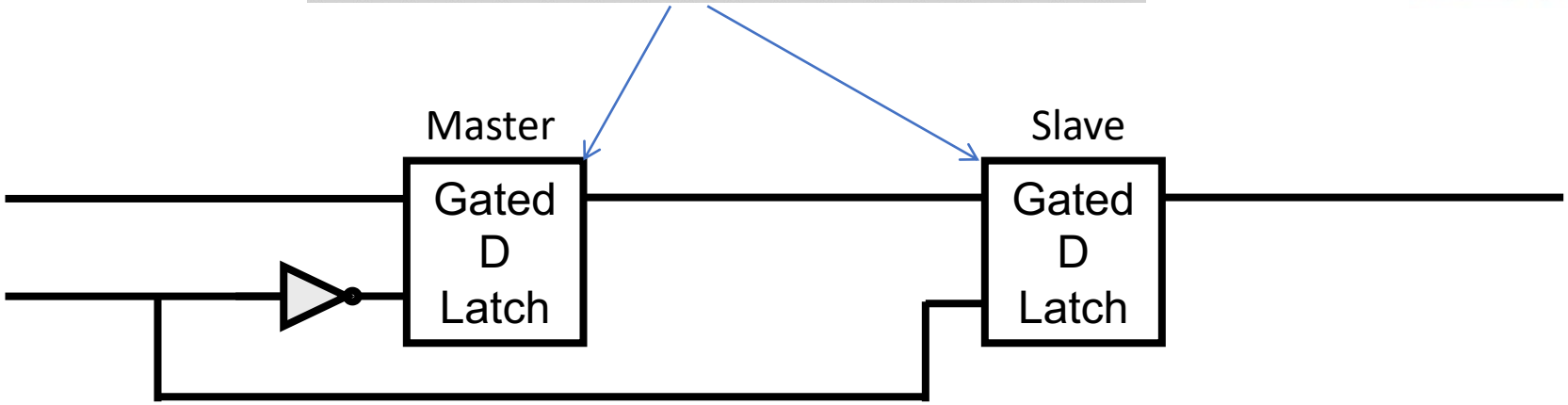
Master-Slave Flip Flop

Remember our Gated D Latch?

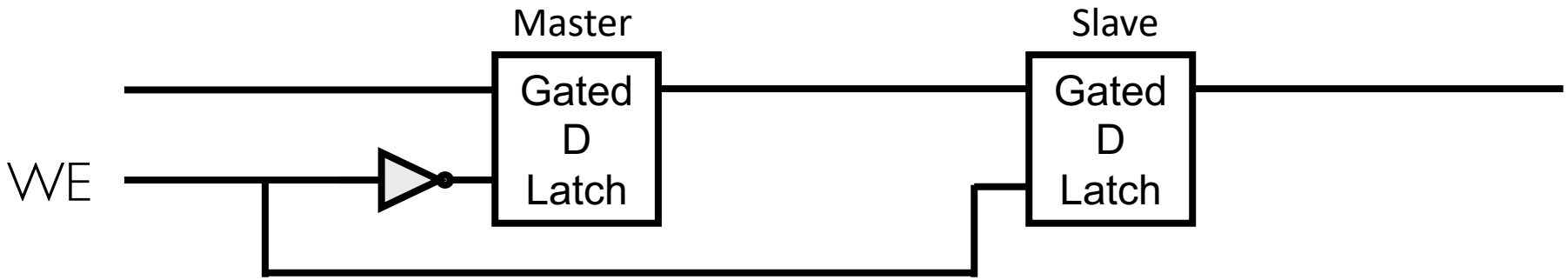


Truth Table

EN	D	Q	Q_{+1}	STATE
1 1	0 0	0 1	0 0	RESET
1 1	1 1	0 1	1 1	SET
0 0	0 0	0 1	0 1	NC
0 0	1 1	0 1	0 1	NC

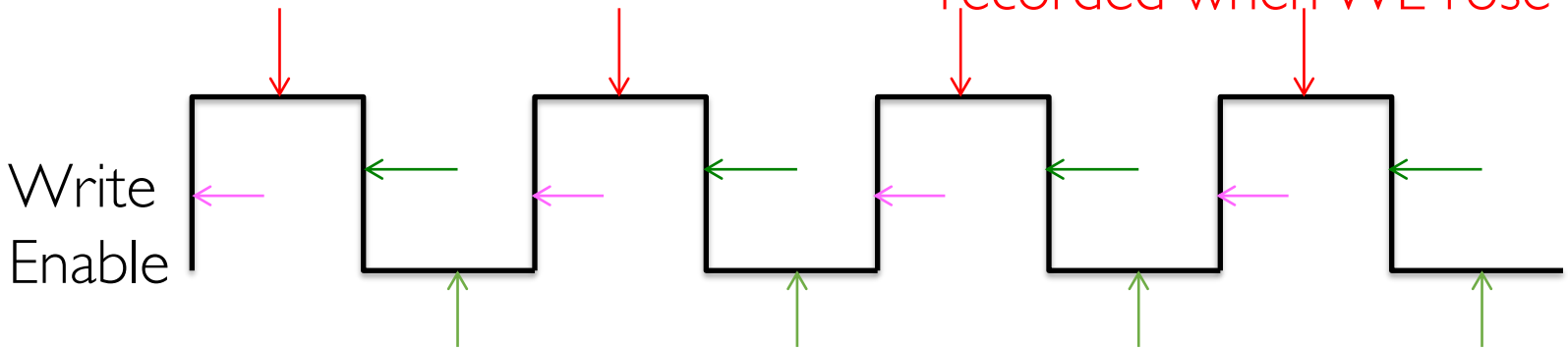


Clock Edges and Master-Slave



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.

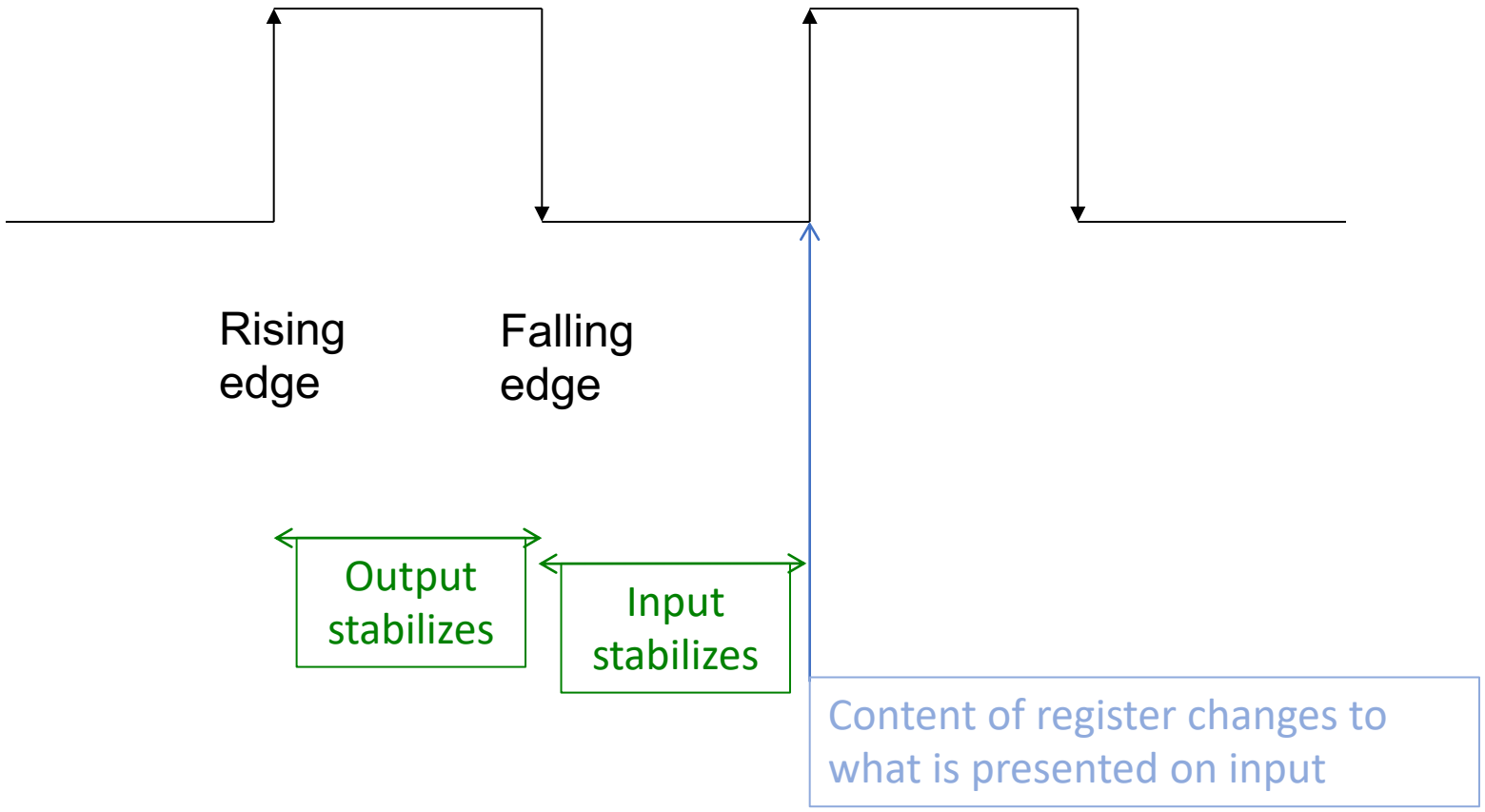


Slave latch records what is presented on its input.

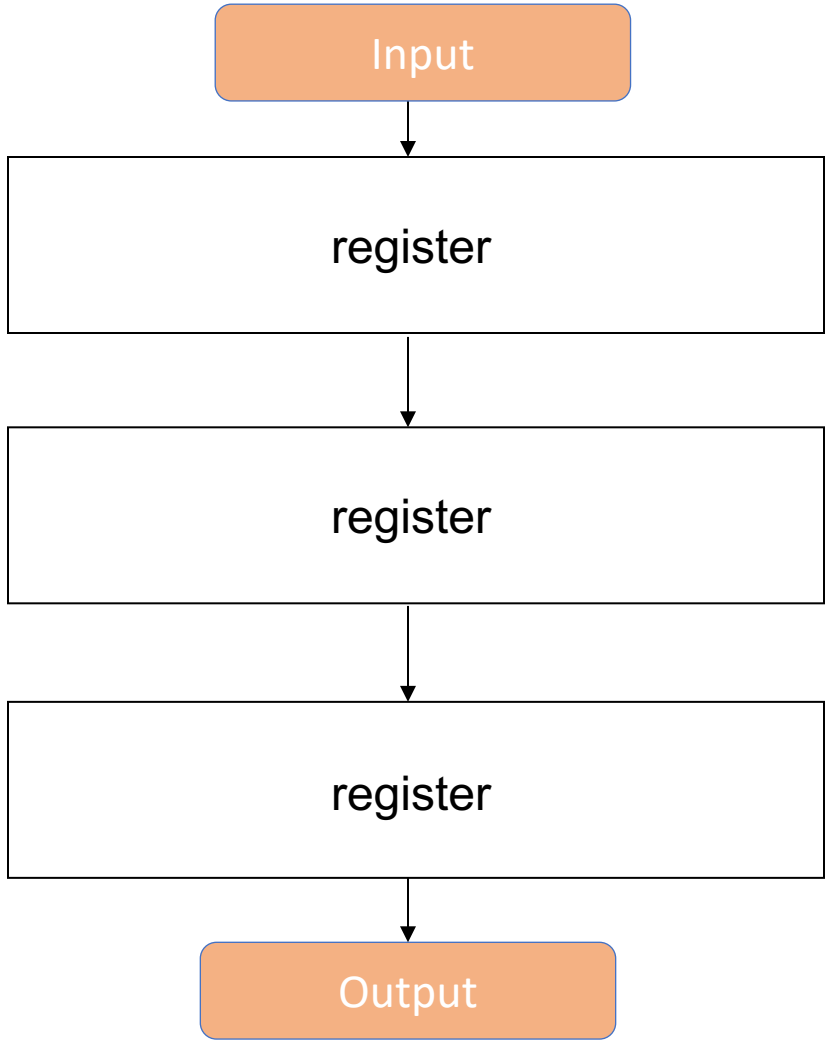
Slave latch outputs what was recorded when WE fell to 0.
Master latch outputs its inputs.

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

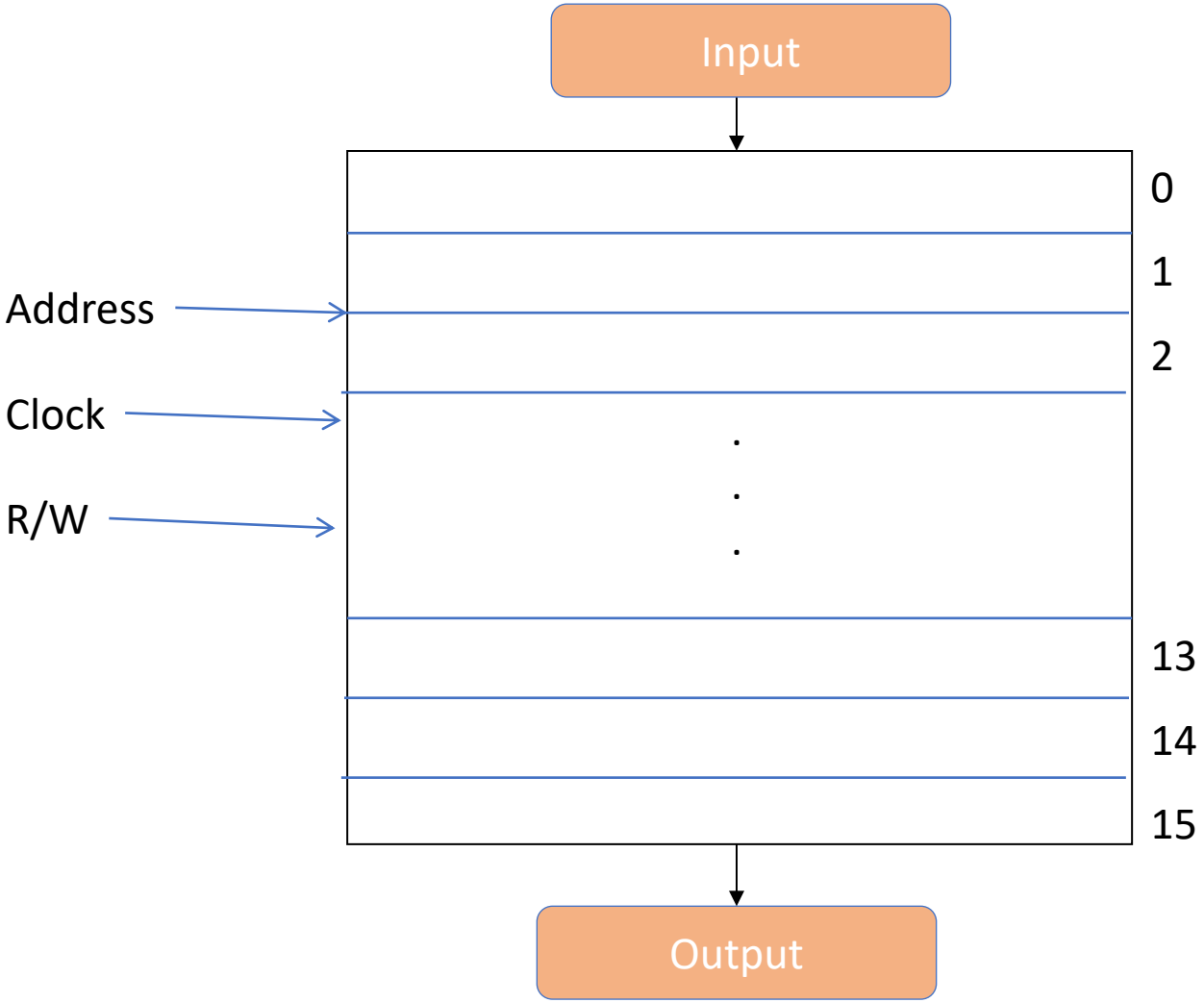
Registers and Clocks



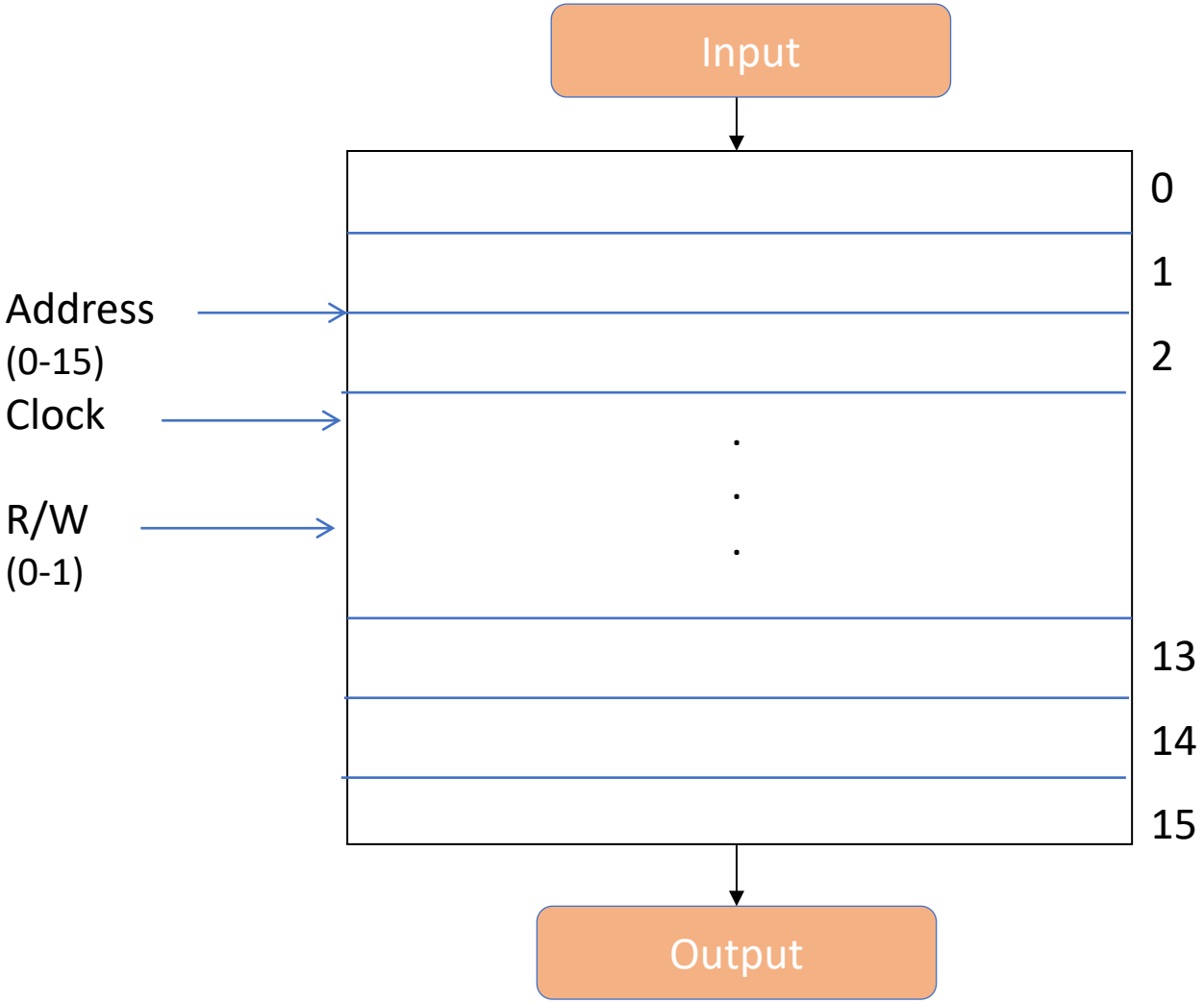
How Many Clock Cycles from Input to Output?



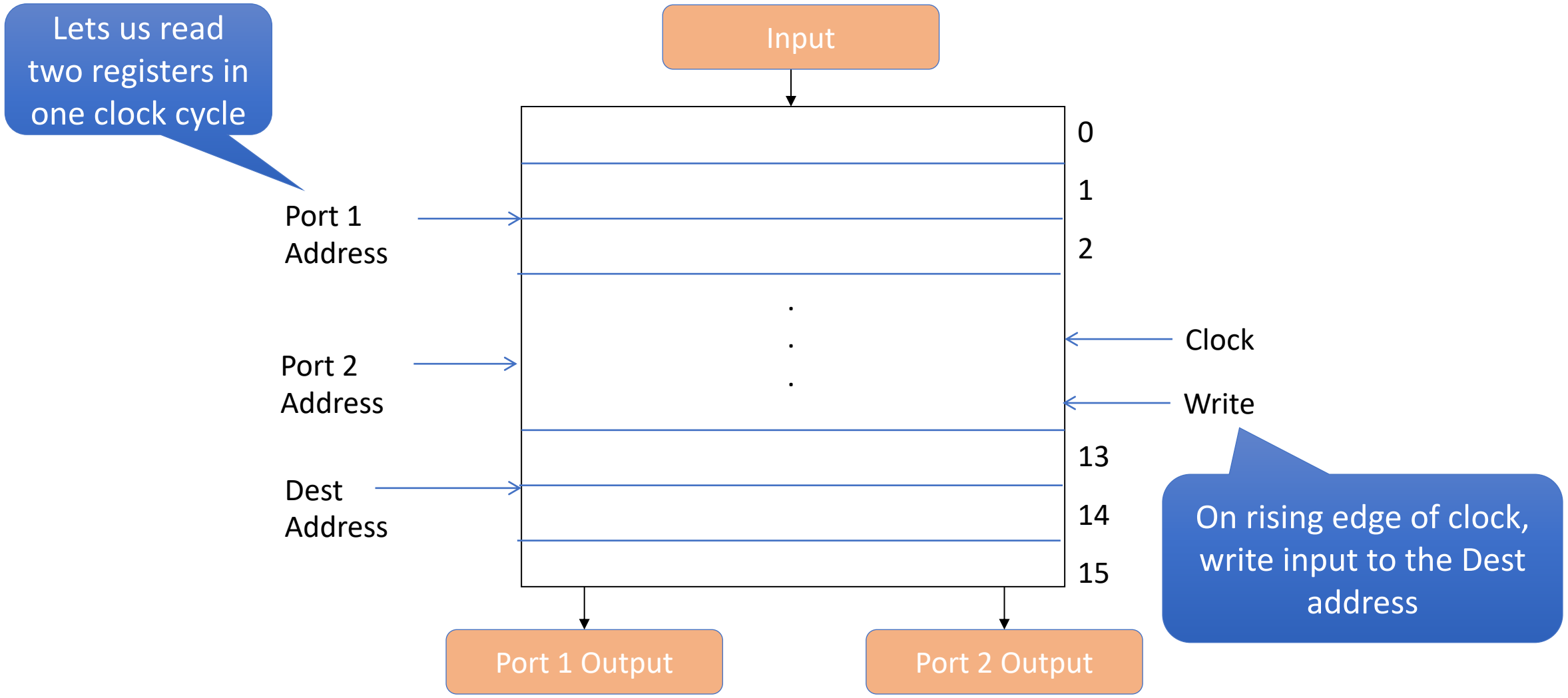
Register File



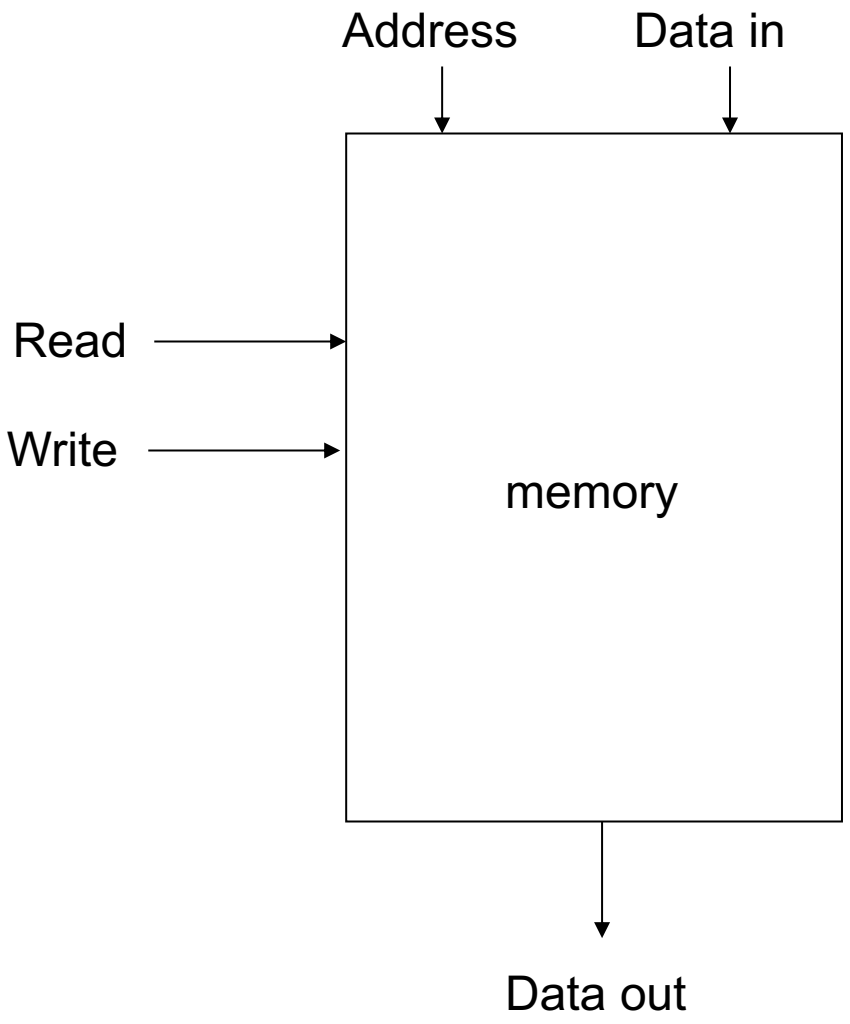
Register File



Dual Ported Register File



Memory (DRAM)

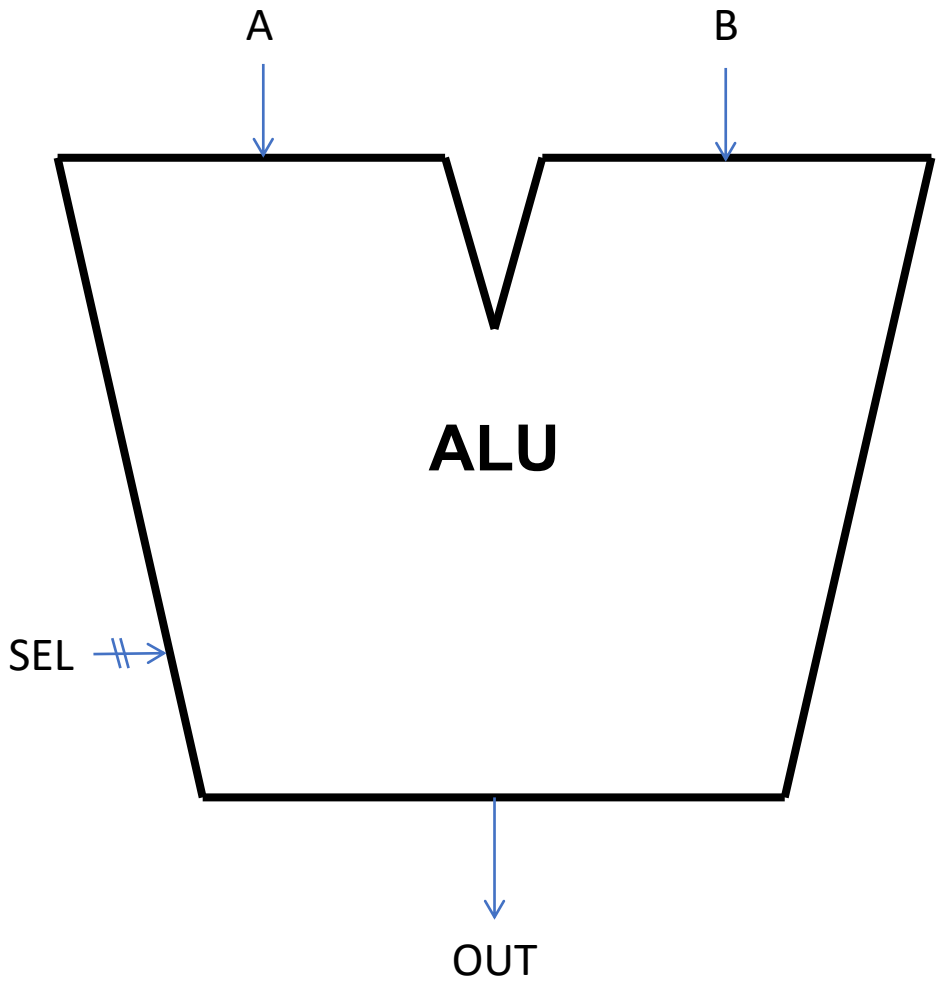


- Works differently from a register
  - Not clocked (for the purposes of cs2200)
  - Level triggered logic

- Each DRAM cell is a single transistor and capacitor → higher density but slower

A Typical ALU

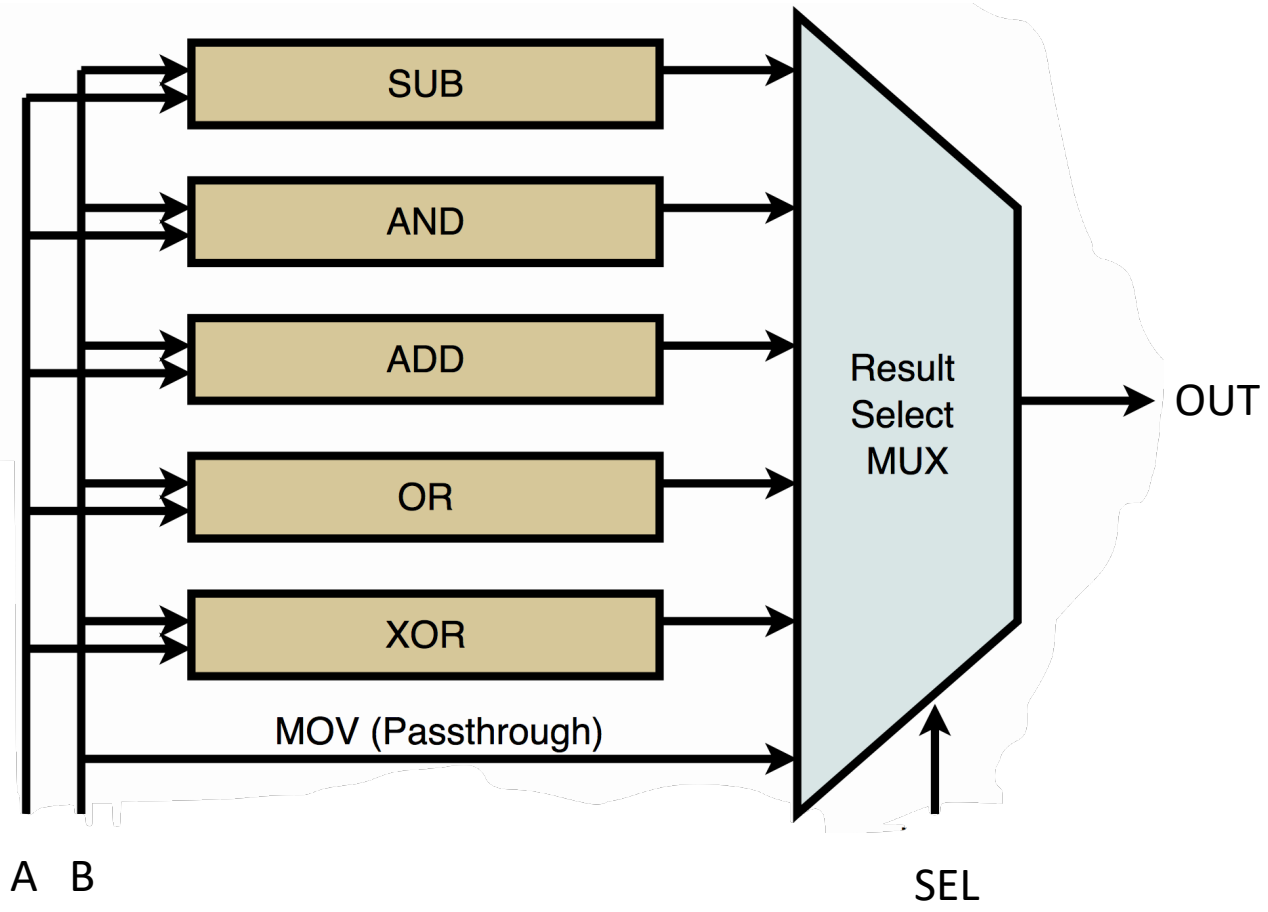
- Combinational circuit
- Two inputs
- One output
- "Function select" selects which functional unit is presented on the output



What's Inside an ALU?

How many lines needed for SEL?

How many bits wide are the data paths?

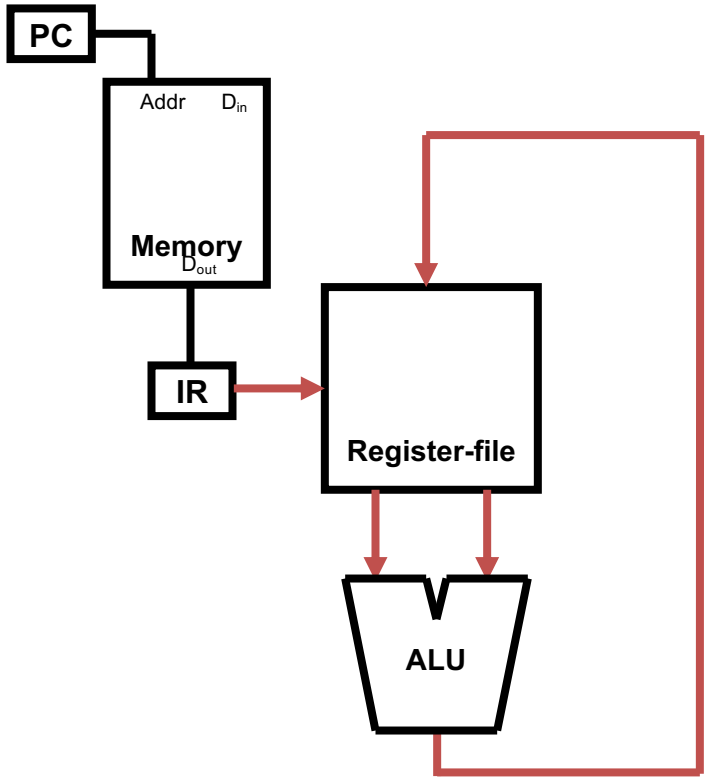


Gisselquist Technology, Inc. https://zipcpu.com/zipcpu/2017/08/11/simple-alu.html

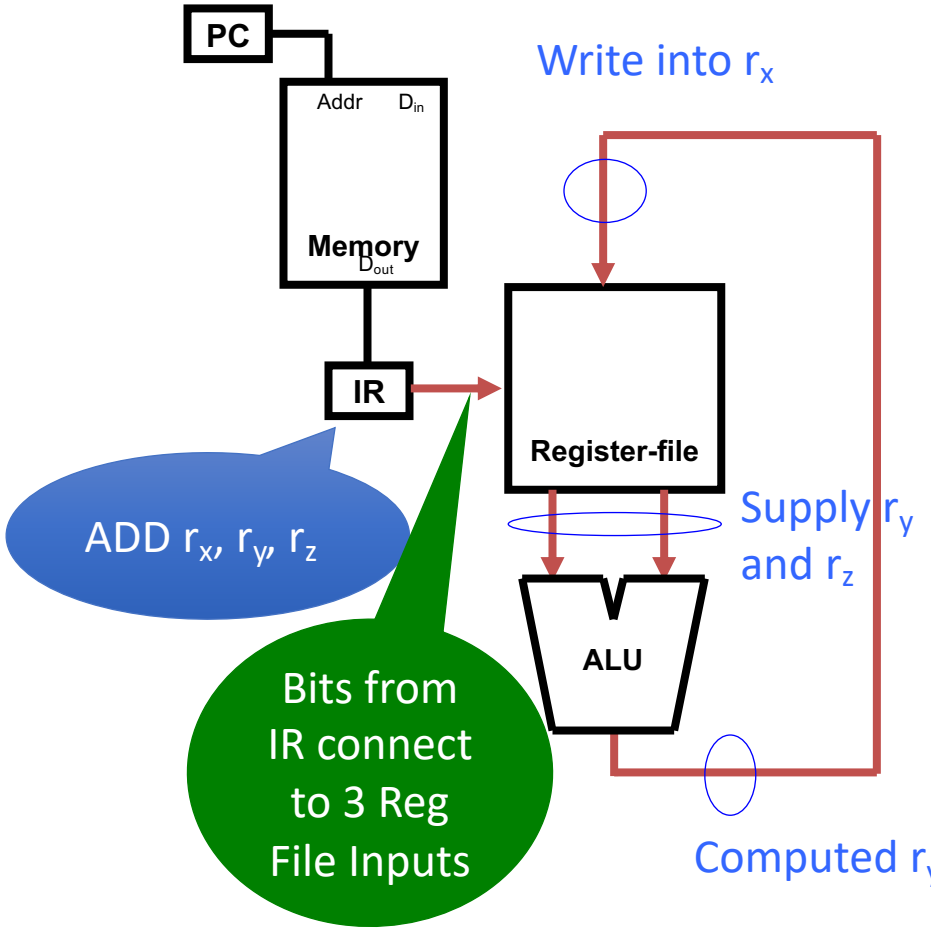
Review Components

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

Connecting the Datapath Elements



Connecting the Datapath Elements



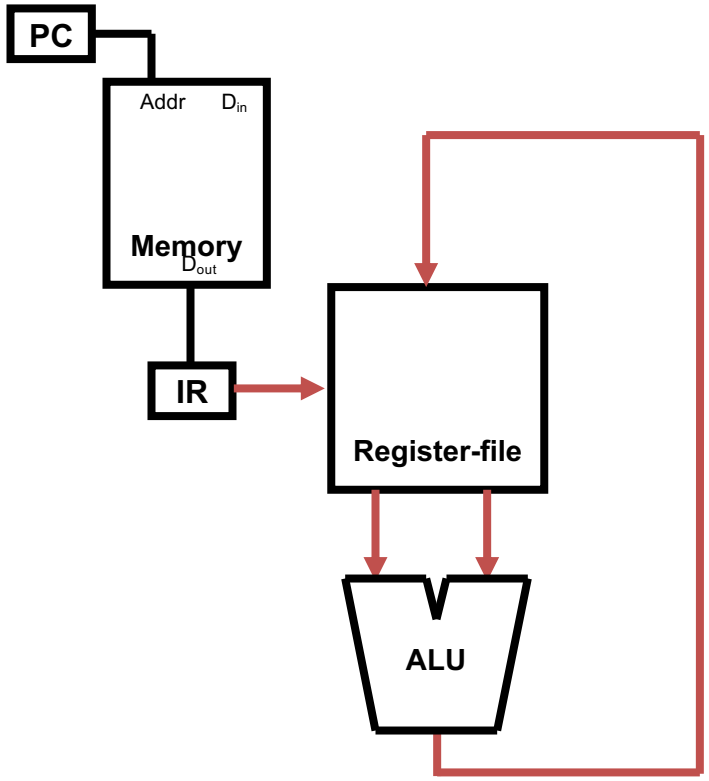
PC ➔ ADD r_x, r_y, r_z

$r_x <= r_y + r_z$

Work done:

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

How Can We Tell?



PC → Mem → IR →

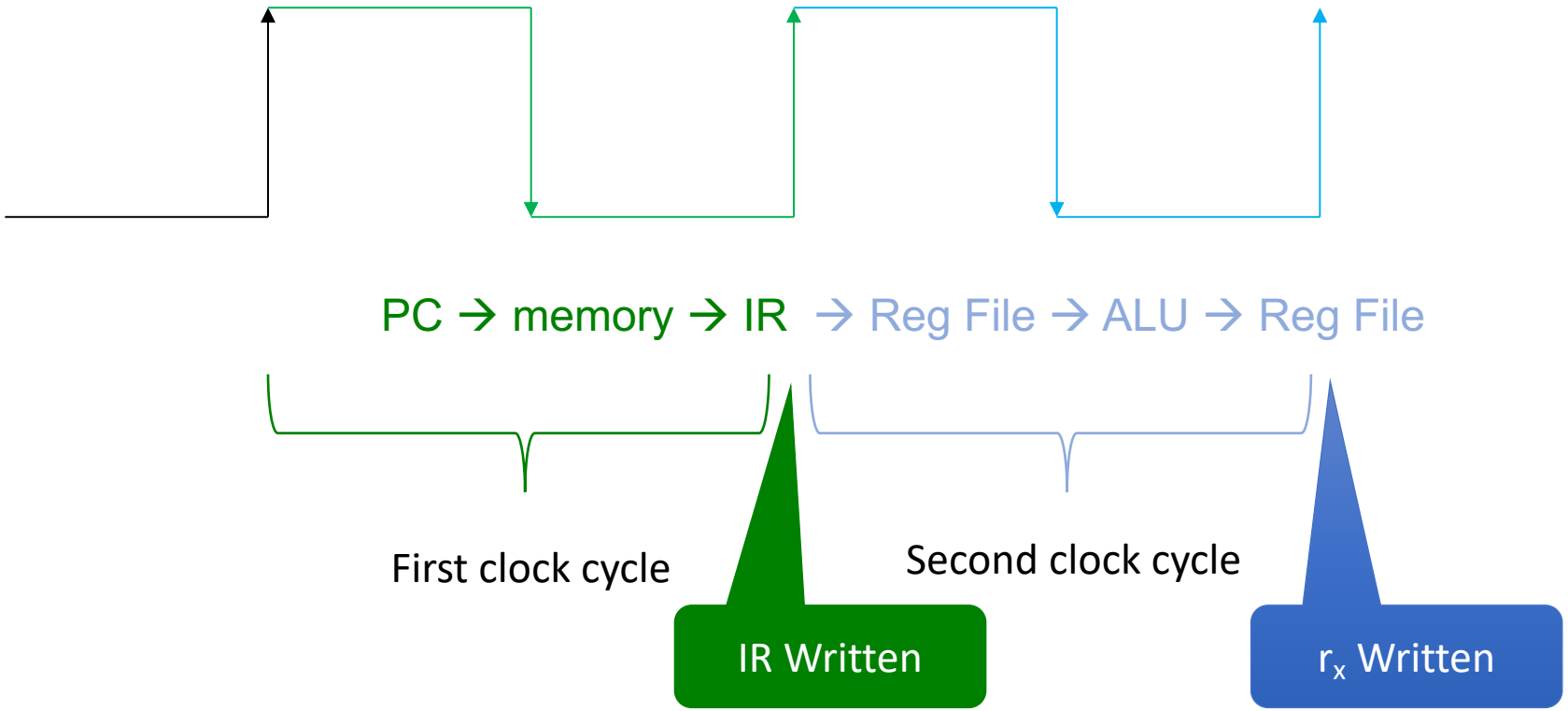
Comb + Level

Rising Edge

Reg File → ALU → Reg File

Comb + Level

Two Clock Cycles

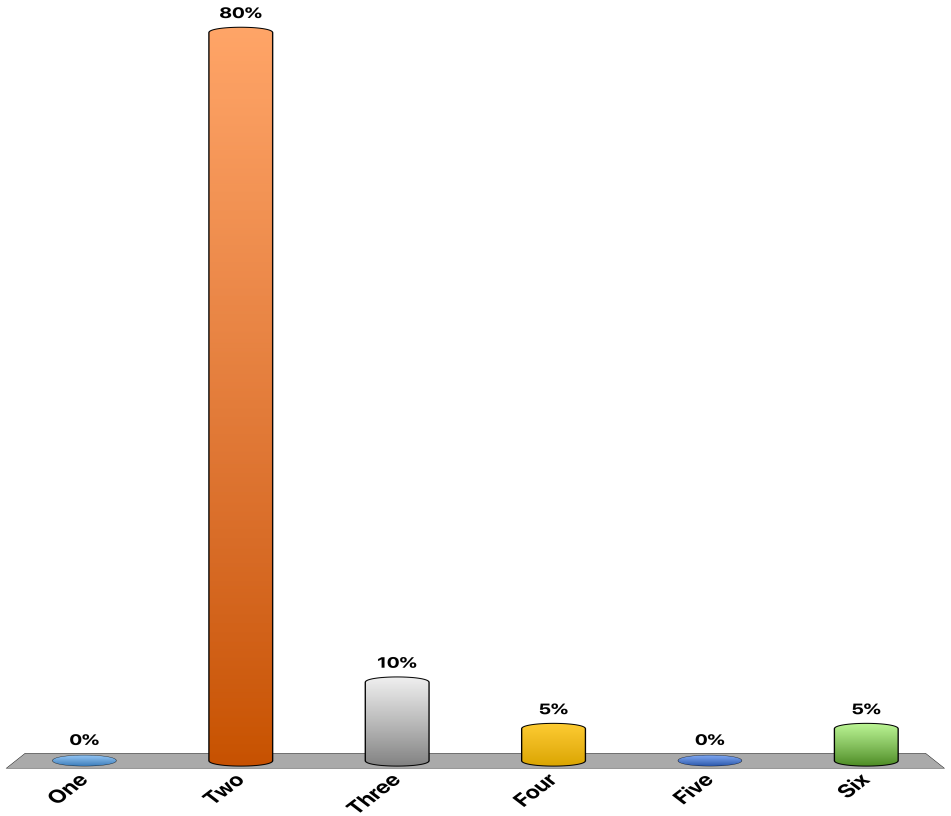


Session ID: dag2200

How Many Clock Cycles?

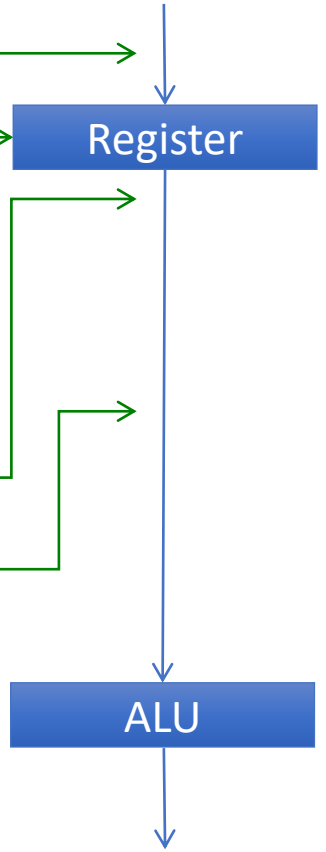
- How many clock cycles did it take to execute
  PC → Mem → IR → Reg File → ALU → Reg File

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

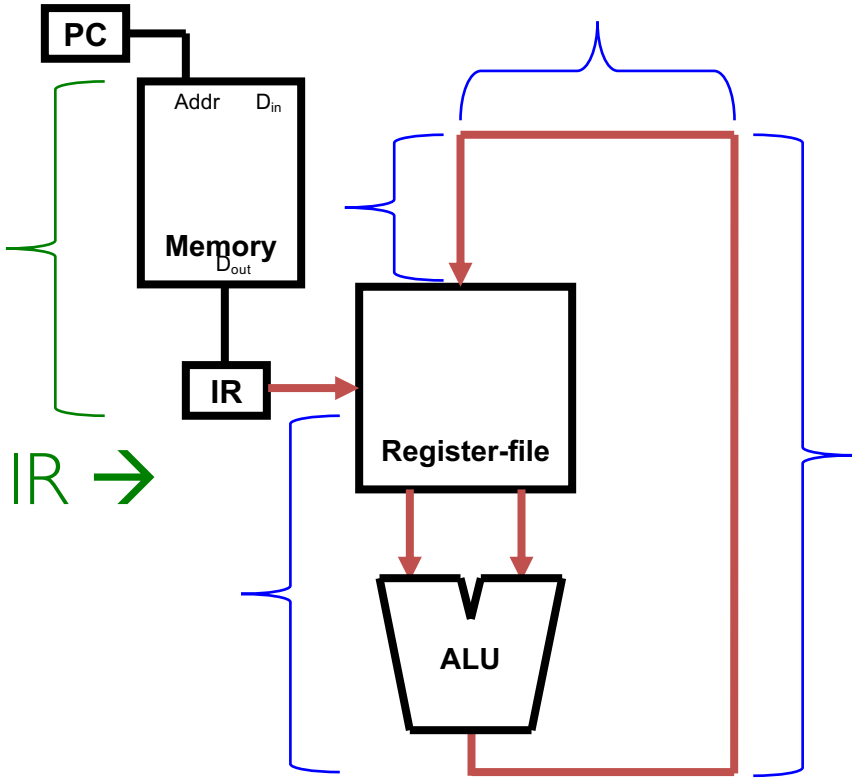


Delays

- 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

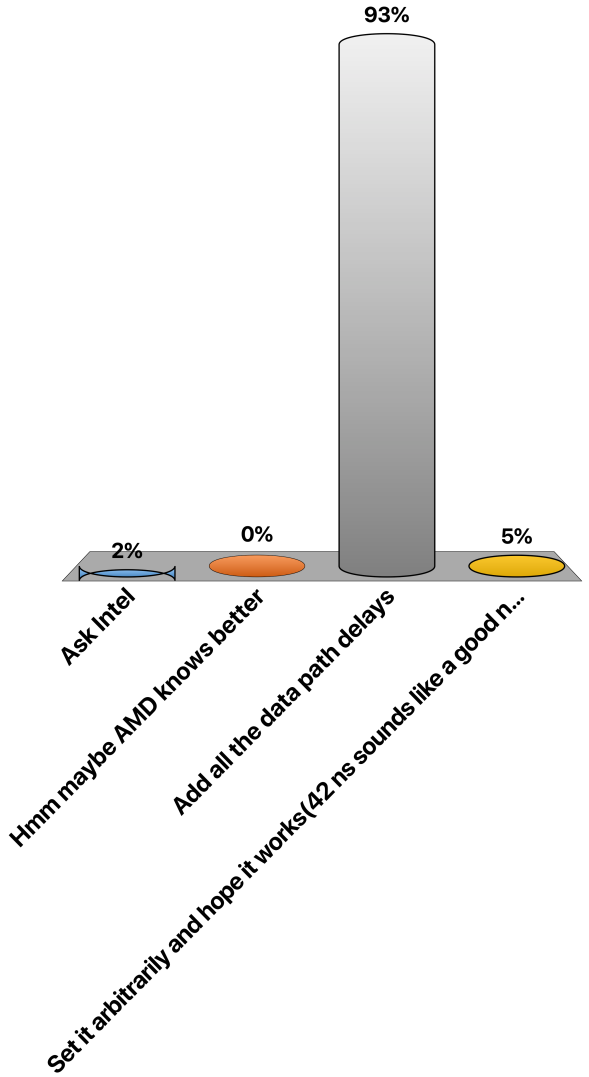
Example Delay Parameters

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

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

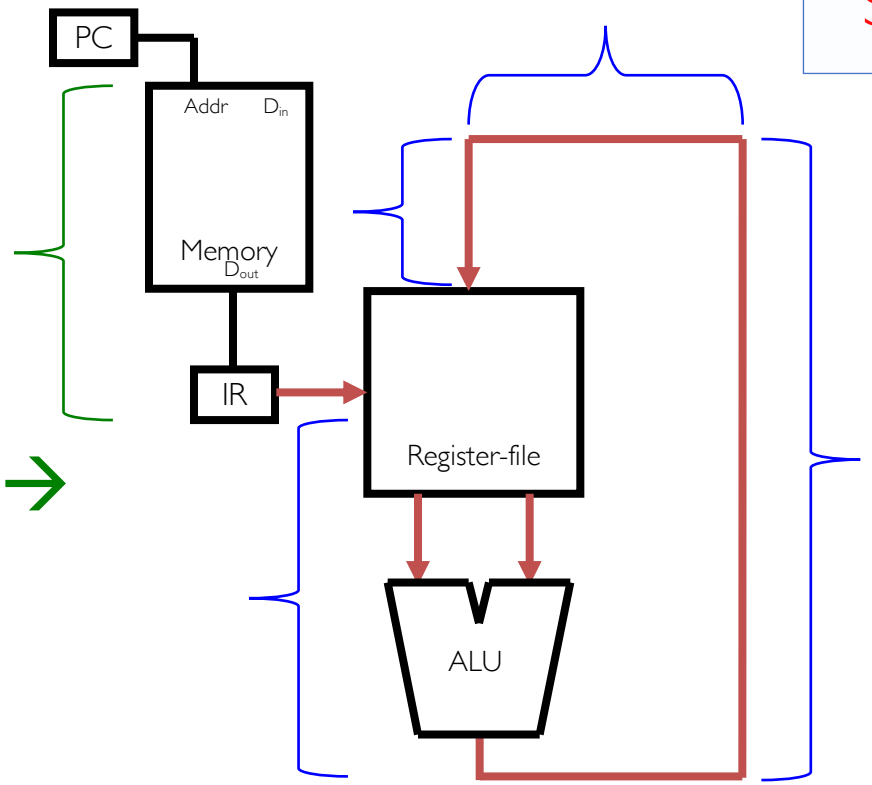
What Should the Duration of Clock Cycle Be?

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)



Calculating Clock Duration

Split circuits at edge-triggered devices

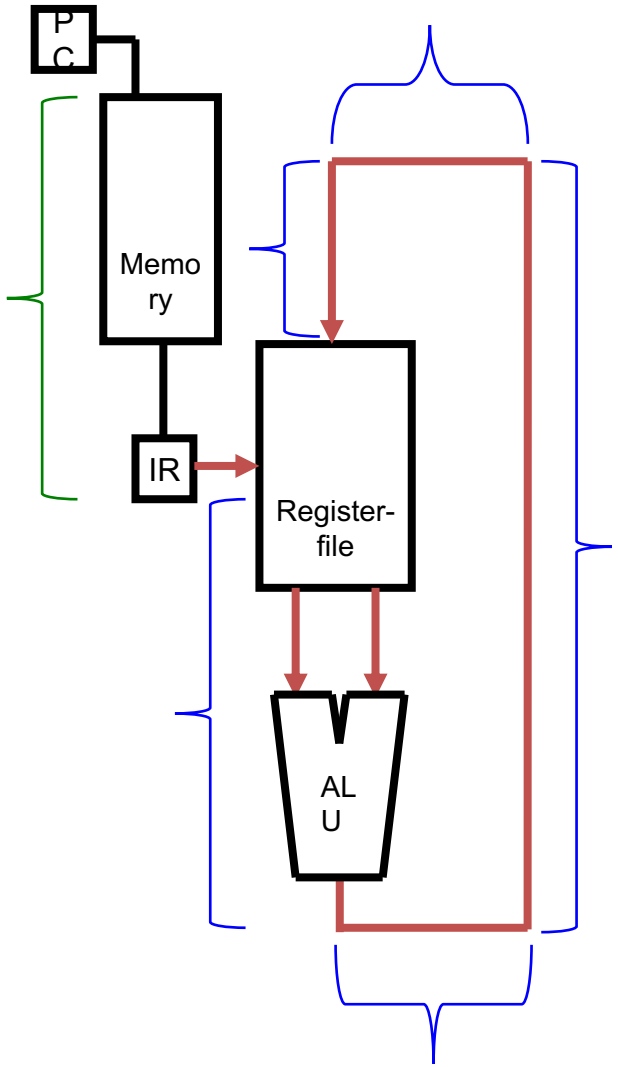


PC → memory → IR →

Reg File → ALU → Reg File

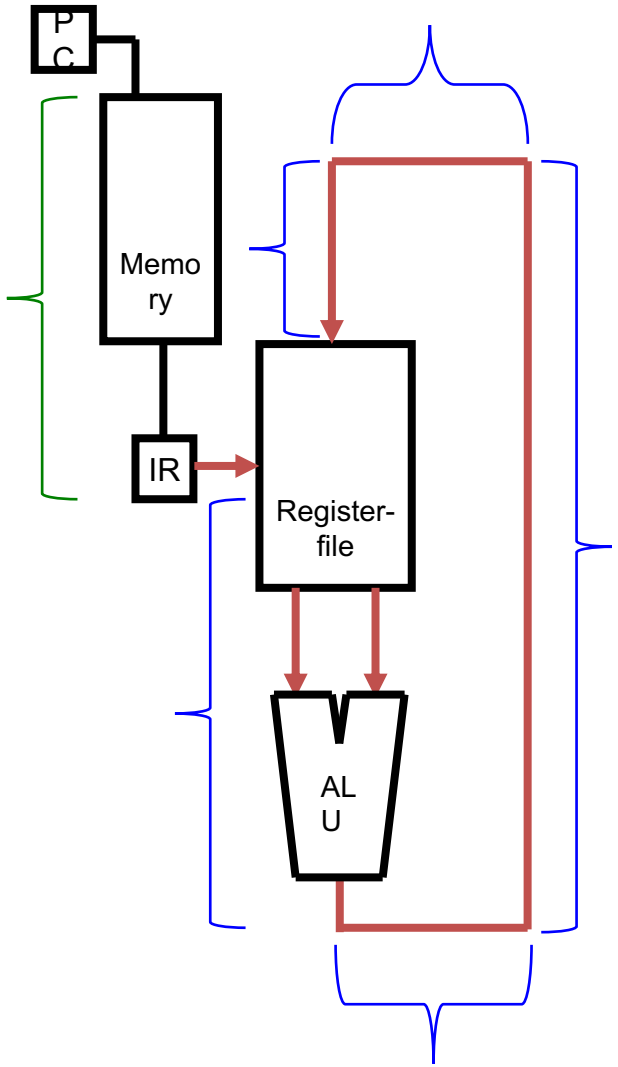
max(green time, blue time)

Calculating Clock Minimums



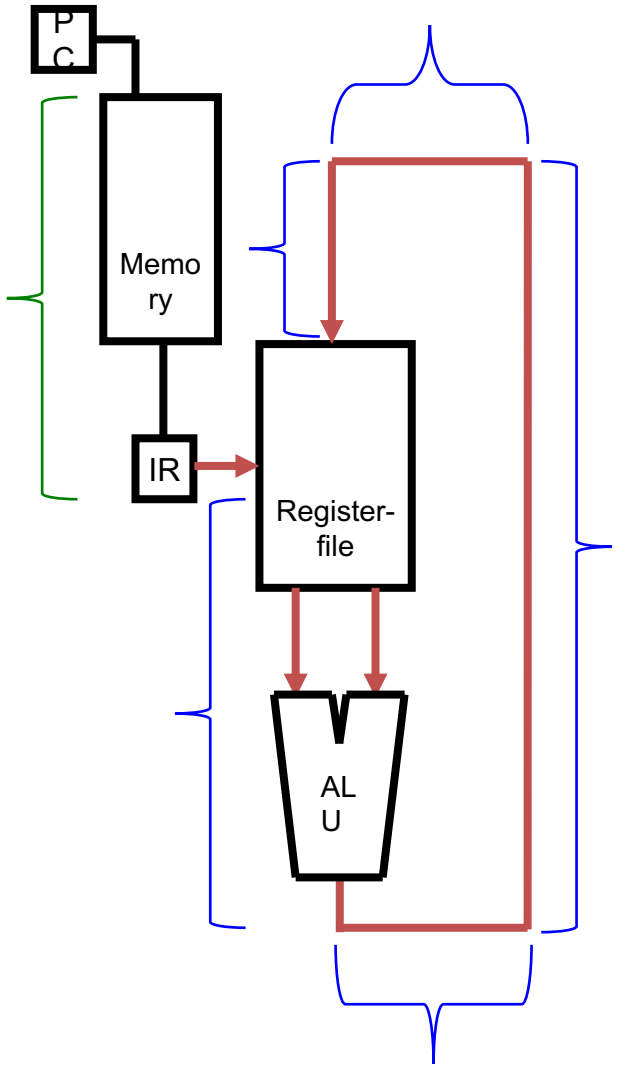
	Delay (ps)	Green bracket	Blue bracket 1	Blue brackets 2-5	
$D_{r\text{-}ouput\text{-}stable}$	20	20			(PC output stable)
$D_{wire\text{-}PC\text{-}Addr}$	250	250			(wire delay from PC to Addr of Memory)
$D_{mem\text{-}read}$	1500	1500			(Memory read)
$D_{wire\text{-}Dout\text{-}IR}$	250	250			(wire delay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

Calculating Clock Minimums



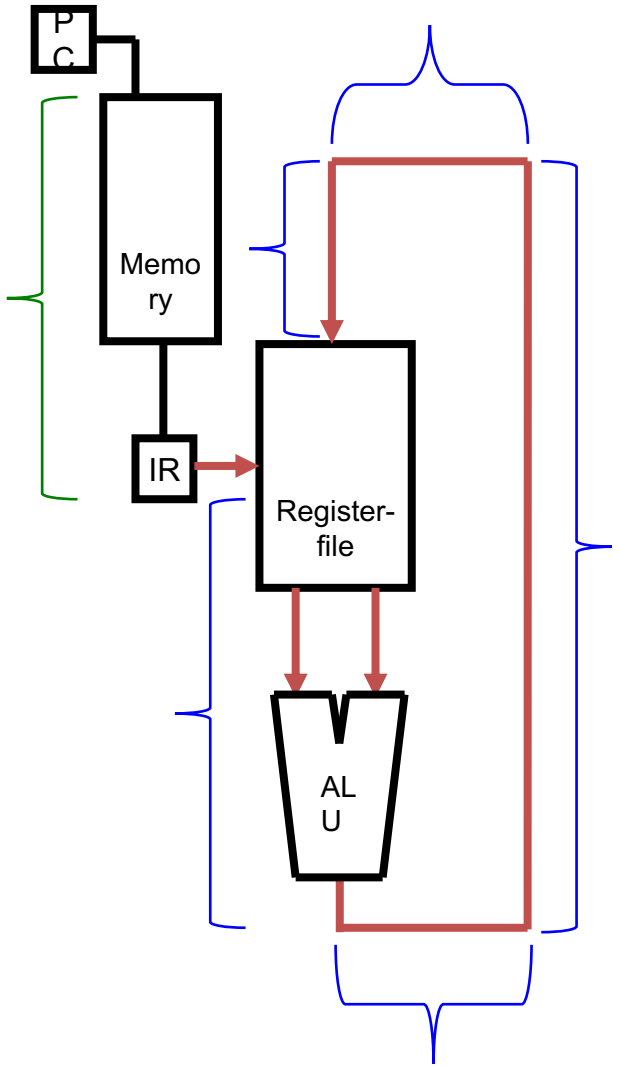
	Delay (ps)	Green bracket	Blue bracket 1	Blue brackets 2-5	
$D_{r\text{-}ouput\text{-}stable}$	20	20			(PC output stable)
$D_{wire\text{-}PC\text{-}Addr}$	250	250			(wire delay from PC to Addr of Memory)
$D_{mem\text{-}read}$	1500	1500			(Memory read)
$D_{wire\text{-}Dout\text{-}IR}$	250	250			(wire delay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

Calculating Clock Minimums



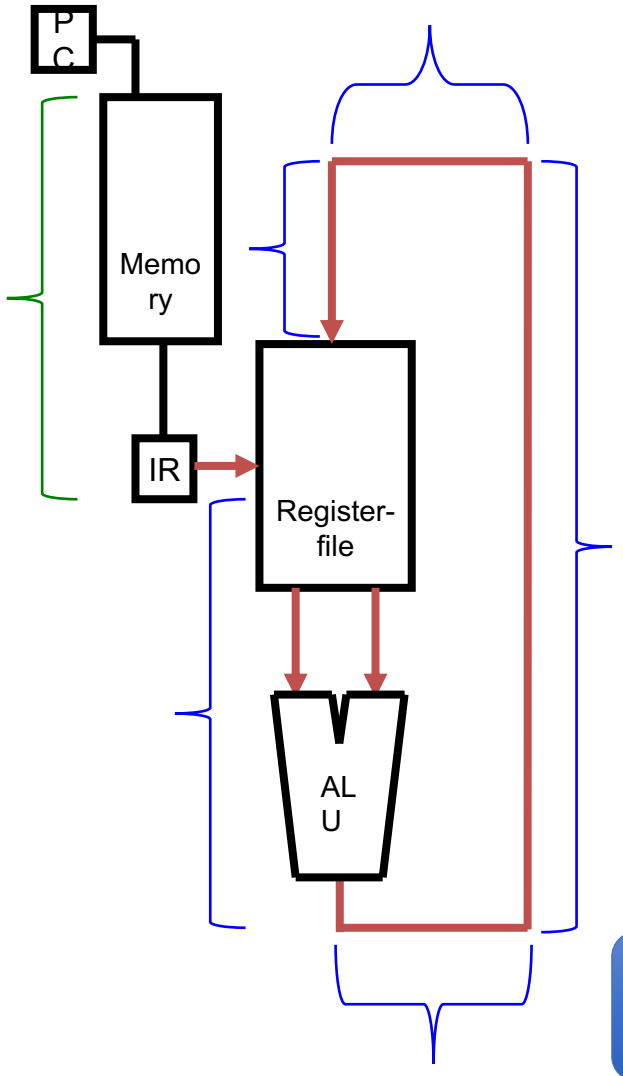
	Delay (ps)	Green bracket	Blue bracket 1	Blue brackets 2-5	
$D_{r\text{-}ouput\text{-}stable}$	20	20			(PC output stable)
$D_{wire\text{-}PC\text{-}Addr}$	250	250			(wire delay from PC to Addr of Memory)
$D_{mem\text{-}read}$	1500	1500			(Memory read)
$D_{wire\text{-}Dout\text{-}IR}$	250	250			(wire delay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

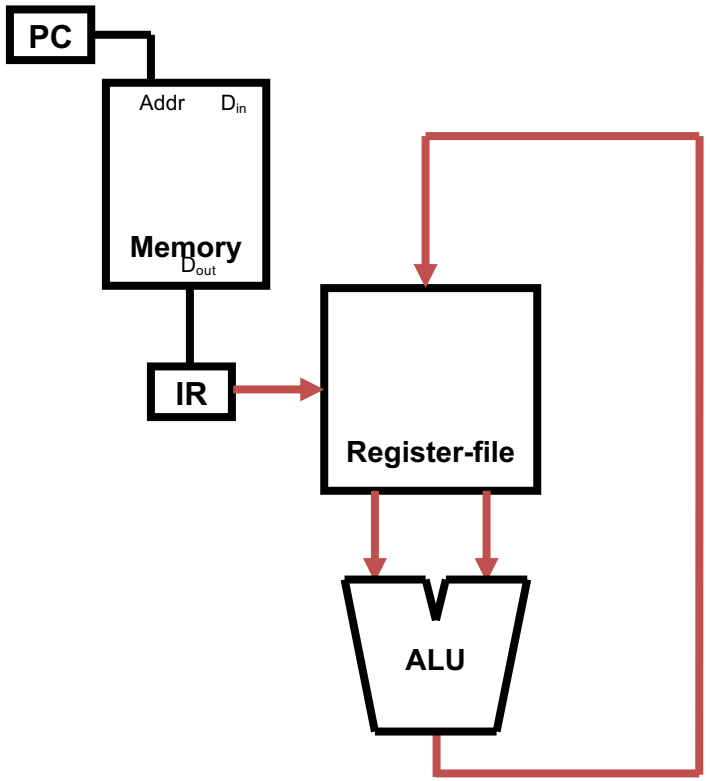
Calculating Clock Minimums



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

Clock must be >= 2060 ps

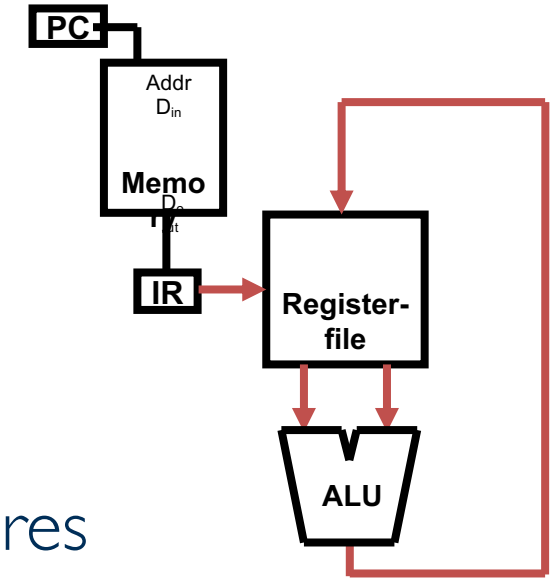
What Other Connections Do We Need?



- 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?

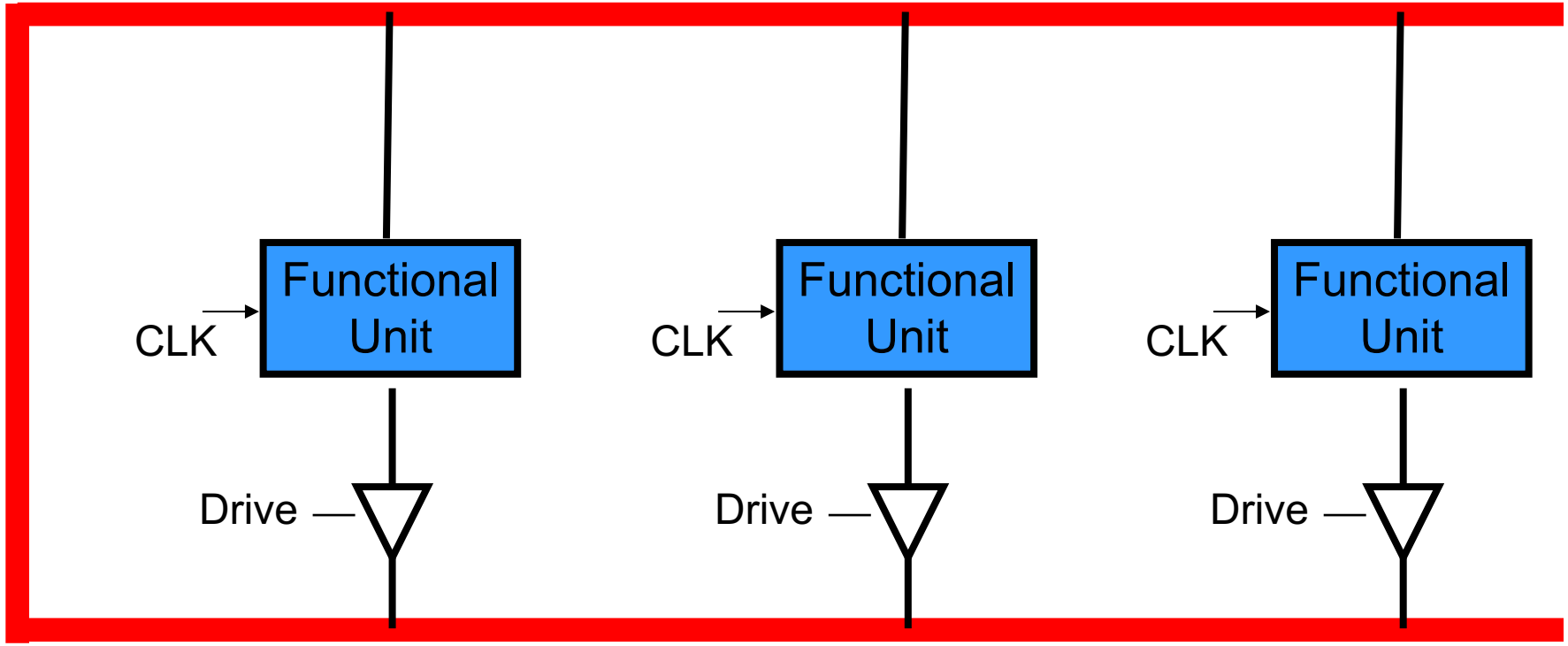
- R-type instructions (add, nand):
  - Opcode, x, y, z
- I-type instructions (addi, lw, sw, beq):
  - Opcode, x, y, offset
- J-type instructions (jalr):
  - Opcode, x, y
- O-type instructions (halt):
  - Opcode

Towards bus-based design

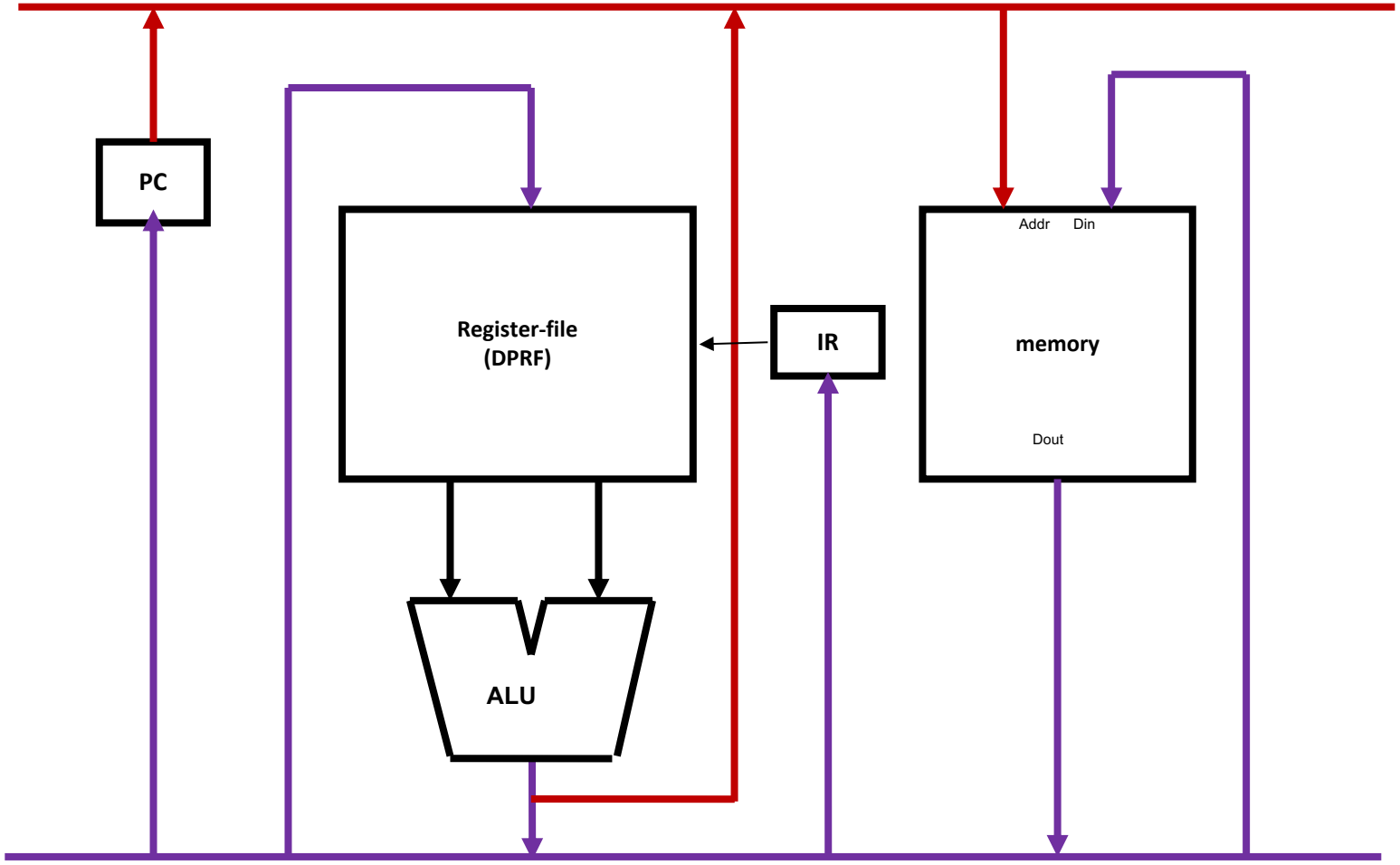
- In principle we must make connections between circuit elements for every instruction
- Numerous connections are expensive and take up valuable space
- Have a set of wires that all elements can connect to and share in order to transfer information

Concept of a Bus

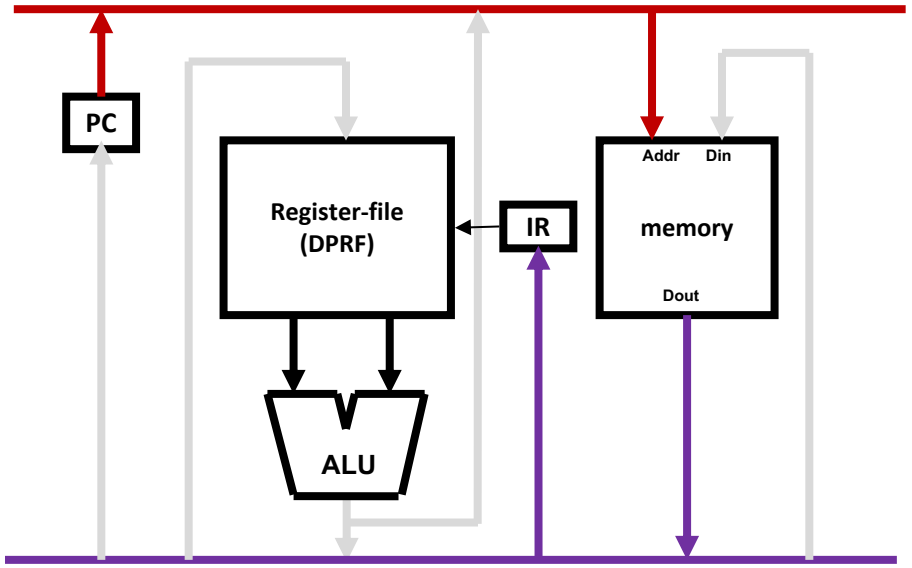
- So what's that "Drive" thing?
- It's another term for a Tri-state Buffer



A Two-Bus Design

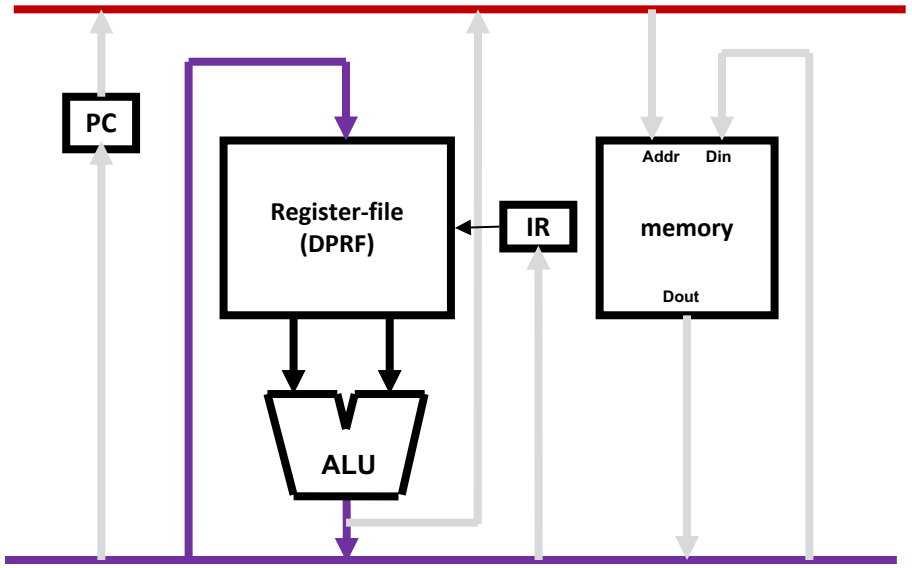


Clock Cycle



PC → MEM → IR

Clock Cycle



IR → Reg File → ALU → Reg File

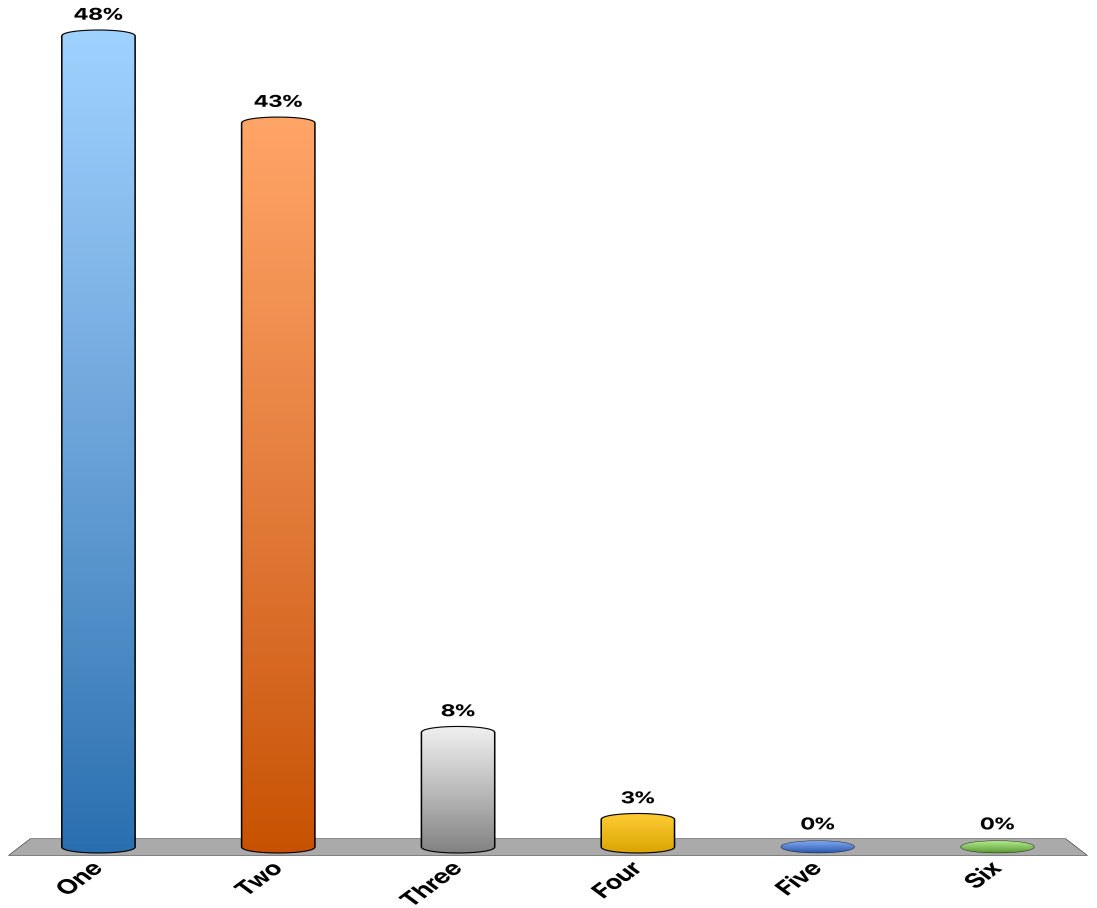
Session ID:
dag2200

Two Busses: How many cycles?

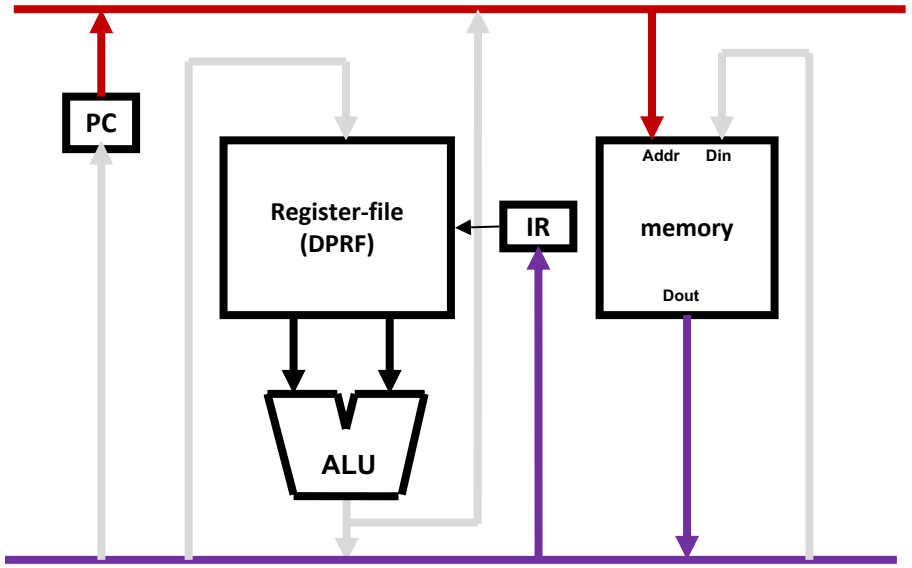
Using just two busses, how many clock cycles does it take to execute

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

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

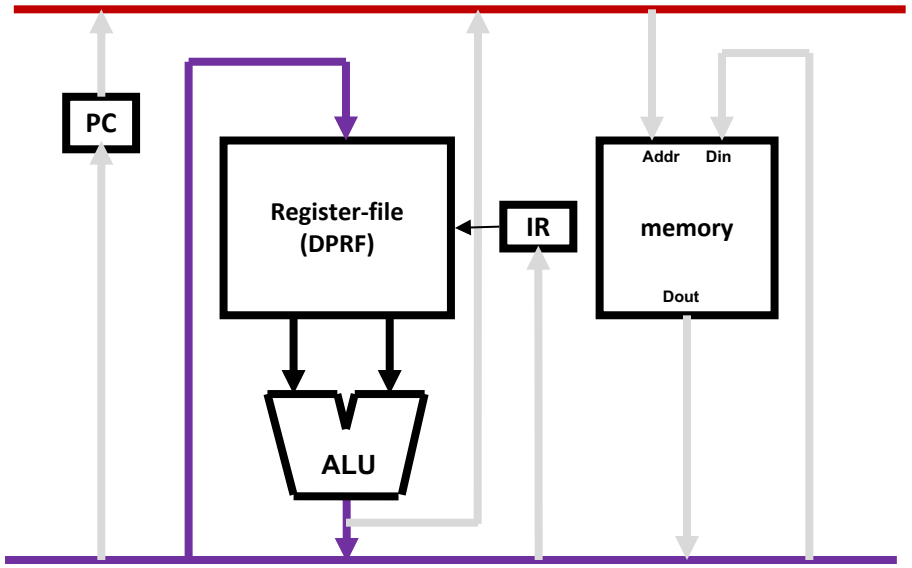


First Clock Cycle



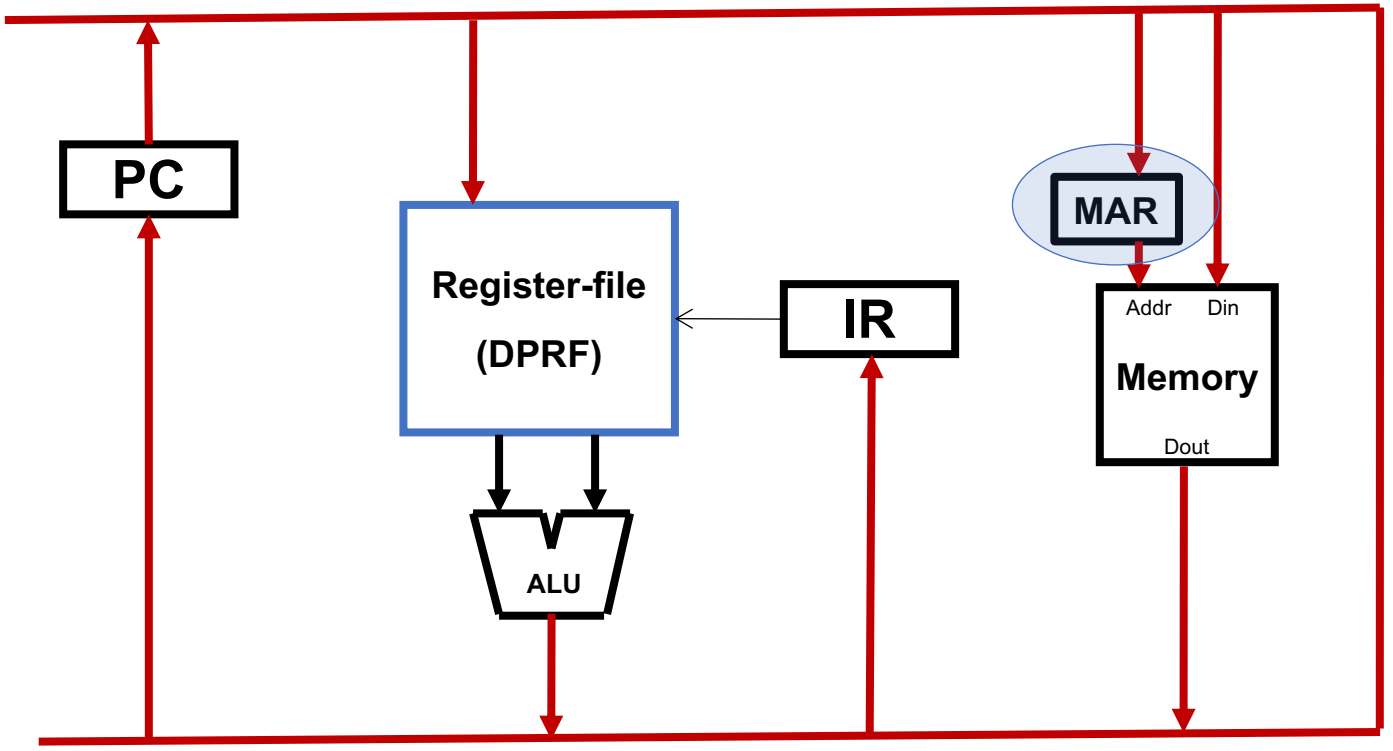
First: PC → MEM → IR

Second Clock Cycle

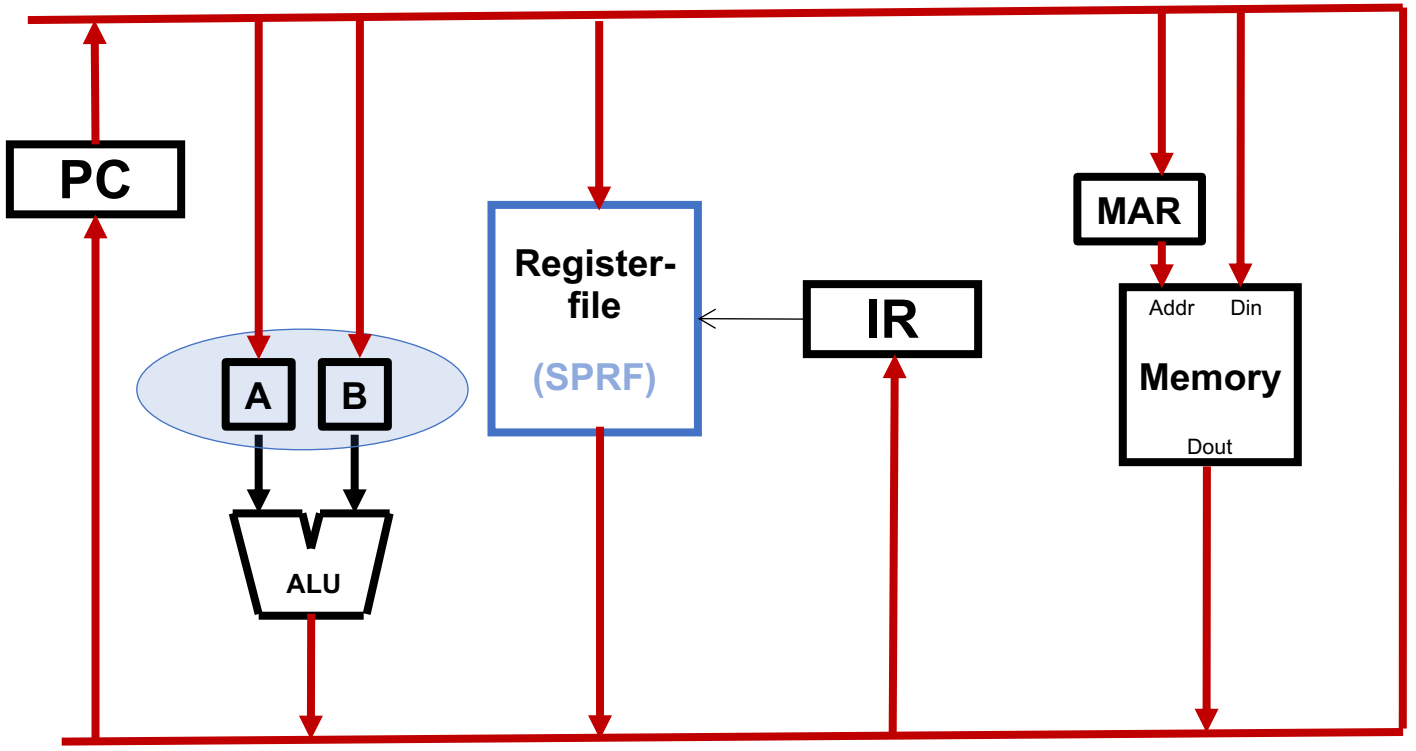


Second: IR → Reg File → ALU → Reg File

Single Bus Design



Single Bus Design w/o DPRF



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

Session ID: dag2200

Question...

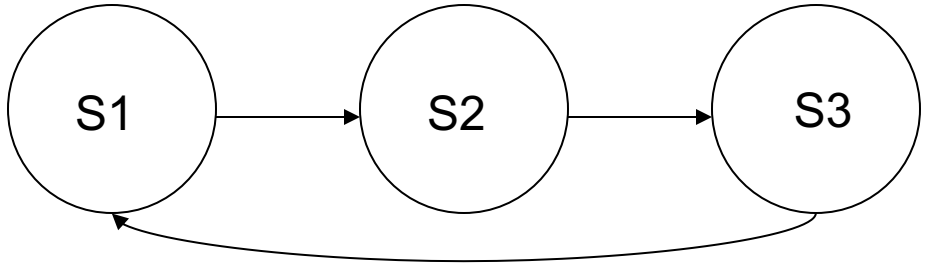
Using a single-ported register file, what is the minimum number of cycles it will take to prepare the ALU for a two-operand calculation?

Rank	Responses

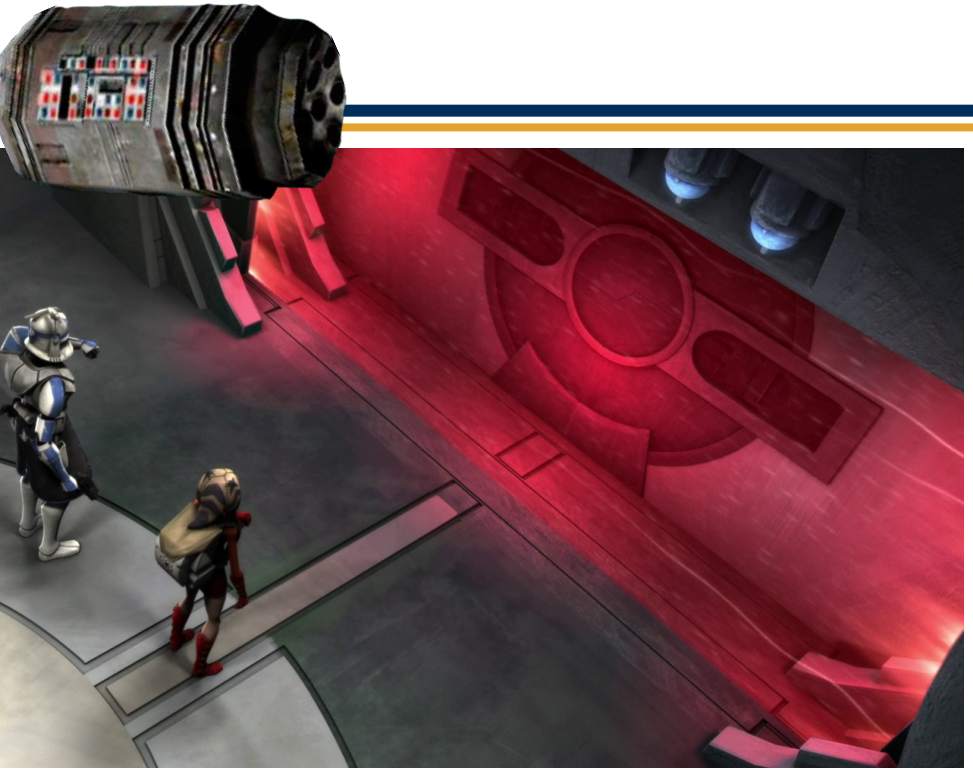
Topics

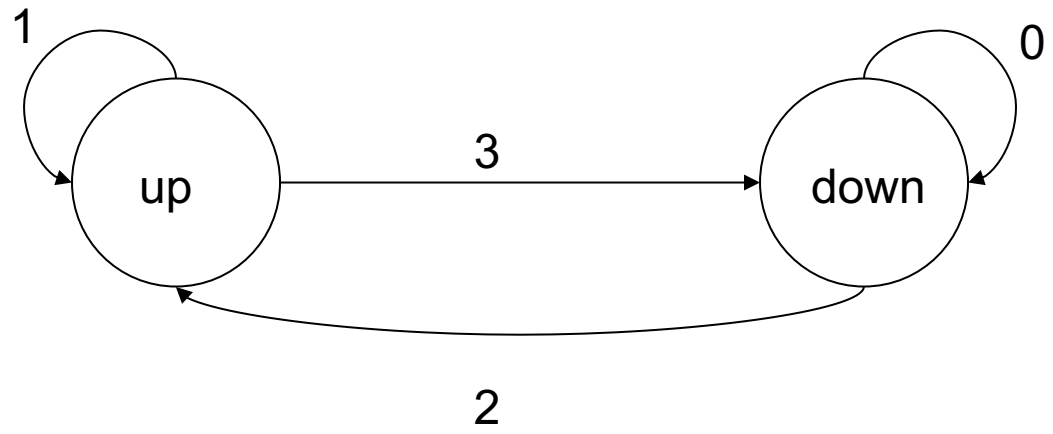
- ~~Logic design review~~
- ~~Data paths~~
- Finite State Machines

Simple FSM Example



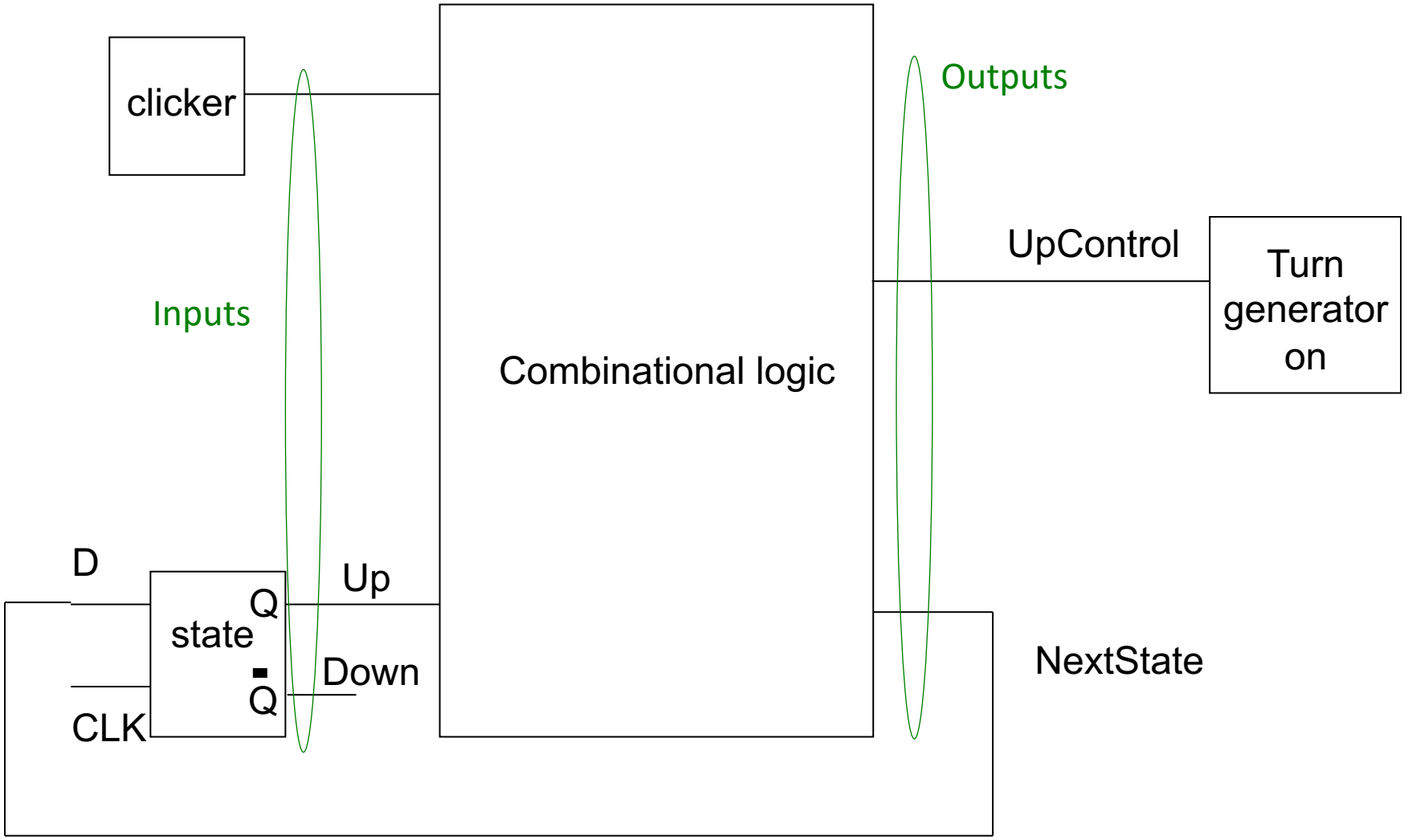
Ray Shield





Transition No	Current State	Clicker	Ray shield generator input	Next State
0	down	none	no change	down
1	up	none	no change	up
2	down	click	on	up
3	up	click	off	down

Ray Shield Controller



Combinational Logic

- RSGSI = CurrentState'&Clicker
- NextState = (CurrentState&Clicker') | (CurrentState'&Clicker)

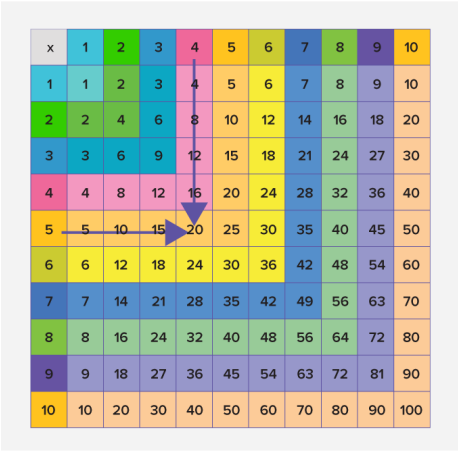
Transition No	Current State	Clicker	Ray shield generator "start" input	Next State
0	0	0	0	0
1	1	0	0	1
2	0	1	1	1
3	1	1	0	0

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?



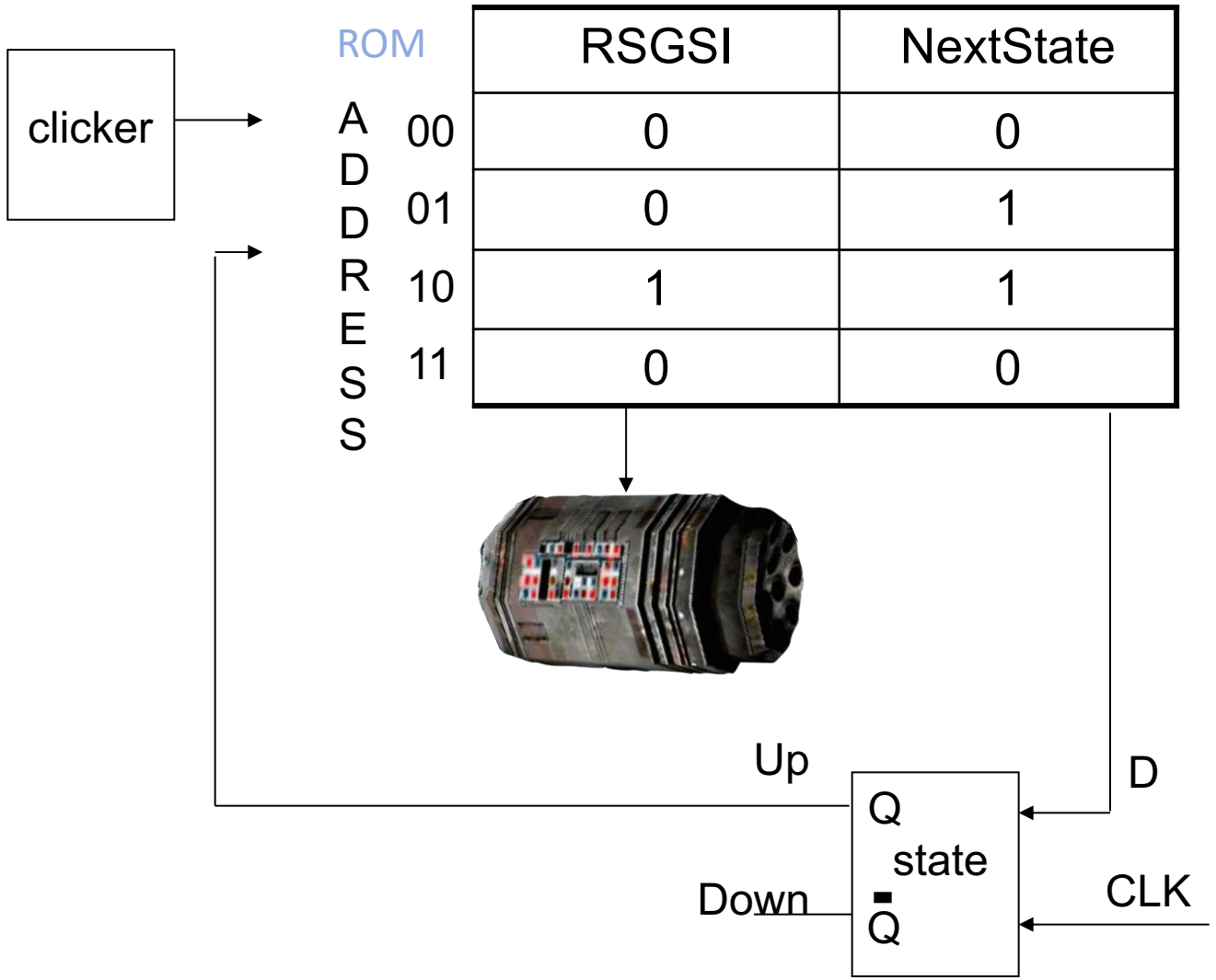
x	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40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

From FSM to ROM

Transition No	Current State	Clicker	Ray shield generator "start" input	Next State
0	0	0	0	0
1	1	0	0	1
2	0	1	1	1
3	1	1	0	0

ROM: 2-bit word x 4 words

Replacing Discrete Logic with a ROM



How large a ROM?

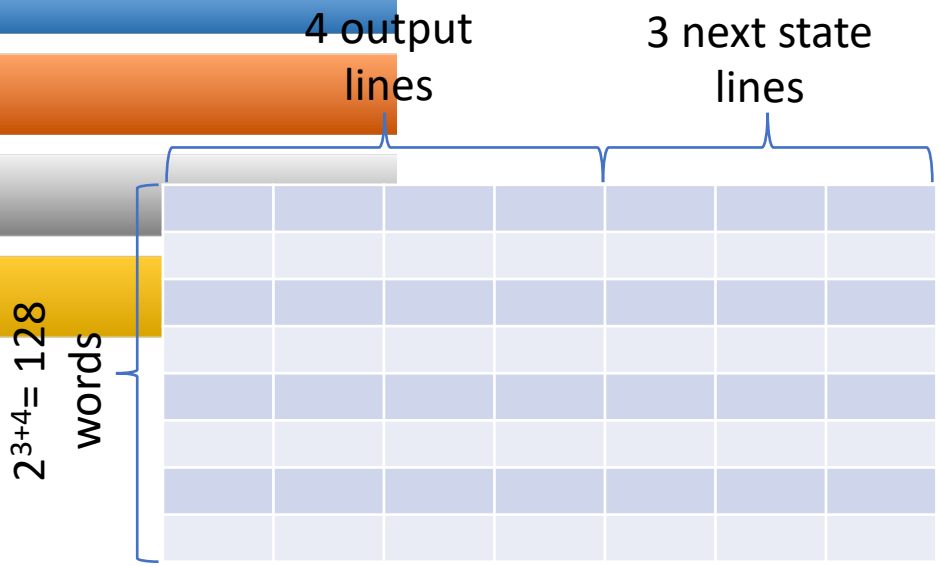
If you have a truth table with 4 binary inputs, 8 states (i.e., 3 state bits), and 4 outputs, what size ROM should you use to encode it?

25% A. 2^7 words of 7 bits

25% B. 2^4 words of 4 bits

25% C. 2^8 words of 7 bits

25% 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)

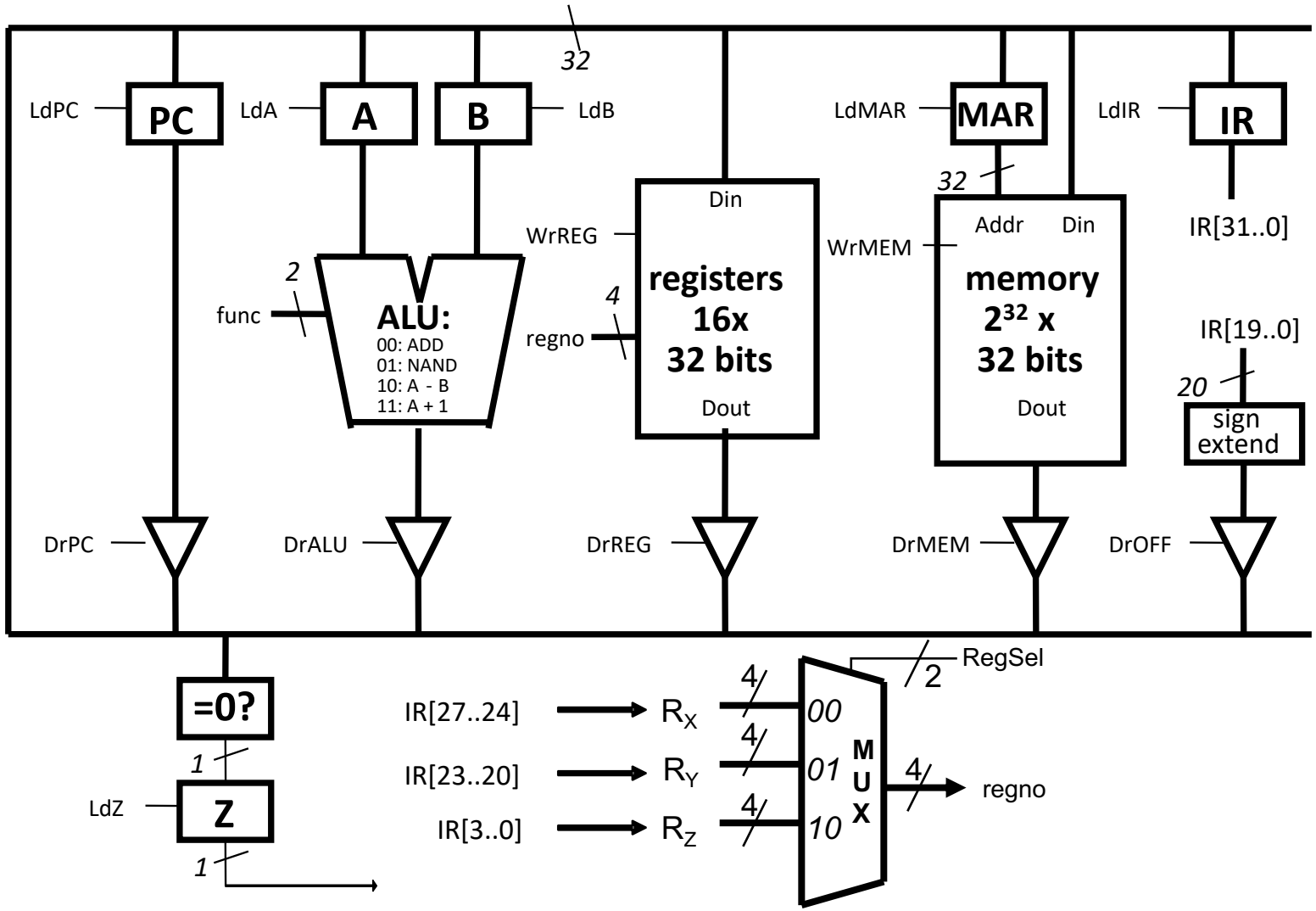
Levels of Abstraction

Application (Algorithms expressed in High Level Language)
System software (Compiler, OS, etc.)
Computer Architecture (ISA)
Machine Organization (Datapath and Control)
Sequential and Combinational Logic Elements
Logic Gates
Transistors
Solid-State Physics (Electrons and Holes)

Levels of Abstraction

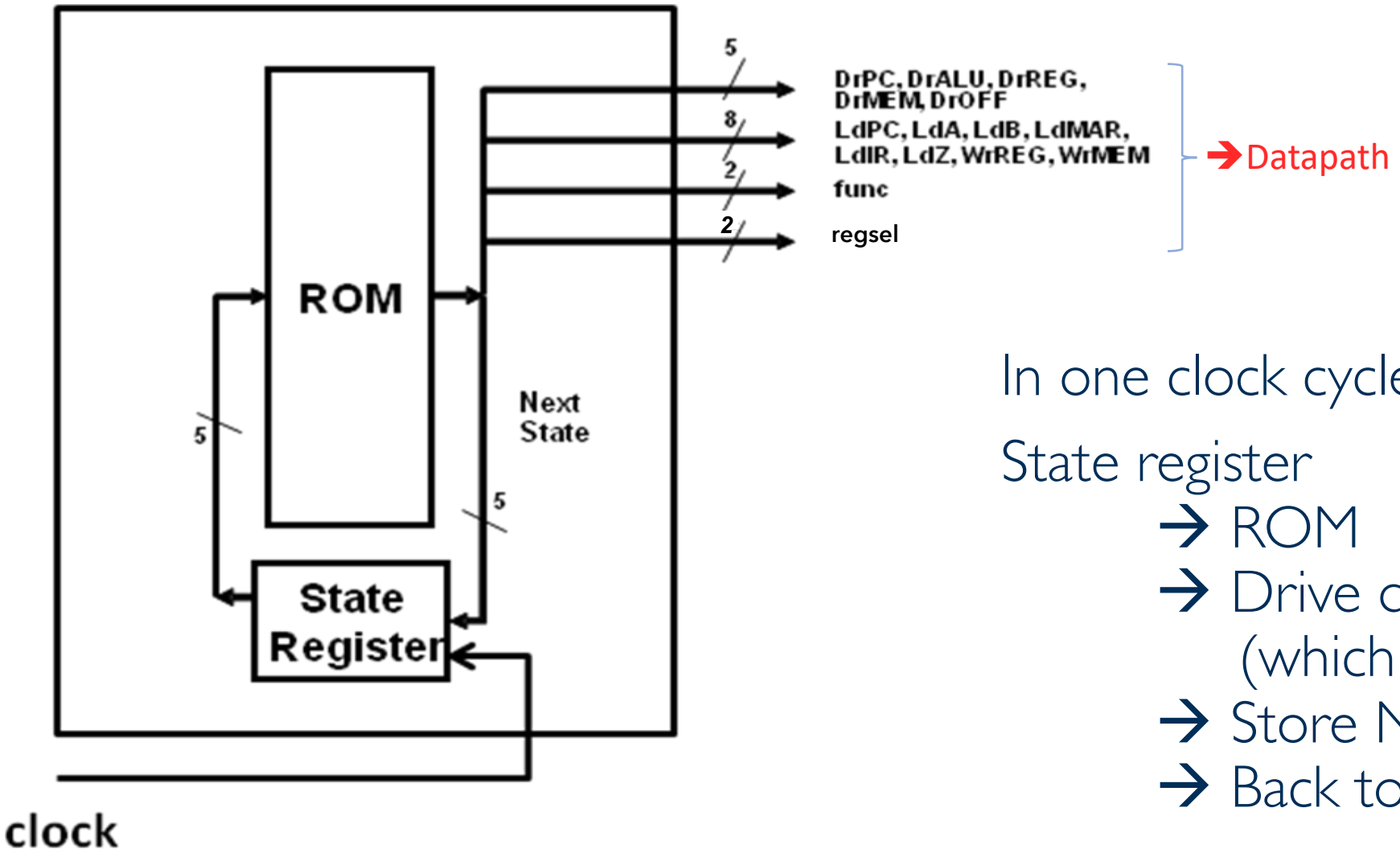
Application (Algorithms expressed in High Level Language)
System software (Compiler, OS, etc.)
Computer Architecture (ISA)
Machine Organization - Control
Machine Organization - Datapath
Sequential and Combinational Logic Elements
Logic Gates
Transistors
Solid-State Physics (Electrons and Holes)

We've Got a Datapath for LC-2200!



What else do we need??

A Control Unit!



In one clock cycle:

State register

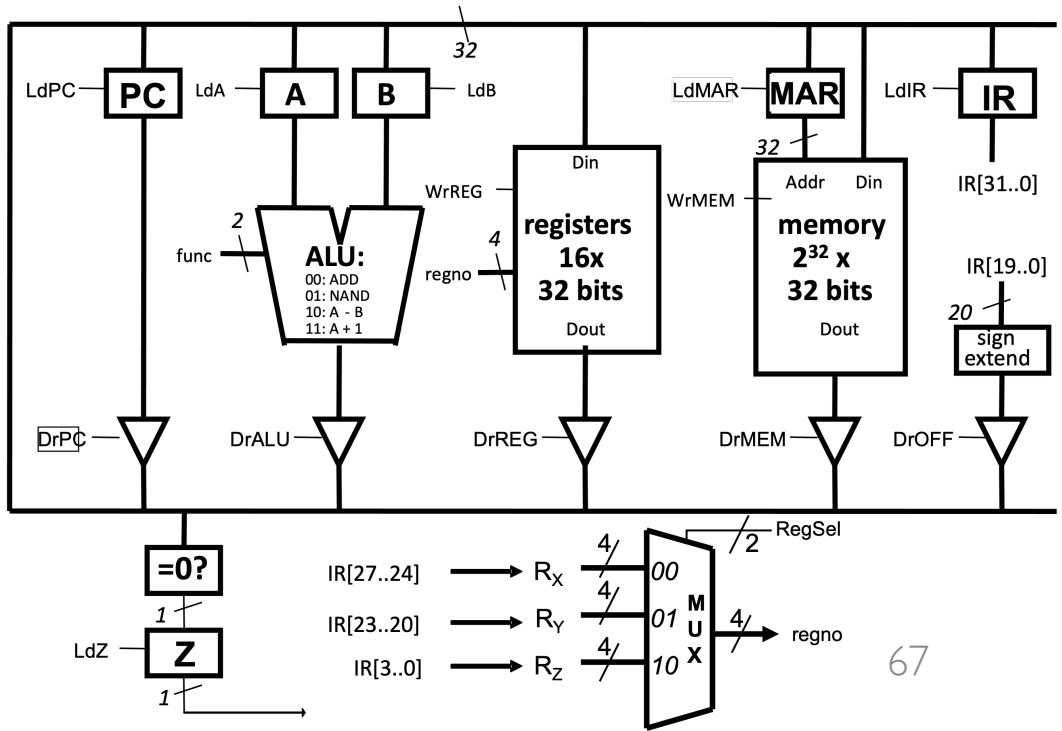
- → ROM
- → Drive datapath (which does the work)
- → Store Next State
- → Back to the start

What's in the ROM?

	Drive Signals					Load Signals						Write Signals			
Current State	PC	ALU	Reg	MEM	OFF	PC	A	B	MAR	IR	Z	MEM	REG	func	regno

Recognize all these as the control signals to drive the datapath

You will find each one on the datapath diagram!



The Next State is Stored in the ROM, Too!

	Drive Signals					Load Signals						Write Signals				
Current State	PC	ALU	Reg	MEM	OFF	PC	A	B	MAR	IR	Z	MEM	REG	func	regno	Next State
...																

In addition to being the Current State, this is also necessarily the address of the word in the ROM

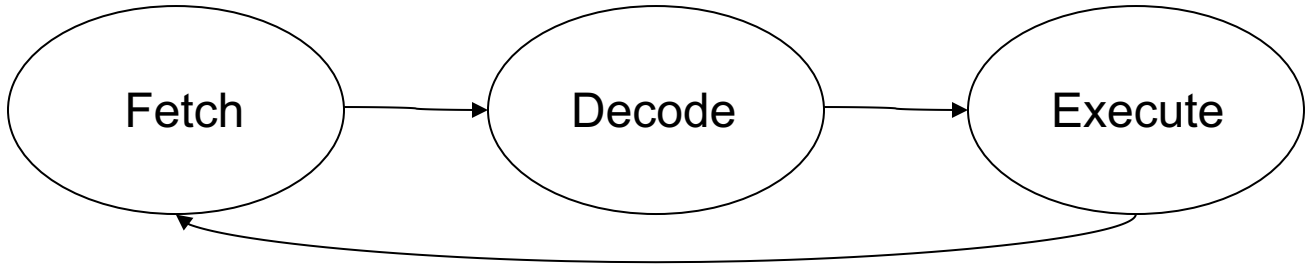
This Means the ROM Contents Are Our Microprogram!

	Drive Signals					Load Signals						Write Signals				
Current State	PC	ALU	Reg	MEM	OFF	PC	A	B	MAR	IR	Z	MEM	REG	func	regno	Next State
00000	1						1		1							00001
...																

For short, we might write this microinstruction as
00000: DrPC LdMAR LdA next=00001

A Familiar State Diagram?

- What state diagram do you think a CPU implementer might be concerned with?



- Is a processor implementation a Finite State Machine?
- What happens in each state?
- What resources are needed to execute each instruction?

Implementing the LC-2200 ISA

- R-type instructions
  - Sequence of machine states are similar
  - Only the ALU op changes
- J-type instructions
  - Straightforward
- I-type instructions (LW, SW, ADDI)
  - Straightforward
- I-type instructions (BEQ)
  - May take some thought…
  - Let's do that first