

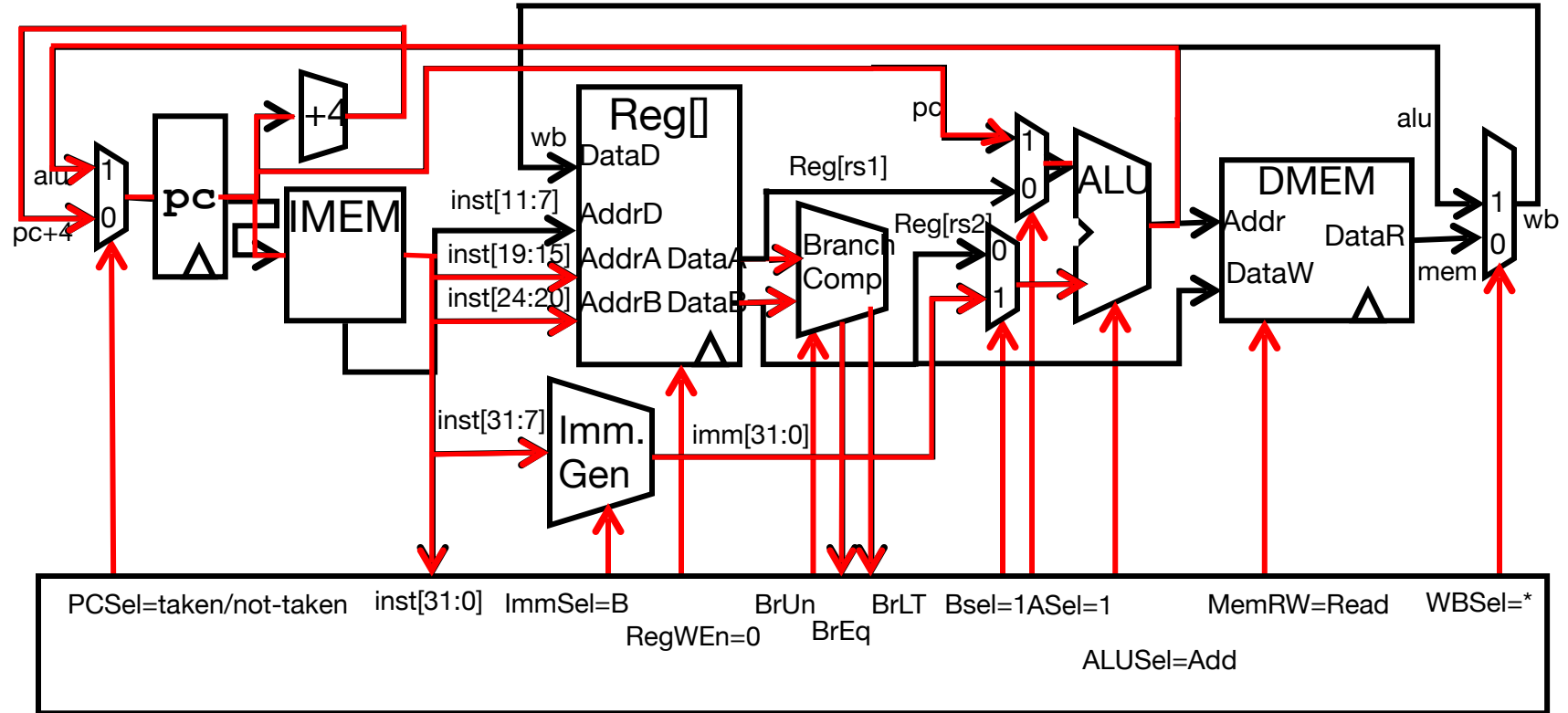
CS 61C:

Great Ideas in Computer Architecture

Lecture 13: RISC-V *Control & Operating Speed*

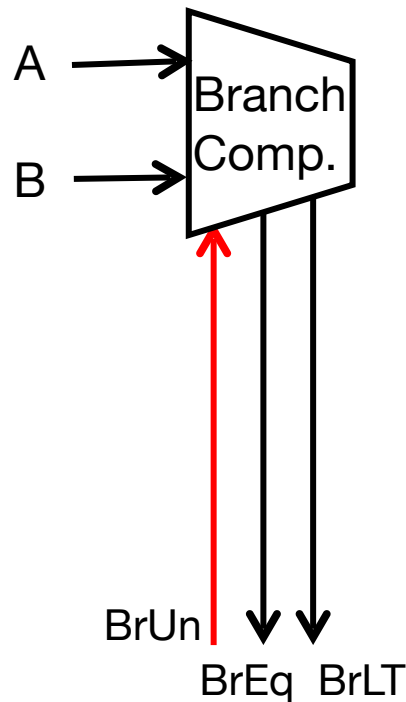
- **Completion of Single-Cycle RISC-V Datapath**
- Controller
- Instruction Timing
- Performance Measures
- Introduction to Pipelining
- Pipelined RISC-V Datapath
- And in Conclusion, ...

Adding branches to datapath

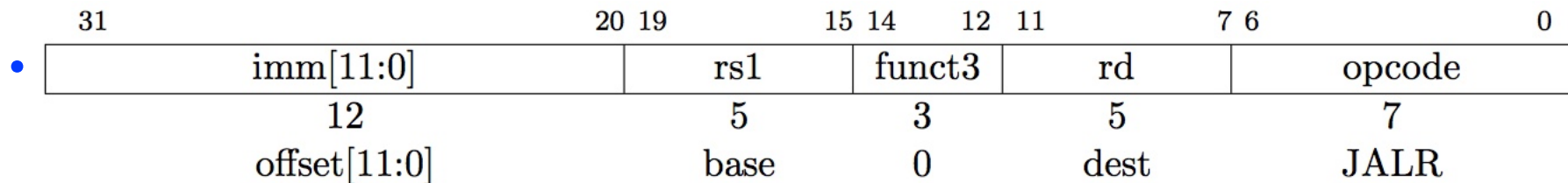


Branch Comparator

- $\text{BrEq} = 1$, if $A=B$
- $\text{BrLT} = 1$, if $A < B$
- $\text{BrUn} = 1$ selects unsigned comparison for BrLT , 0=signed
- BGE branch: $A \geq B$, if $\neg(A < B)$

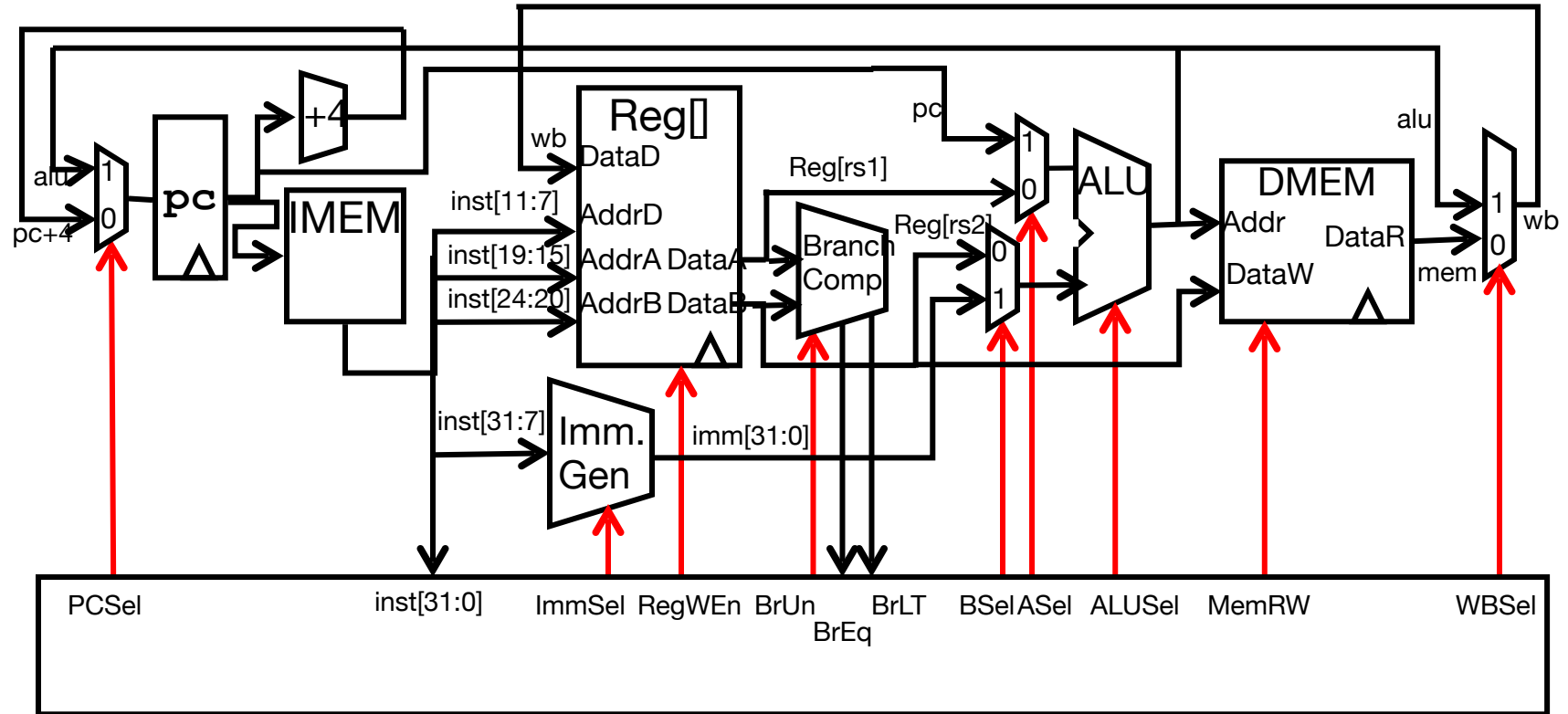


Implementing **JALR** Instruction (I-Format)

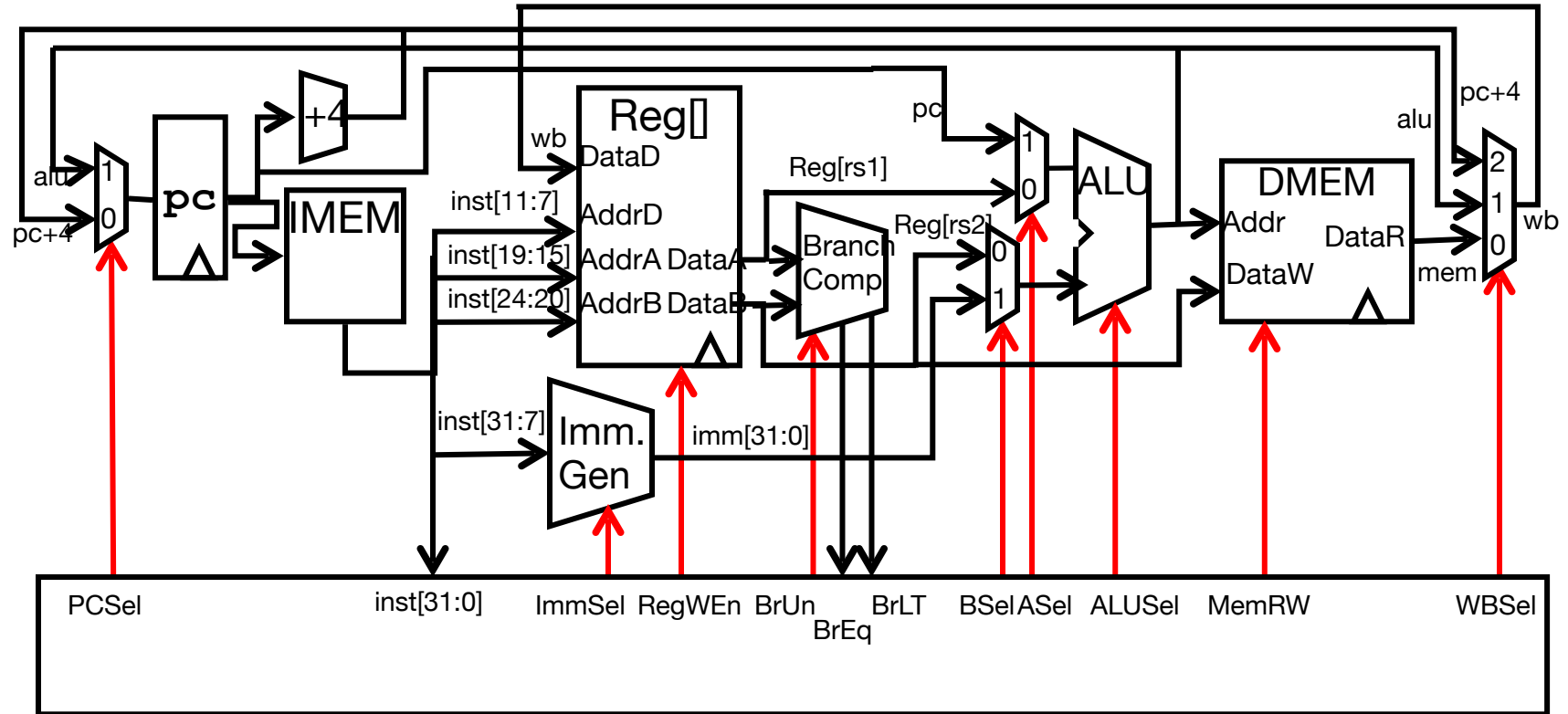


- $PC = \text{Reg}[\text{rs1}] + \text{immediate}$
 - Uses same immediates as arithmetic and loads
 - **no** multiplication by 2 bytes
- $\text{Reg}[\text{rd}] = PC + 4$
 - return address - the “link” - is saved

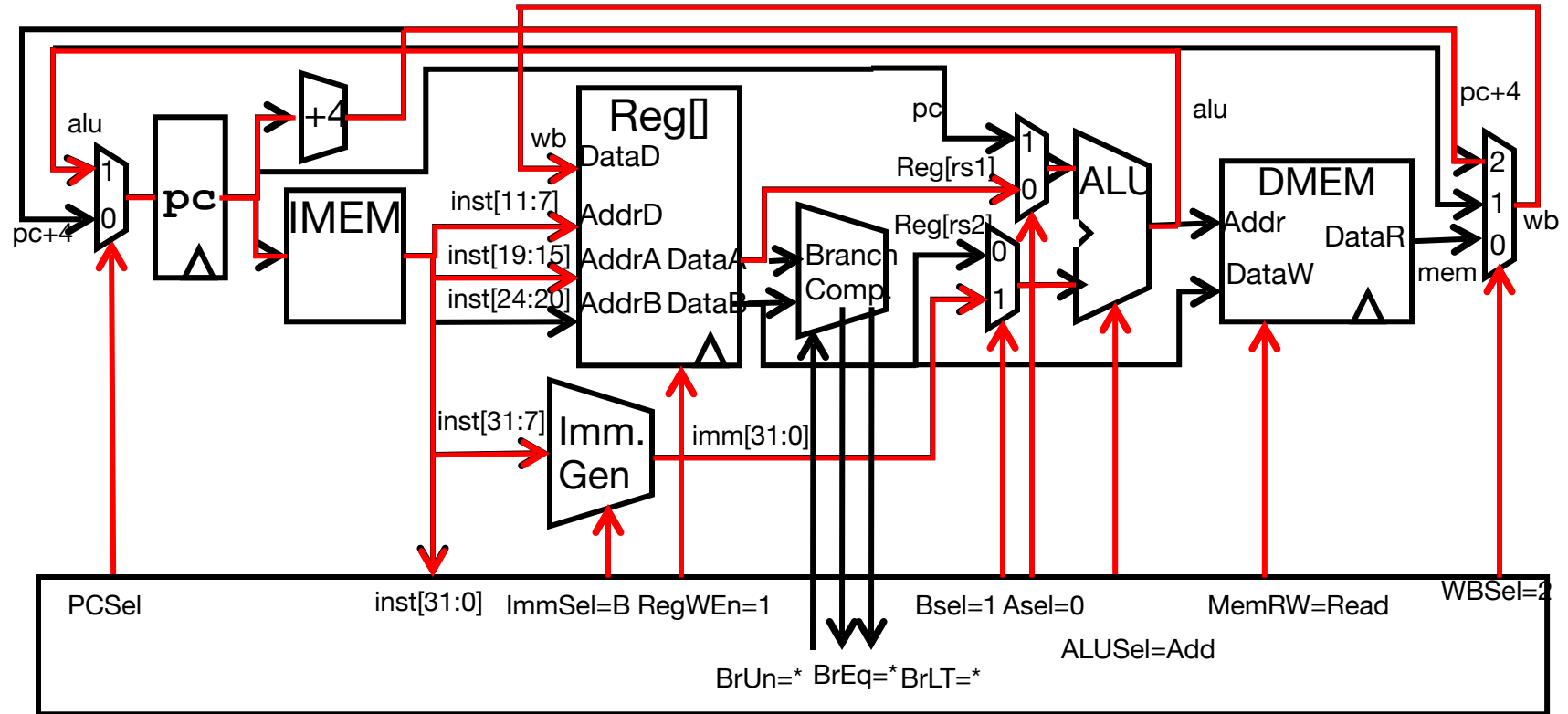
Adding branches to datapath



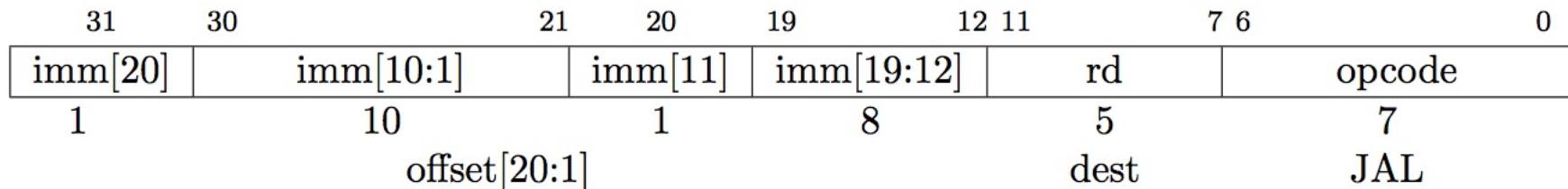
Adding jalr to datapath



Adding jalr to datapath

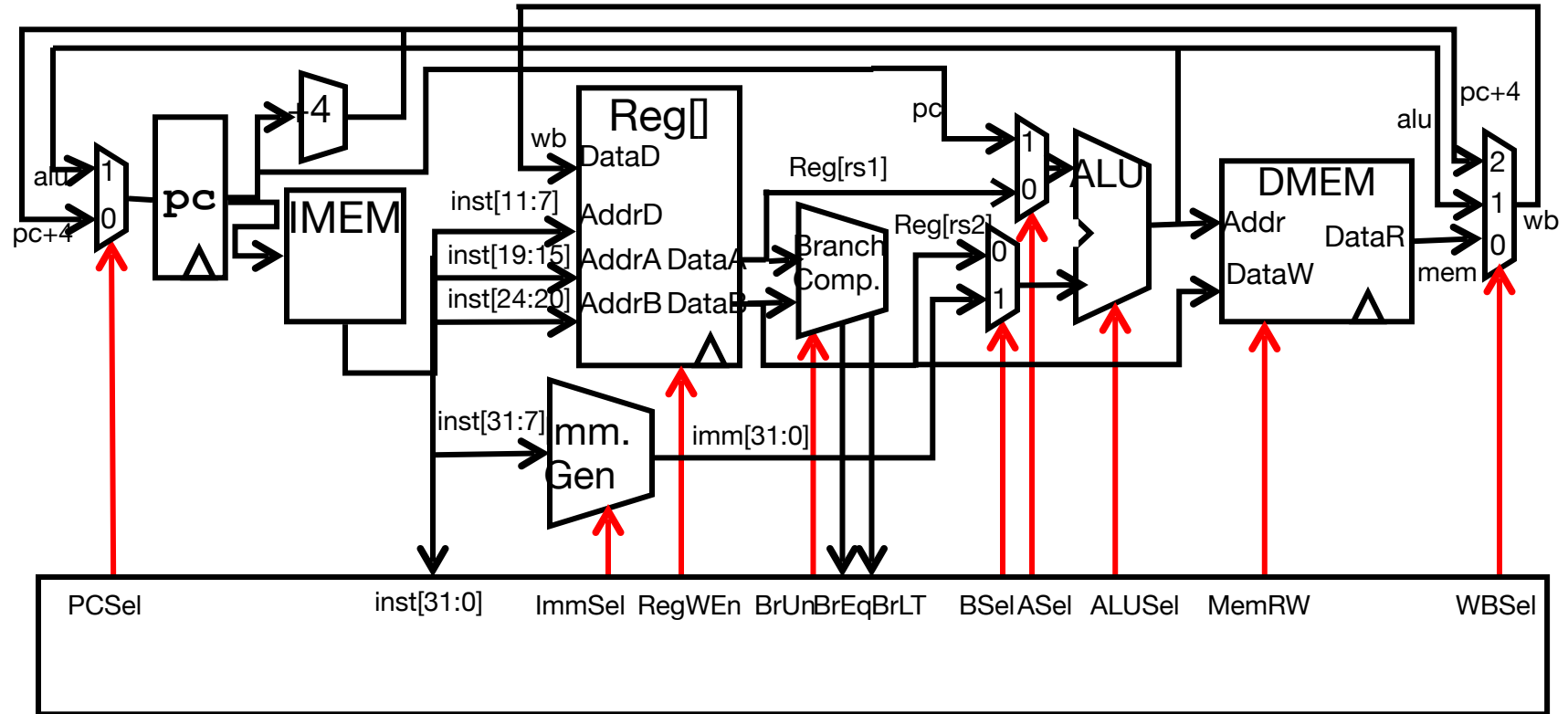


Implementing jal Instruction

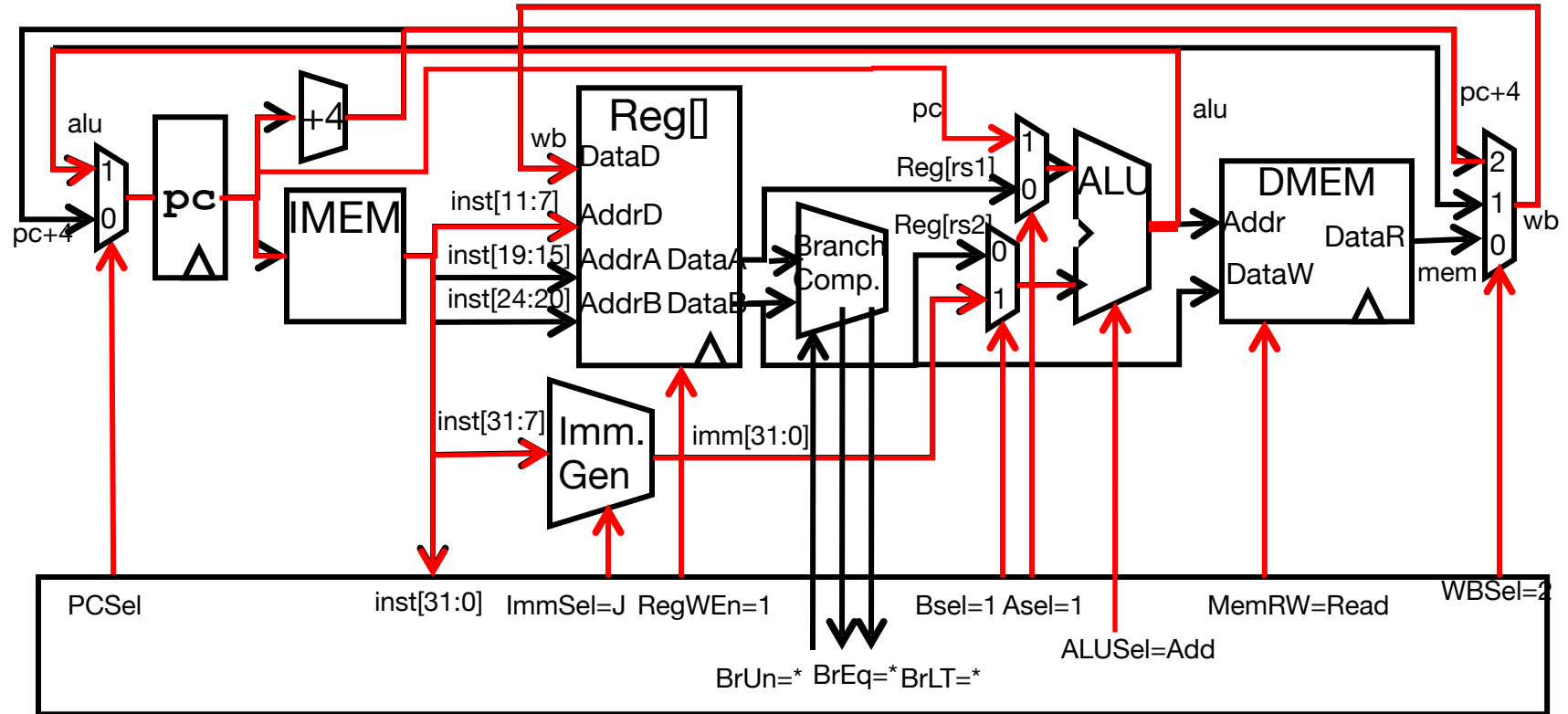


- JAL saves PC+4 in Reg[rd] (the return address)
- Set PC = PC + offset (PC-relative jump)
 - Target somewhere within $\pm 2^{19}$ locations, 2 bytes apart
 - $\pm 2^{18}$ 32-bit instructions
 - Immediate encoding optimized similarly to branch instruction to reduce hardware cost

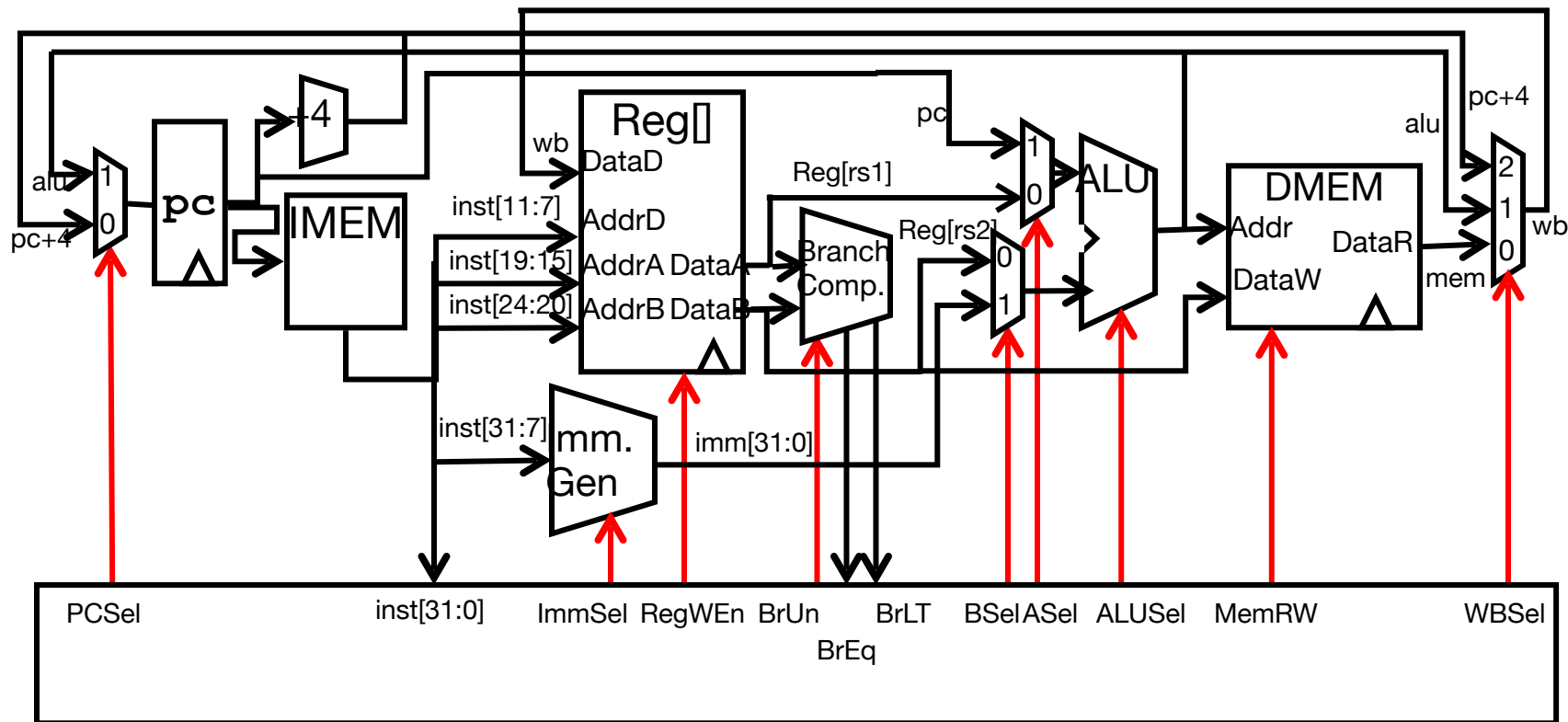
Adding jal to datapath



Adding jal to datapath



Single-Cycle RISC-V RV32I Datapath



Recap: Complete RV32I ISA

imm[31:12]					rd	0110111
imm[31:12]					rd	0010111
imm[20:10:11:19:12]					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

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	001	rd	1110011	CSRRS	

Not in CS61C

RV32I has 47 instructions total
 37 instructions covered in CS61C

Remaining instructions (ex: lui, auipc) can be implemented with no significant additions to the datapath: adding a “pass B” option to the ALU and another immediate decoding option. Rest is all control logic

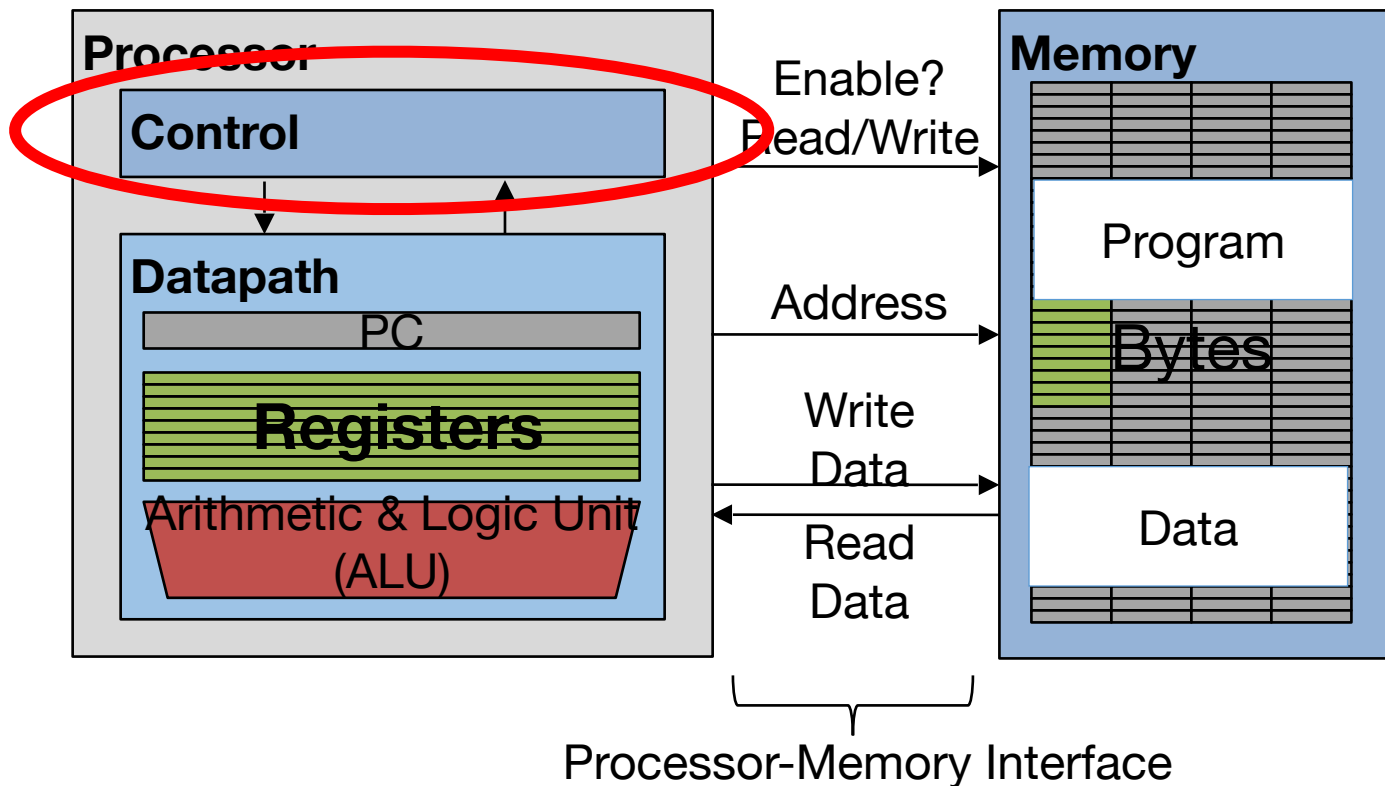
And in Conclusion, ...

- Universal datapath
 - Capable of executing all RISC-V instructions in one cycle each
 - datapath is the “union” of all the units used by all the instructions. Muxes provide the options.
 - Not all units (hardware) used by all instructions
- 5 Phases of execution
 - IF, ID, EX, MEM, WB
 - Not all instructions are active in all phases
- Controller specifies how to execute instructions

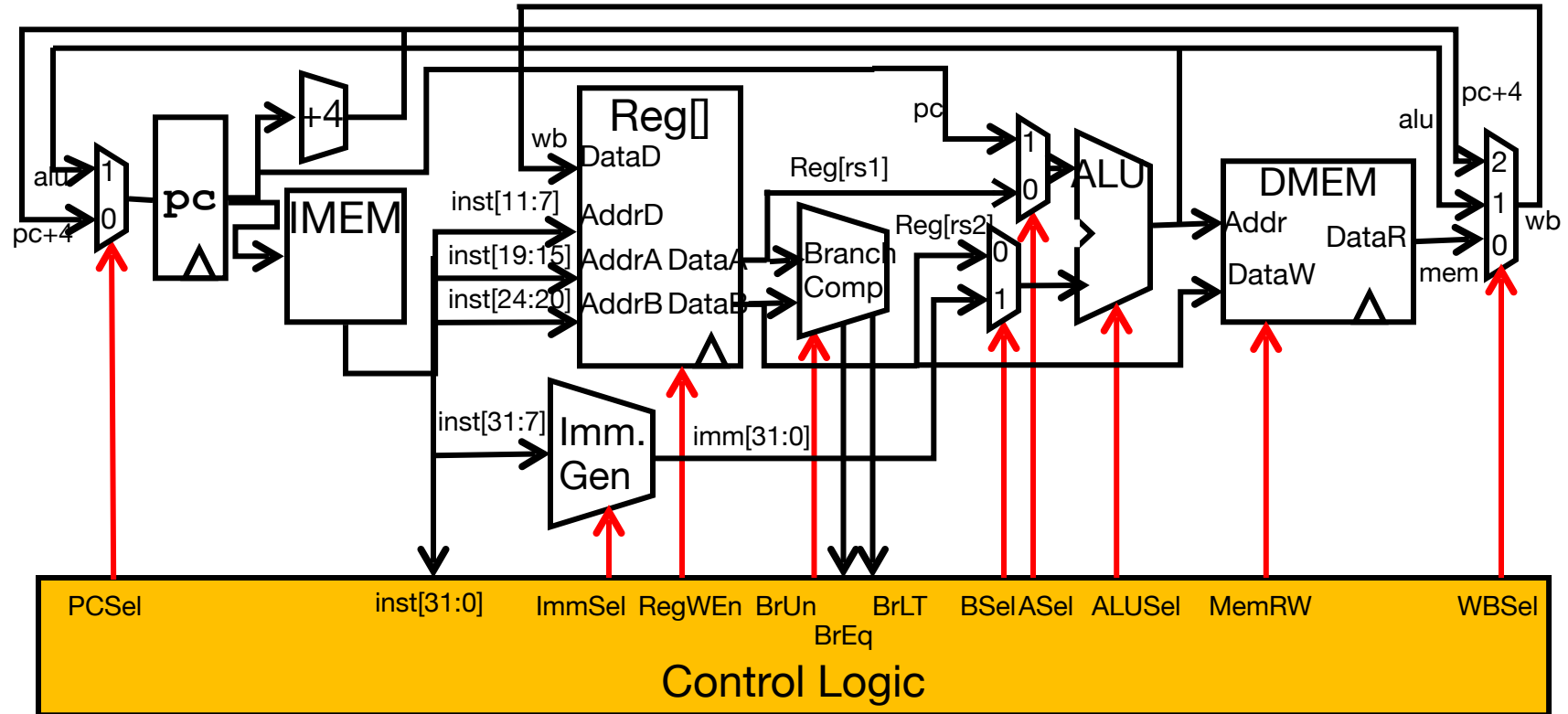
Agenda

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

Processor



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

* means “for all values”
 - means “don’t care, use any value”

Note: Instruction type encoded using only 9 bits

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

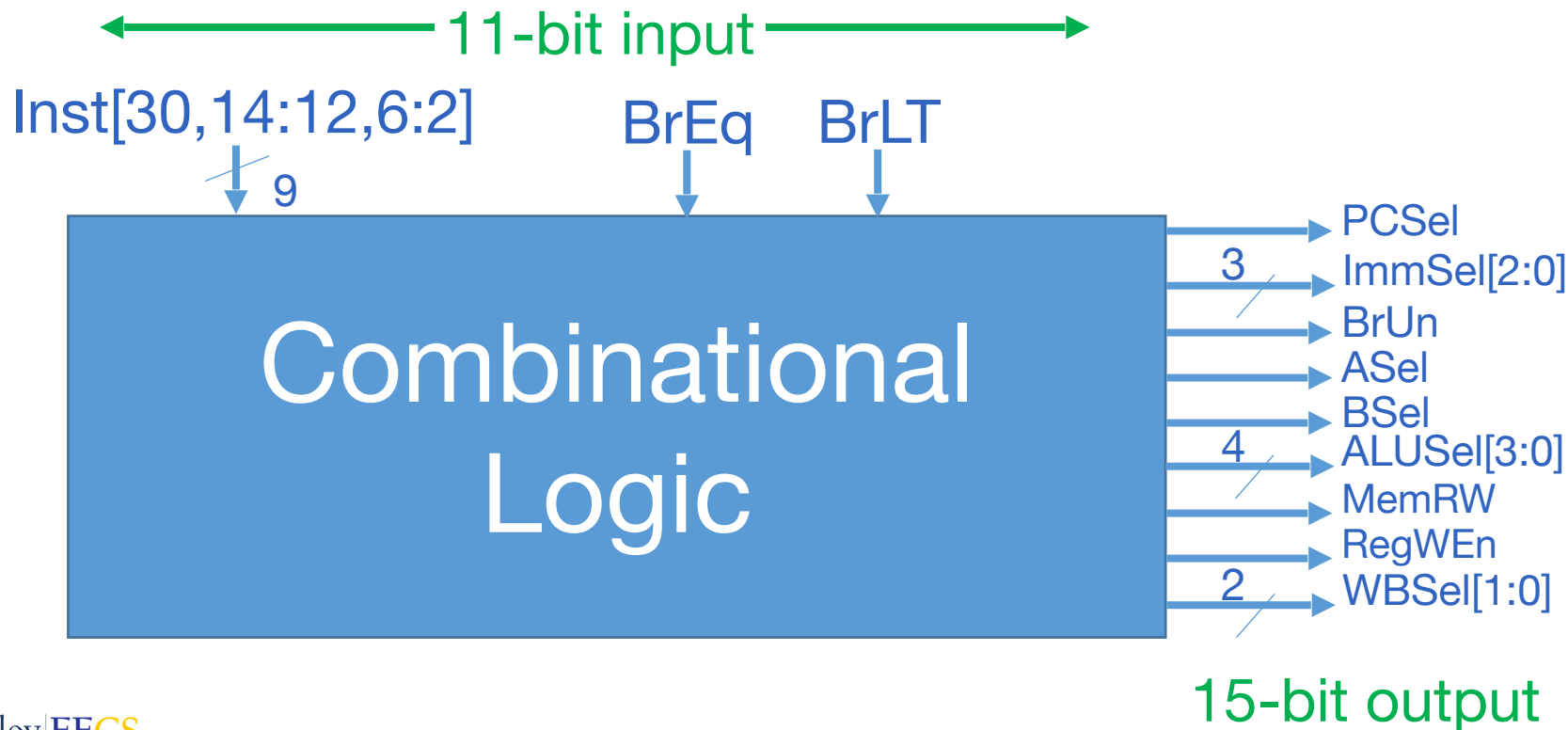
imm[31:12]				rd	011011	LUI
imm[31:12]				rd	001011	AUIPC
imm[20:10:11:19:12]				rd	110111	JAL
imm[11:0]		rs1	000	rd	110011	JALR
imm[12:10:5]	rs2	rs1	000	imm[4:1:11]	110001	BEQ
imm[12:10:5]	rs2	rs1	001	imm[4:1:11]	110001	BNE
imm[12:10:5]	rs2	rs1	100	imm[4:1:11]	110001	BLT
imm[12:10:5]	rs2	rs1	101	imm[4:1:11]	110001	BGE
imm[12:10:5]	rs2	rs1	110	imm[4:1:11]	110001	BLTU
imm[12:10:5]	rs2	rs1	111	imm[4:1:11]	110001	BGEU
imm[11:0]		rs1	000	rd	000001	LB
imm[11:0]		rs1	001	rd	000001	LH
imm[11:0]		rs1	010	rd	000001	LW
imm[11:0]		rs1	100	rd	000001	LBU
imm[11:0]		rs1	101	rd	000001	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	010001	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	010001	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	010001	SW
imm[11:0]		rs1	000	rd	001001	ADDI
imm[11:0]		rs1	010	rd	001001	SLTI
imm[11:0]		rs1	011	rd	001001	SLTIU
imm[11:0]		rs1	100	rd	001001	XORI
imm[11:0]		rs1	110	rd	001001	ORI
imm[11:0]		rs1	111	rd	001001	ANDI

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

0000000	shamt	rs1	00	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
00000000000000			00000	000	00000	1110011	ECALL
00000000000001			00000	000	00000	1110011	EBREAK
csr	rs1	001	rd	1110011			CSRRW
csr	rs1	0	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

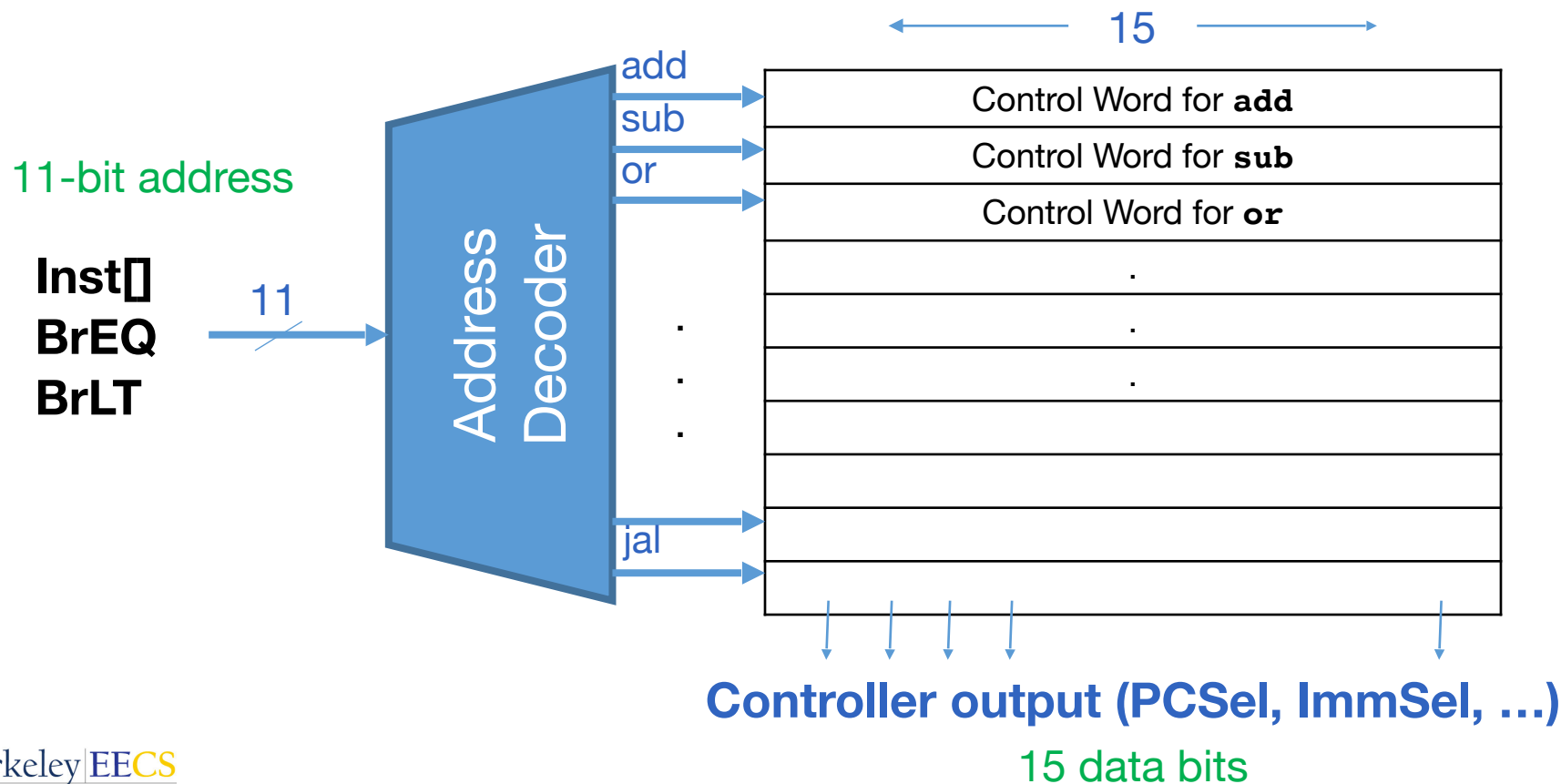
Control Block Design



Controller Realization Options

- **ROM (Read-Only Memory)**
 - Regular structure made from transistors
 - Can be easily reprogrammed during the design process to
 - fix errors
 - add instructions
 - Popular when designing control logic manually
- **Combinatorial Logic**
 - Today, chip designers often use logic synthesis tools to convert truth tables to networks of gates
 - Logic equation for each control signal (common sub-expressions shared among control signal equations)
 - Can exploit output “don’t cares” and input “for all values” to simplify circuit.

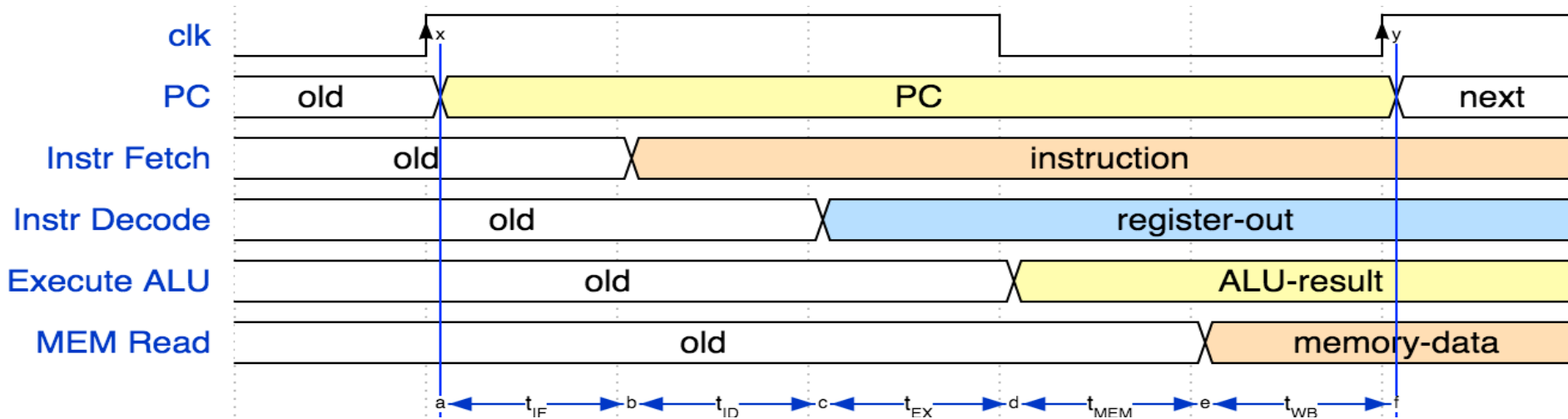
ROM (read only memory) Controller Implementation



Agenda

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

Typical Approximate Worst-Case 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

- How can we keep data-path resources (such as ALU) busy all the time?
- For ALU could have 5 billion adds/sec, rather than just 1.25 billion?
- Idea: Factories “assembly line” - all 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:
 - Less time for each instruction?
 - More instructions per unit time?
 - Aren’t these the same? Yes, for our simple single-cycle processor, but not so when we employ **parallelism**.
 - Is energy efficiency a measure of performance?

Transportation Analogy

Computer Science 61C Fall 2021



Weaver

	Race Car	Bus
Passenger Capacity	1	50
Travel Speed	200 mph	50 mph
Gas Mileage	5 mpg	2 mpg

50 Mile trip:

	Race Car	Bus
Travel Time	15 min	60 min
Time for 100 passengers	1500 min	120 min
Gallons per passenger	10 gallons	0.5 gallons

Processor Analogy

Transportation	Computer
Trip Time	Instruction execution time (latency)
Time for 100 passengers	Total number of instructions executed per unit time (throughput)
Gallons per passenger	Energy per instruction (energy efficiency): e.g. how many total instructions executed per battery charge or per unit on energy bill for datacenter

Computer Task-level Analogy

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

“Iron Law” of Processor Performance

$$\frac{\textit{time}}{\textit{program}} = \frac{\textit{instructions}}{\textit{program}} \cdot \frac{\textit{cycles}}{\textit{instruction}} \cdot \frac{\textit{time}}{\textit{cycle}}$$

Instructions per Program

- Determined by

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

$$\frac{\text{time}}{\text{program}} = \boxed{\frac{\text{instructions}}{\text{program}}} \cdot \frac{\text{cycles}}{\text{instruction}} \cdot \frac{\text{time}}{\text{cycle}}$$

(Average) Clock cycles per Instruction

- Determined by

$$\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \cdot \frac{\text{cycles}}{\text{instruction}} \cdot \frac{\text{time}}{\text{cycle}}$$

- ISA (CISC versus RISC)
- Processor implementation (or **microarchitecture**)
 - E.g. for “our” single-cycle RISC-V design, CPI = 1
- Pipelined processors, CPI > 1 (next lecture)
- Superscalar processors, CPI < 1 (next lecture)

Time per Cycle (1/Frequency)

$$\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \cdot \frac{\text{cycles}}{\text{instruction}} \cdot \boxed{\frac{\text{time}}{\text{cycle}}}$$

- Determined by
 - Processor microarchitecture (determines critical path through logic gates)
 - Technology (e.g. 5nm versus 14nm)
 - Supply voltage (lower voltage reduces transistor speed, but improves energy efficiency)

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

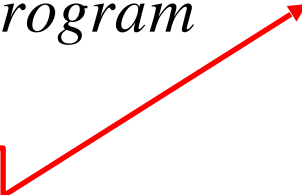
Processor B is faster for this task, despite executing more instructions and having a lower clock rate!

Energy per Task

$$\frac{\text{energy}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \cdot \frac{\text{energy}}{\text{instruction}}$$

$$\frac{\text{energy}}{\text{program}} \propto \frac{\text{instructions}}{\text{program}} \cdot CV_{dd}^2$$

“Capacitance” depends on technology, microarchitecture, circuit details



Supply voltage, e.g. 1V



Want to reduce capacitance and voltage to reduce energy/task

Energy Tradeoff Example

- For instance, “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”

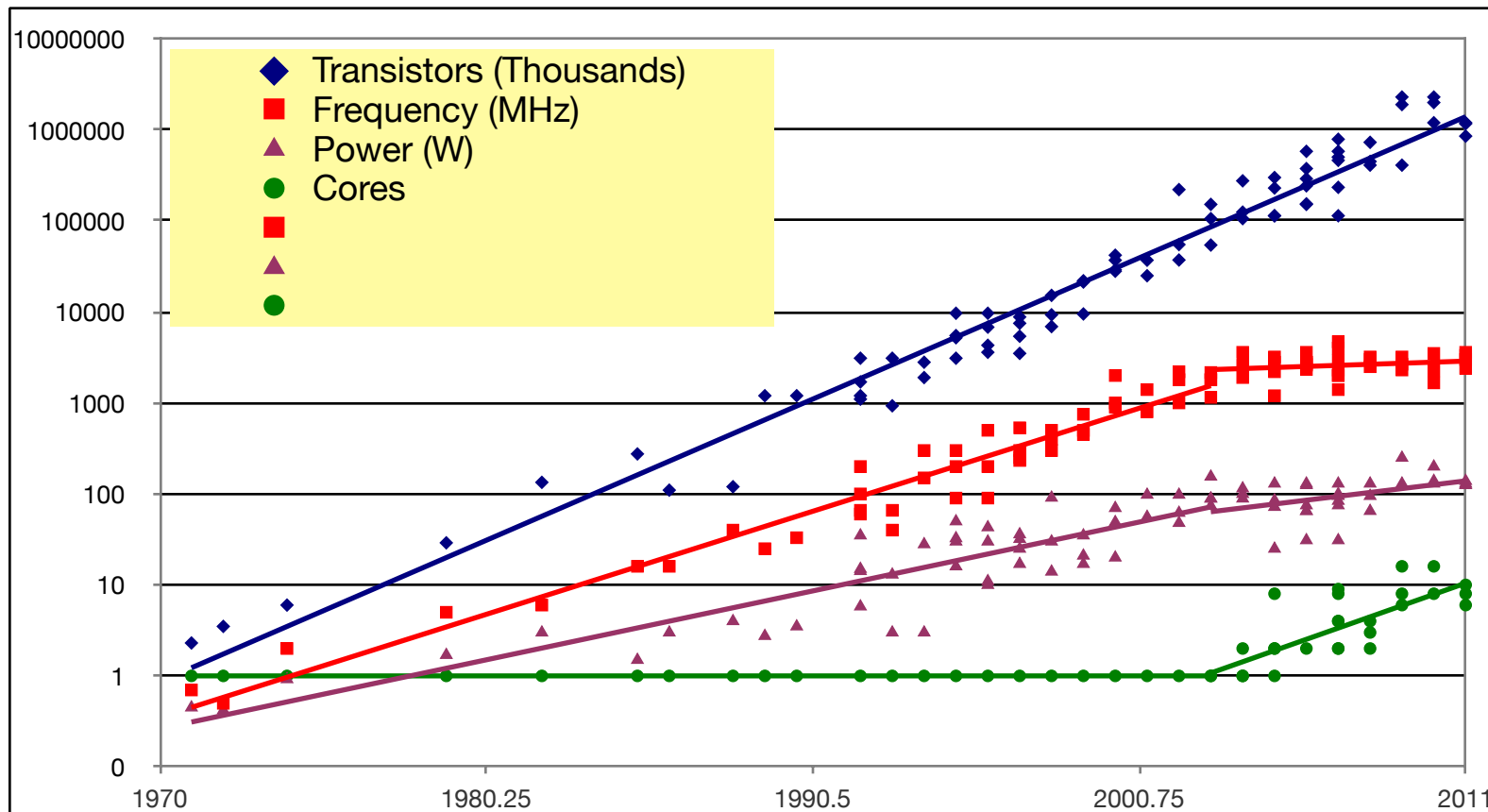
- Energy efficiency (e.g., instructions/Joule) is key metric in all computing devices
- For power-constrained systems (e.g., 20MW datacenter), need better energy efficiency to get more performance at same power
- For energy-constrained systems (e.g., 1W phone), need better energy efficiency to prolong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

$$\begin{array}{lll} \textit{performance} & = & \textit{power} \cdot \textit{energy efficiency} \\ (\textit{tasks/second}) & & (\textit{Joules/sec}) \quad (\textit{tasks/Joule}) \end{array}$$

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
 - Rather than horizontal modern CMOS uses vertically-aligned transistors to pack them closer together... But that doesn't reduce capacitance just allows for higher density
- Power becomes a growing concern – the “power wall”
- Cost-effective air-cooled chip limit around ~150W

Processor Trends



[Olukotun, Hammond, Sutter, Smith, Batten]

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



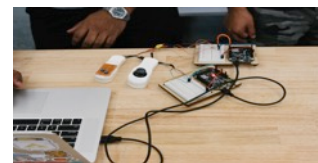
Year 1



Year 2



Year 3

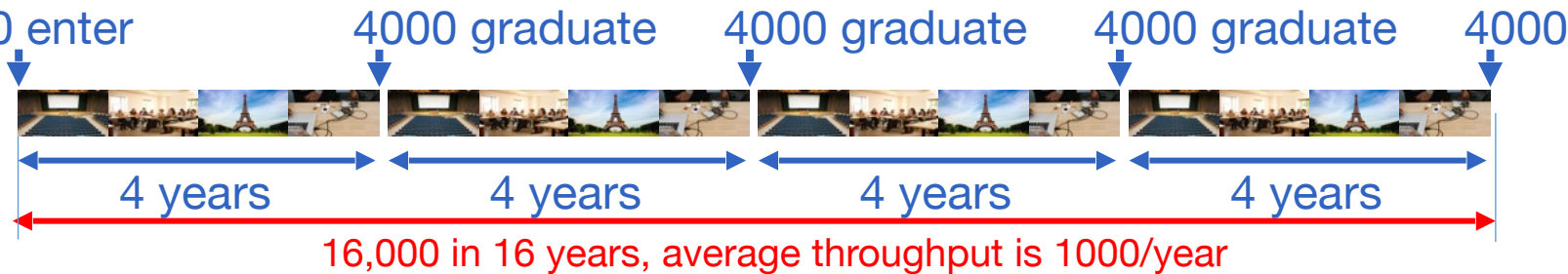


Year 4

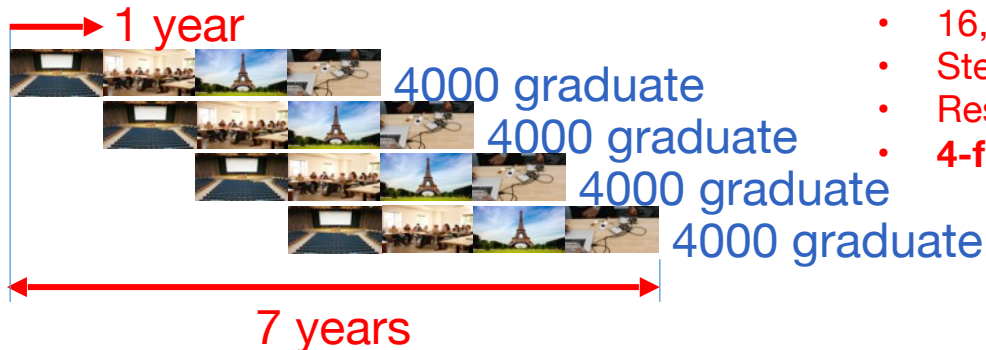
- Shortage of Computer scientists (your startup is growing):
 - How long does it take to educate 16,000 students?

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**

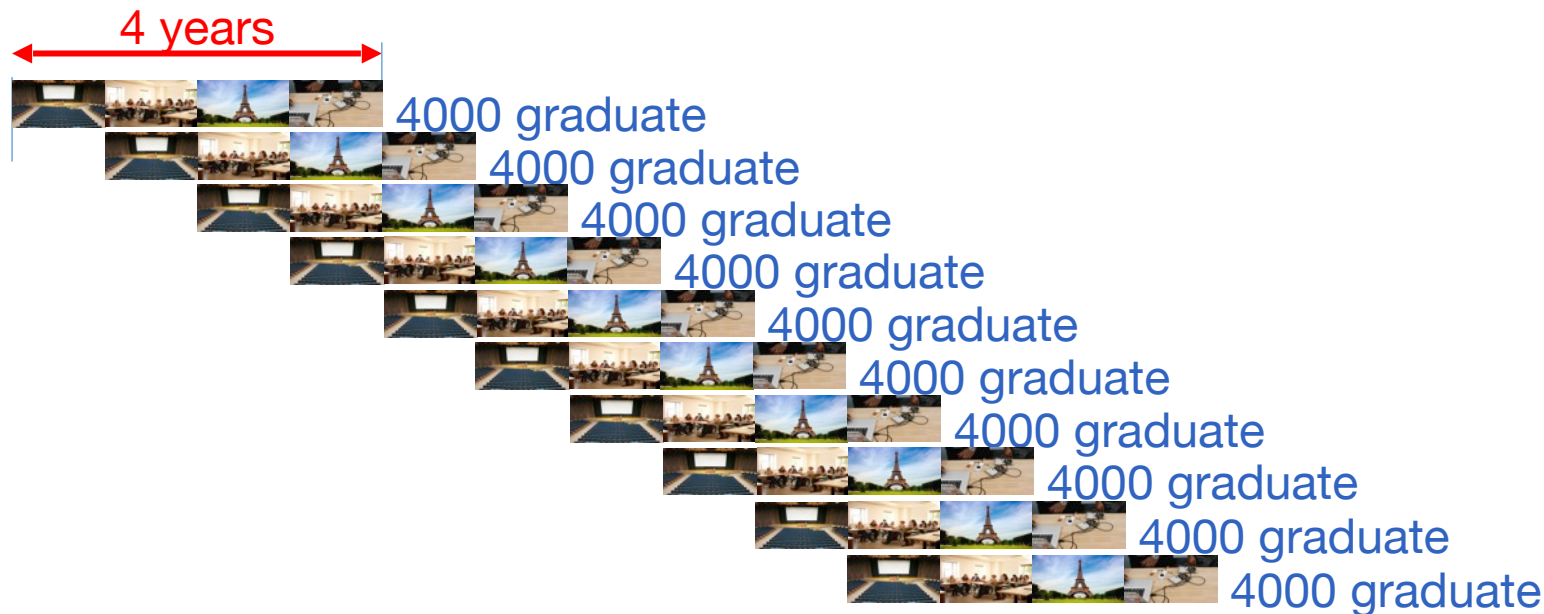
- Average number of students graduating each year
- Serial 1000
- Pipelining 4000

- **Pipelining**

- Increases throughput (4x in this example)
- But can ***never improve*** latency
 - sometimes worse (additional overhead)

Simultaneous versus Sequential







- What happens *sequentially*?
- What happens *simultaneously*? *A form of parallel processing!*



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

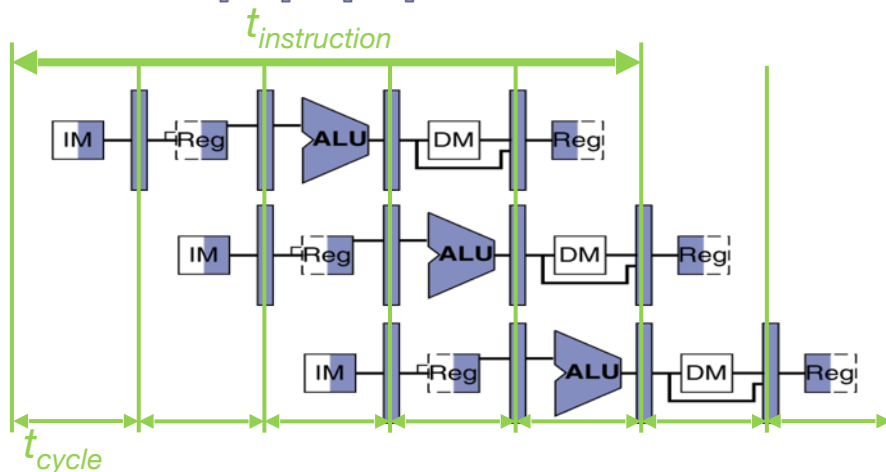
Phase	Pictogram	t_{step} Serial	t_{cycle} Pipelined
Instruction Fetch		200 ps	200 ps
Reg Read		100 ps	200 ps
ALU		200 ps	200 ps
Memory		200 ps	200 ps
Register Write		100 ps	200 ps
$t_{instruction}$		800 ps	1000 ps

instruction sequence

add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3



Pipelining with RISC-V

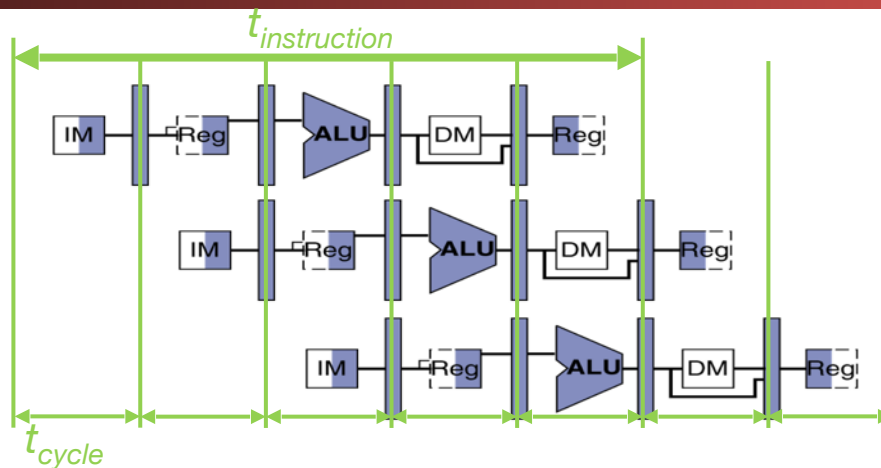
instruction sequence



add t0, t1, t2

or t3, t4, t5

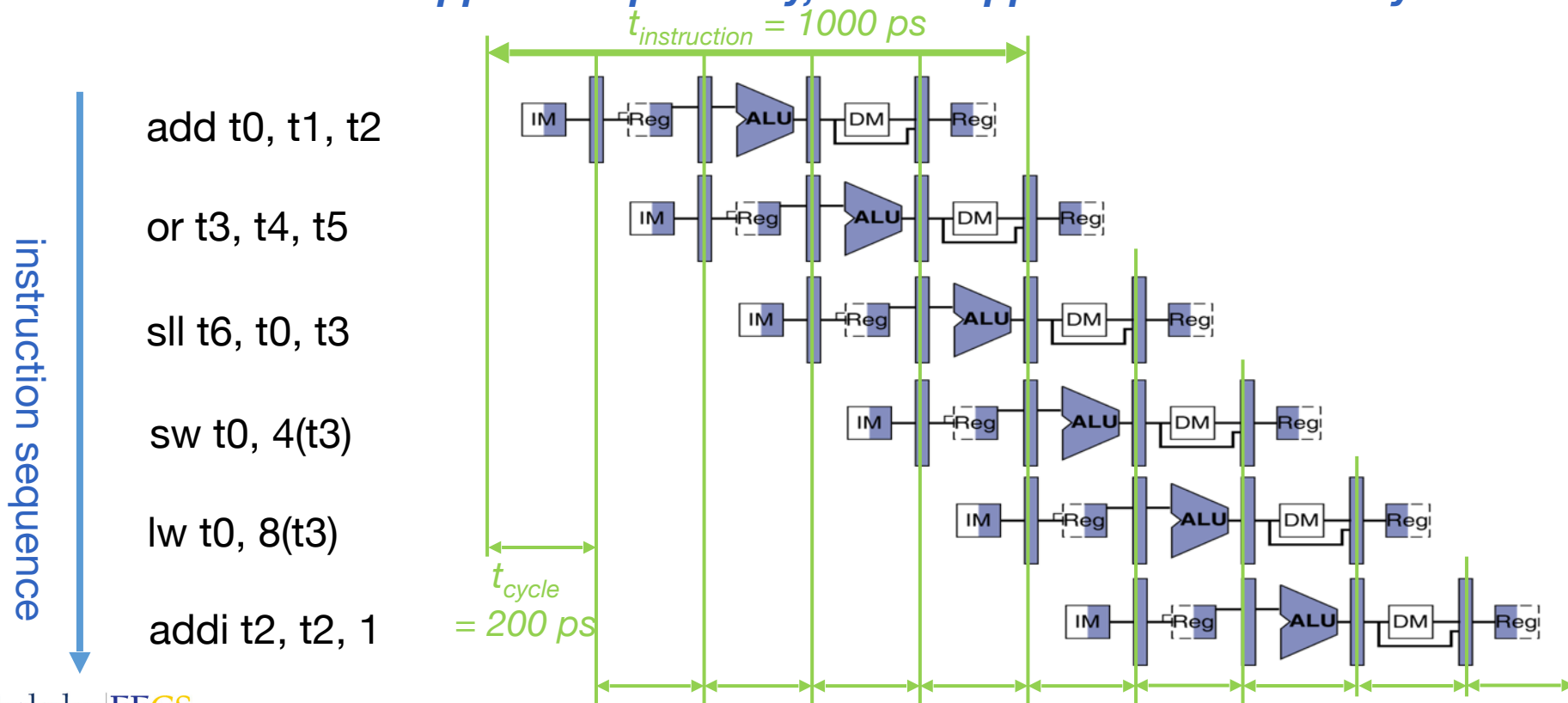
sll t6, t0, t3



	Single Cycle	Pipelining
Timing	$t_{step} = 100 \dots 200 \text{ ps}$ (Register access only 100 ps)	$t_{cycle} = 200 \text{ ps}$ 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