

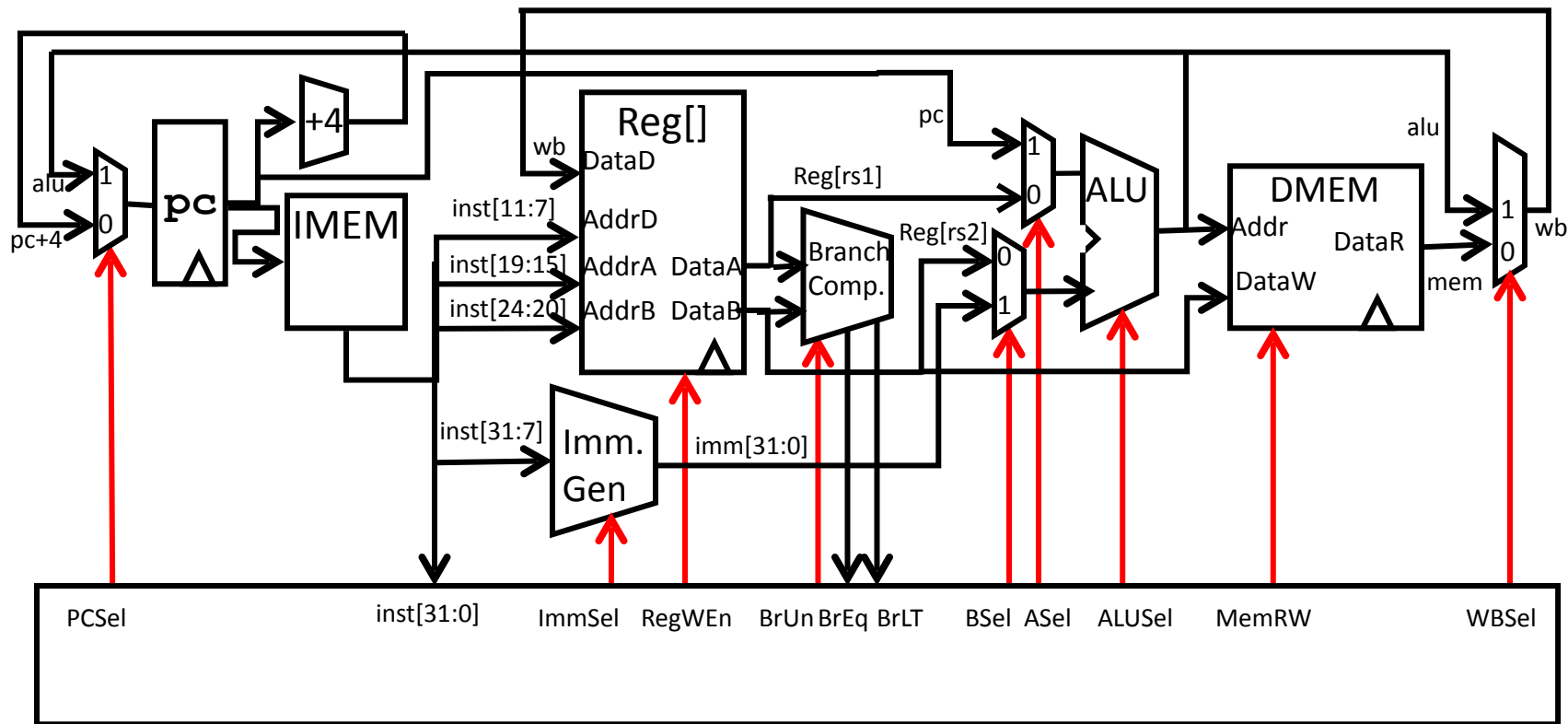
CS 61C:
Great Ideas in Computer Architecture
Lecture 12: *Control & Operating Speed*

John Wawrzynek & Nick Weaver
<http://inst.eecs.berkeley.edu/~cs61c/sp18>

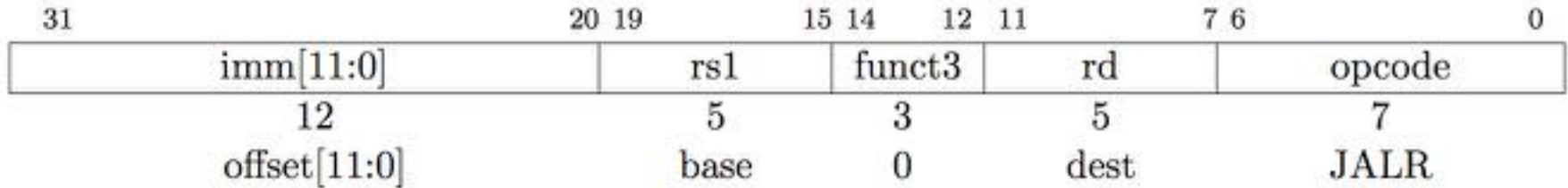
Agenda

- **Finish Single-Cycle RISC-V Datapath**
- Controller
- Instruction Timing
- Performance Measures
- Introduction to Pipelining
- Pipelined RISC-V Datapath
- And in Conclusion, ...

Recap: Adding branches to datapath

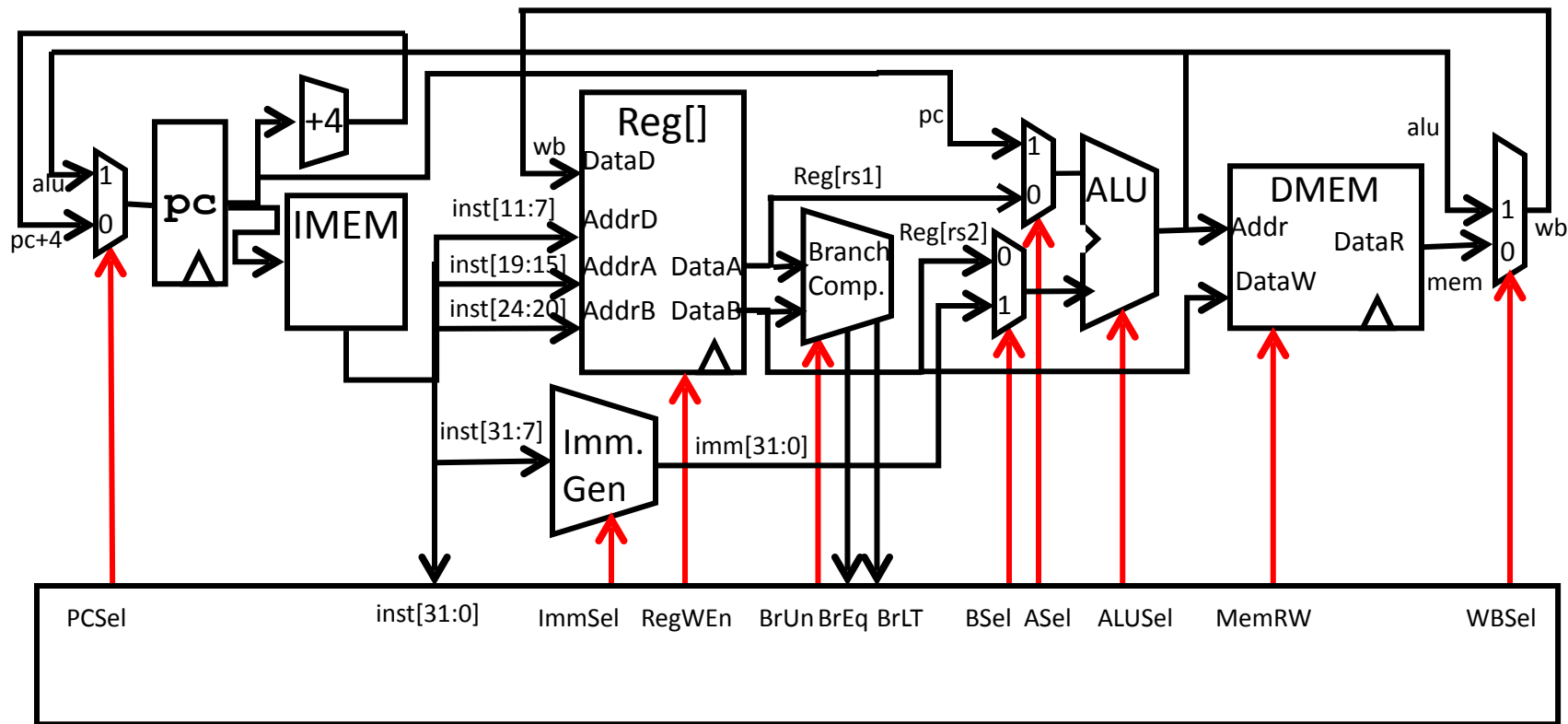


Implementing **JALR** Instruction (I-Format)

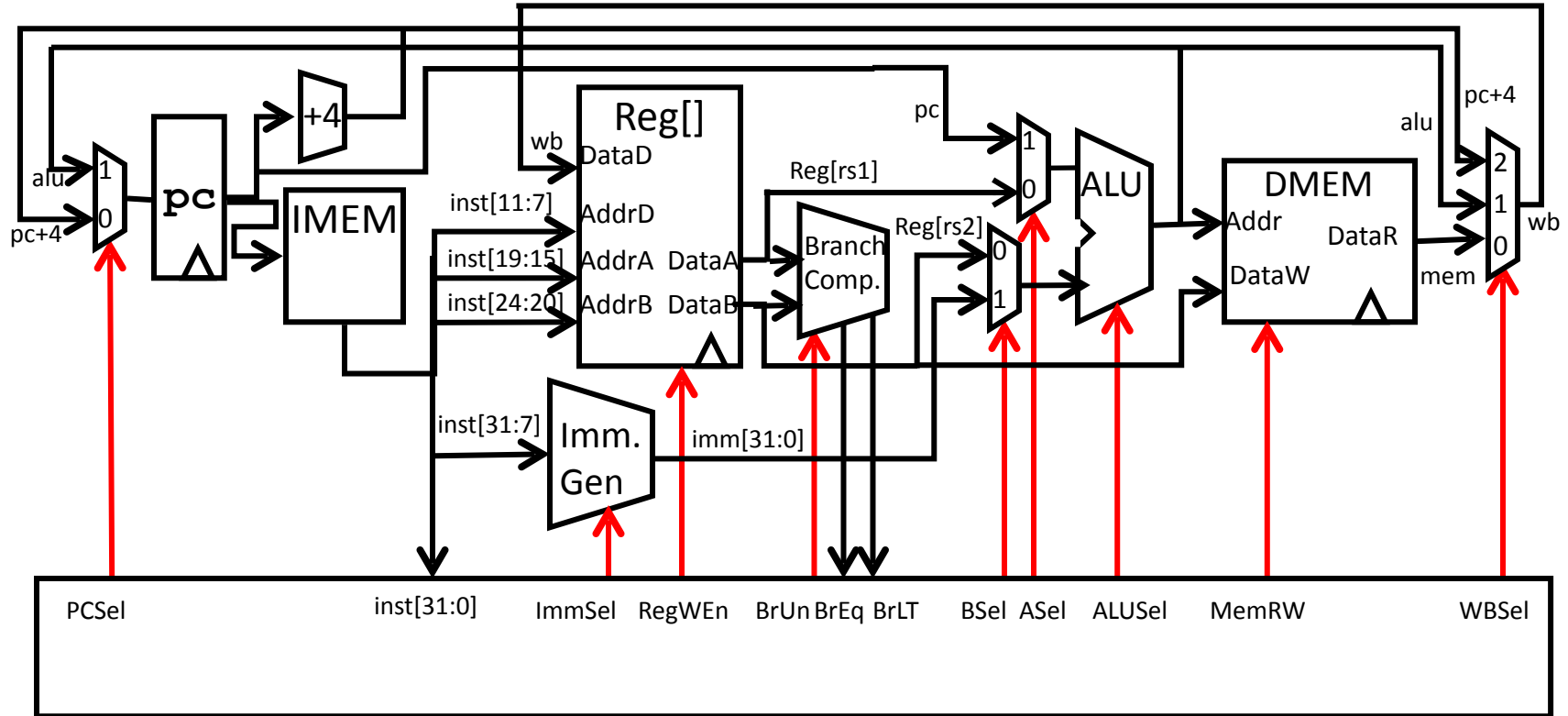


- JALR rd, rs, immediate
 - Writes PC+4 to Reg[rd] (return address)
 - Sets PC = Reg[rs1] + immediate
 - Uses same immediates as arithmetic and loads
 - **no** multiplication by 2 bytes

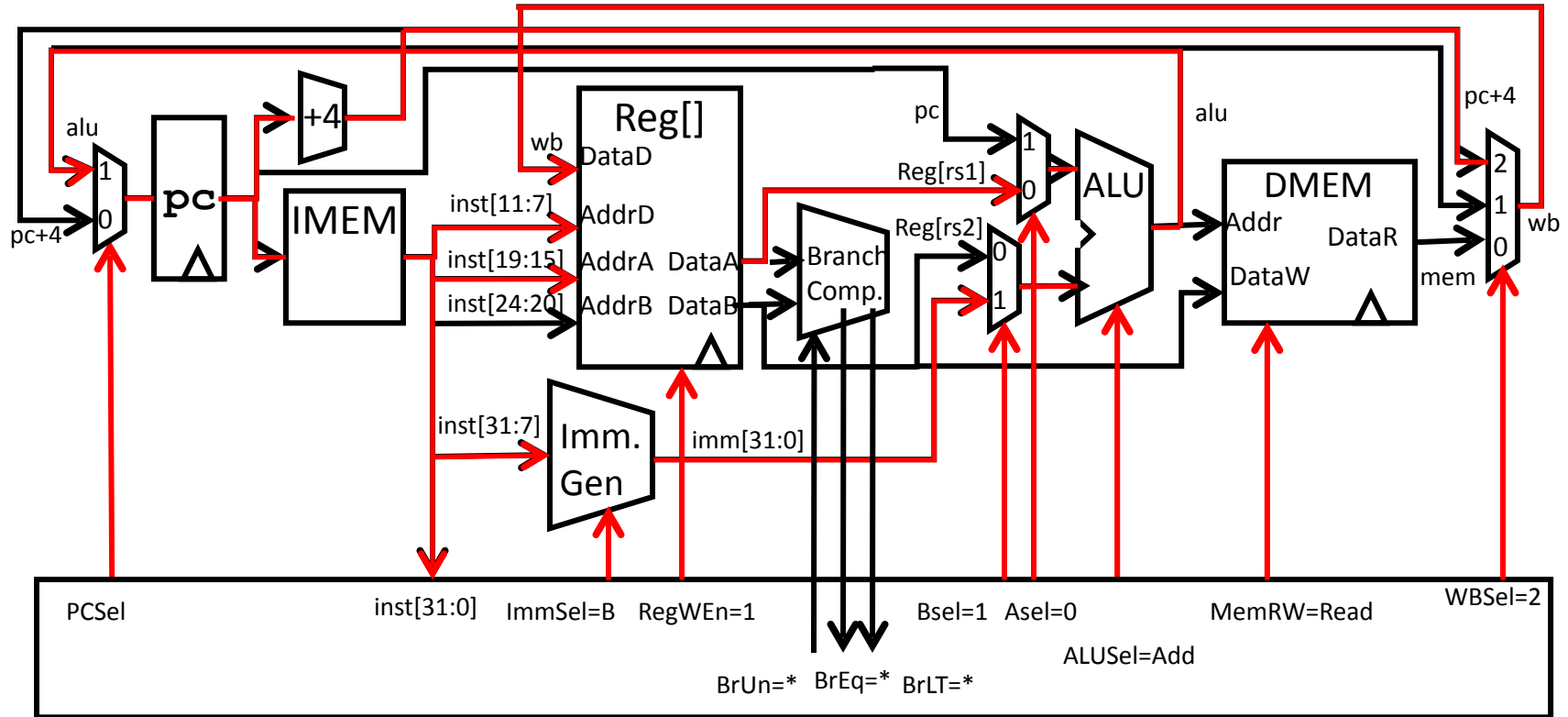
Datapath with Branches



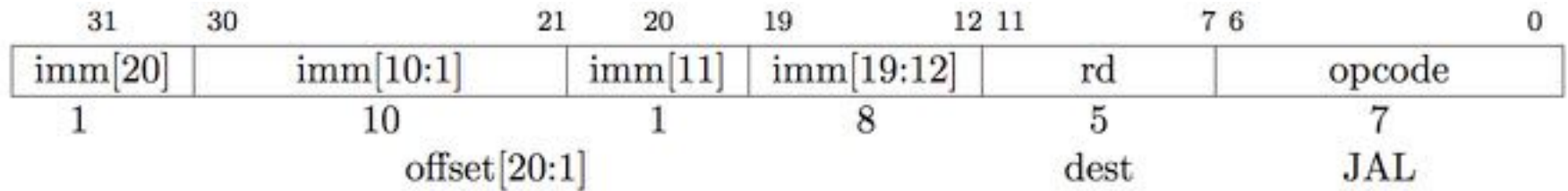
Adding jalr to datapath



Adding jalr to datapath

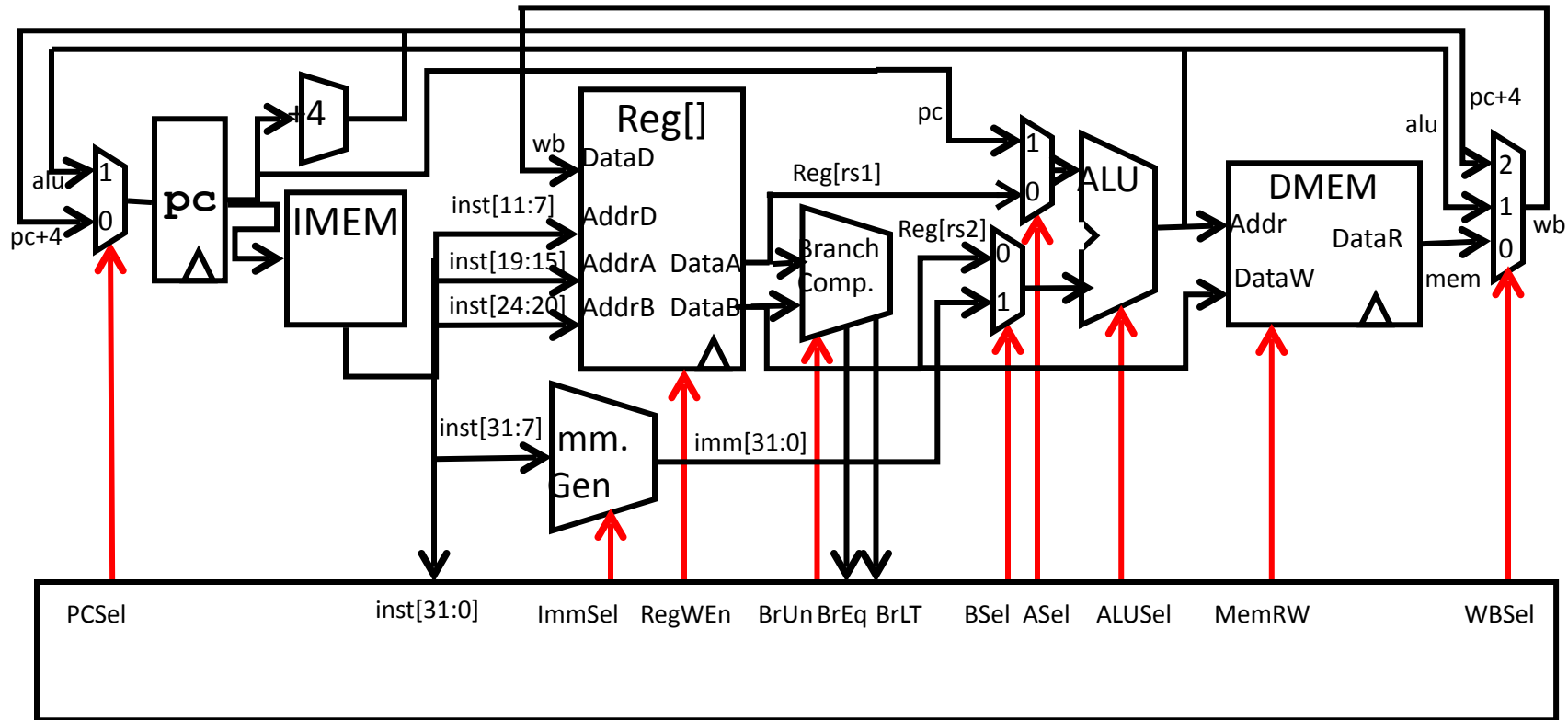


Implementing jal Instruction

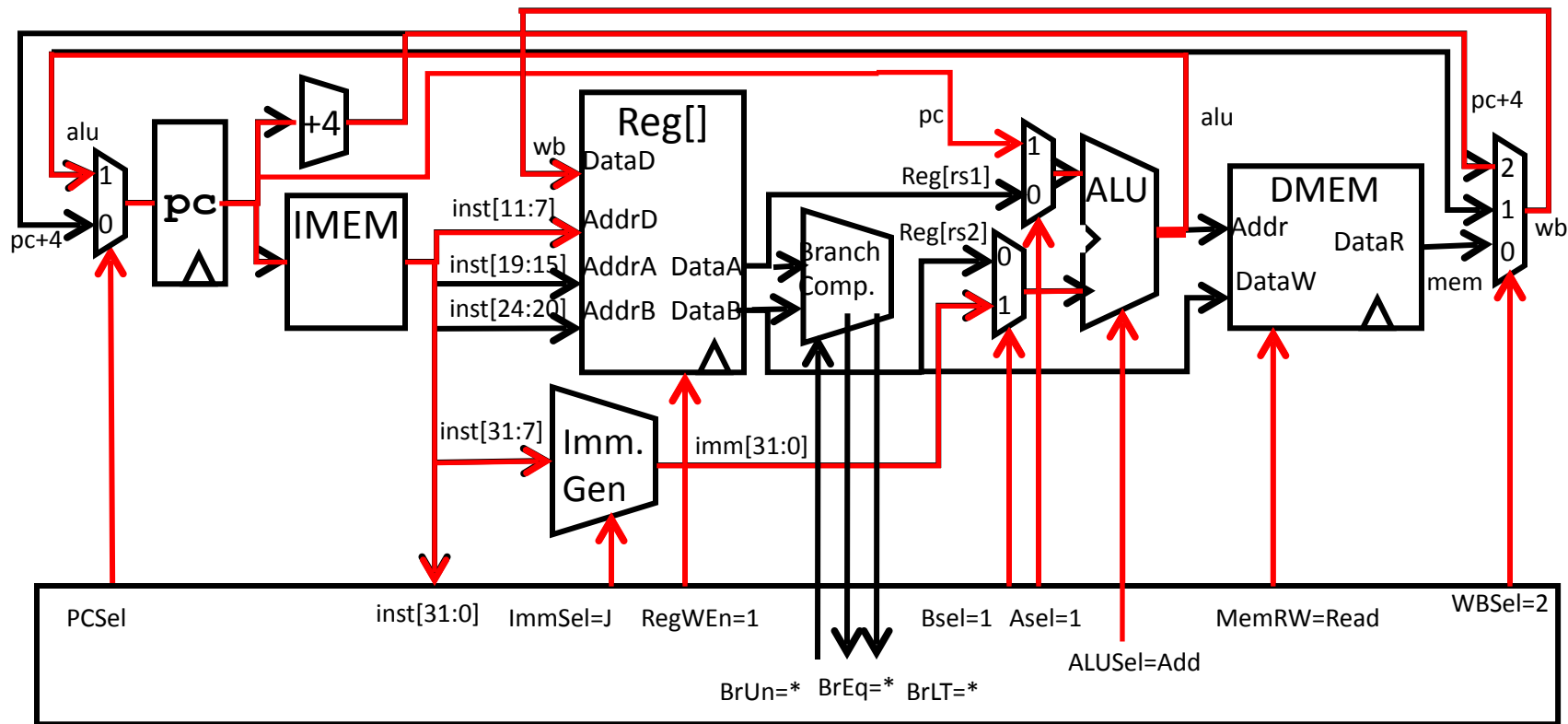


- JAL saves PC+4 in Reg[rd] (the return address)
- Set PC = PC + offset (PC-relative jump)
- Target somewhere within $\pm 2^{19}$ locations, 2 bytes apart
 - $\pm 2^{18}$ 32-bit instructions
- Immediate encoding optimized similarly to branch instruction to reduce hardware cost

Adding jal to datapath



Adding jal to datapath



Recap: Complete RV32I ISA

imm[31:12]					rd	0110111
imm[31:12]					rd	0010111
imm[30:10:11:19:12]					rd	1101111
imm[11:0]					rd	1100111
imm[12:10:5]	rs2	rs1	000	imm[4:1:11]	1100011	
imm[12:10:5]	rs2	rs1	001	imm[4:1:11]	1100011	
imm[12:10:5]	rs2	rs1	100	imm[4:1:11]	1100011	
imm[12:10:5]	rs2	rs1	101	imm[4:1:11]	1100011	
imm[12:10:5]	rs2	rs1	110	imm[4:1:11]	1100011	
imm[12:10:5]	rs2	rs1	111	imm[4:1:11]	1100011	
imm[11:0]					rd	0000011
imm[11:0]					rd	0000011
imm[11:0]					rd	0000011
imm[11:0]					rd	0000011
imm[11:0]					rd	0000011
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	
imm[11:0]					rd	0010011
imm[11:0]					rd	0010011
imm[11:0]					rd	0010011
imm[11:0]					rd	0010011
imm[11:0]					rd	0010011
imm[11:0]					rd	0010011
imm[11:0]					rd	0010011

LUI
AUIPC
JAL
JALR
BEQ
BNE
BLT
BGE
BLTU
BGEU
LB
LH
LW
LBU
LHU
SB
SH
SW
ADDI
SLTI
SLTIU
XORI
ORI
ANDI

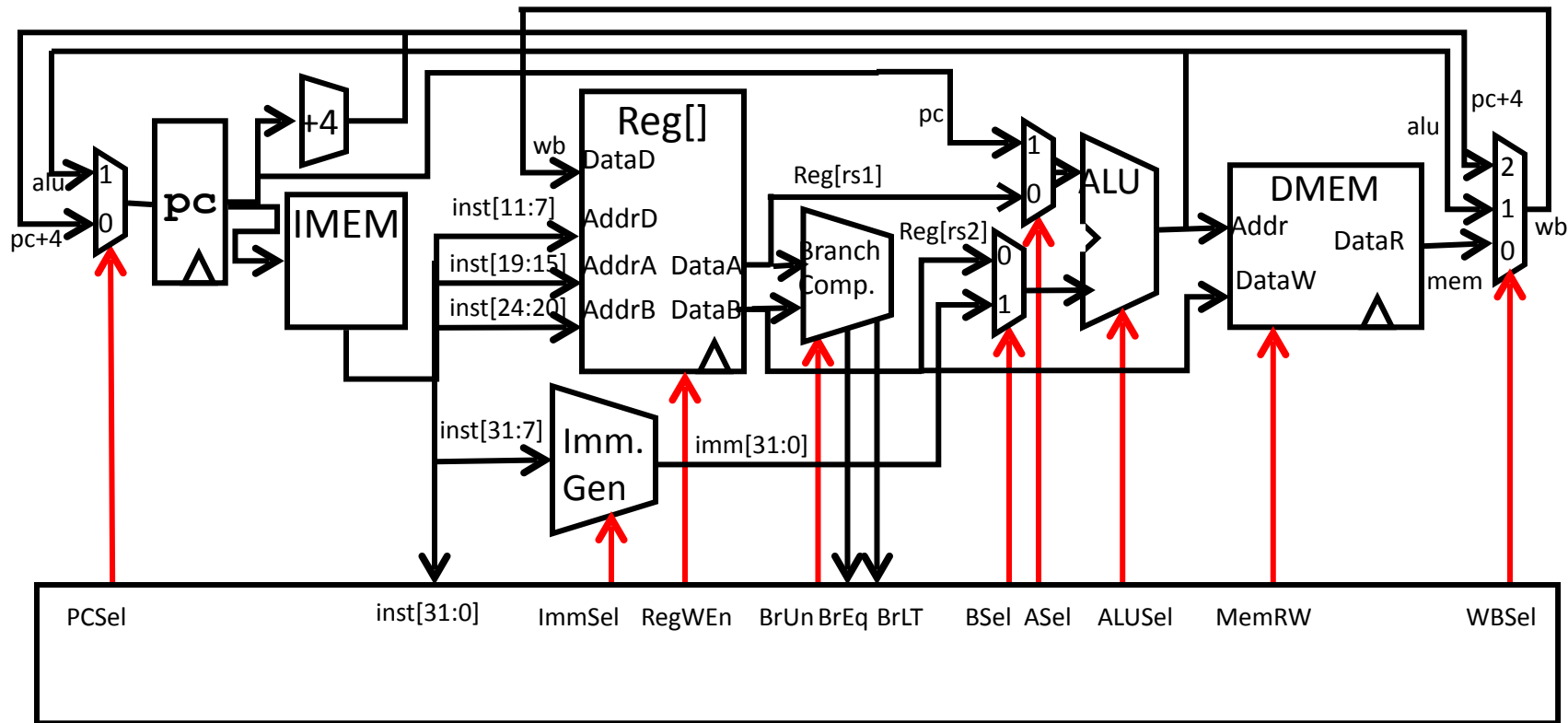
000000	shamt	rs1	001	rd	0010011	SLLI
000000	shamt	rs1	101	rd	0010011	SRLI
010000	shamt	rs1	101	rd	0010011	SRAI
000000	rs2	rs1	000	rd	0110011	ADD
010000	rs2	rs1	000	rd	0110011	SUB
000000	rs2	rs1	001	rd	0110011	SLL
000000	rs2	rs1	010	rd	0110011	SLT
000000	rs2	rs1	011	rd	0110011	SLTU
000000	rs2	rs1	100	rd	0110011	XOR
000000	rs2	rs1	101	rd	0110011	SRL
010000	rs2	rs1	101	rd	0110011	SRA
000000	rs2	rs1	110	rd	0110011	OR
000000	rs2	rs1	111	rd	0110011	AND
0000	pred	succ	0000	000	0000	FENCE
0000	0000	0000	0000	001	0000	FENCE.I
000000000000			0000	000	0000	ECALL
000000000001			0000	000	0000	EBREAK
csr	rs1	rd	001	rd	1110011	CSRWR
csr	rs1	rd	010	rd	1110011	CSRRS
csr	rs1	rd	011	rd	1110011	CSRWC
csr	zimm	rd	101	rd	1110011	CSRRIWI
csr	zimm	rd	110	rd	1110011	CSRRSI
csr	zimm	rd	111	rd	1110011	CSRRCI

Not in CS61C

RV32I has 47 instructions total
37 instructions covered in CS61C

*Other instructions (ex: lui, auipc)
can be implemented with no new additions to
the datapath and only by changing control*

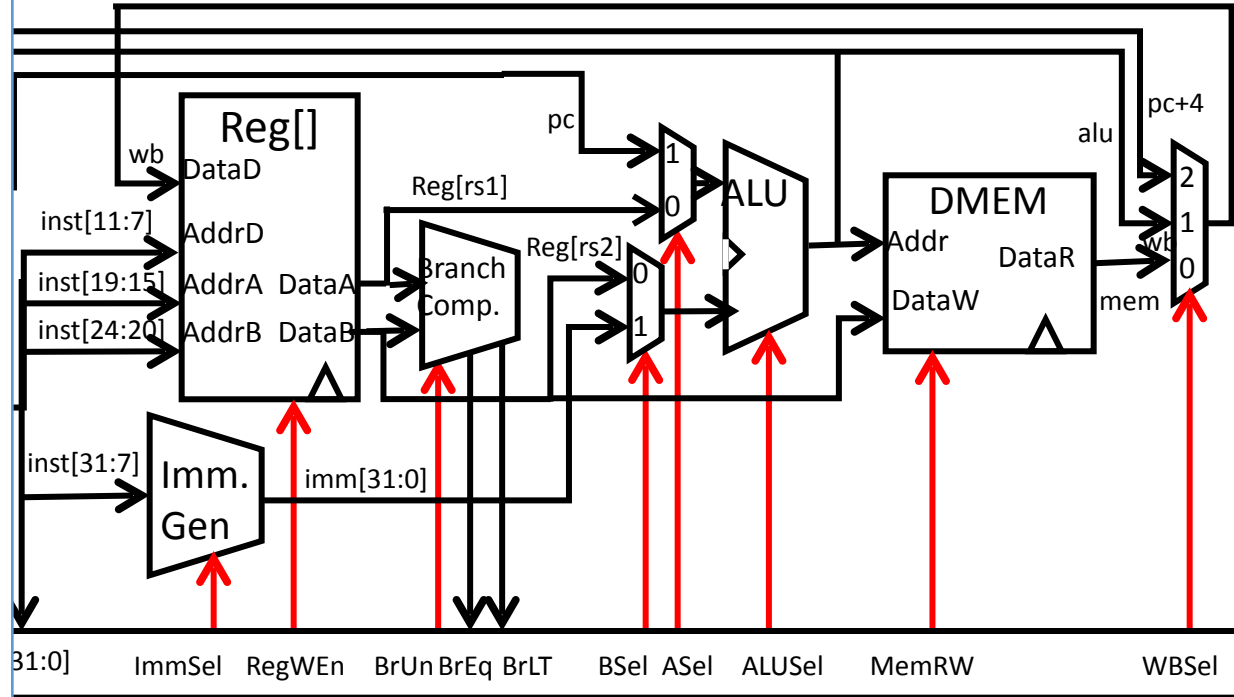
Single-Cycle RISC-V RV32I Datapath



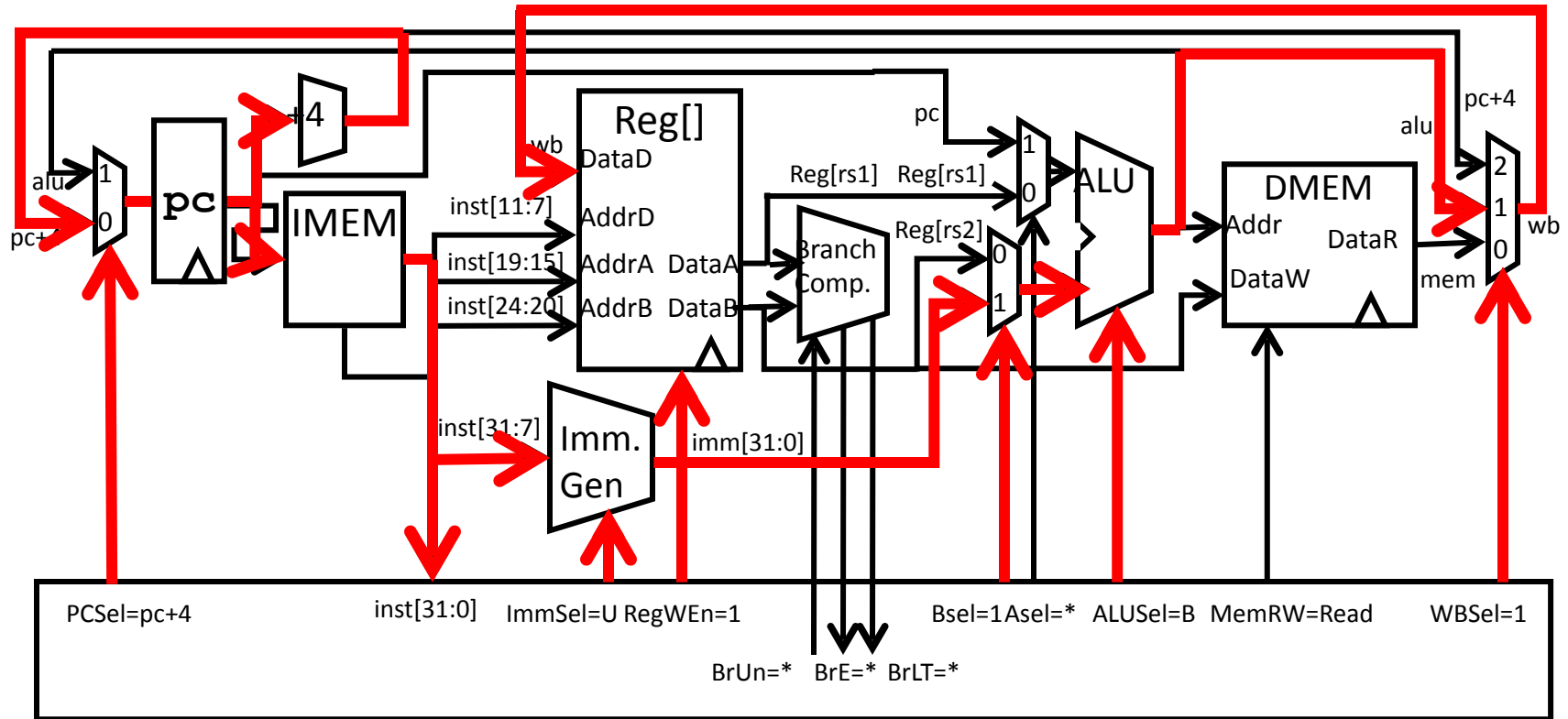
Clicker Question

What are proper control signals for **lui** instruction?

- A:** BSel=0, ASel=0, WBSel=0
- B:** BSel=0, ASel=0, WBSel=1
- C:** BSel=0, ASel=1, WBSel=1
- D:** BSel=1, ASel=1, WBSel=1



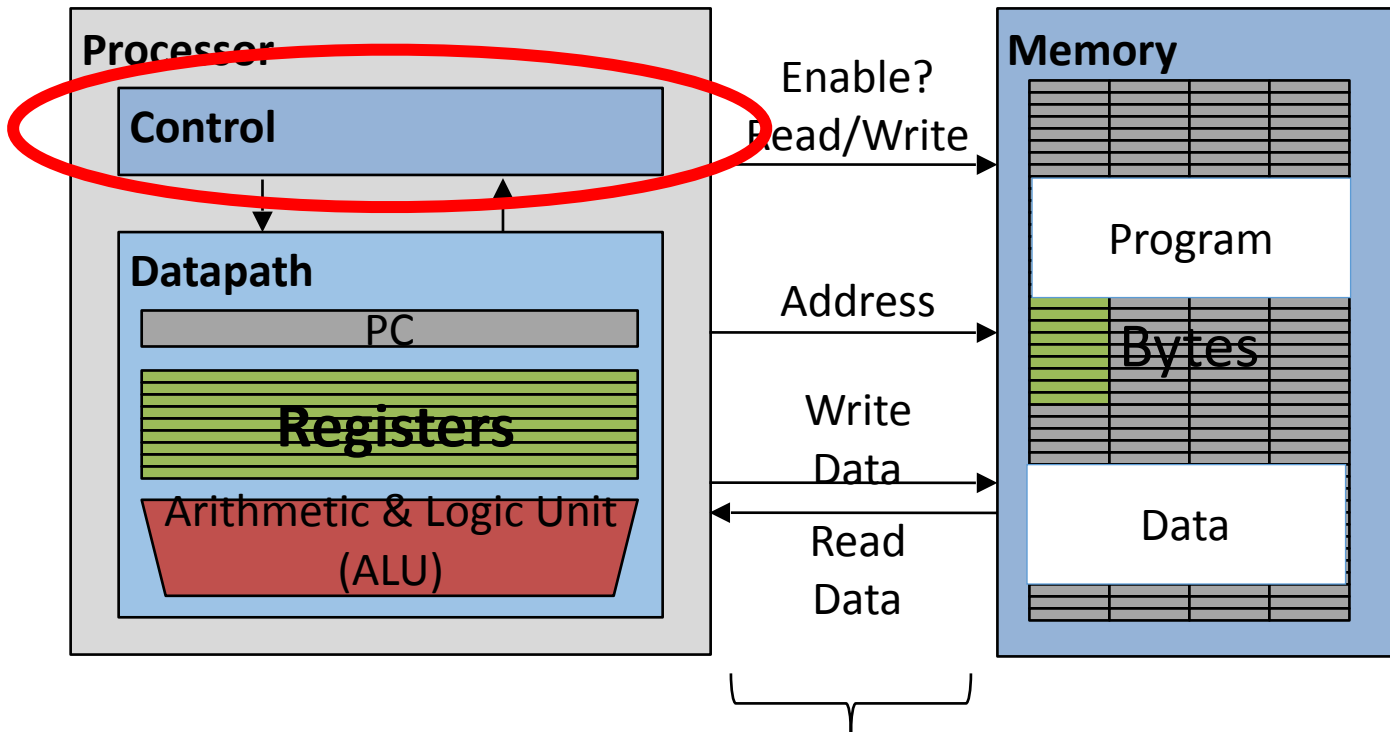
Implementing lui



Agenda

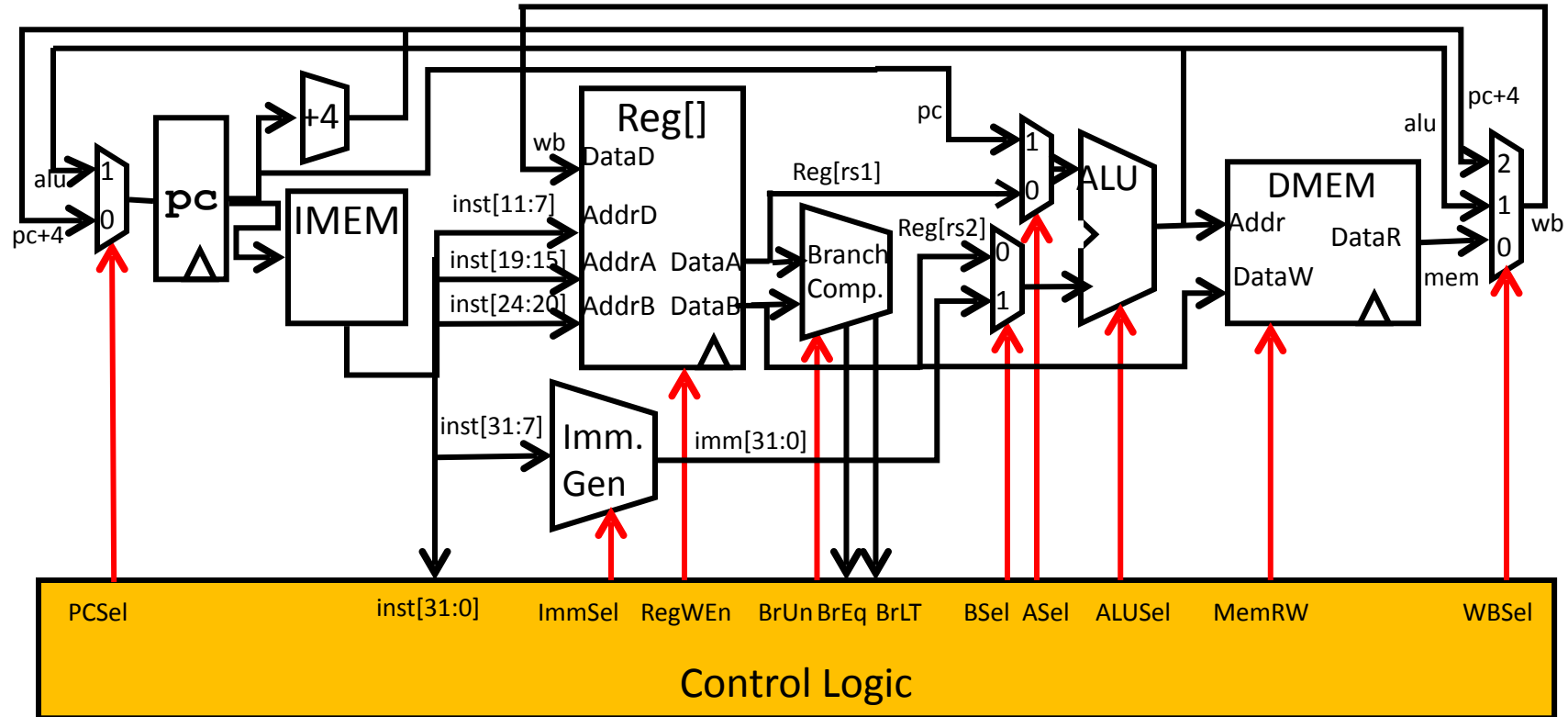
- Finish Single-Cycle RISC-V Datapath
- **Controller**
- Instruction Timing
- Performance Measures
- Introduction to Pipelining
- Pipelined RISC-V Datapath
- And in Conclusion, ...

Processor



Processor-Memory Interface

Single-Cycle RISC-V RV32I Datapath



Control Logic Truth Table (incomplete)

Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
sub	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	(Op)	Read	1	ALU
addi	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
lw	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
sw	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
beq	0	*	+4	B	*	PC	Imm	Add	Read	0	*
beq	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	1	*	+4	B	*	PC	Imm	Add	Read	0	*
blt	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
bltu	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
jalr	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
jal	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
auipc	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

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

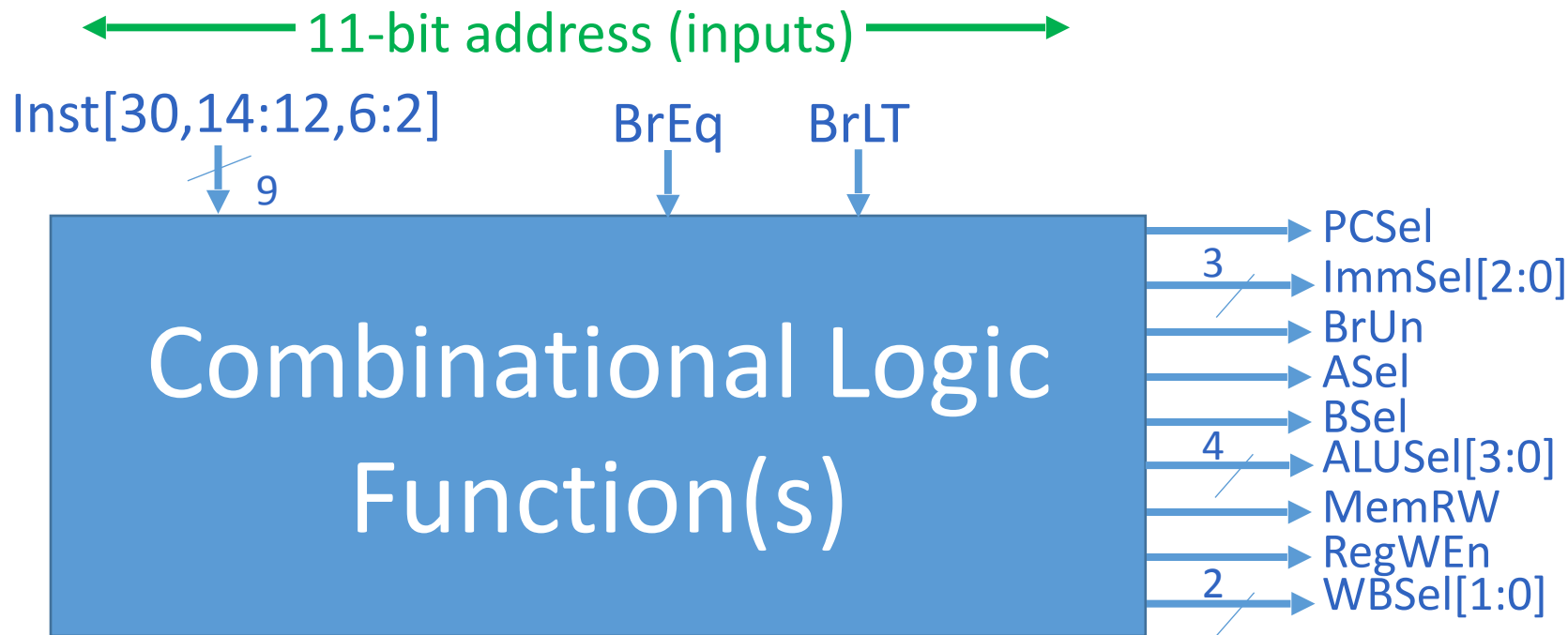
imm[31:12]					rd	0110111	LUI	
imm[31:12]					rd	0010111	AUIPC	
imm[20:10:11:19:12]					rd	1100111	JAL	
imm[11:0]				rs1	000	rd	1100111	JALR
imm[12:10:5]		rs2	rs1	000	imm[4:1:11]	1100011	BEQ	
imm[12:10:5]		rs2	rs1	001	imm[4:1:11]	1100011	BNE	
imm[12:10:5]		rs2	rs1	100	imm[4:1:11]	1100011	BLT	
imm[12:10:5]		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	BGTU	
imm[11:0]				rs1	000	rd	0000011	LB
imm[11:0]				rs1	001	rd	0000011	LH
imm[11:0]				rs1	010	rd	0000011	LW
imm[11:0]				rs1	100	rd	0000011	LBU
imm[11:0]				rs1	101	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]				rs1	000	rd	0010011	ADDI
imm[11:0]				rs1	010	rd	0010011	SLTI
imm[11:0]				rs1	011	rd	0010011	SLTIU
imm[11:0]				rs1	100	rd	0010011	XORI
imm[11:0]				rs1	110	rd	0010011	ORI
imm[11:0]				rs1	111	rd	0010011	ANDI

inst[30] inst[14:12] inst[6:2]

000000	shamt	rs1	001	rd	0010011	SLLI	
000000	shamt	rs1	101	rd	0010011	SRLI	
010000	shamt	rs1	101	rd	0010011	SRAI	
000000	rs2	rs1	000	rd	0110011	ADD	
010000	rs2	rs1	000	rd	0110011	SUB	
000000	rs2	rs1	001	rd	0110011	SLL	
000000	rs2	rs1	010	rd	0110011	SLT	
000000	rs2	rs1	011	rd	0110011	SLTU	
000000	rs2	rs1	100	rd	0110011	XOR	
000000	rs2	rs1	101	rd	0110011	SRL	
010000	rs2	rs1	101	rd	0110011	SRA	
000000	rs2	rs1	110	rd	0110011	OR	
000000	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
000000000000			00000	000	00000	1110011	ERRRACK
csr		rs1	001	rd	1110011	CSRWR	
csr		rs1	010	rd	1110011	CSRRS	
csr		rs1	011	rd	1110011	CSRRC	
csr		zimm	101	rd	1110011	CSRRIWI	
csr		zimm	110	rd	1110011	CSRRI	
csr		zimm	111	rd	1110011	CSRRCI	

Not in CS61C

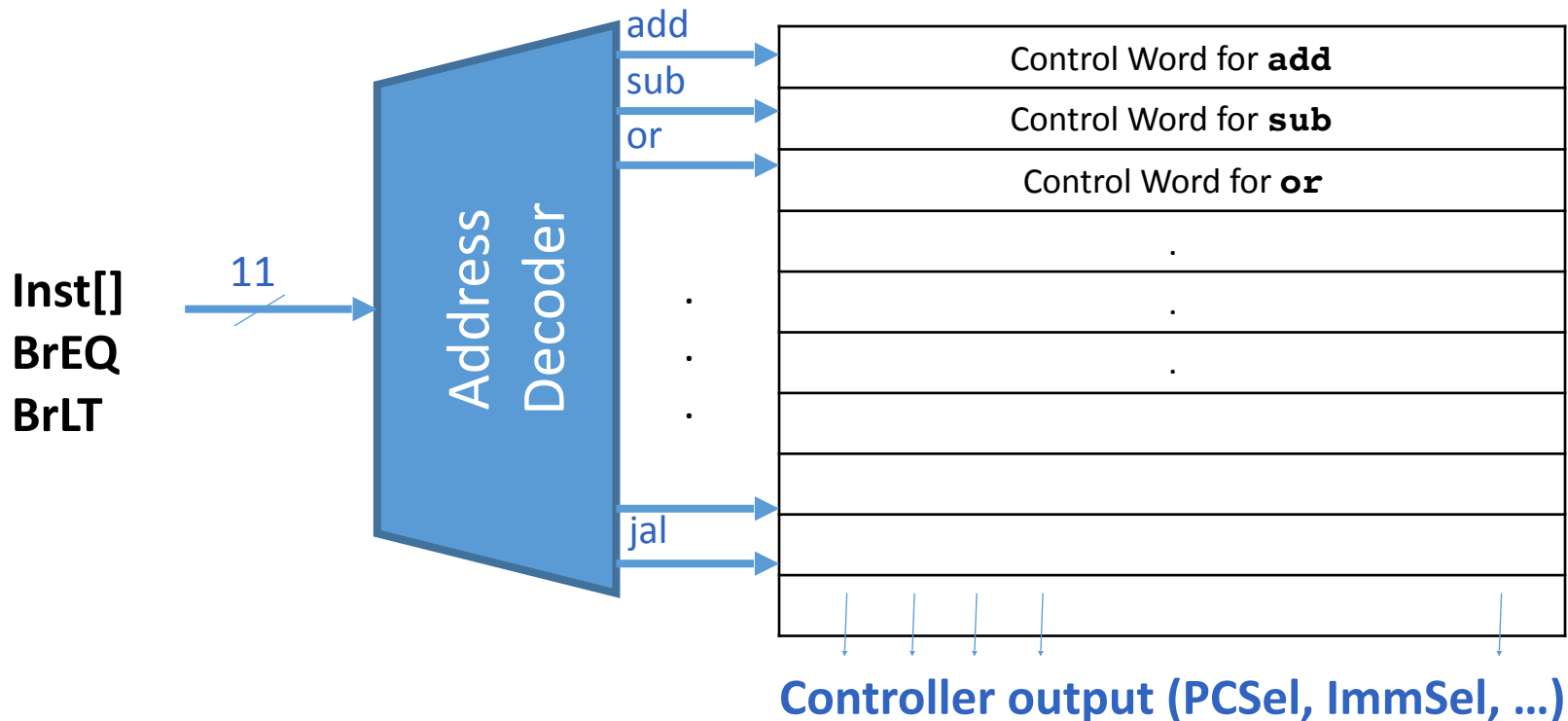
Control Block Design



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

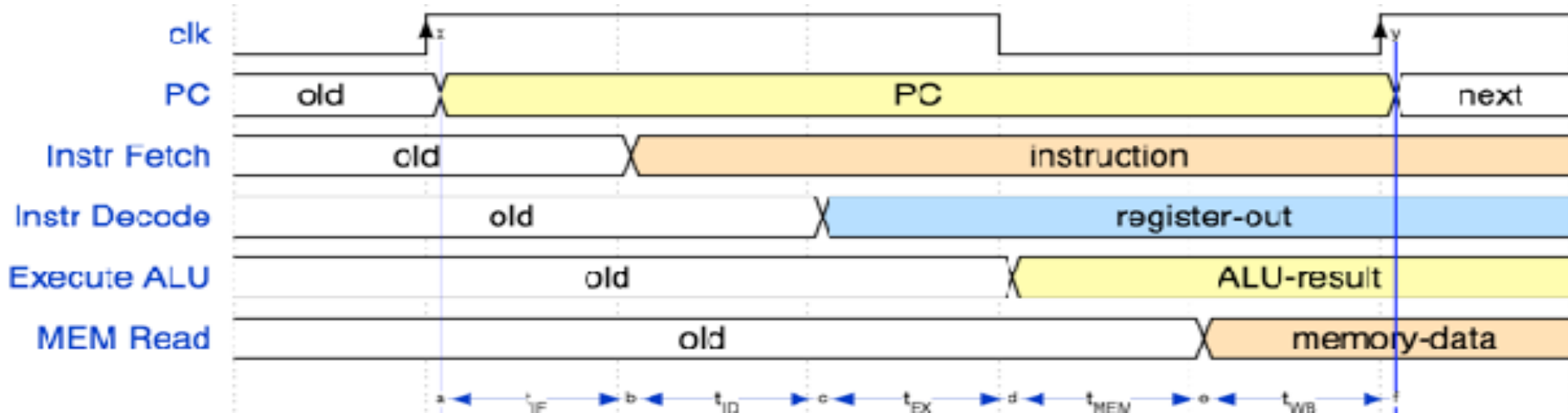
ROM Controller Implementation



Agenda

- Finish Single-Cycle RISC-V Datapath
- Controller
- **Instruction Timing**
- Performance Measures
- Introduction to Pipelining
- Pipelined RISC-V Datapath
- And in Conclusion, ...

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!

Agenda

- Finish Single-Cycle RISC-V Datapath
- Controller
- Instruction Timing
- **Performance Measures**
- Introduction to Pipelining
- Pipelined RISC-V Datapath
- And in Conclusion, ...

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 (CISC versus RISC)
- Processor implementation (or *microarchitecture*)
- E.g. for “our” single-cycle RISC-V design, CPI = 1
- Superscalar processors, $\text{CPI} < 1$ (next lecture)

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

Determined by

- Processor microarchitecture (determines critical path through logic gates)
- Technology (e.g. 14nm versus 28nm)
- Power budget (lower voltages reduce transistor speed)

Speed Tradeoff Example

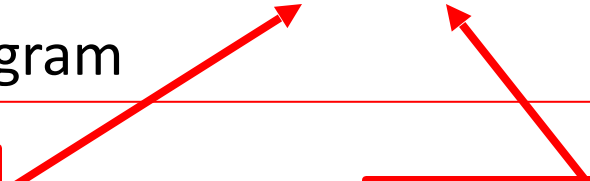
- For some task (e.g. image compression) ...

	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, microarchitecture, circuit details

Supply voltage, e.g. 1V

Want to reduce capacitance and voltage to reduce energy/task₃₅

Energy Tradeoff Example

- “Next-generation” processor (Moore’s law)
 - Capacitance, C : reduced by 15 %
 - Supply voltage, V_{sup} : reduced by 15 %
 - Energy consumption: $(.85C)(.85V)^2 = .63E \Rightarrow$ -39 % reduction
- Significantly improved energy efficiency thanks to
 - Moore’s Law AND
 - Reduced supply voltage

Energy “Iron Law”

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

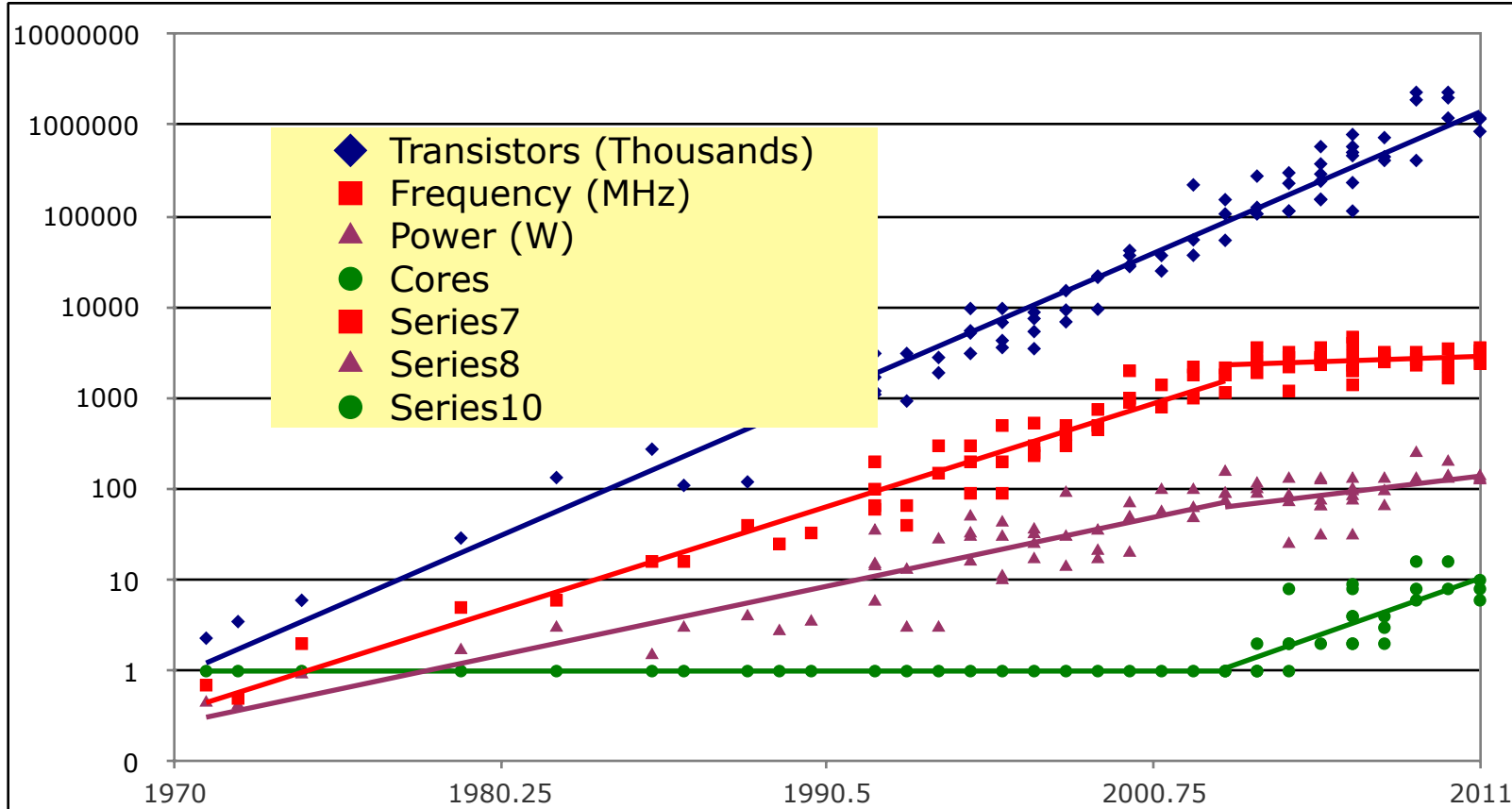
(Tasks/Second) (Joules/Second) (Tasks/Joule)

- 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 $\sim 150\text{W}$

Processor Trends



Break!



Administrivia

- Project 3.1 released Tuesday night, due next Wednesday (3/7).
- Homework 2 due Friday
- Project party on Wednesday 7-10
- Guerrilla session Thursday 7-9pm
- Midterm 2, March 20, is moved to 8-10PM (was 7-9 on the website)
 - Alternative exam earlier, 6-8PM (so people don't need to be in exams until midnight :)
 - submit exam conflict form if they haven't

Agenda

- Finish Single-Cycle RISC-V Datapath
- Controller
- Instruction Timing
- Performance Measures
- **Introduction to Pipelining**
- Pipelined RISC-V Datapath
- And in Conclusion, ...

Pipelining

- A familiar example:
 - Getting a university degree



Year 1



Year 2



Year 3

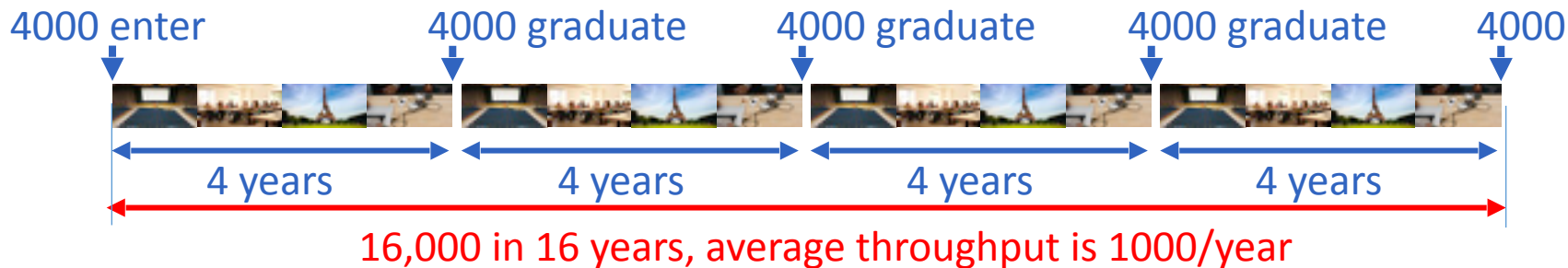


Year 4

- Shortage of Computer scientists (your startup is growing):
 - How long does it take to educate 16,000 students?

Computer Scientist Education

- Option 1: *serial*



- Option 2: *pipelining*



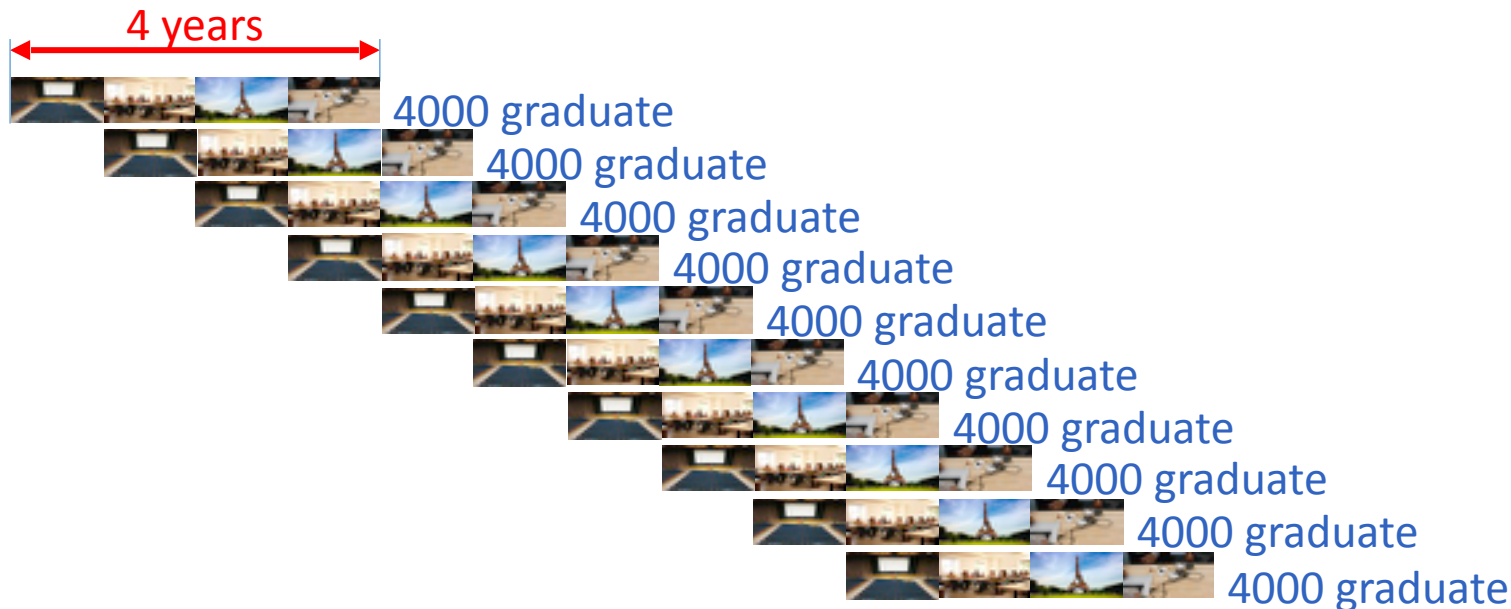
- 16,000 in 7 years
- Steady state throughput is 4000/year
- Resources used efficiently
- **4-fold improvement over serial education**

Latency versus Throughput

- Latency
 - Time from entering college to graduation
 - Serial 4 years
 - Pipelining 4 years
- Throughput
 - Average number of students graduating each year
 - Serial 1000
 - Pipelining 4000
- *Pipelining*
 - Increases throughput (4x in this example)
 - But does nothing to latency
 - sometimes worse (additional overhead e.g. for shift transition)

Simultaneous versus Sequential







- What happens *sequentially*?
- What happens *simultaneously*?



Agenda

- Finish Single-Cycle RISC-V Datapath
- Controller
- Instruction Timing
- Performance Measures
- Introduction to Pipelining
- **Pipelined RISC-V Datapath**
- And in Conclusion, ...

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
$t_{instruction}$		800 ps	1000 ps

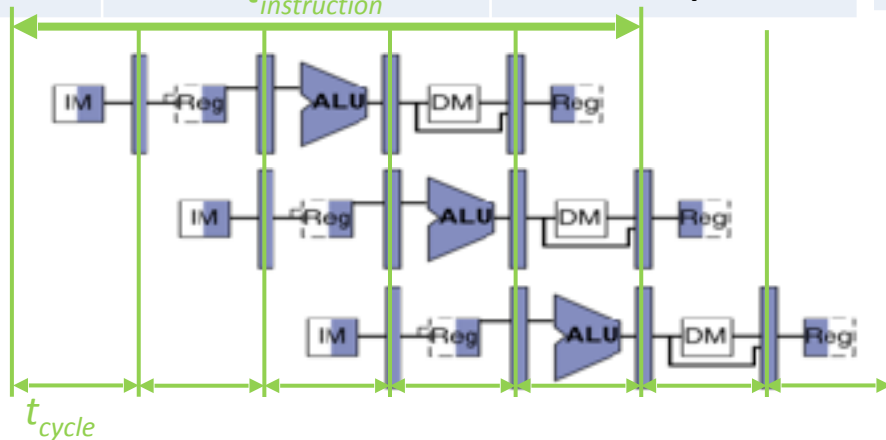
instruction sequence



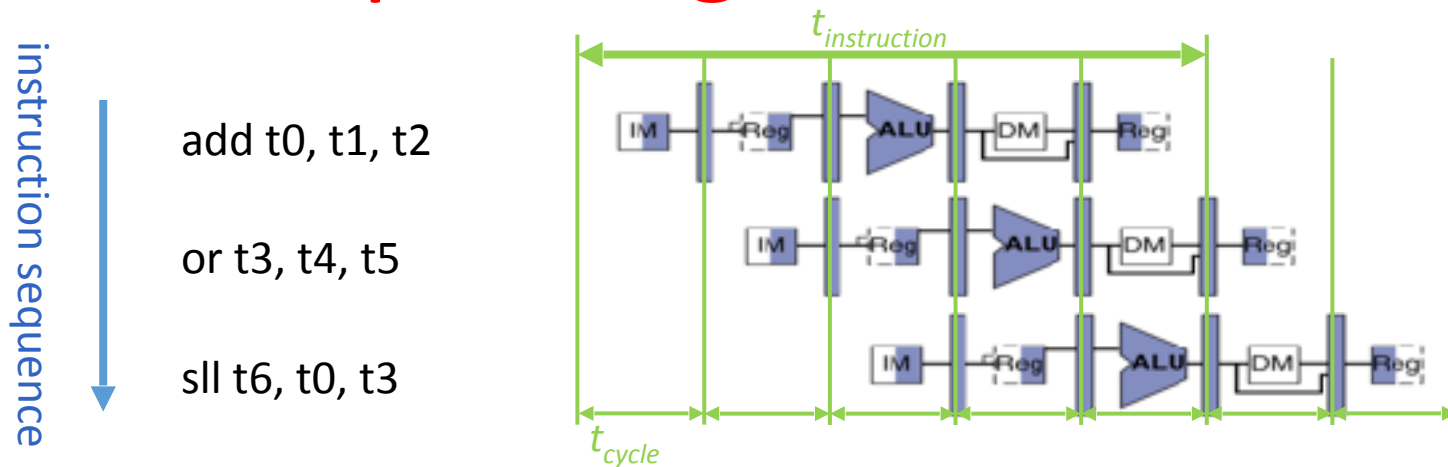
add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3



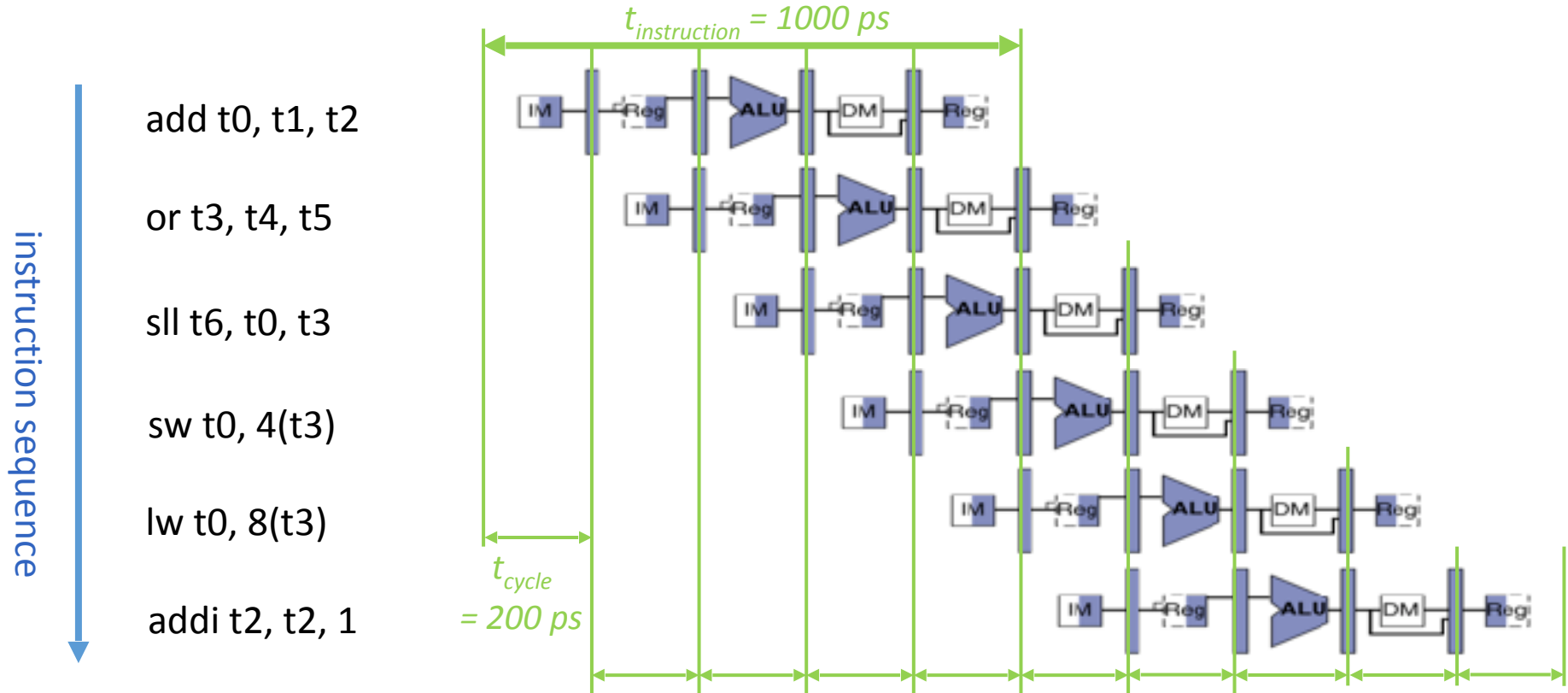
Pipelining with RISC-V



	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, $t_{instruction}$	$= t_{cycle} = 800 \text{ ps}$	1000 ps
Clock rate, f_s	$1/800 \text{ ps} = 1.25 \text{ GHz}$	$1/200 \text{ ps} = 5 \text{ GHz}$
Relative speed	1 x	4 x

Sequential vs Simultaneous

What happens sequentially, what happens simultaneously?



Agenda

- Finish Single-Cycle RISC-V Datapath
- Controller
- Instruction Timing
- Performance Measures
- Introduction to Pipelining
- Pipelined RISC-V Datapath
- **And in Conclusion, ...**

And in Conclusion, ...

- Controller
 - Tells universal datapath how to execute each instruction
- Instruction timing
 - Set by instruction complexity, architecture, technology
 - Pipelining increases clock frequency, “instructions per second”
 - But does not reduce time to complete instruction
- Performance measures
 - Different measures depending on objective
 - Response time
 - Jobs / second
 - Energy per task