

# Great Ideas in Computer Architecture

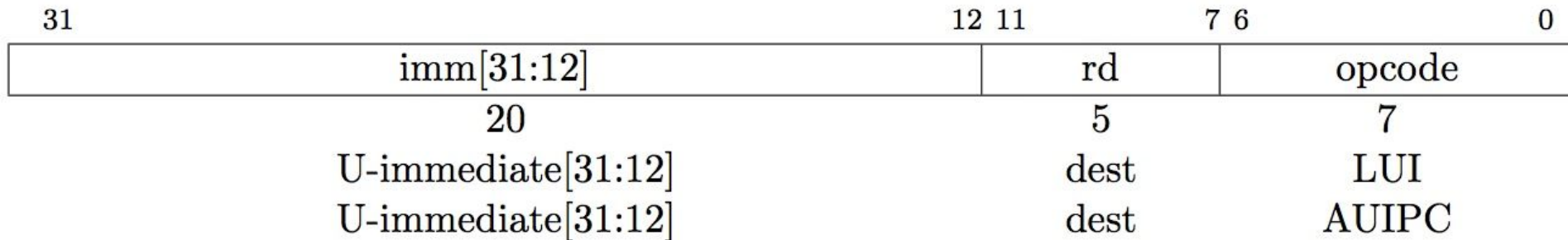
## *RISC-V CPU Control, Pipelining*

Instructor: Nick Riasanovsky

# Agenda

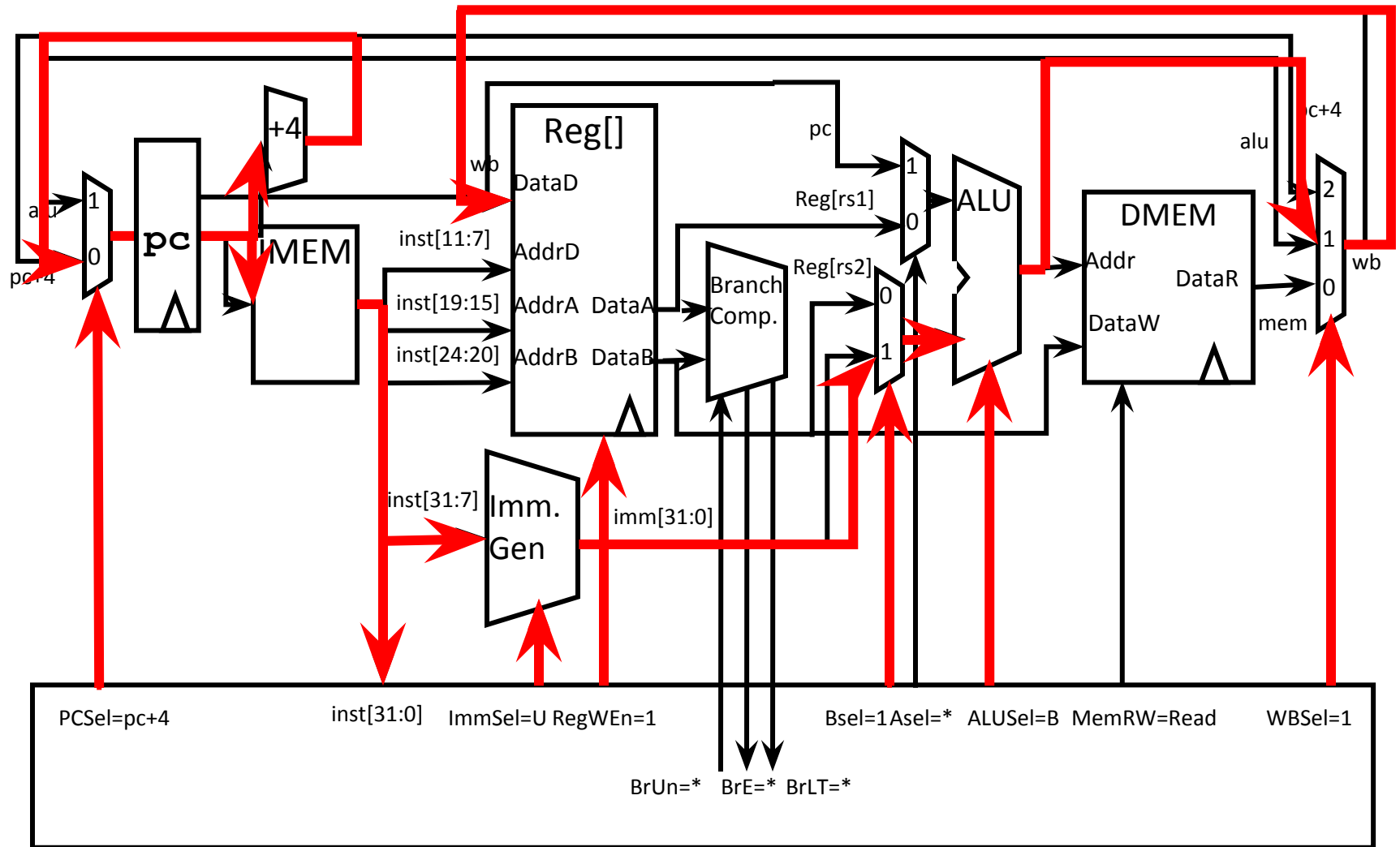
- Datapath Review
- Control Implementation
- Administrivia
- Performance Analysis
- Pipelined Execution
- Pipelined Datapath

# “Upper Immediate” instructions

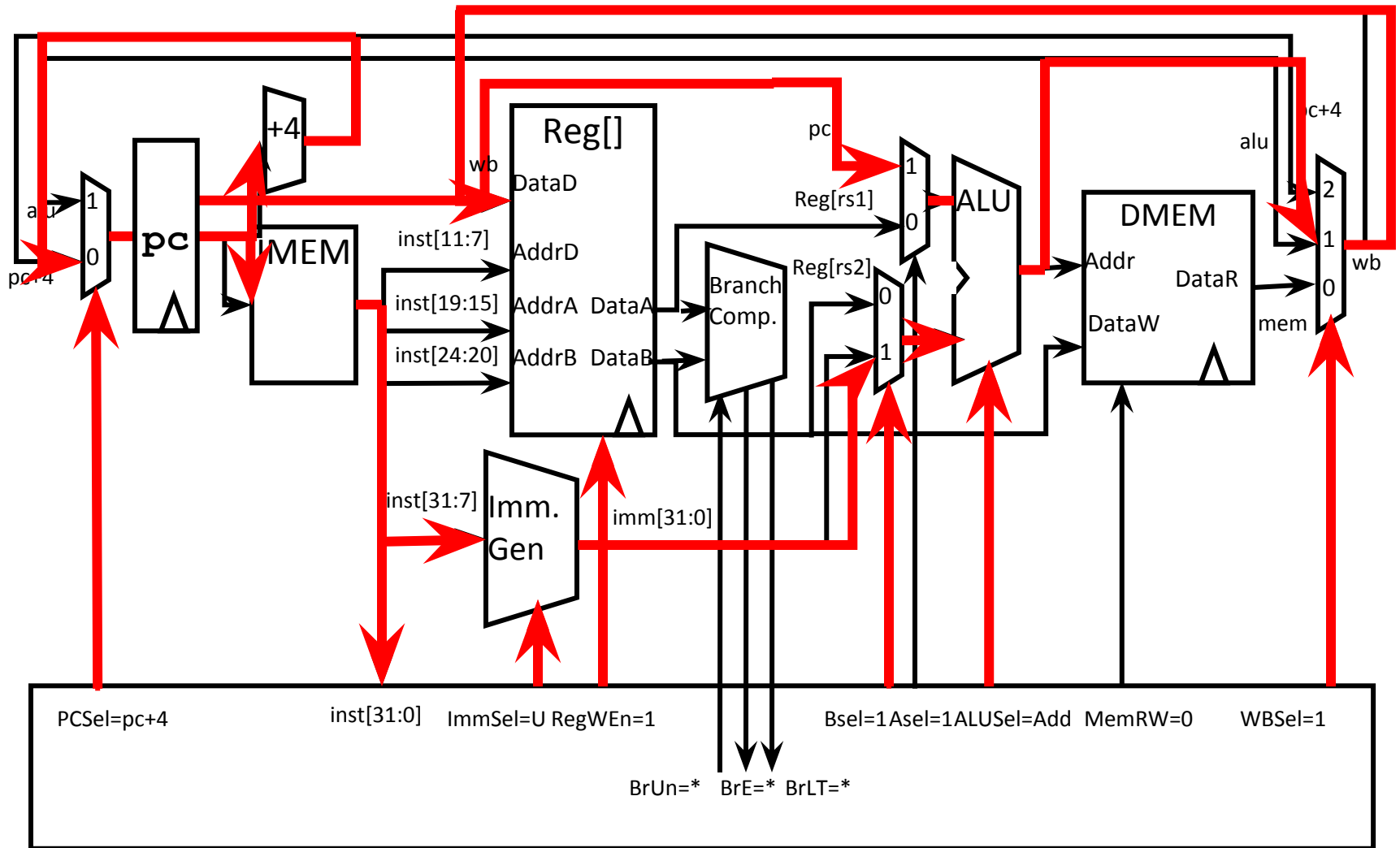


- Has 20-bit immediate in upper 20 bits of 32-bit instruction word
- One destination register, rd
- Used for two instructions
  - LUI – Load Upper Immediate (add to zero)
  - AUIPC – Add Upper Immediate to PC

# Implementing lui



# Implementing auipc



# All immediates

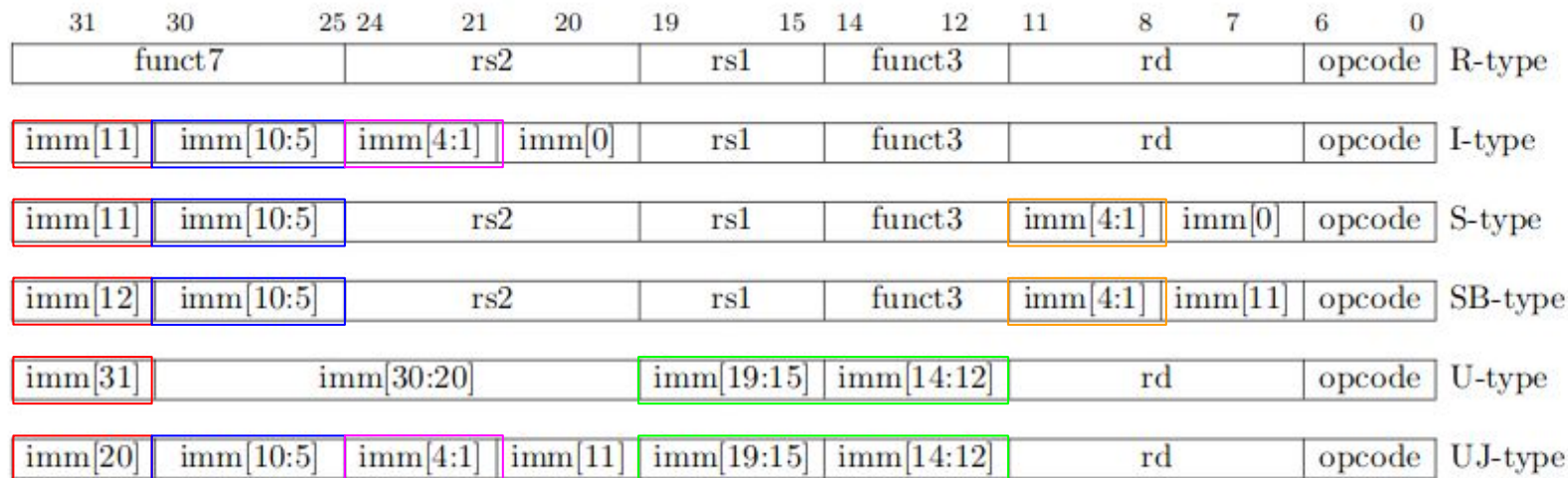
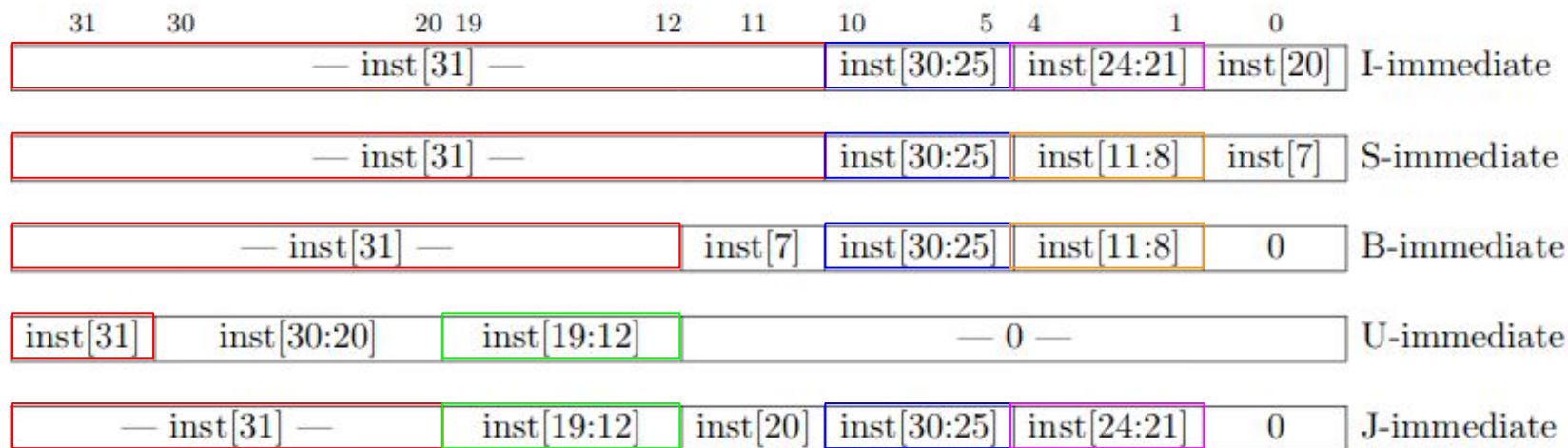
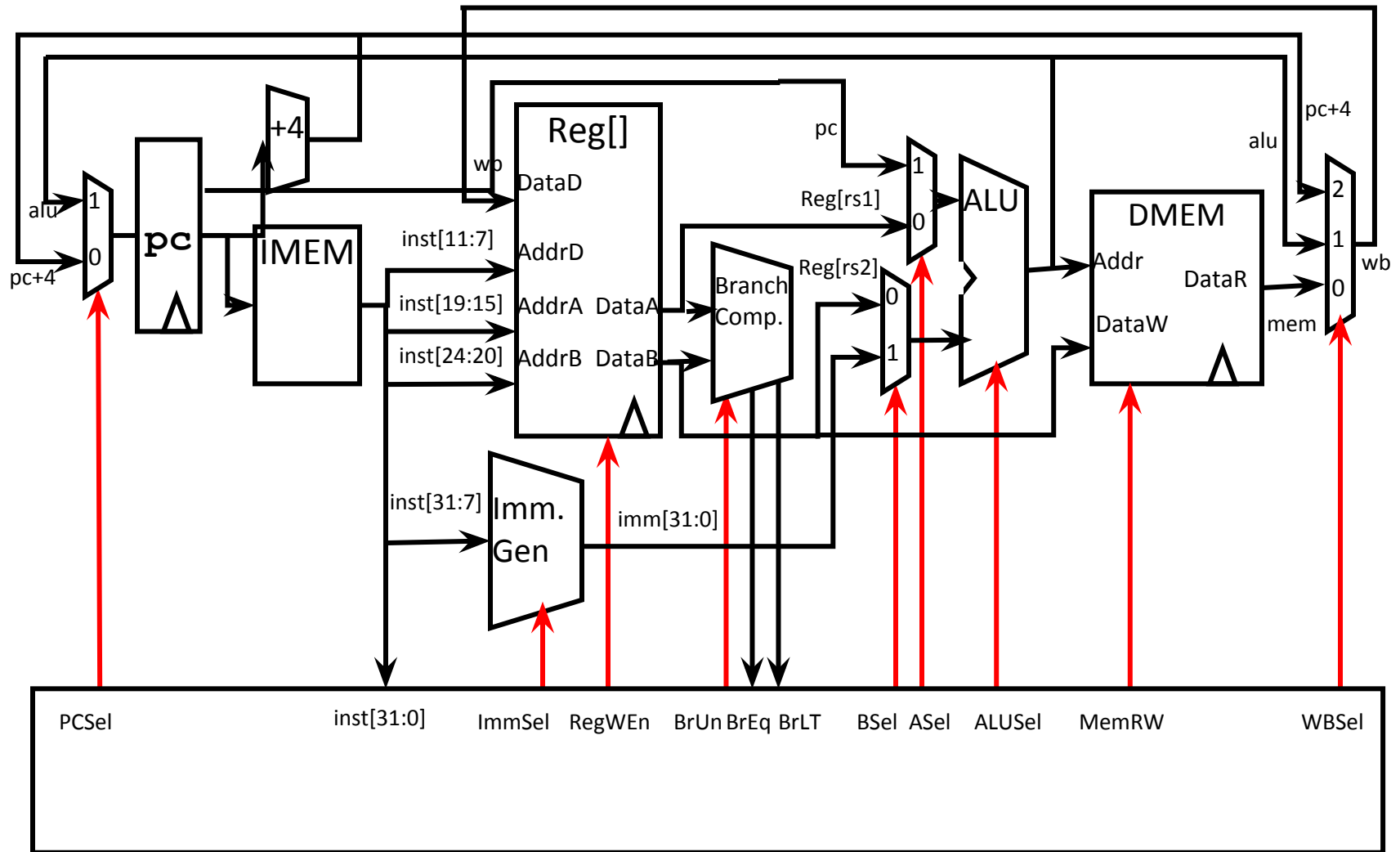


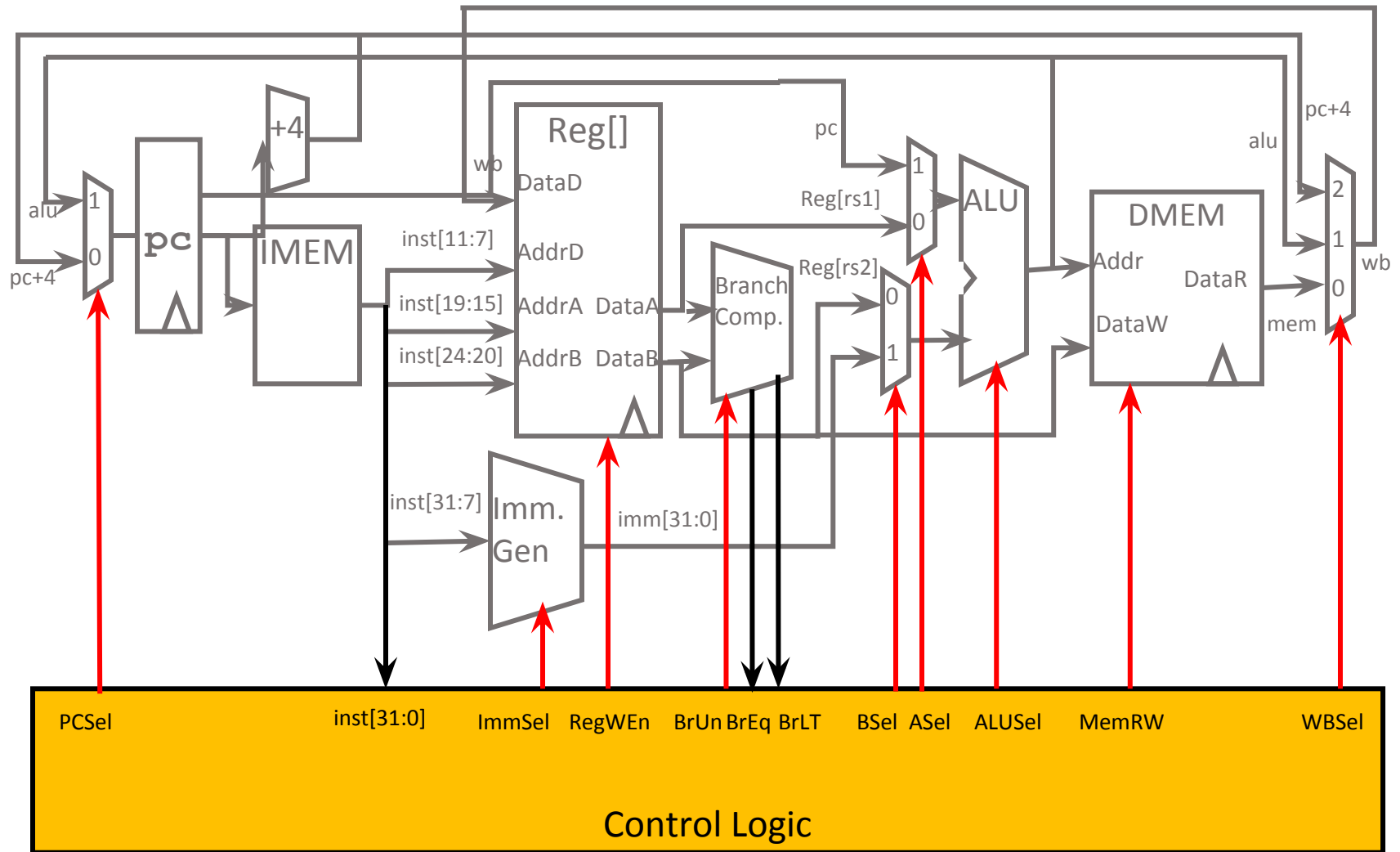
Figure 2.3: RISC-V base instruction formats showing immediate variants.



# Single-Cycle RISC-V RV32I Datapath



# Single-Cycle RISC-V RV32I Datapath





**Question:** Which statement is TRUE about our RV32I ISA?

- (A)** When not in use, parts of the datapath cease to carry a value.
- (B)** Adding the instruction `lbu` will not change the datapath.
- (C)** All control signals will be don't care ('X') in at least one instruction.
- (D)** Adding the instruction `bge` will not change the datapath.

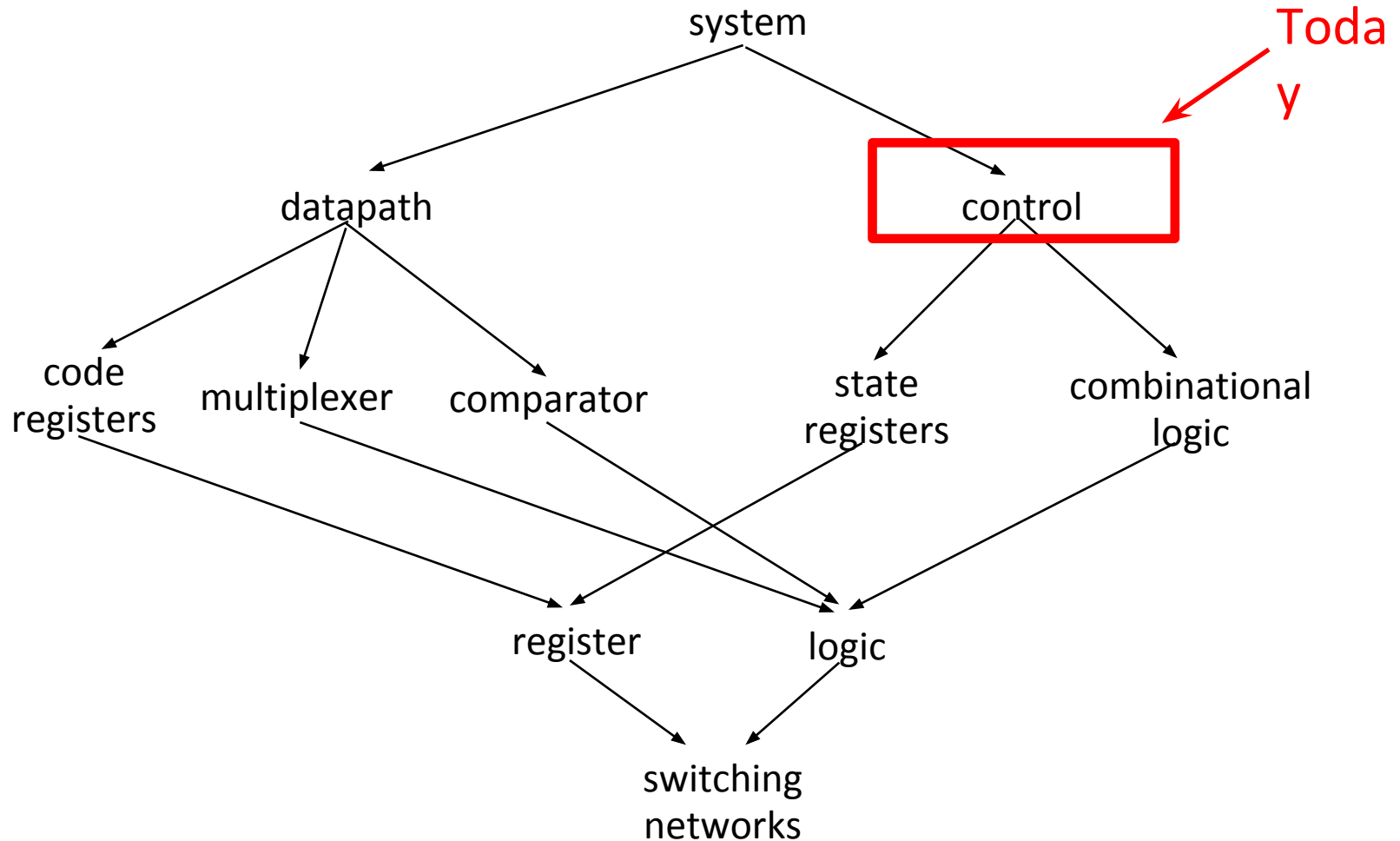
**Question:** Which statement is TRUE about the RV32I ISA?

- (A) When not in use, parts of the datapath cease to carry a value.
- (B) Adding the instruction `lbu` will not change the datapath.
- (C) All control signals will be don't care ('X') in at least one instruction.
- (D) Adding the instruction `bge` will not change the datapath.

# Agenda

- Quick Datapath Review
- **Control Implementation**
- Administrivia
- Performance Analysis
- Pipelined Execution
- Pipelined Datapath

# Hardware Design Hierarchy



# Control Logic Truth Table (incomplete)

Inst[31:0]	BrE q	BrLT	PCSel	ImmSe l	BrU n	ASel	BSel	ALUSel	MemR W	RegWE n	WBSel
<b>add</b>	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
<b>sub</b>	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
<b>(R-R Op)</b>	*	*	+4	*	*	Reg	Reg	(Op)	Read	1	ALU
<b>addi</b>	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
<b>lw</b>	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
<b>sw</b>	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
<b>beq</b>	0	*	+4	B	*	PC	Imm	Add	Read	0	*
<b>beq</b>	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
<b>bne</b>	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
<b>bne</b>	1	*	+4	B	*	PC	Imm	Add	Read	0	*
<b>blt</b>	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
<b>bltu</b>	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
<b>jalr</b>	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
<b>jal</b>	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
<b>auipc</b>	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

# RV32I, a nine-bit ISA!

inst[30]

inst[14:12]

inst[6:2]

imm[31:12]					rd	011011	LUI
imm[31:12]					rd	001011	AUIPC
imm[20 10:1 11 19:12]					rd	110111	JAL
imm[11:0]					rd	110011	JALR
imm[12 10:5]	rs2	rs1	000	imm[4:1 11]	1100011	BEQ	
imm[12 10:5]	rs2	rs1	001	imm[4:1 11]	1100011	BNE	
imm[12 10:5]	rs2	rs1	100	imm[4:1 11]	1100011	BLT	
imm[12 10:5]	rs2	rs1	101	imm[4:1 11]	1100011	BGE	
imm[12 10:5]	rs2	rs1	110	imm[4:1 11]	1100011	BLTU	
imm[12 10:5]	rs2	rs1	111	imm[4:1 11]	1100011	BGEU	
imm[11:0]					rd	0000011	LB
imm[11:0]					rd	0000011	LH
imm[11:0]					rd	0000011	LW
imm[11:0]					rd	0000011	LBU
imm[11:0]					rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB	
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH	
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW	
imm[11:0]					rd	0010011	ADDI
imm[11:0]					rd	0010011	SLTI
imm[11:0]					rd	0010011	SLTIU
imm[11:0]					rd	0010011	XORI
imm[11:0]					rd	0010011	ORI
imm[11:0]					rd	0010011	ANDI

0000000	shamt	rs1	001	rd	0010011	SLLI	
0000000	shamt	rs1	101	rd	0010011	SRLI	
0100000	shamt	rs1	101	rd	0010011	SRAI	
0000000	rs2	rs1	000	rd	0110011	ADD	
0100000	rs2	rs1	000	rd	0110011	SUB	
0000000	rs2	rs1	001	rd	0110011	SLL	
0000000	rs2	rs1	010	rd	0110011	SLT	
0000000	rs2	rs1	011	rd	0110011	SLTU	
0000000	rs2	rs1	100	rd	0110011	XOR	
0000000	rs2	rs1	101	rd	0110011	SRL	
0100000	rs2	rs1	101	rd	0110011	SRA	
0000000	rs2	rs1	110	rd	0110011	OR	
0000000	rs2	rs1	111	rd	0110011	AND	
0000	pred	succ	00000	000	00000	0001111	FENCE
0000	0000	0000	00000	001	00000	0001111	FENCE.I
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK
csr	rs1	001	rd	1110011			CSRRW
csr	rs1	010	rd	1110011			CSRRS
csr	rs1	011	rd	1110011			CSRRC
csr	zimm	101	rd	1110011			CSRRWI
csr	zimm	110	rd	1110011			CSRRSI
csr	zimm	111	rd	1110011			CSRRCI

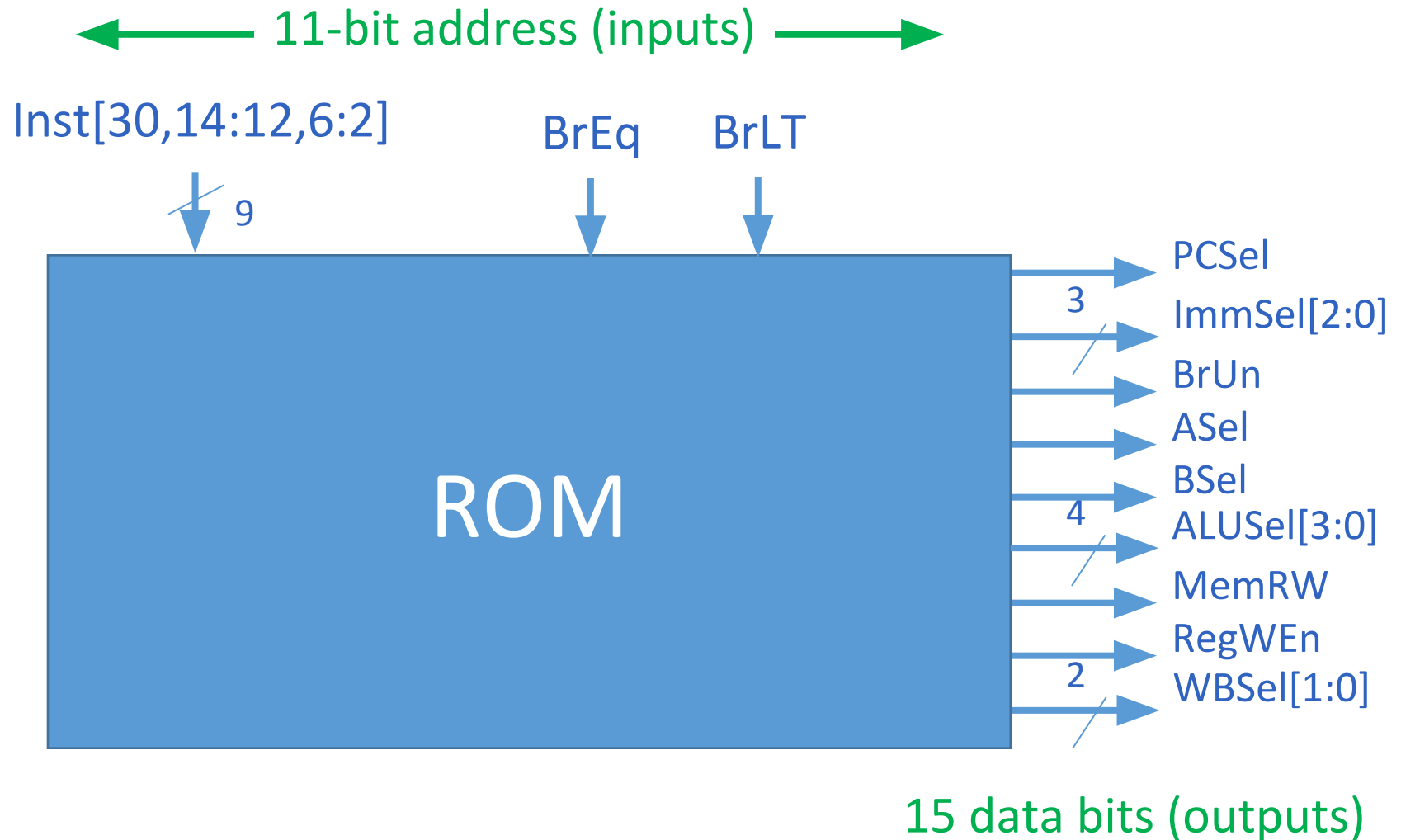
Not in CS61C

Instruction type encoded using only 9 bits  
inst[30],inst[14:12], inst[6:2]

# Control Realization Options

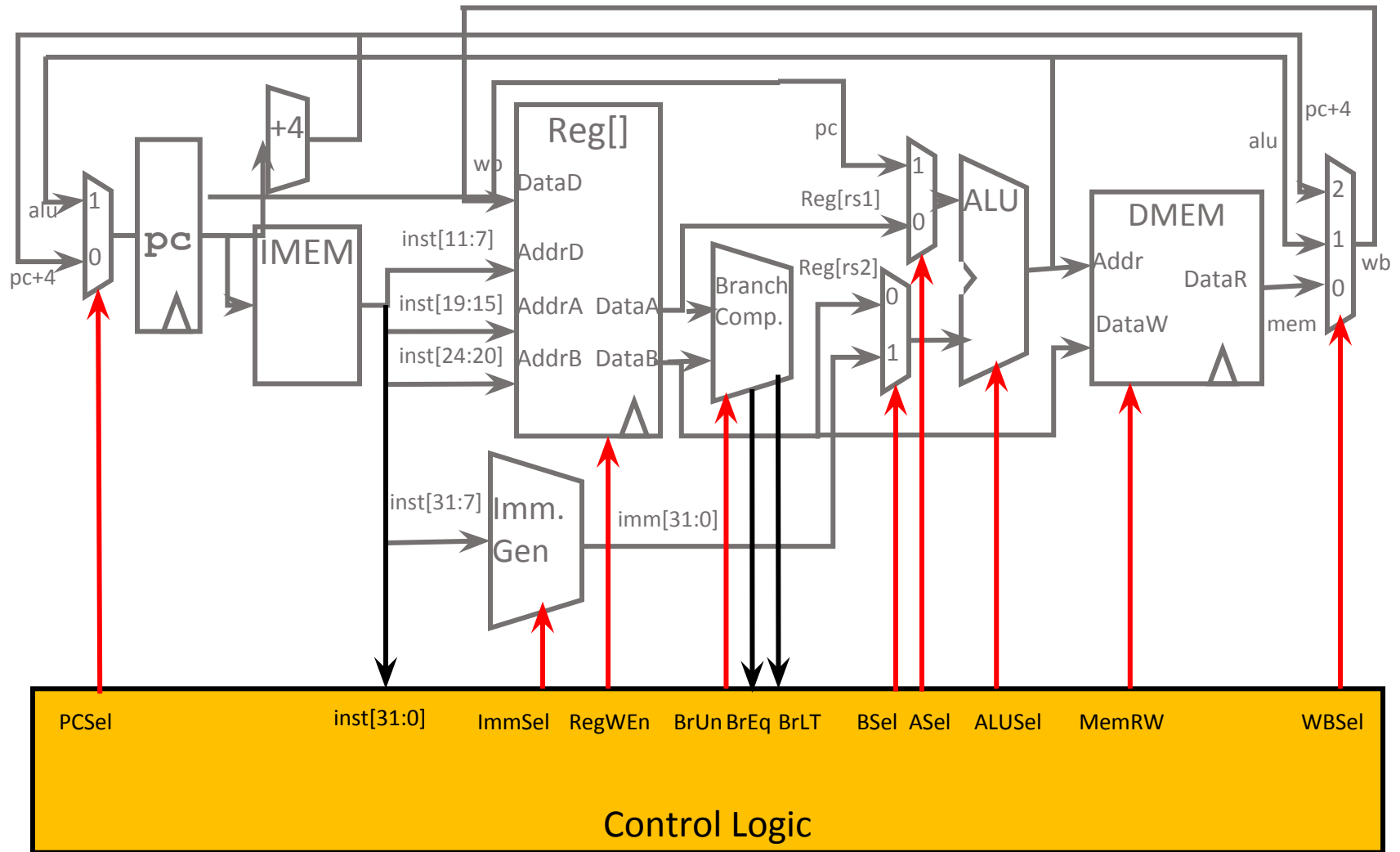
- ROM
  - “Read-Only Memory”
  - Regular structure
  - Can be easily reprogrammed
    - fix errors
    - add instructions
  - Popular when designing control logic manually
- Combinatorial Logic
  - Today, chip designers use logic synthesis tools to convert truth tables to networks of gates

# ROM-based Control

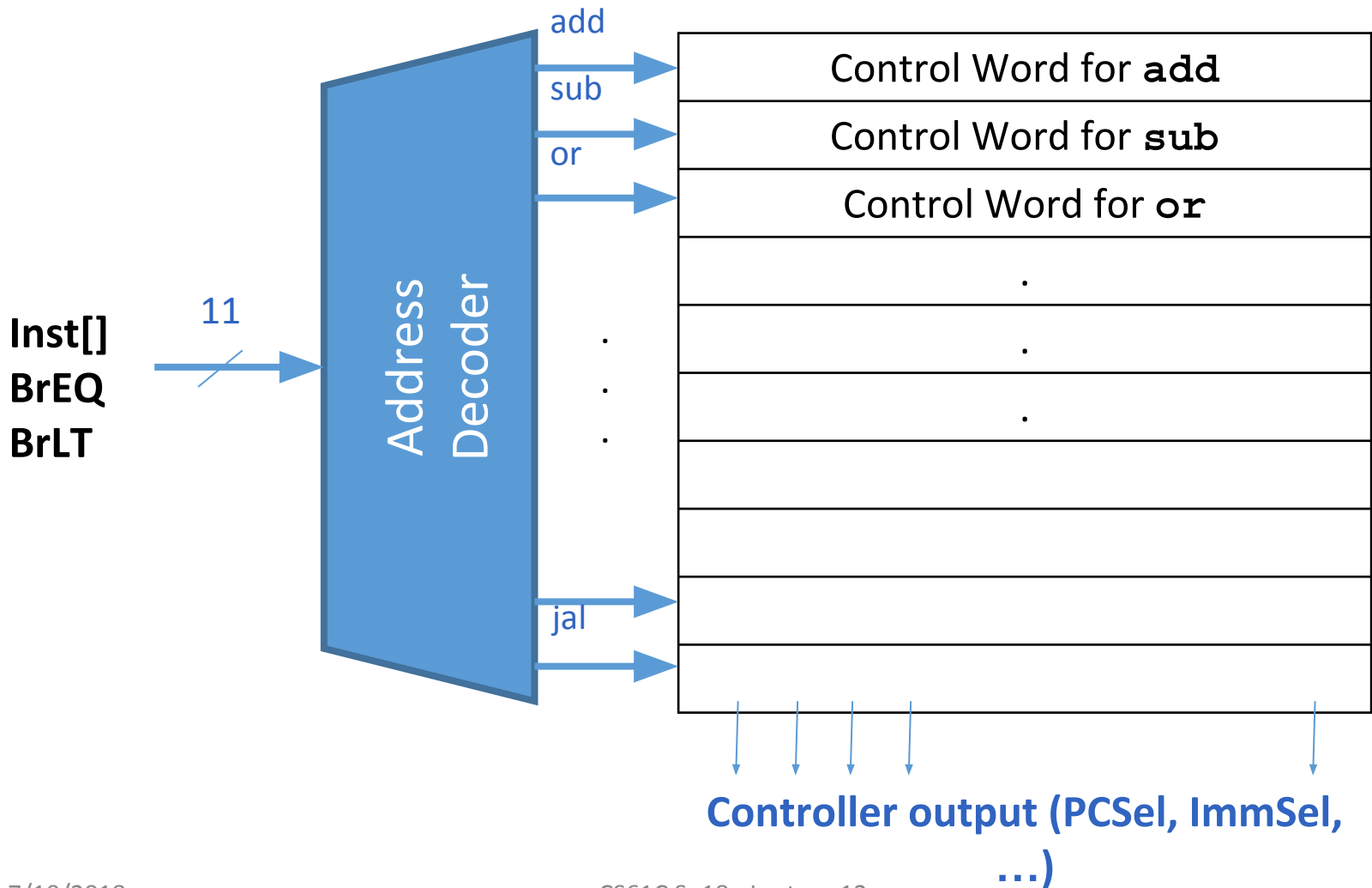




# Single-Cycle RISC-V RV32I Datapath



# ROM Controller Implementation



# Agenda

- Quick Datapath Review
- Control Implementation
- **Administrivia**
- Performance Analysis
- Pipelined Execution
- Pipelined Datapath

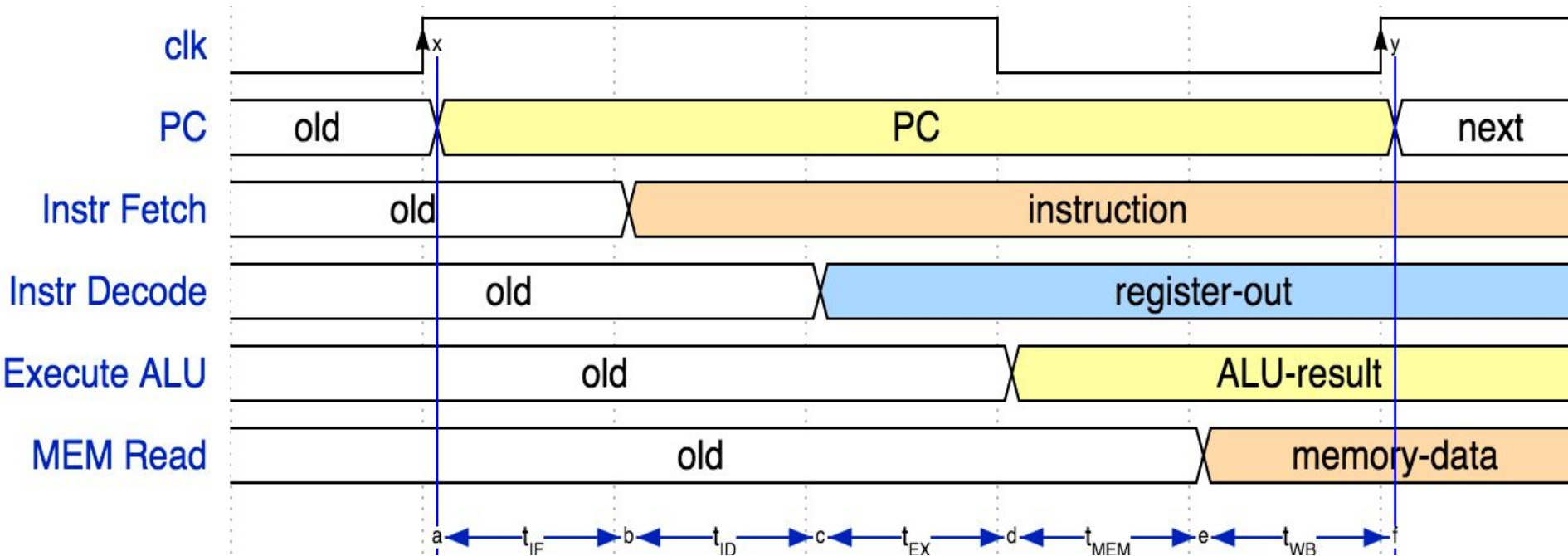
# Administrivia

- Regrade requests are due tonight
- Homework 3/4 due 7/16! (NOT 7/13, oops)
- Project 2-2 due Friday
- Project 3 released on Thurs, will rely on lab 6, so make sure you're caught up on labs!
- Guerilla session tomorrow night 7/11
- HW Grades
  - Make sure edx/instructional account emails match!

# Agenda

- Quick Datapath Review
- Control Implementation
- Administrivia
- **Performance Analysis**
- Pipelined Execution
- Pipelined Datapath

# Instruction Timing



IF	ID	EX	MEM	WB	Total
I-MEM	Reg Read	ALU	D-MEM	Reg W	
200 ps	100 ps	200 ps	200 ps	100 ps	800 ps

# Instruction Timing

Instr	IF = 200ps	ID = 100ps	ALU = 200ps	MEM=200ps	WB = 100ps	Total
add	X	X	X		X	600ps
beq	X	X	X			500ps
jal	X	X	X			500ps
lw	X	X	X	X	X	800ps
sw	X	X	X	X		700ps

- Maximum clock frequency

- $f_{\max} = 1/800\text{ps} = 1.25 \text{ GHz}$

- Most blocks idle most of the time

- E.g.  $f_{\max, \text{ALU}} = 1/200\text{ps} = 5 \text{ GHz!}$

- How can we keep ALU busy all the time?

- 5 billion adds/sec, rather than just 1.25 billion?

- Idea: Factories use three employee shifts - equipment is always busy!

# Performance Measures

- “Our” RISC-V executes instructions at 1.25 GHz
  - 1 instruction every 800 ps
- Can we improve its performance?
  - What do we mean with this statement?
  - Not so obvious:
    - Quicker response time, so one job finishes faster?
    - More jobs per unit time (e.g. web server returning pages)?
    - Longer battery life?





# Transportation Analogy



	Sports Car	Bus
Passenger Capacity	2	50
Travel Speed	200 mph	50 mph
Gas Mileage	5 mpg	2 mpg

## 50 Mile trip:

	Sports Car	Bus
Travel Time	15 min	60 min
Time for 100 passengers	750 min	120 min
Gallons per passenger	5 gallons	0.5 gallons

# Computer Analogy

Transportation	Computer
Trip Time	Program execution time: e.g. time to update display
Time for 100 passengers	Throughput: e.g. number of server requests handled per hour
Gallons per passenger	Energy per task*: e.g. how many movies you can watch per battery charge or energy bill for datacenter

\* Note: power is not a good measure, since low-power CPU might run for a long time to complete one task consuming more energy than faster computer running at higher power for a shorter time

# “Iron Law” of Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

# Instructions per Program

Determined by

- Task
- Algorithm, e.g.  $O(N^2)$  vs  $O(N)$
- Programming language
- Compiler
- Instruction Set Architecture (ISA)

# (Average) Clock cycles per Instruction

Determined by

- ISA
- Processor implementation (or *microarchitecture*)
- E.g. for “our” single-cycle RISC-V design, CPI = 1
- Complex instructions (e.g. **strcpy**), CPI  $\gg 1$
- Superscalar processors, CPI  $< 1$  (next lecture)

# Time per Cycle ( $1/\text{Frequency}$ )

## Determined by

- Processor microarchitecture (determines critical path through logic gates)
- Technology (e.g. transistor size)
- Power budget (lower voltages reduce transistor speed)

# Speed Tradeoff Example

- For some task (e.g. image compression) ...

	Processor A	Processor B
# Instructions	1 Million	1.5 Million
Average CPI	2.5	1
Clock rate $f$	2.5 GHz	2 GHz
Execution time	1 ms	0.75 ms

Processor B is faster for this task, despite executing more instructions and having a lower clock rate!

# Energy per Task

$$\frac{\text{Energy}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Energy}}{\text{Instruction}}$$

$$\frac{\text{Energy}}{\text{Program}} \propto \frac{\text{Instructions}}{\text{Program}} * C V^2$$

“Capacitance” depends on technology, processor features e.g. # of cores

Supply voltage, e.g. 1V

Want to reduce capacitance and voltage to reduce energy/task



# Energy Tradeoff Example

- “Next-generation” processor
  - C (Moore’s Law): -15 %
  - Supply voltage,  $V_{\text{sup}}$ : -15 %
  - Energy consumption:  $1 - (1-0.85)^3 = -39 \%$
- Significantly improved energy efficiency thanks to
  - Moore’s Law **AND**
  - Reduced supply voltage
- We will cover Moore’s Law later in the course

# Energy “Iron Law”

$$\text{Performance} = \text{Power} * \text{Energy Efficiency}$$

*(Tasks/Second) (Joules/Second) (Tasks/Joule)*

- Energy efficiency (e.g., instructions/Joule) is key metric in all computing devices
- For power-constrained systems (e.g., 20MW datacenter), need better energy efficiency to get more performance at same power
- For energy-constrained systems (e.g., 1W phone), need better energy efficiency to prolong battery life

# End of Scaling

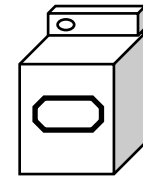
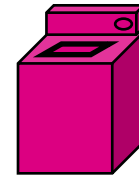
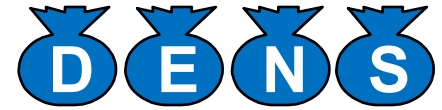
- In recent years, industry has not been able to reduce supply voltage much, as reducing it further would mean increasing “leakage power” where transistor switches don’t fully turn off (more like dimmer switch than on-off switch)
- Also, size of transistors and hence capacitance, not shrinking as much as before between transistor generations
- Power becomes a growing concern – the “power wall”
- Cost-effective air-cooled chip limit around ~150W

# Agenda

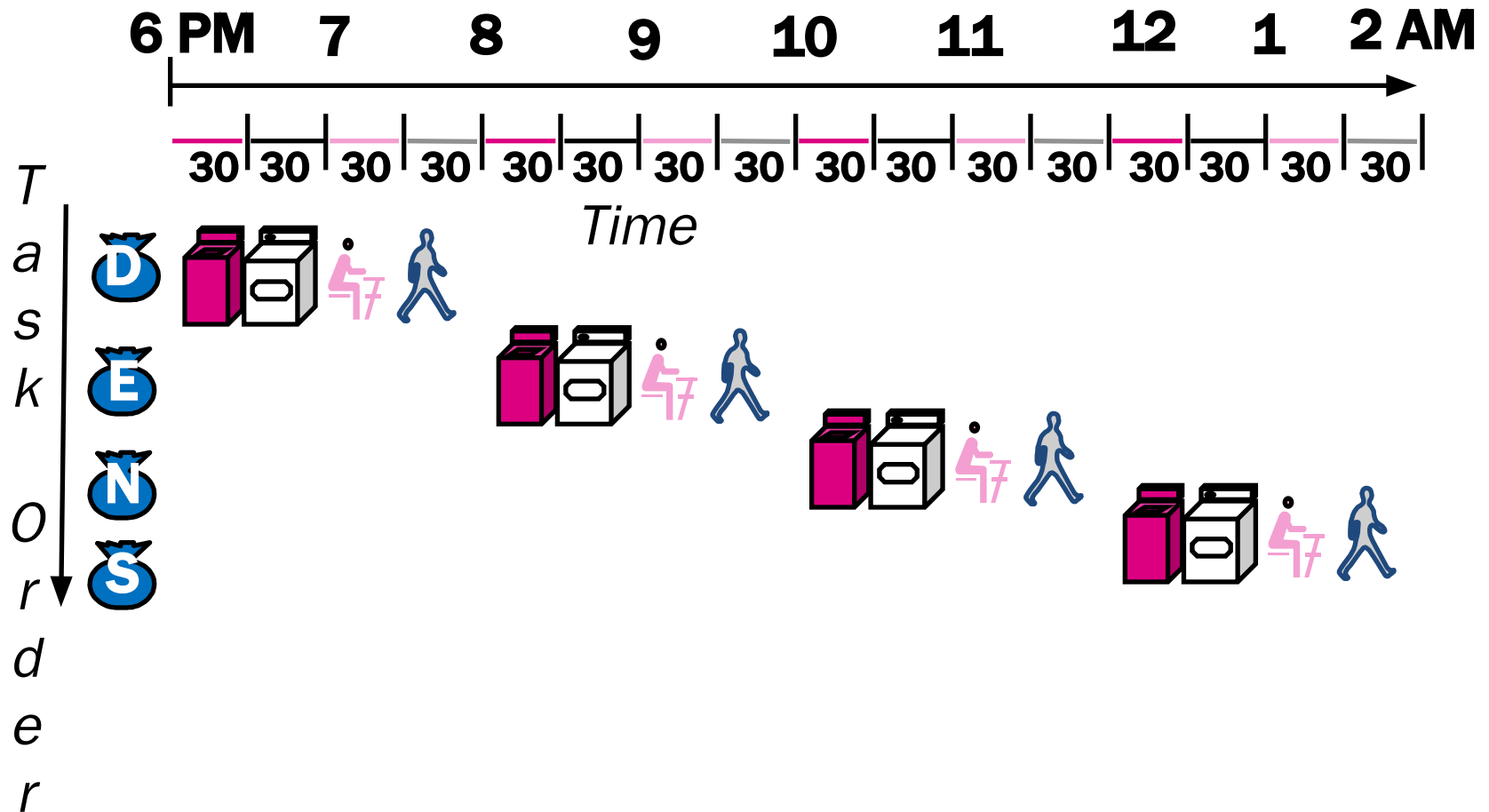
- Quick Datapath Review
- Control Implementation
- Administrivia
- Performance Analysis
- **Pipelined Execution**
- Pipelined Datapath

# Pipeline Analogy: Doing Laundry

- Damon, Emaan, Nick, and Steven each have one load of clothes to wash, dry, fold, and put away
  - Washer takes 30 minutes
  - Dryer takes 30 minutes
  - “Folder” takes 30 minutes
  - “Stasher” takes 30 minutes to put clothes into drawers

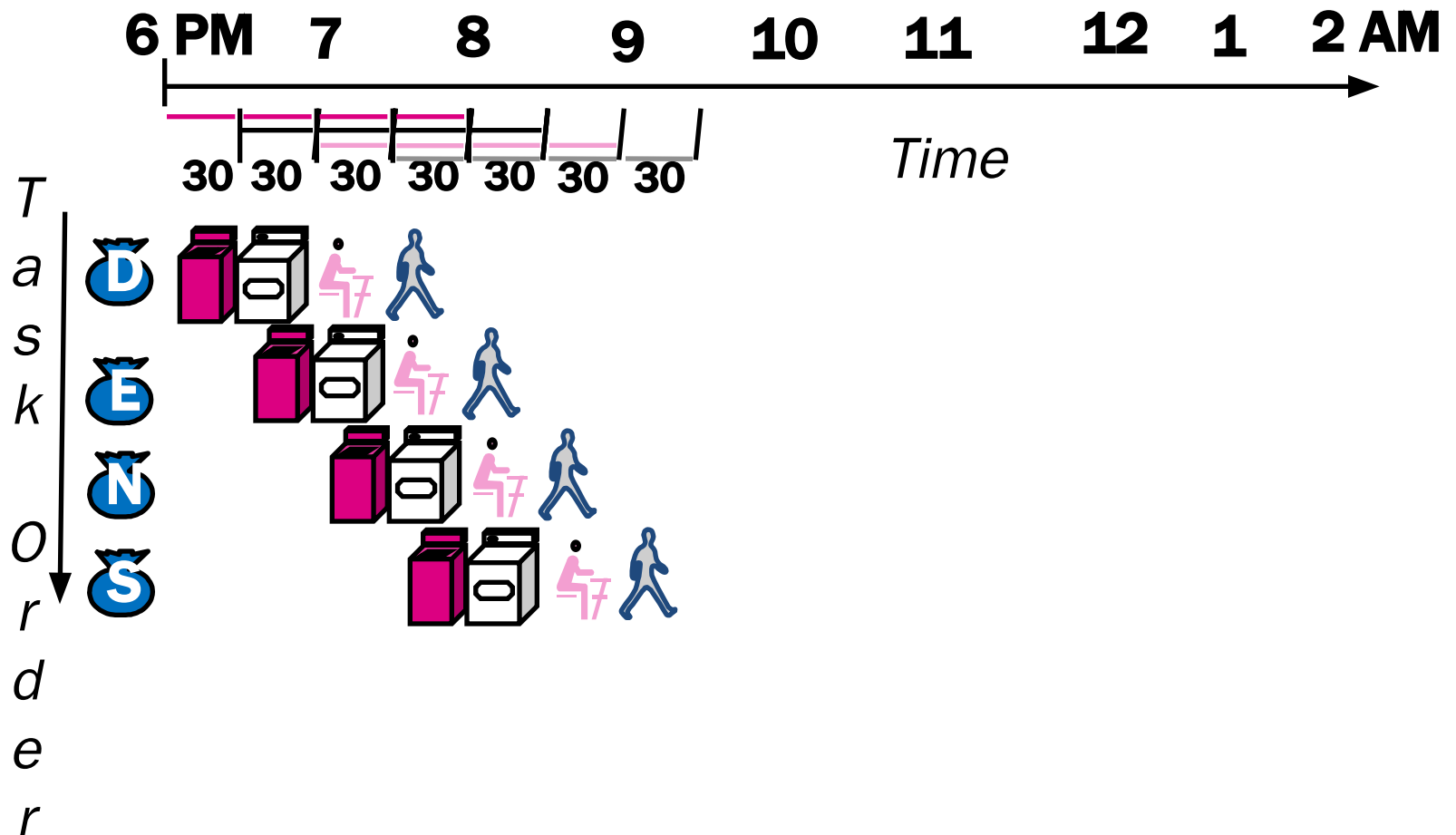


# Sequential Laundry



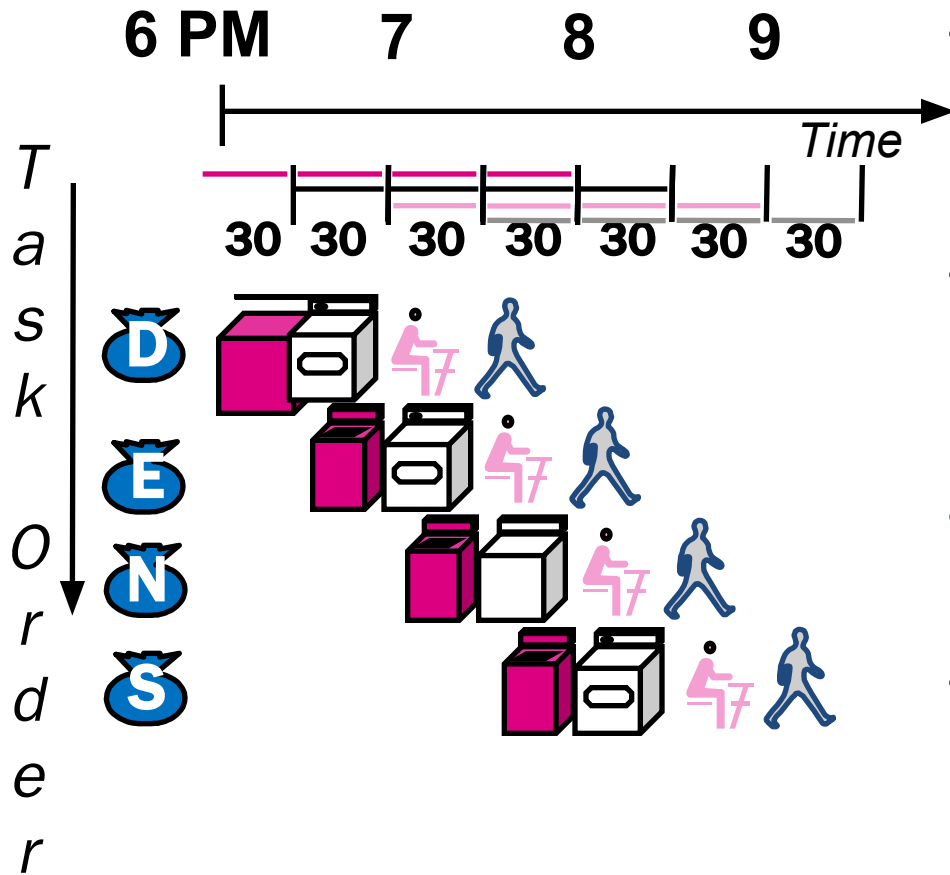
- Sequential laundry takes 8 hours for 4 loads

# Pipelined Laundry



- *Pipelined laundry takes 3.5 hours for 4 loads!*

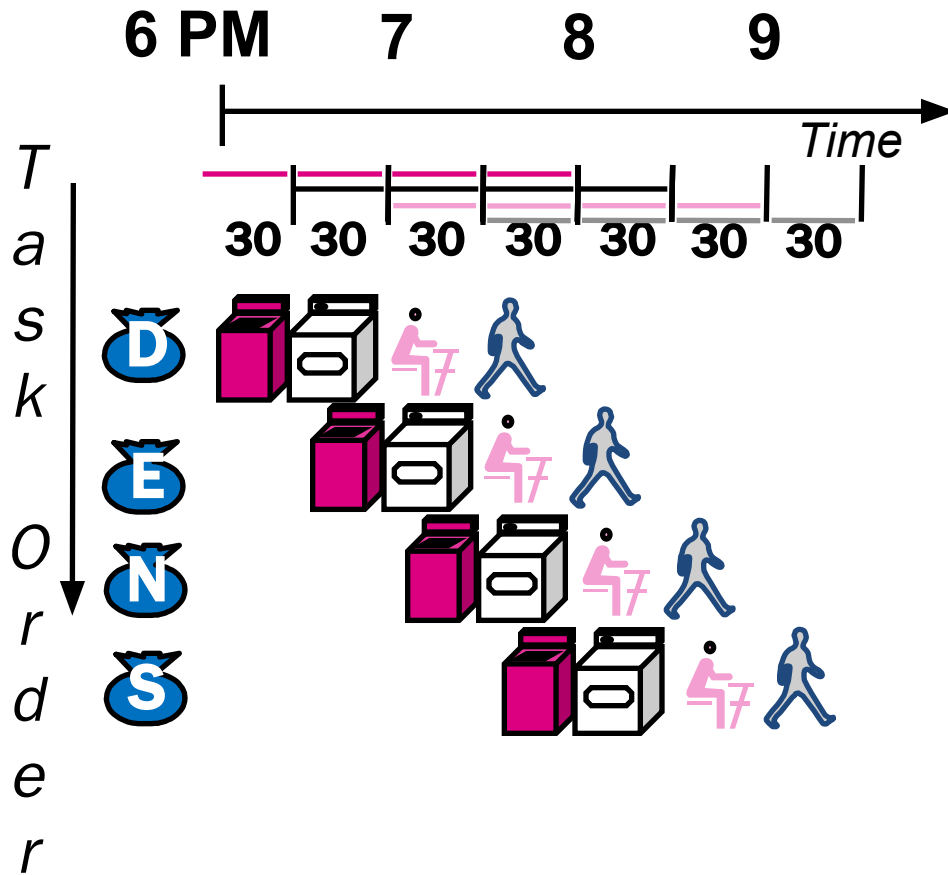
# Pipelining Lessons (1/2)



- Pipelining doesn't help *latency* of single task, just *throughput* of entire workload
- *Multiple* tasks operating simultaneously using different resources
- **Potential speedup = number of pipeline stages**
- Speedup reduced by time to *fill* and *drain* the pipeline: 8 hours/3.5 hours or 2.3X v. potential 4X in this example



# Pipelining Lessons (2/2)






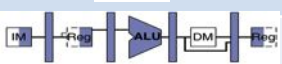


- Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?
  - Pipeline rate limited by *slowest* pipeline stage
  - Unbalanced lengths of pipeline stages reduces speedup

# Agenda

- Quick Datapath Review
- Control Implementation
- Administrivia
- Performance Analysis
- Pipelined Execution
- **Pipelined Datapath**

# Pipelining with RISC-V

Phase	Pictogram	$t_{step}$ Serial	$t_{cycle}$ Pipelined
Instruction Fetch		200 ps	200 ps
Reg Read		100 ps	200 ps
ALU		200 ps	200 ps
Memory		200 ps	200 ps
Register Write		100 ps	200 ps
$t_{instruction}$		<b>800 ps</b>	<b>1000 ps</b>

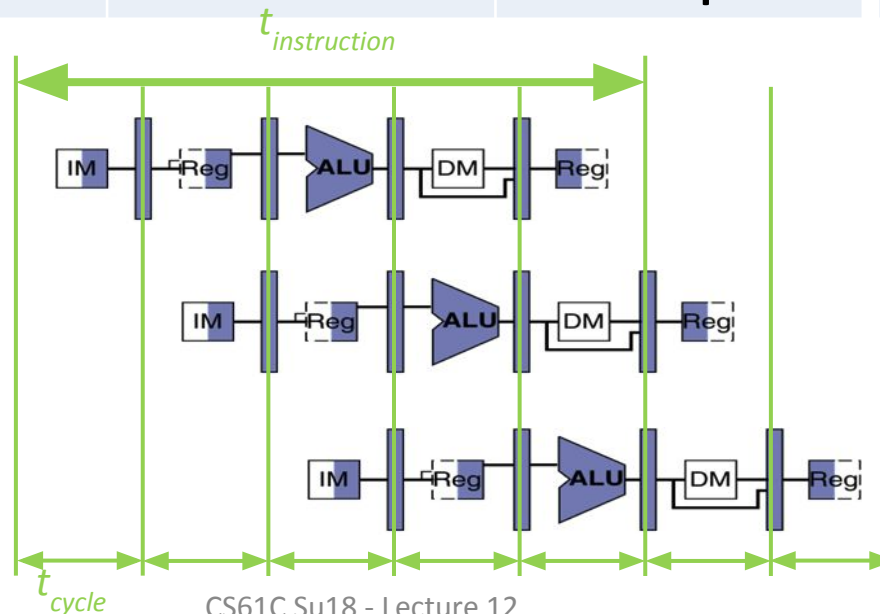
instruction sequence



add t0, t1, t2

or t3, t4, t5

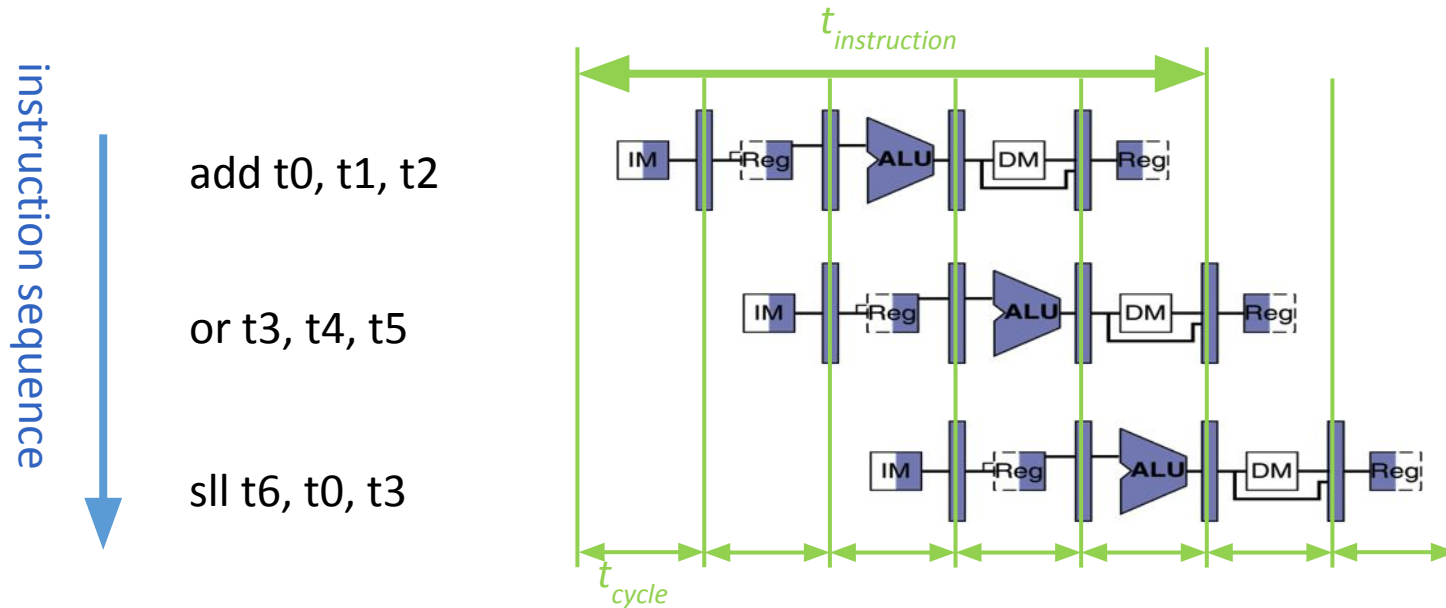
sll t6, t0, t3



# Pipeline Performance

- Use  $T_c$  (“time between completion of instructions”) to measure speedup
  - $T_{c,\text{pipelined}} \geq \frac{T_{c,\text{single-cycle}}}{\text{Number of stages}}$
  - Equality only achieved if stages are *balanced* (i.e. take the same amount of time)
- If not balanced, speedup is reduced
- Speedup due to increased *throughput*
  - *Latency* for each instruction does not decrease

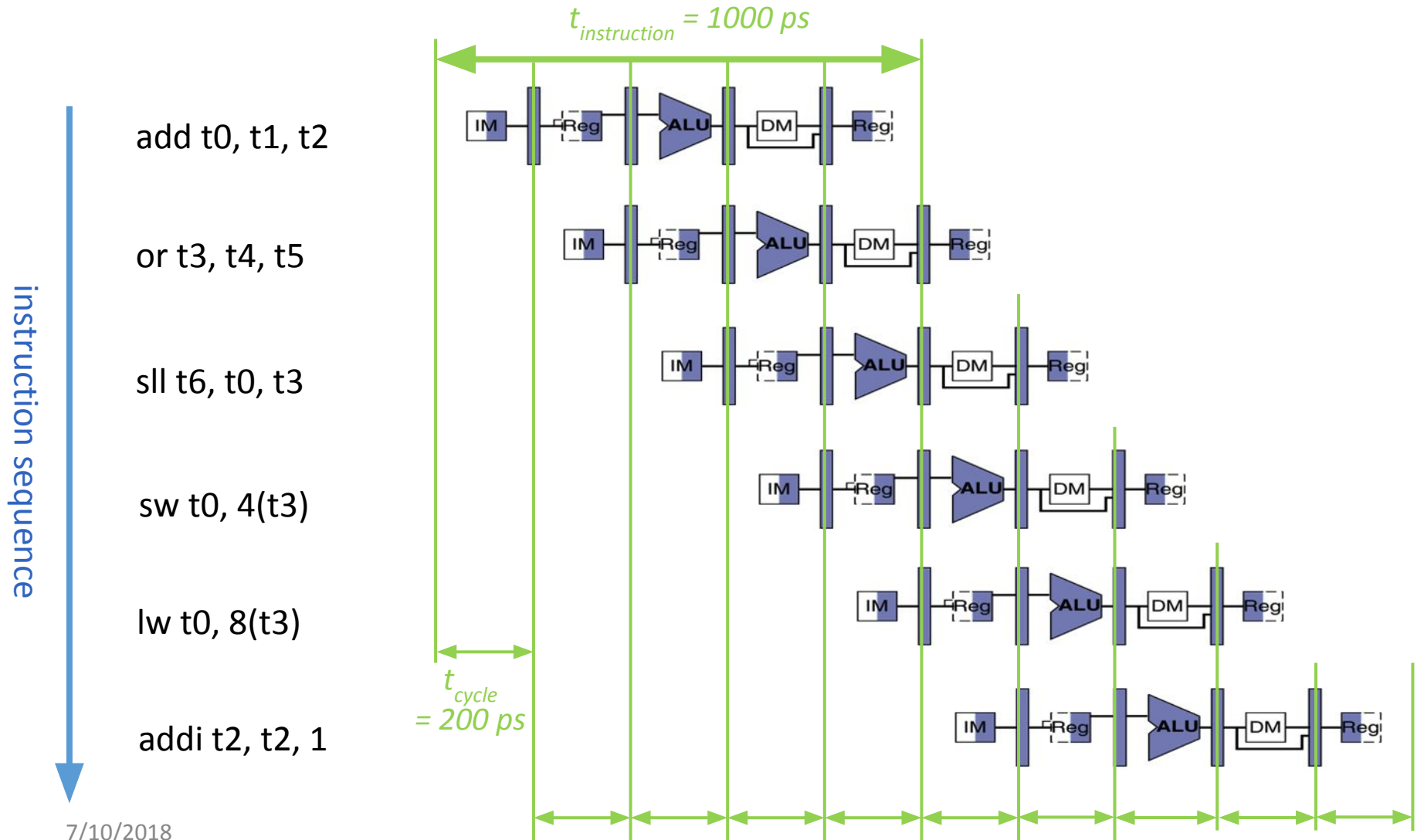
# Pipelining with RISC-V



	Single Cycle	Pipelining
Timing	$t_{step} = 100 \dots 200 \text{ ps}$	$t_{cycle} = 200 \text{ ps}$
	Register access only 100 ps	All cycles same length
Instruction time, $t_{instruction}$	$= t_{cycle} = 800 \text{ ps}$	<b>1000 ps</b>
Clock rate, $f_s$	$1/800 \text{ ps} = 1.25 \text{ GHz}$	$1/200 \text{ ps} = \mathbf{5 \text{ GHz}}$
Relative speed	1 x	<b>4 x</b>

# Sequential vs Simultaneous

*What happens sequentially, what happens simultaneously?*

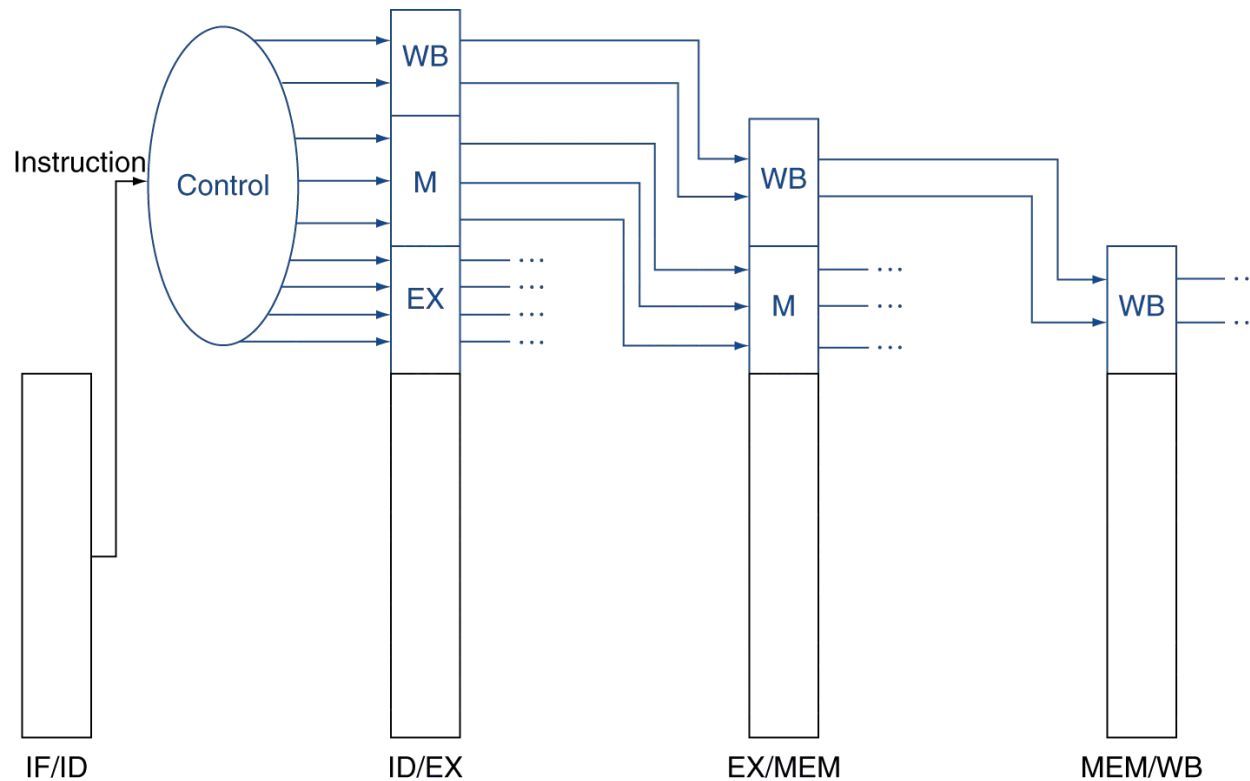


# Instruction Level Parallelism (ILP)

- Pipelining allows us to execute parts of multiple instructions at the same time using the same hardware!
  - This is known as *instruction level parallelism*
- Later: Other types of parallelism
  - DLP: same operation on lots of data (SIMD)
  - TLP: executing multiple threads “simultaneously” (OpenMP)

# Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation
  - Information is stored in pipeline registers for use by later stages





**Question:** Assume the stage times shown below. Suppose we *remove loads and stores* from our ISA. Consider going from a single-cycle implementation to a **4-stage** pipelined version.

Instr Fetch	Reg Read	ALU Op	Mem Access	Reg Write
200ps	100 ps	200ps	200ps	100 ps

- 1) The *latency* will be 1.25x slower.
- 2) The *throughput* will be 3x faster.

	1	2
(A)	F	F
(B)	F	T
(C)	T	F
(D)	T	T

**Question:** Assume the stage times shown below. Suppose we *remove loads and stores* from our ISA. Consider going from a single-cycle implementation to a **4-stage** pipelined version.

Instr Fetch	Reg Read	ALU Op	Mem Access	Reg Write
200ps	100 ps	200ps	200ps	100 ps

- 1) The *latency* will be 1.25x slower.
- 2) The *throughput* will be 3x faster.

	1	2
(A)	F	F
(B)	F	T
(C)	T	F
(D)	T	T

**No mem access**

throughput:

$$1/(IF+ID+EX+WB) = 1/600 \rightarrow$$

$$4/(4*\max\_stage) = 1/200$$

$$1/200*600/1 = 3x \text{ faster}$$

**Question:** Assume the stage times shown below. Suppose we *remove loads and stores* from our ISA. Consider going from a single-cycle implementation to a **4-stage** pipelined version.

Instr Fetch	Reg Read	ALU Op	Mem Access	Reg Write
200ps	100 ps	200ps	200ps	100 ps

- 1) The *latency* will be 1.25x slower.
- 2) The *throughput* will be 3x faster.

	1	2
(A)	F	F
(B)	F	T
(C)	T	F
(D)	T	T

**No mem access**

latency:

$$IF+ID+EX+WB = 600 \rightarrow$$

$$4 * \max\_stage = 800$$

$$800/600 = 1.33x \text{ slower!}$$

**Question:** Assume the stage times shown below. Suppose we *remove loads and stores* from our ISA. Consider going from a single-cycle implementation to a **4-stage** pipelined version.

Instr Fetch	Reg Read	ALU Op	Mem Access	Reg Write
200ps	100 ps	200ps	200ps	100 ps

- 1) The *latency* will be 1.25x slower.
- 2) The *throughput* will be 3x faster.

	1	2
(A)	F	F
(B)	F	T
(C)	T	F
(D)	T	T

# Summary

- Implementing controller for your datapath
  - Take decoded signals from instruction and generate control signals
- Pipelining improves performance by exploiting Instruction Level Parallelism
  - 5-stage pipeline for RV32I: IF, ID, EX, MEM, WB
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
  - Be careful of signal passing (*more on this next lecture*)