UNIVERSITY of WASHINGTON

Roadmap

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

Assembly language:

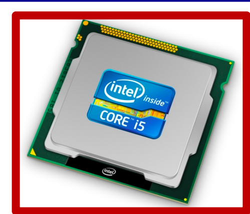
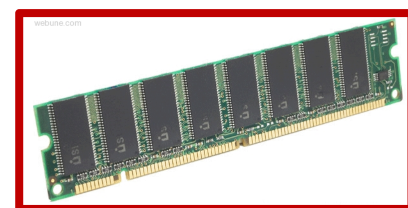
```
get_mpg(car*):
        lw      a5,0(a0)
        lw      a4,4(a0)
        divw    a5,a5,a4
        fcvt.s.w        fa0,a5
        ret
```

Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
110000011111101000011111
```

OS:

Computer system:

- Memory & data
- Arrays & structs
- Integers & floats
- RISC V assembly
- Procedures & stacks
- Executables
- Memory & caches
- **Processor Pipeline**
- Performance
- Parallelism

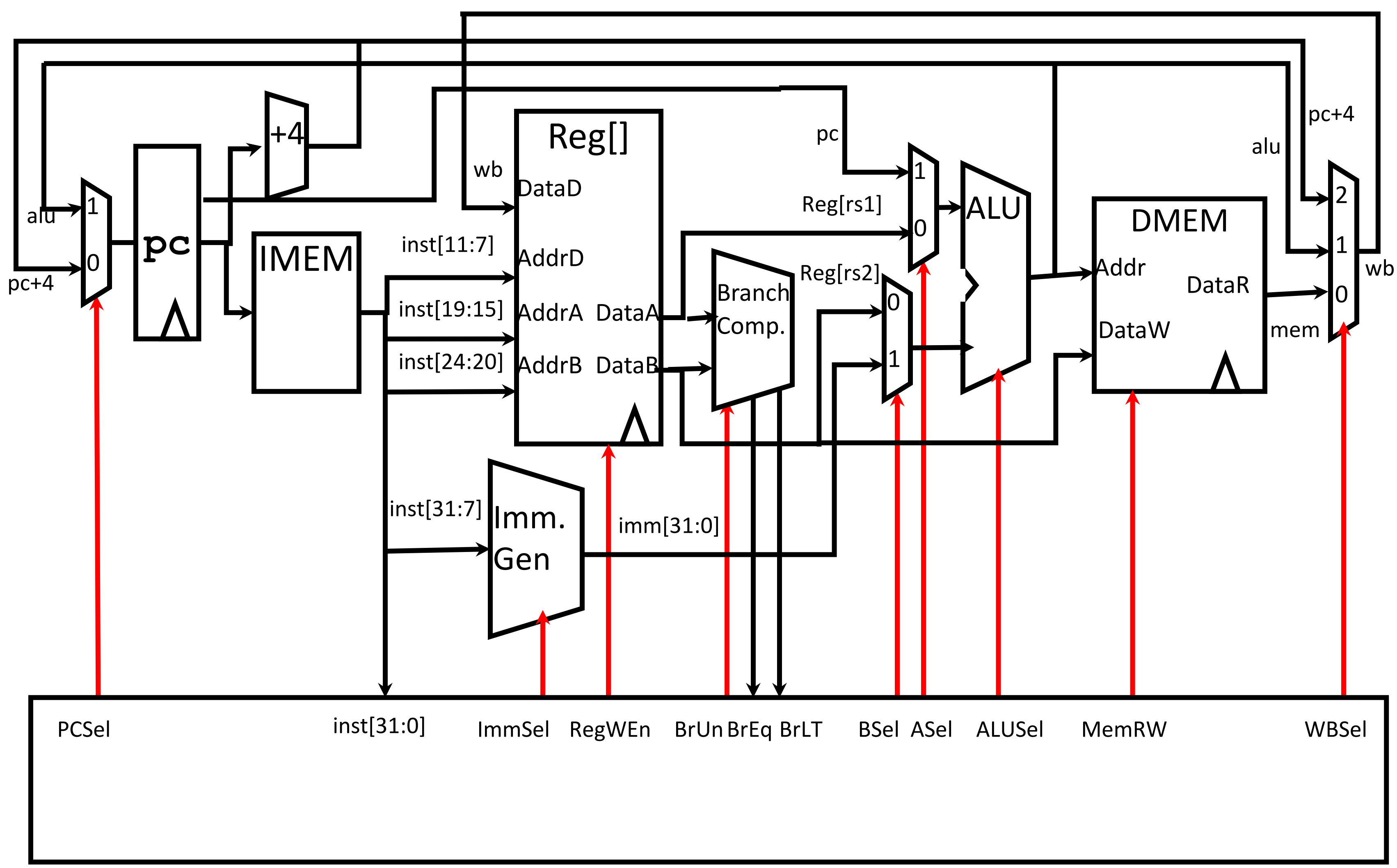
UNIVERSITY of WASHINGTON

Agenda

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

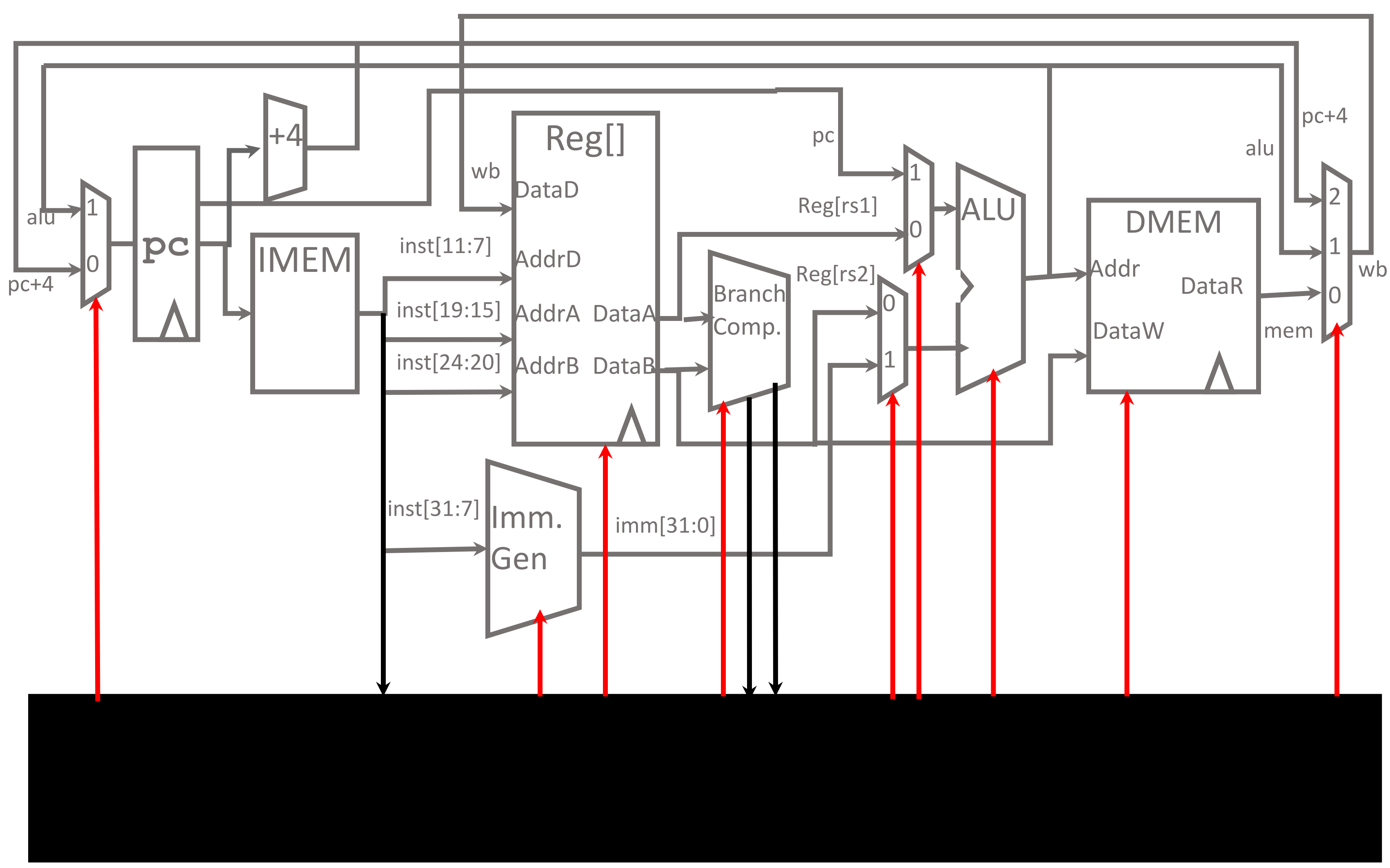
UNIVERSITY of WASHINGTON

Single-Cycle RISC-V RV32I Datapath



UNIVERSITY of WASHINGTON

Single-Cycle RISC-V RV32I Datapath



UNIVERSITY of WASHINGTON

Agenda

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

Our Control Bits

- PCSel
  - Does this instruction change my control flow?
  - What is the address of my next instruction?
- ImmSel
  - Does this instruction have/use an immediate?
    - If yes, what type of instruction is it? How is the immediate stored?
- RegWEn
  - Does this instruction write to the destination register rd?
- BrUn
  - Does this instruction do a branch? If so, is it unsigned or signed?

UNIVERSITY of WASHINGTON

Our Control Bits

- BSel
  - Does this instruction operate on R[rs2] or an immediate?
- ASel
  - Does this instruction operate on R[rs1] or PC?
- ALUSel
  - What operation should we perform on the selected operands?
- MemRW
  - Do we want to write to memory? Do we want to read from memory?
    - If we don’t care about the memory output, what should we do?
- WBSel
  - Which value do we want to write back to rd?
    - If we aren’t writing back (RegWEn = 0) does this value matter?

UNIVERSITY of WASHINGTON

Designing Control Signals

- Questions you should ask:
  - Is this control signal the same for every instruction of the same type? (I, R, S, SB, etc.) If so, can you use a combination of opcode/funct3/funct7 to encode the value?
  - Is this control signal dependent on *other* controls?
    - ie. PCSel and BrEq, BrLT
  - Does the value of this control signal alter the execution of the instruction?
    - Some cases: yes! (MemRW, for example)
    - Some cases: no! (ImmSel in R-type inst, for example)

UNIVERSITY of WASHINGTON

Let’s try an example!

Design PCSel yourself!

I recognise this is hard, given you don’t all have logic simulator in front of you :(describe the circuit and its dependencies as best you can !

Might help to split it into three cases:

- Regular (non-control) instructions
- Branches
- Jumps

You may assume PCSel = 0 → PC = PC + 4, and PCSel = 1 → PC = ALUout

UNIVERSITY of WASHINGTON

PCSel: Regular Instructions

- Assumption: PCSel = 0 → PC = PC + 4, and PCSel = 1 → PC = ALUout
  - This isn't the case in every datapath! How can you check? Look at what the PC-input MUX maps 0 and 1 to. Its controlled by PCSel! (Review lecture 9, 10)
- For regular instructions, PCSel is always 0, so our circuit looks like this (pretty boring, huh?)

UNIVERSITY of WASHINGTON

PCSel: Branch Instructions

- How do we know if an instruction is a branch?
- Intuition: check the green sheet!

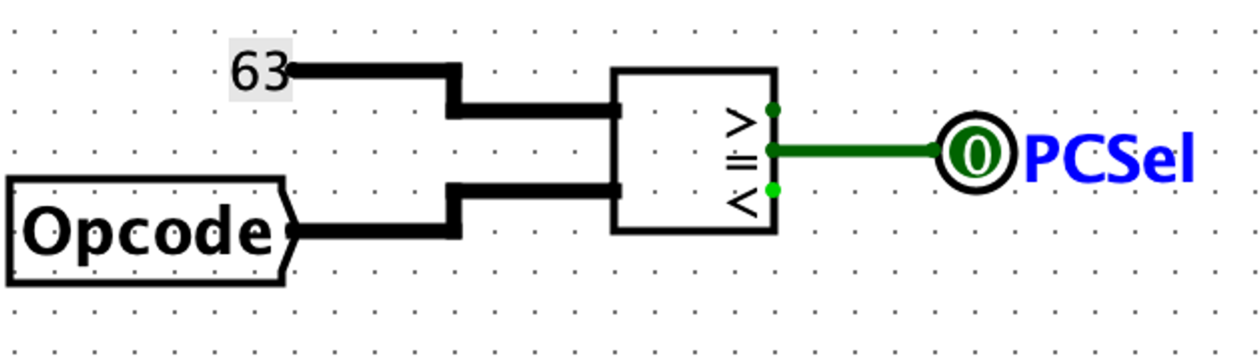
beq	SB	1100011	000	63/0
bne	SB	1100011	001	63/1
blt	SB	1100011	100	63/4
bge	SB	1100011	101	63/5
bltu	SB	1100011	110	63/6
bgeu	SB	1100011	111	63/7

- In order: opcode, func3 → same fields but in hex
- Look at that! they all have the same opcode! We should also check to make sure no other instructions have the same one!
  - spoiler: they don’t, but you should check!

UNIVERSITY of WASHINGTON

PCSel: Branch Instructions

- Let's describe our desired behaviour in words:
  - If we are a regular instruction, choose PC+4. If we are a branch instruction, choose ALUout
  - If we are a regular instruction, set PCSel = 0. If we are a branch instruction, set PCSel = 1
- We can identify a branch instruction by doing an equality check on the opcode. Here's our sub circuit:



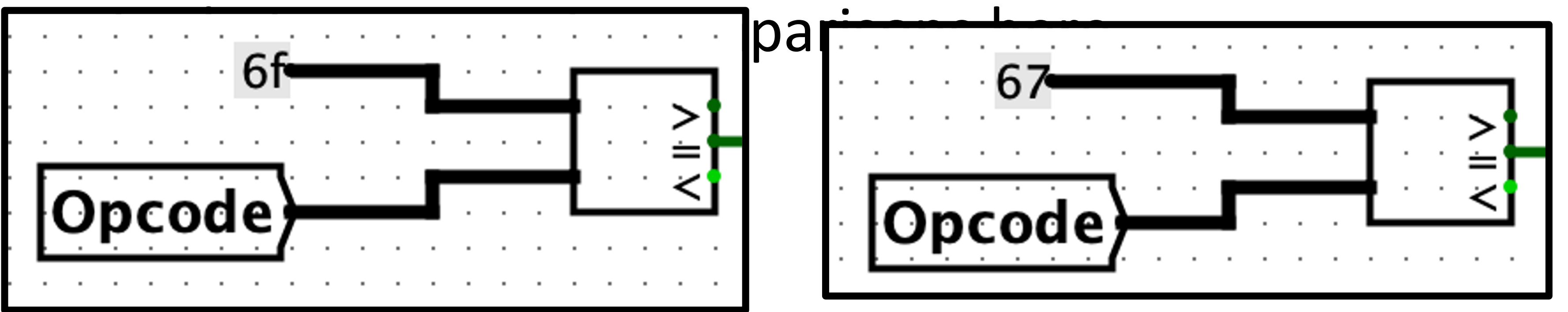
UNIVERSITY of WASHINGTON

PCSel: Jumps

- Which instructions are our jump instructions?

jalr	I	1100111	000	67/0
jal	UJ	1101111		6F

- In order, opcode, func3 (none for jal) → hex
  - Oh no! These are different, so no easy generalisation here.
- Same as with branching, though, we can do an equality check on the opcode using a comparator. We'll have



UNIVERSITY of WASHINGTON

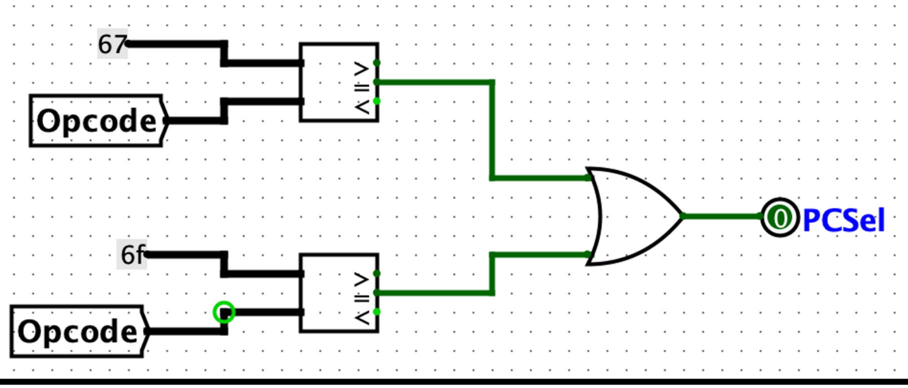
Peer Question

Which of the following circuits is the correct PCSel for jumps?

A

67
Opcode
6f
Opcode
PCSel

B



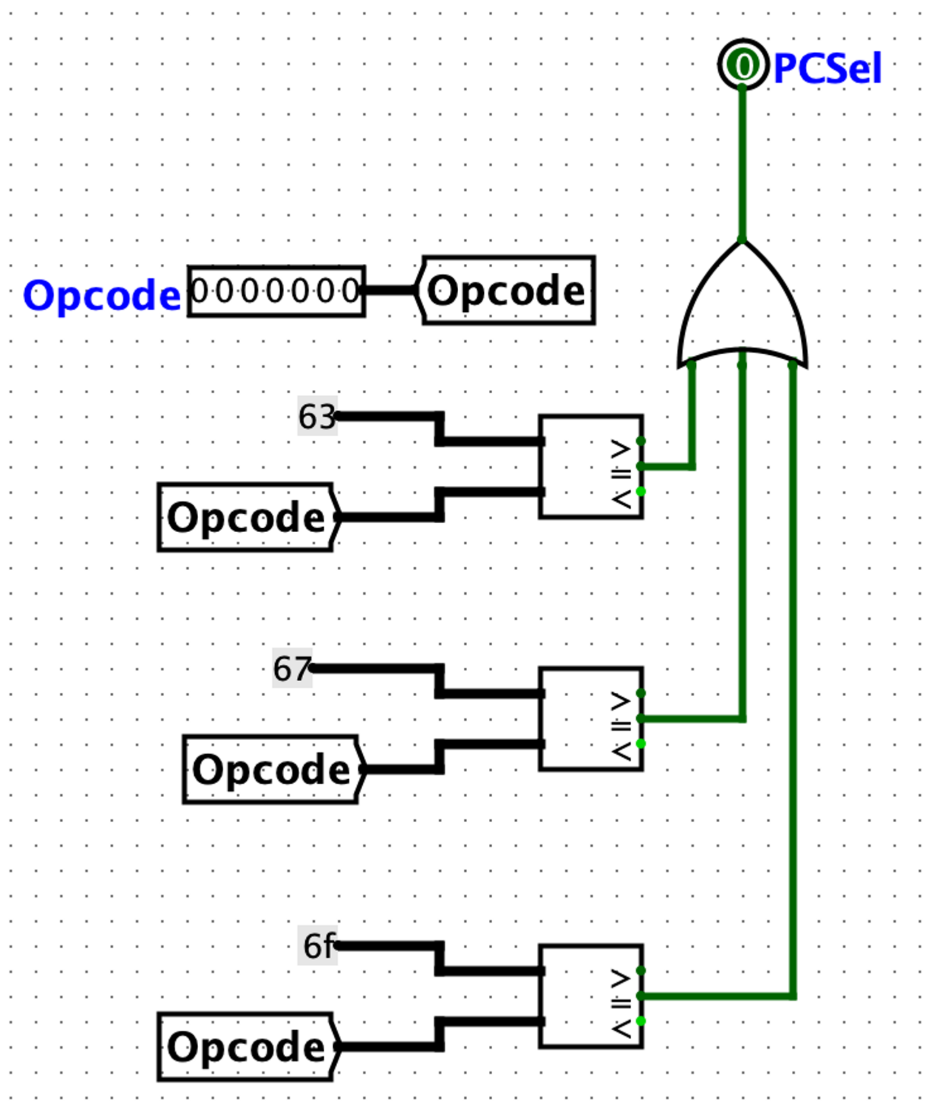
UNIVERSITY of WASHINGTON

Putting them all together

- We can have a regular instruction OR a branch instruction OR a jump instruction. To combine all our signals together and retain the functionality of each individual piece, we'll OR them!
  - Describing your circuit aloud, and keying in on the words you use, might be a helpful design/debugging strategy!
- If any of the sub-circuits are true, PCSel will become (1)
  - Otherwise, it'll be 0
- Because we only have sub-circuits for the branch and jump cases, all normal instructions will have PCSel = 0, while branch, jump will have PCSel = 1 as desired :)

UNIVERSITY of WASHINGTON

PCSel: Final Circuit



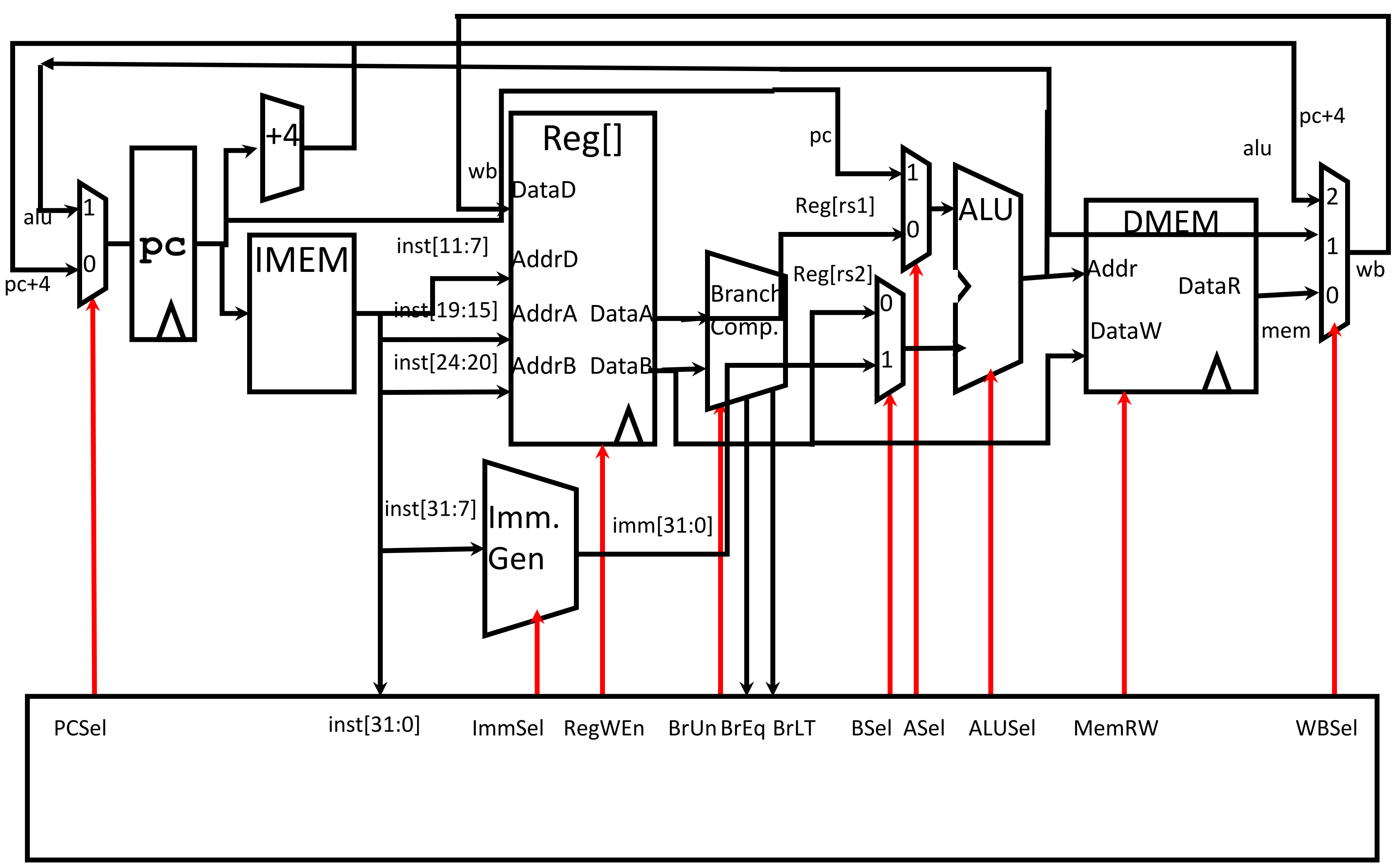
UNIVERSITY of WASHINGTON

Control Signals: Big picture!

- Control signals are how we get the same hardware to behave differently and produce different instructions
- For every instruction, all control signals are set to one of their possible values (Not always 0 or 1!) or an indeterminate (*) value indicating the control signal doesn't affect the instruction's execution
- Each control signal has a sub-circuit based on ~nine bits from the instruction format:
  - Upper 5 func7 bits (lower 2 are the same for all 295 instructions)
  - All func3 bits
  - "2nd" upper opcode bit (others are the same for all 295 instructions)

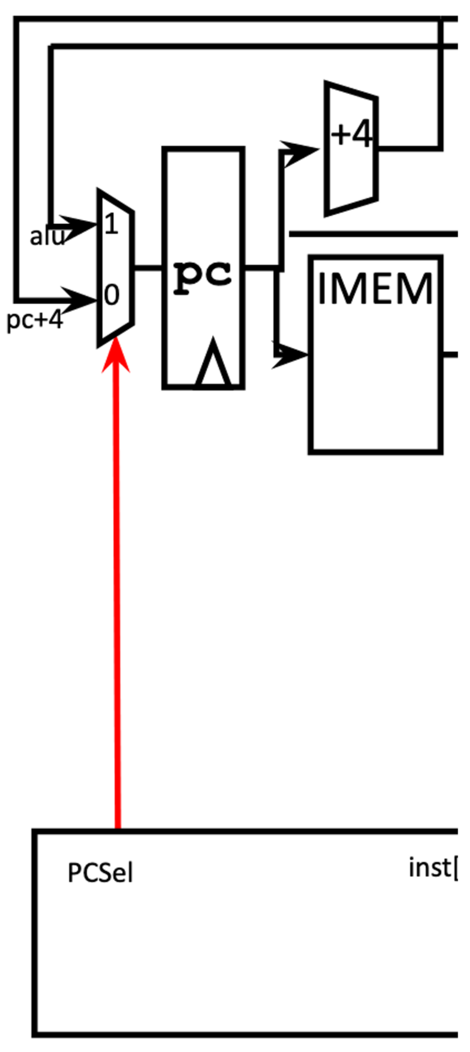
UNIVERSITY of WASHINGTON

Control Signals: ADD



UNIVERSITY of WASHINGTON

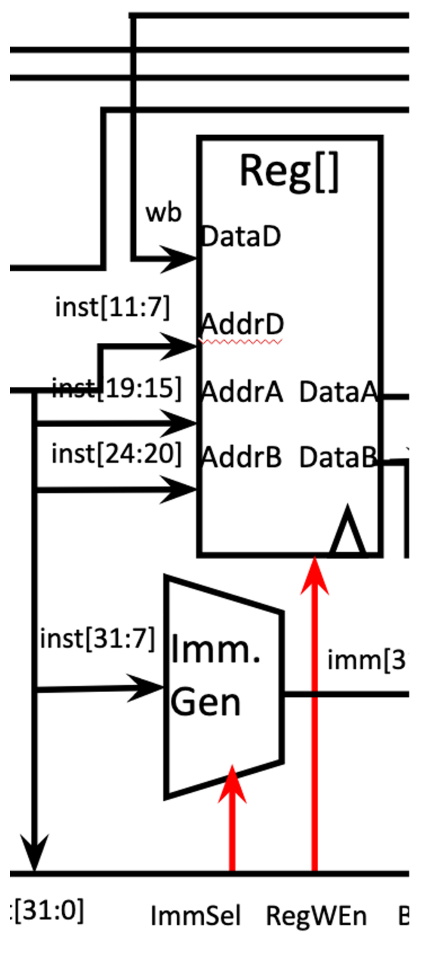
ADD: PCSel



- Should we execute the next instruction (0), or jump control flow to the address given by our ALU output (1)?
- We aren't a branch or jump!
- PCSel = 0

UNIVERSITY of WASHINGTON

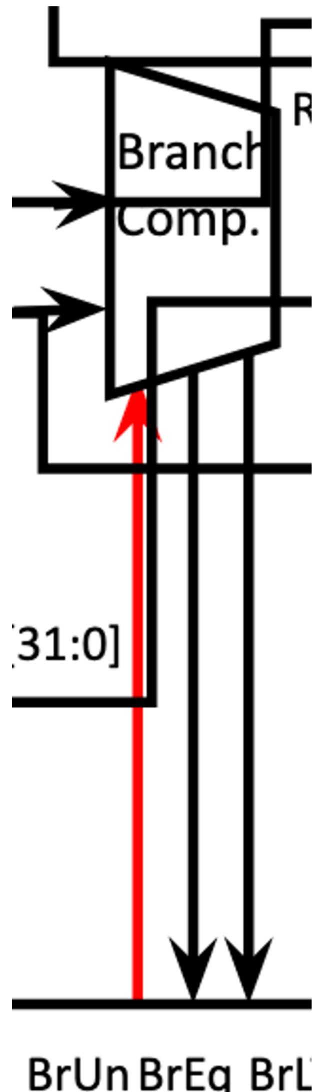
ADD: ImmSel, RegWEn



- How do we want to assemble our immediate?
  - Wait… we don't ? have one?
  - We DON'T CARE about this signal
- ImmSel = *
- Do we want to write (1) to our destination register rd, or not (0)?
  - Add should write!
- RegWEn = 1

UNIVERSITY of WASHINGTON

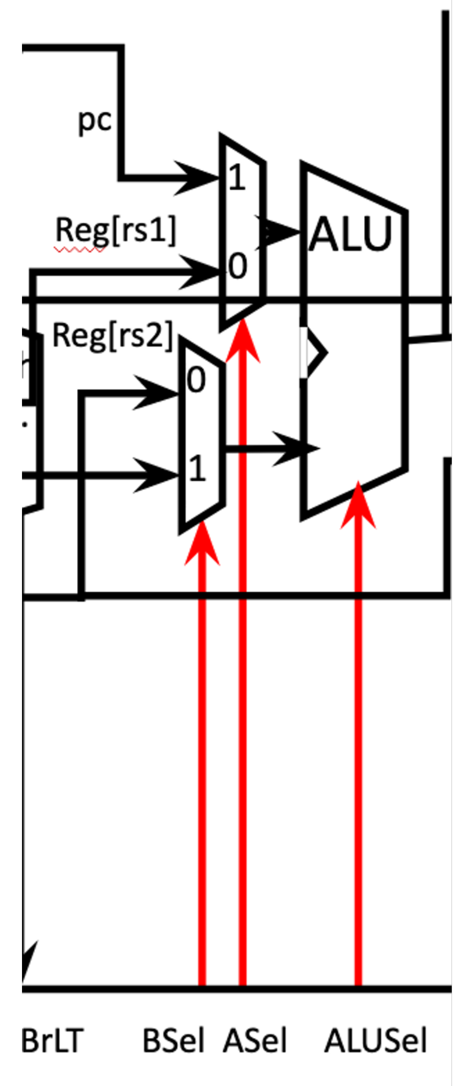
ADD: BrUn



- When we compare R[rs1] and R[rs2], should the comparison be signed (0), or unsigned (1)?
  - We aren’t doing a branch !
  - This value doesn’t matter
- BrUn = *

UNIVERSITY of WASHINGTON

ADD: ASel, BSel, ALUSel



- Which operands do we want to operate on?
  - ADD requires rs1 and rs2
- ASel = 0 (rs1)
- BSel = 0 (rs2)

- What operation do we want to perform?
  - ADD == uh… add ?
- ALUSel = "Add"
  - but wait… that's not binary, how does that work?

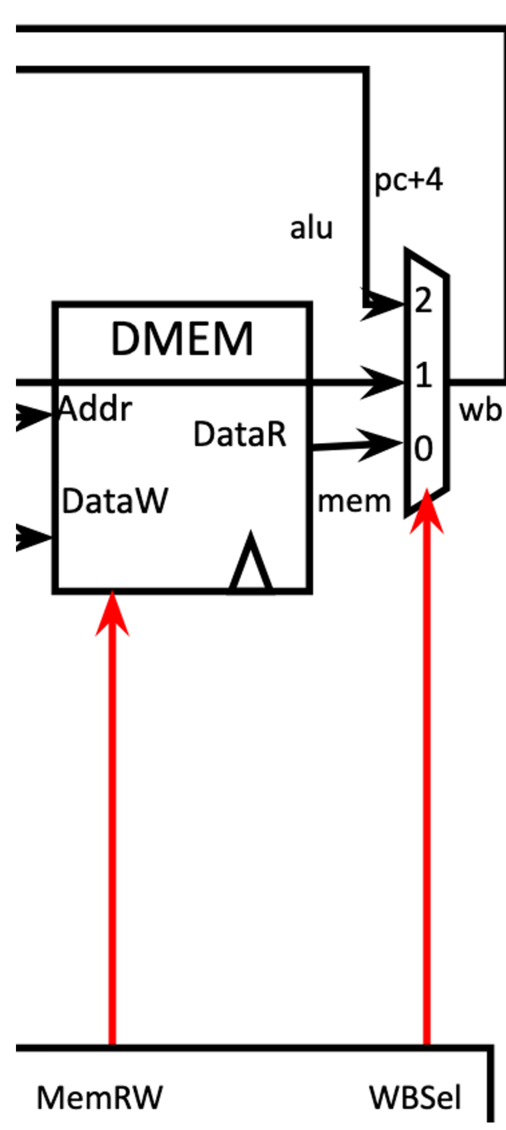
W UNIVERSITY of WASHINGTON

ALUSel

- For diagramming purposes, we set ALUSel on examples and exam questions to an english value (add, sub, or, etc.)
- In your CPU, it'll have a binary value (and so will all other signals!)
- The mapping between english words and binary values depends on how you build your ALU!
  - These mappings are arbitrary! As long as you're consistent (all add-based instructions have the same ALUSel) things will work just fine

UNIVERSITY of WASHINGTON

ADD: MemRW, WBSel



- Are we reading (0) or writing (1) memory?
  - Wait, we're not doing anything with memory. Can this be a "don't care" value?
    - NO NO NO NO NO ! :(
    - We never want to "accidentally" write memory! This has to be a "passive read".
- MemRW = 0
- What value do we want to write back to rd?
  - ALU Out!
- WBSel = 2

UNIVERSITY of WASHINGTON

ADD: Control Signals

Here are the signals and values we've compiled for our ADD instruction:

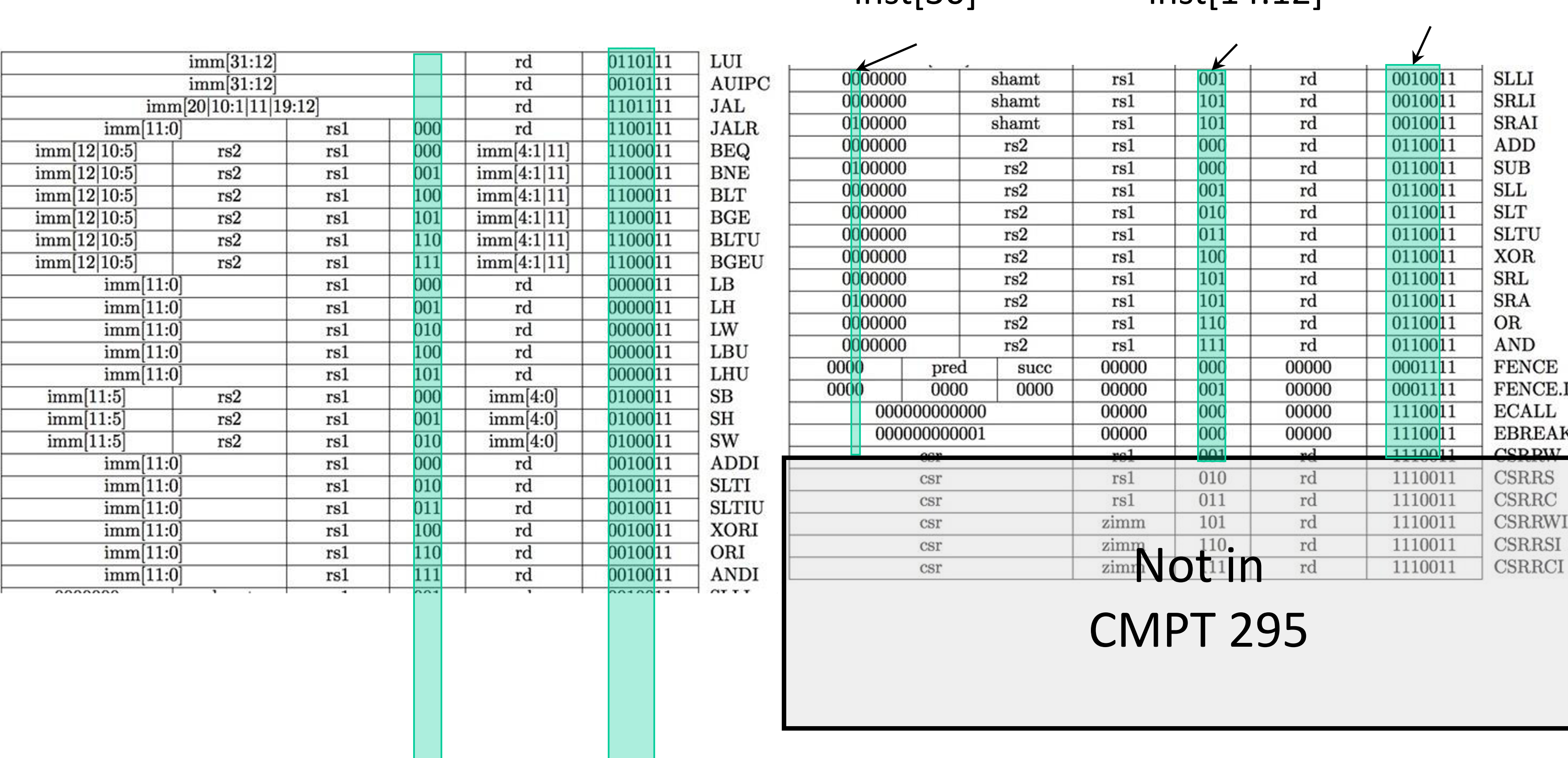
Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	*	*	+4	*	*	Reg	Reg	Add	Read	1 (Y)	ALU

(green = left 3 cols = control INPUTS)

(orange = right 9 cols = control OUTPUTS)

UNIVERSITY of WASHINGTON

RV32I, a nine-bit ISA!



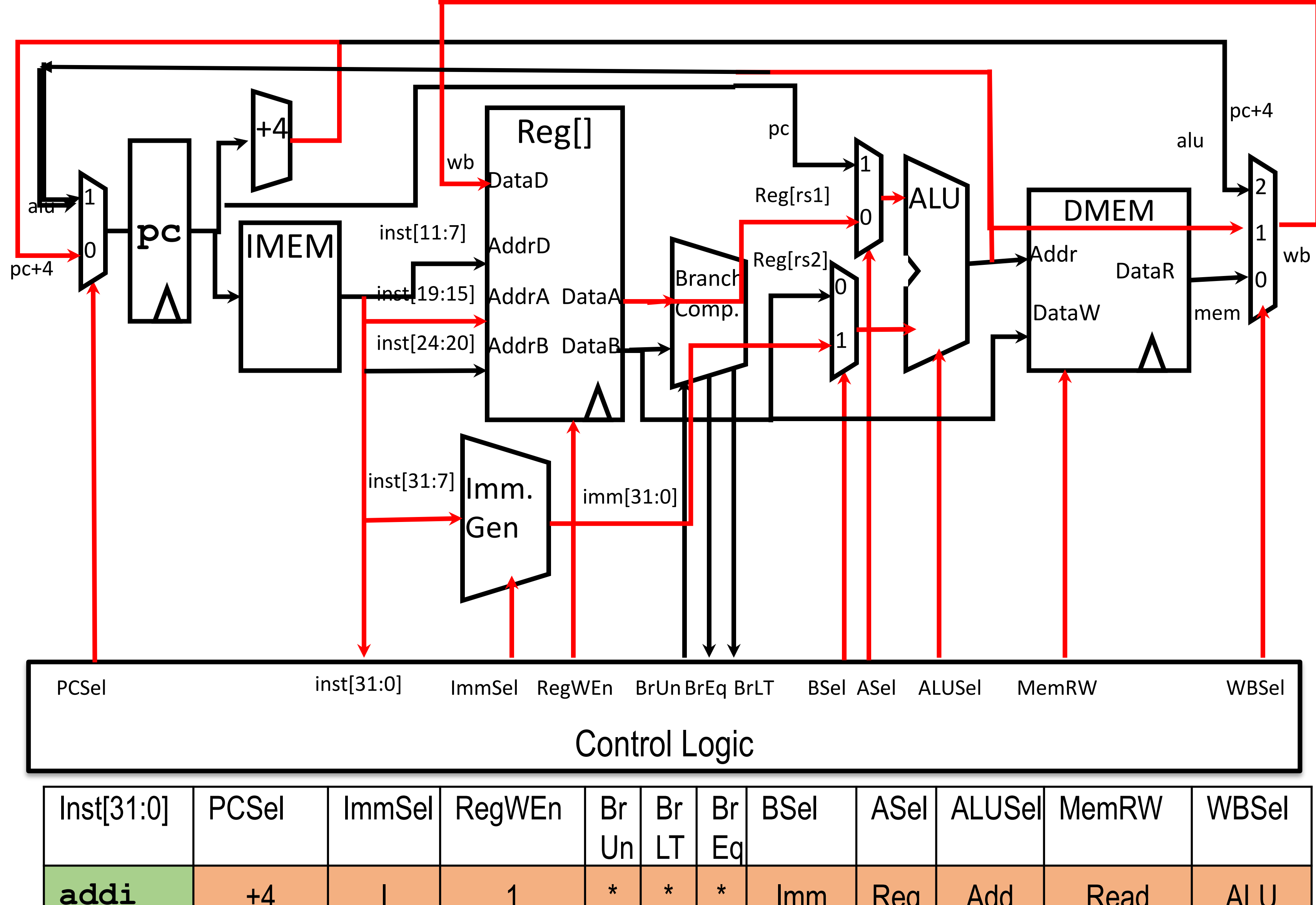
imm / funct7	rs2	rs1	funct3	rd	opcode	Instruction
imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	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	BGEU
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

funct7 / imm	rs2 / shamt	rs1	funct3	rd	opcode	Instruction
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

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

UNIVERSITY of WASHINGTON

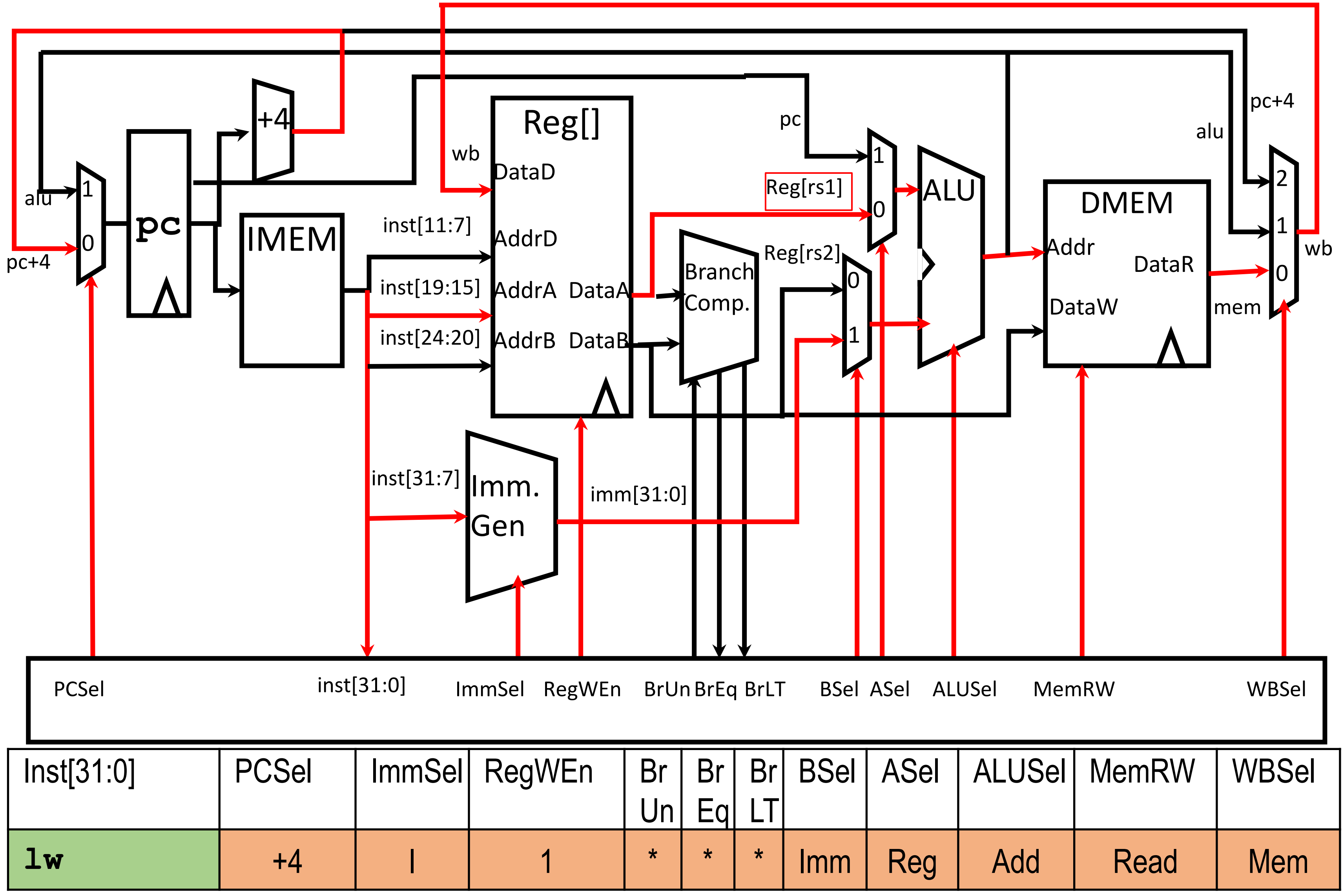
addi datapath



Inst[31:0]	PCSel	ImmSel	RegWEn	Br Un	Br LT	Br Eq	BSel	ASel	ALUSel	MemRW	WBSel
addi	+4	I	1	*	*	*	Imm	Reg	Add	Read	ALU

UNIVERSITY of WASHINGTON

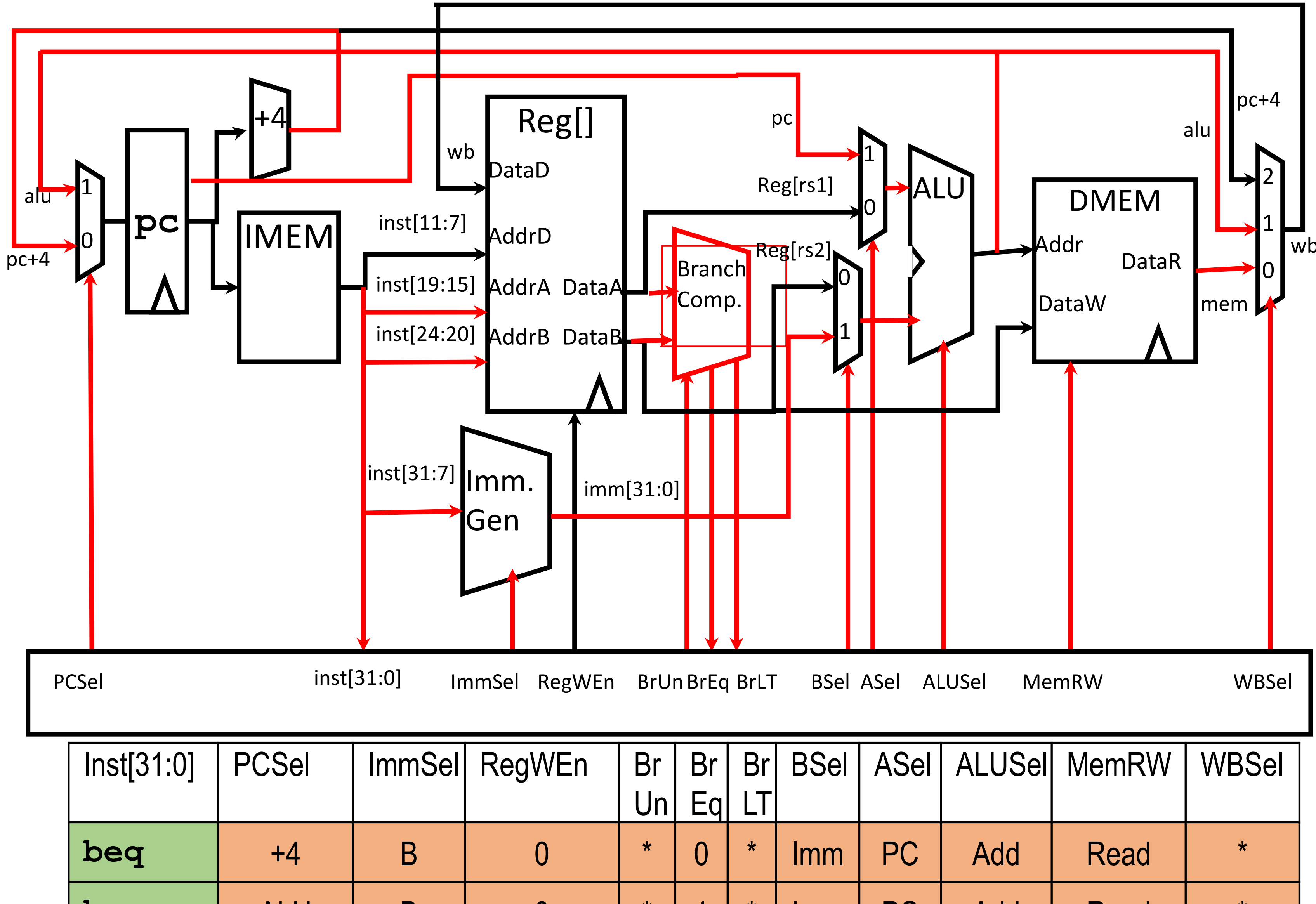
lw datapath



Inst[31:0]	PCSel	ImmSel	RegWEn	Br Un	Br Eq	Br LT	BSel	ASel	ALUSel	MemRW	WBSel
lw	+4	I	1	*	*	*	Imm	Reg	Add	Read	Mem

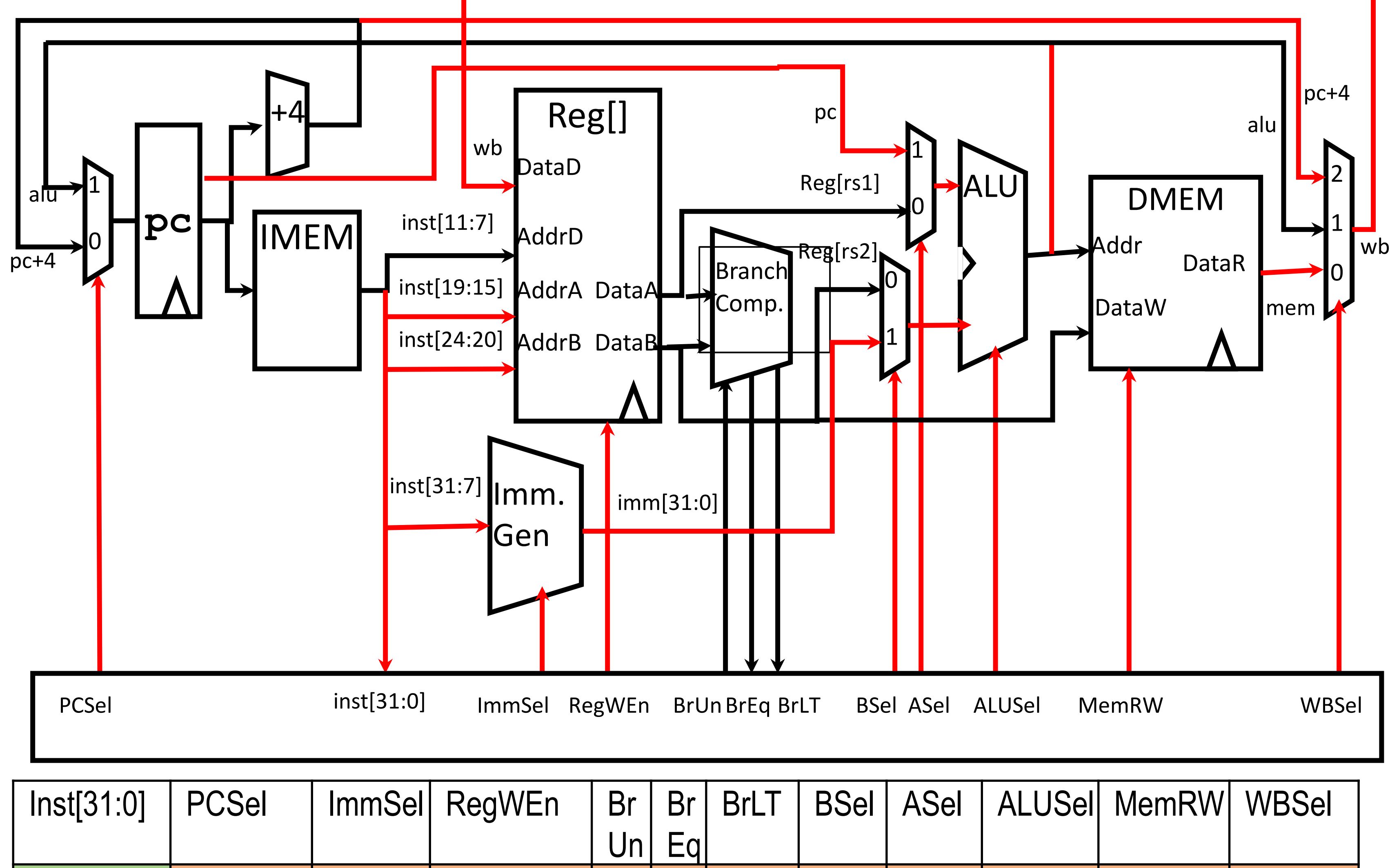
UNIVERSITY of WASHINGTON

Br datapath



Inst[31:0]	PCSel	ImmSel	RegWEn	Br Un	Br Eq	Br LT	BSel	ASel	ALUSel	MemRW	WBSel
beq	+4	B	0	*	0	*	Imm	PC	Add	Read	*
beq	ALU	B	0	*	1	*	Imm	PC	Add	Read	*

Jal datapath



Inst[31:0]	PCSel	ImmSel	RegWEn	Br Un	Br Eq	BrLT	BSel	ASel	ALUSel	MemRW	WBSel
jal	ALU	J	1	*	*	*	Imm	PC	Add	Read	PC+4

UNIVERSITY of WASHINGTON

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

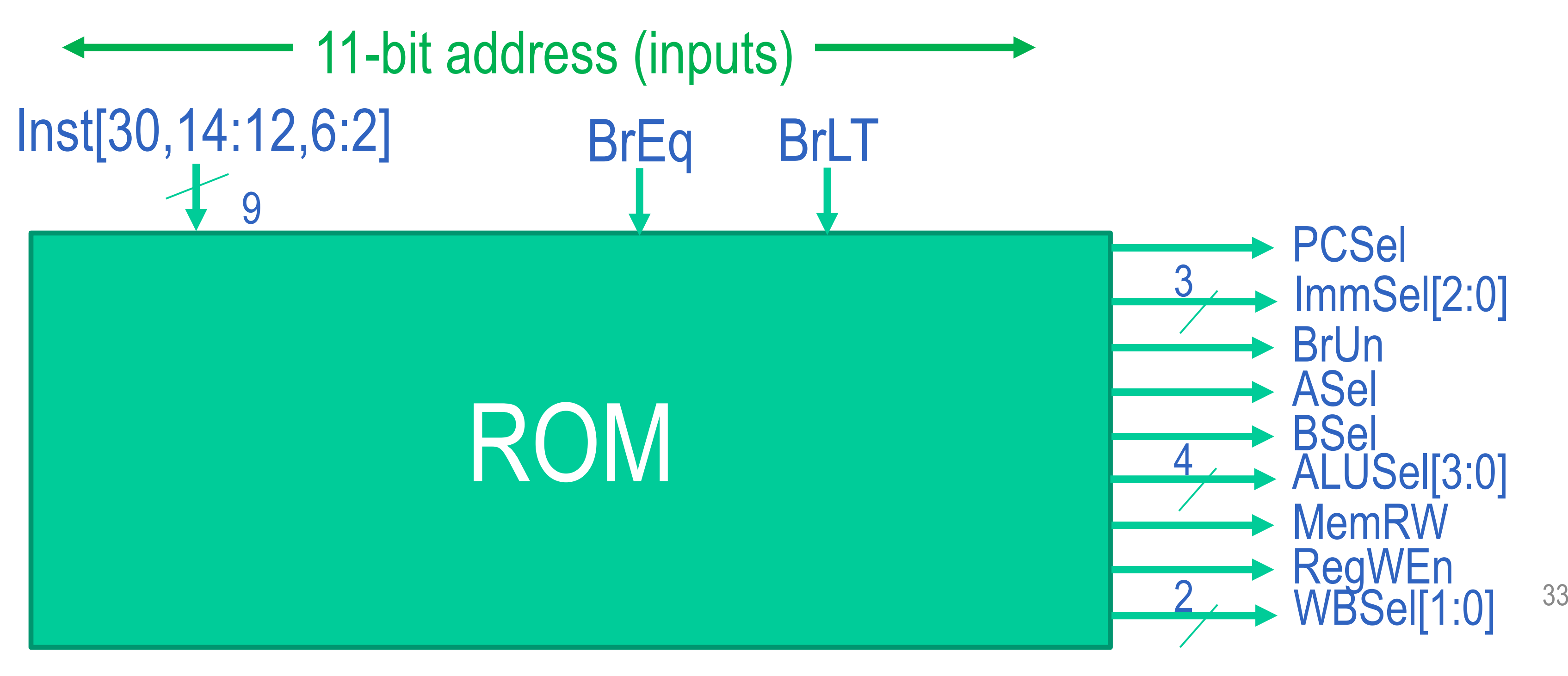
UNIVERSITY of WASHINGTON

Control Construction Options

- ROM
  - "Read-Only Memory"
  - Regular structure (like previous slide's table)
  - 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
  - Not easily changeable/re-programmable because requires modifying hardware
    - But! Likely less expensive, more complex

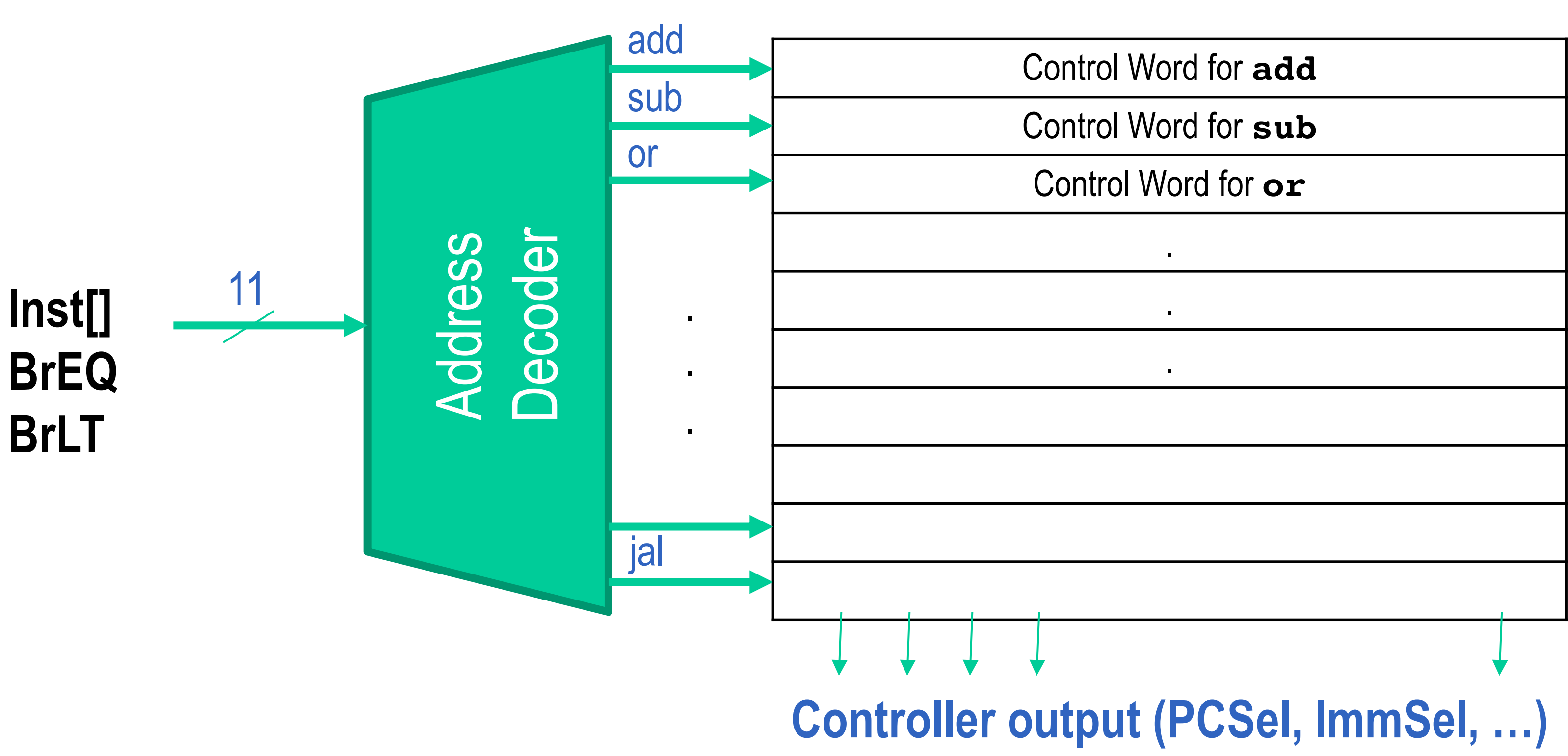
UNIVERSITY of WASHINGTON

ROM-based Control



UNIVERSITY of WASHINGTON

ROM Controller Implementation



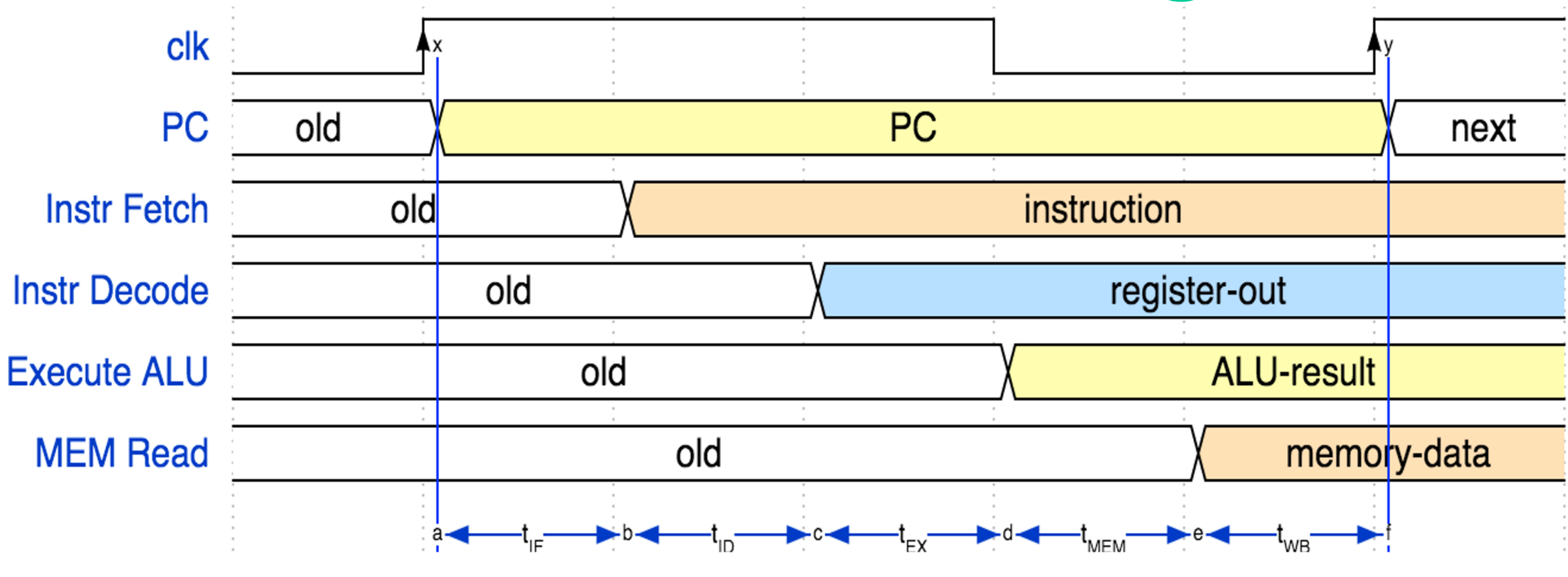
UNIVERSITY of WASHINGTON

Agenda

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

UNIVERSITY of WASHINGTON

Instruction Timing



IF	ID	EX	MEM	WB	Total
IMEM	Reg Read	ALU	DMEM	Reg W	
200 ps	100 ps	200 ps	200 ps	100 ps	800 ps

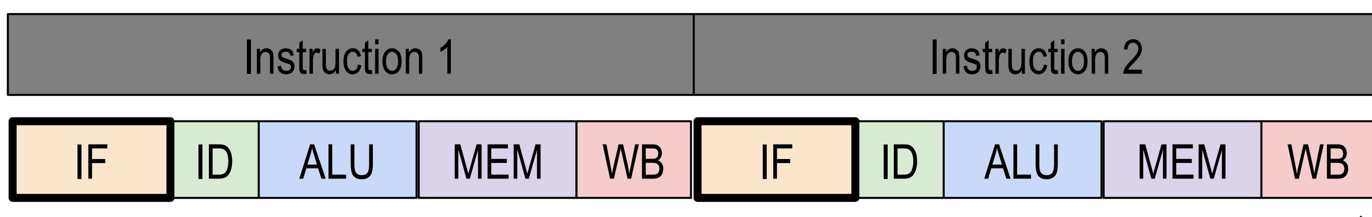
1. Instruction Fetch
2. Decode/ Register Read
3. Execute
4. Memory
5. Reg. Write

UNIVERSITY of WASHINGTON

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		X	600ps
lw	X	X	X	X	X	800ps
sw	X	X	X	X		700ps

- Maximum clock frequency
  - f_{max} = 1/800ps = 1.25 GHz
- Most blocks idle most of the time! ex. "IF" active every 600ps



UNIVERSITY of WASHINGTON

Performance Measures

- In our example, CPU 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?

UNIVERSITY of WASHINGTON

"Iron Law" of Processor Performance

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

UNIVERSITY of WASHINGTON

Instructions per Program

Determined by

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

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

UNIVERSITY of WASHINGTON

(Average) Clock cycles per Instruction, or CPI

Determined by

- ISA and processor implementation (or *microarchitecture*)
  - E.g. for "our" single-cycle RISC-V design, CPI = 1
- Complex instructions (e.g. **`strcpy`**), CPI >> 1
  - True for most CISC languages
- Superscalar processors, CPI < 1 (next lecture)

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

UNIVERSITY of WASHINGTON

Time per Cycle (1/Frequency)

Determined by

- Processor microarchitecture (processor critical path)
- Technology (e.g. transistor size)
- Power budget (lower voltages reduce transistor speed)

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

UNIVERSITY of WASHINGTON

Speed Trade-off 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! Why? Each instruction is less complex! (~2.5 B instructions = 1 A instruction)

UNIVERSITY of WASHINGTON

Poll

If we reduce the time a program takes to execute by pipelining our CPU, which factor is likely to shrink?

A) **Instructions per program**
B) **Cycles per instruction**
C) **Time per cycle**

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

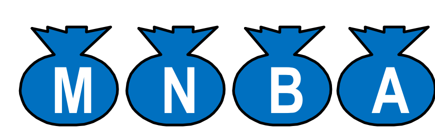
UNIVERSITY of WASHINGTON

Agenda

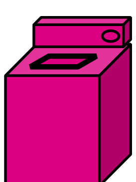
- Quick Datapath Review
- Control Implementation
- Administrivia
- Performance Analysis
- Pipelined Execution
- Pipelined Datapath

UNIVERSITY of WASHINGTON

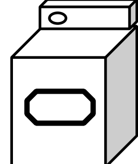
Pipeline Analogy: Doing Laundry



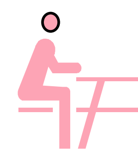
- Minh, Nick, Brandon, and Ali each have one load of clothes to wash, dry, fold, and put away
  - Washer takes 30 minutes



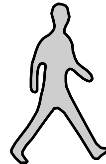
  - Dryer takes 30 minutes



  - “Folder” takes 30 minutes

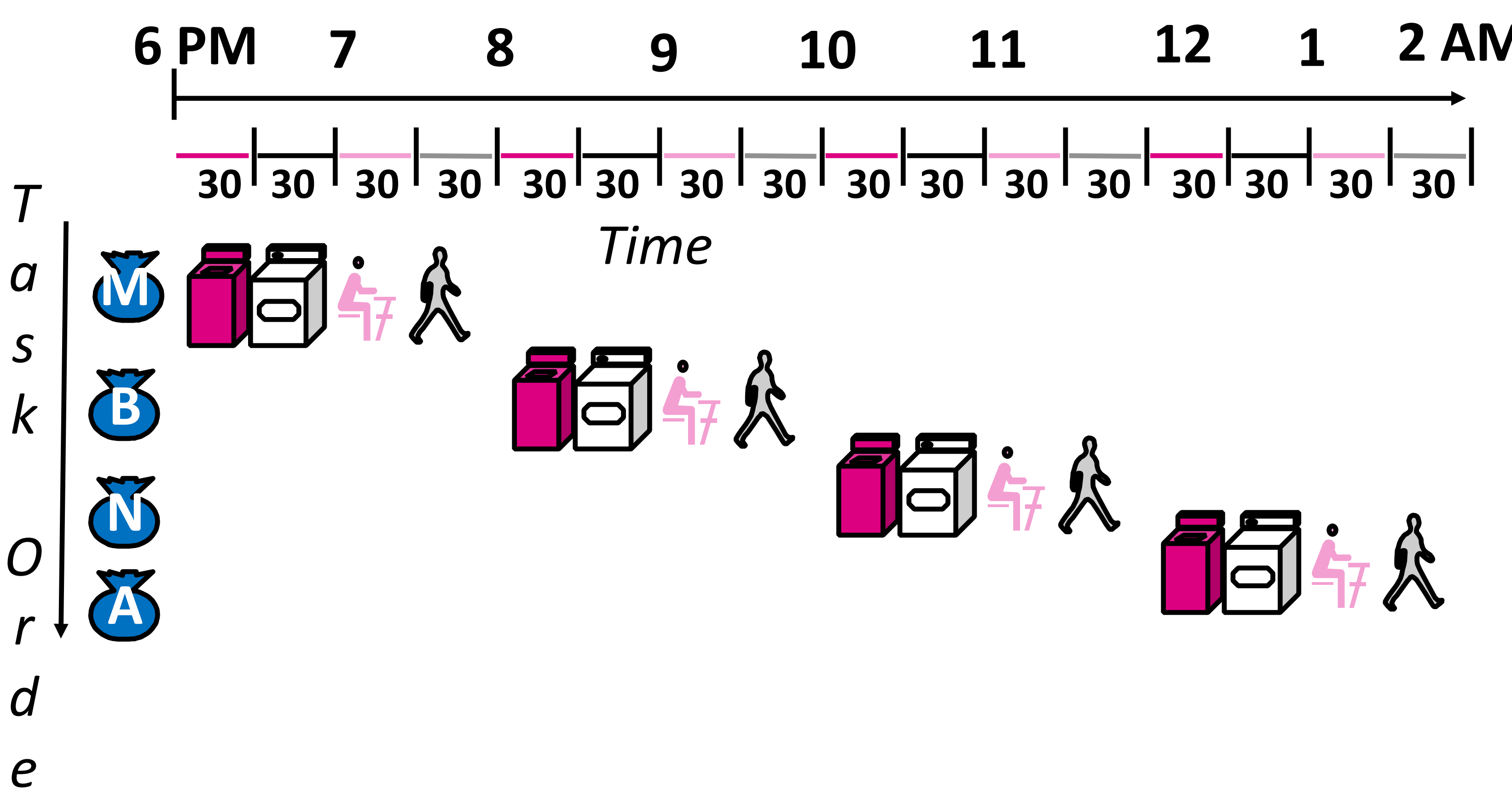


  - “Stasher” takes 30 minutes to put clothes into drawers



UNIVERSITY of WASHINGTON

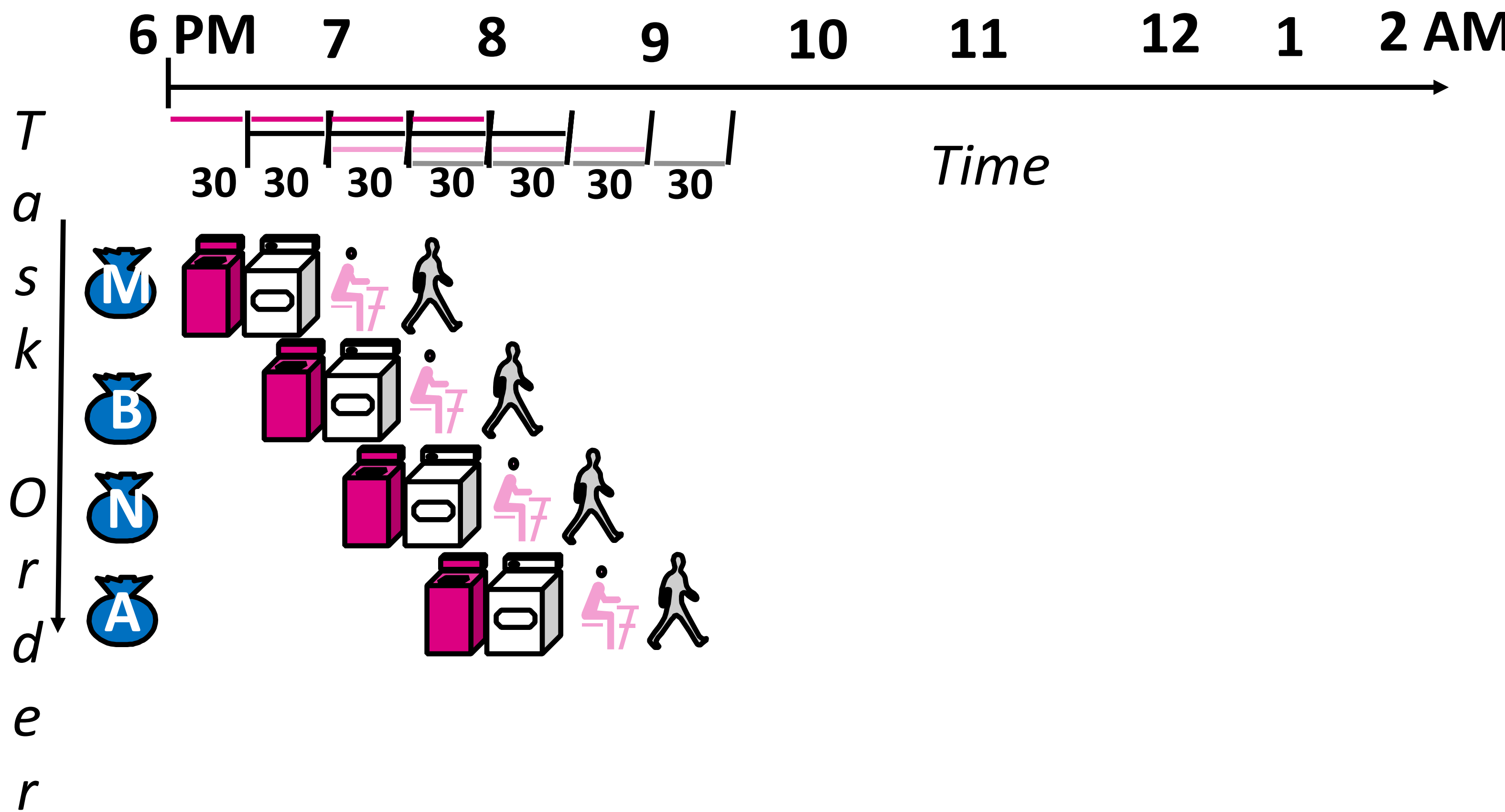
Sequential Laundry



- Sequential laundry takes 8 hours for 4 loads
- 1 load finishes every 2 hours,

UNIVERSITY of WASHINGTON

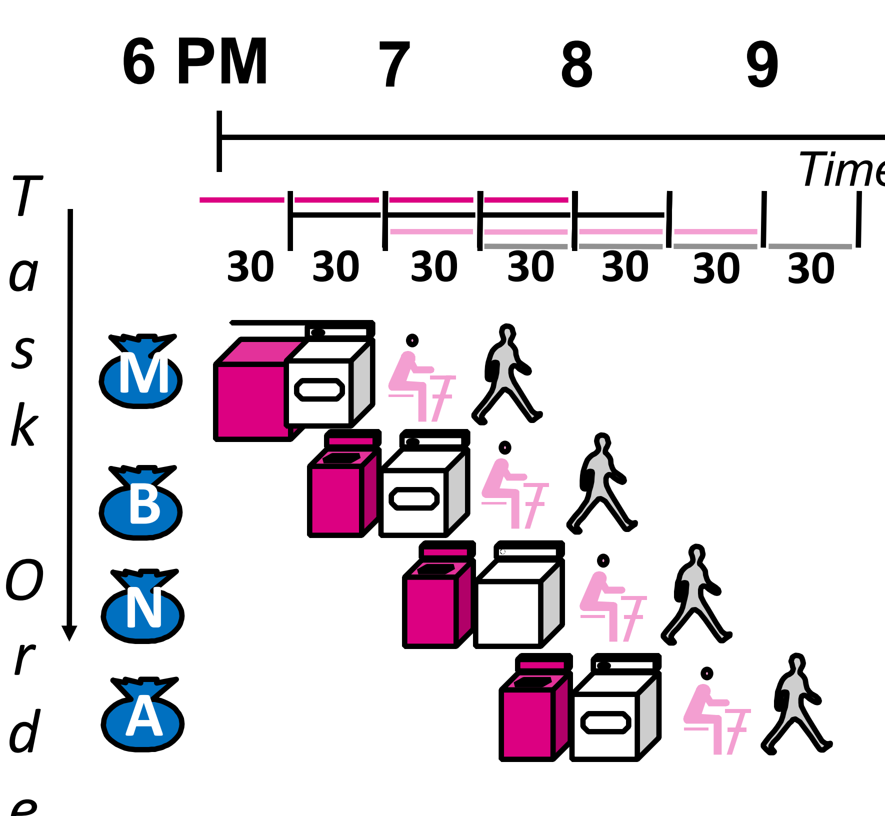
Pipelined Laundry



- Pipelined laundry takes 3.5 hours for 4 loads!
- 1 load finishes every half hour (after the first load, which takes 2 hours)

UNIVERSITY of WASHINGTON

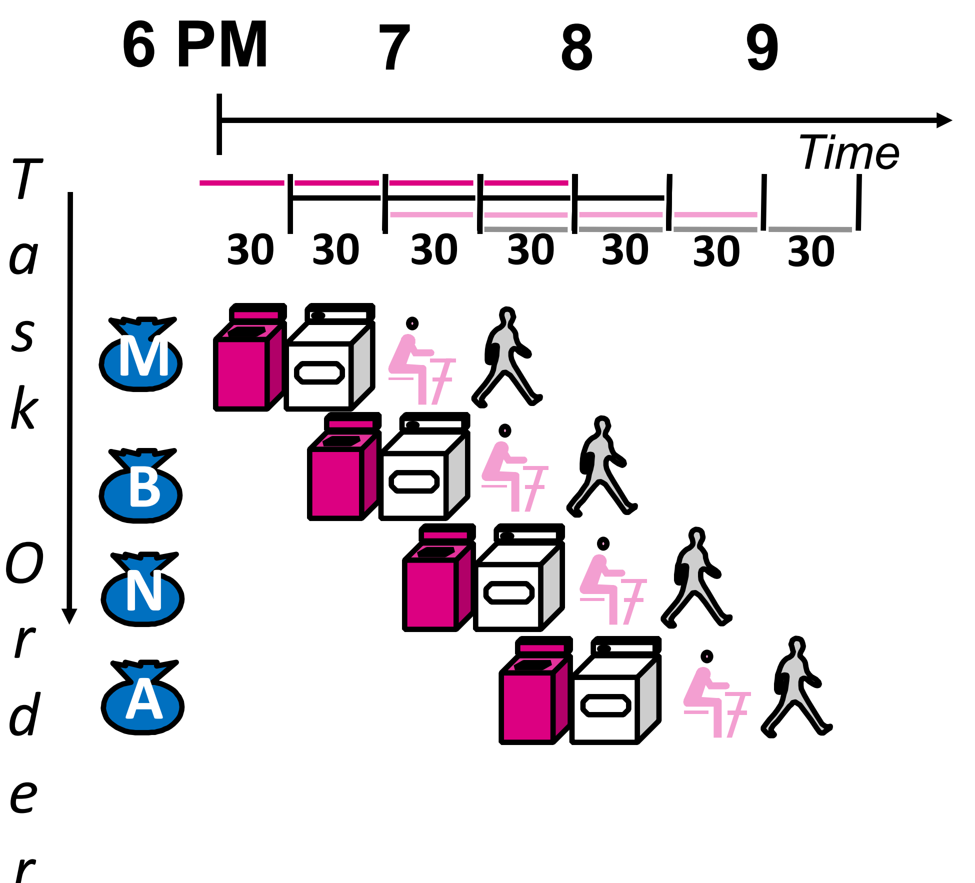
Pipelining Lessons (1/2)



- Pipelining doesn't decrease *latency* of single task; it increases *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 which gives 2.3X speedup v. potential 4X in this example

UNIVERSITY of WASHINGTON

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

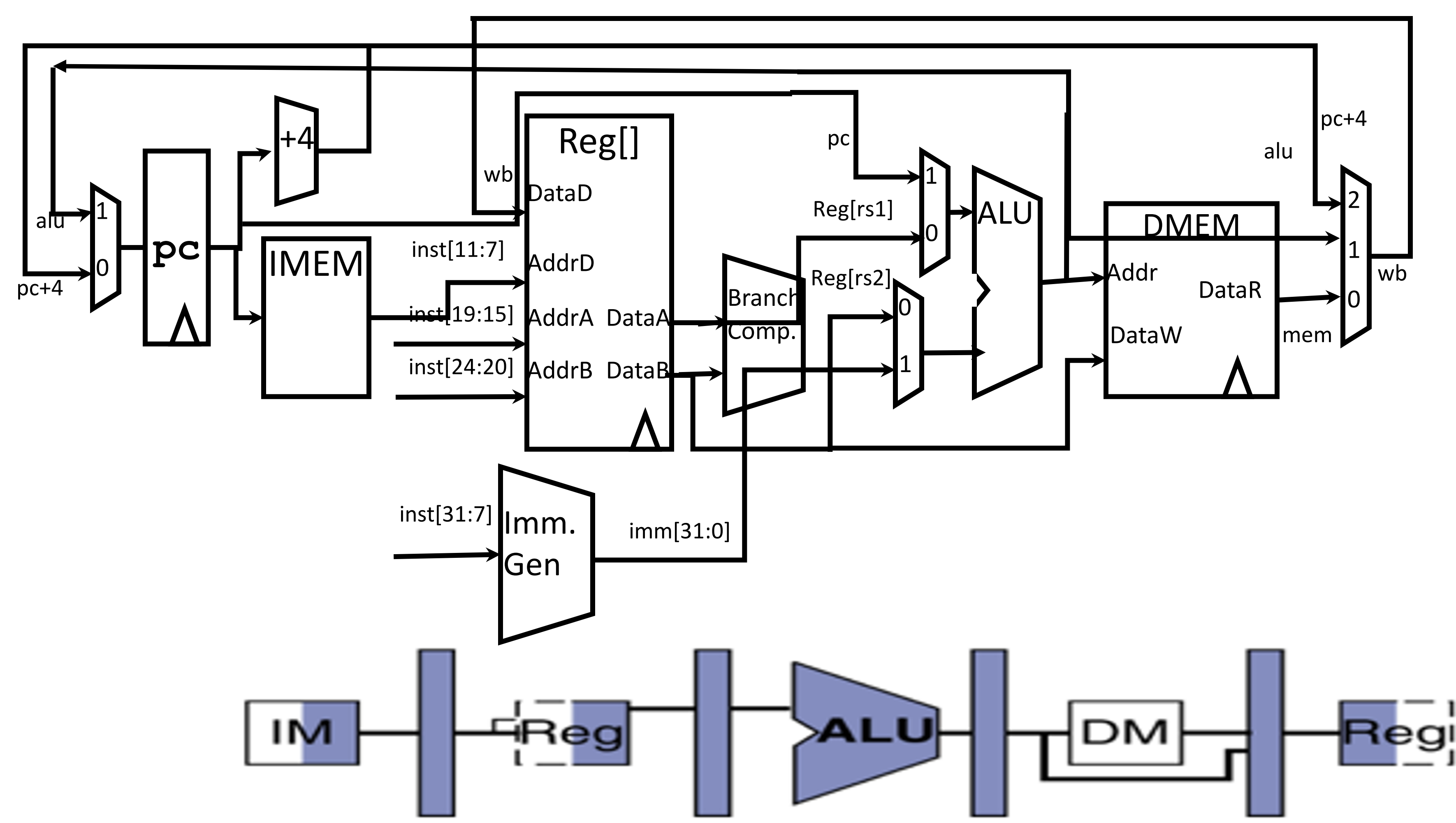
UNIVERSITY of WASHINGTON

Agenda

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

UNIVERSITY of WASHINGTON

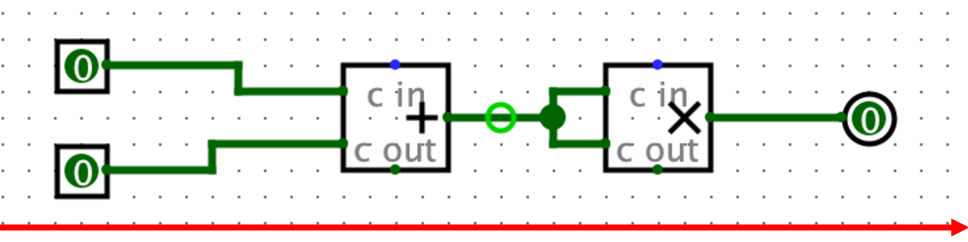
Pipelining RISC-V CPU



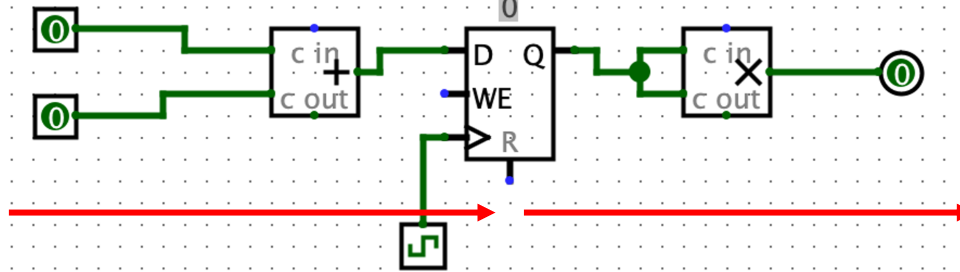
UNIVERSITY of WASHINGTON

Quick review: Circuit pipelining!

- When we calculate cycle time, or critical path, we do so between state elements, inputs, and outputs.



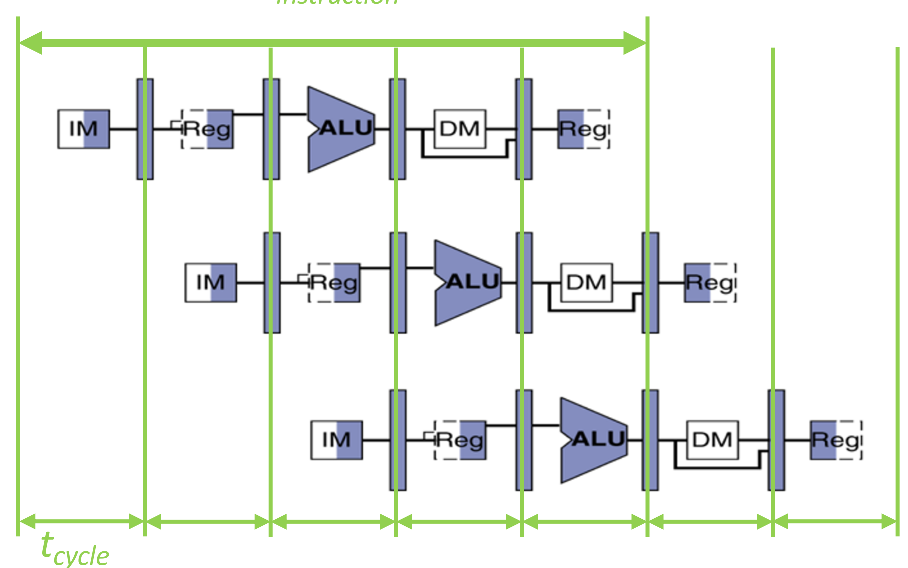
- Adding registers between circuit components *decreases* our critical path and *increases* our frequency



UNIVERSITY of WASHINGTON

Pipelining with RISC-V

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



instruction sequence

add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3

UNIVERSITY of WASHINGTON

Pipeline Performance

- Use T_c ("time between completion of instructions") to measure speedup
- Speedup due to increased *throughput*
  - *Latency* for each instruction does not decrease, in fact it may increase if our stages are uneven!
- It takes longer for the *first* instruction to finish, but every instruction after finishes *faster*!

UNIVERSITY of WASHINGTON

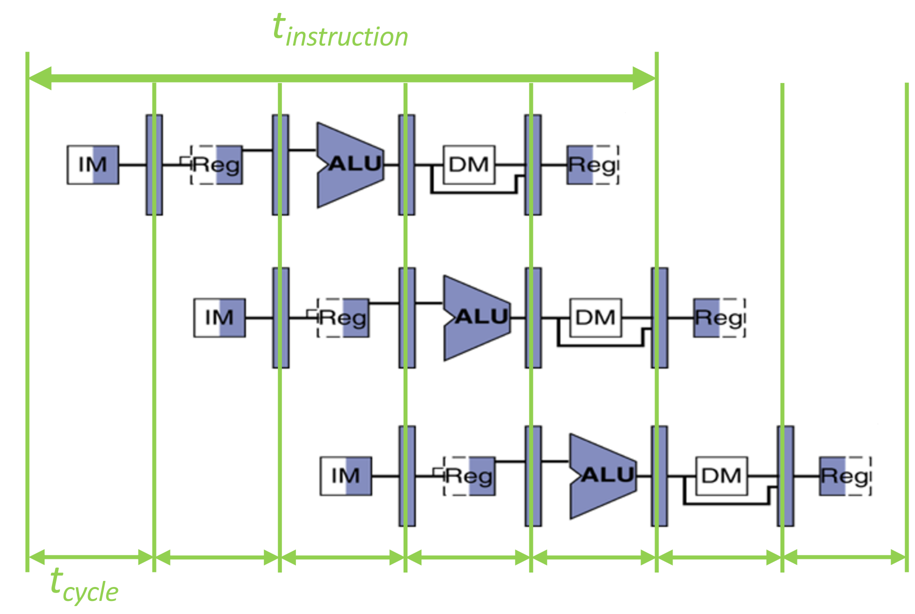
Pipelining with RISC-V

instruction sequence

add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3

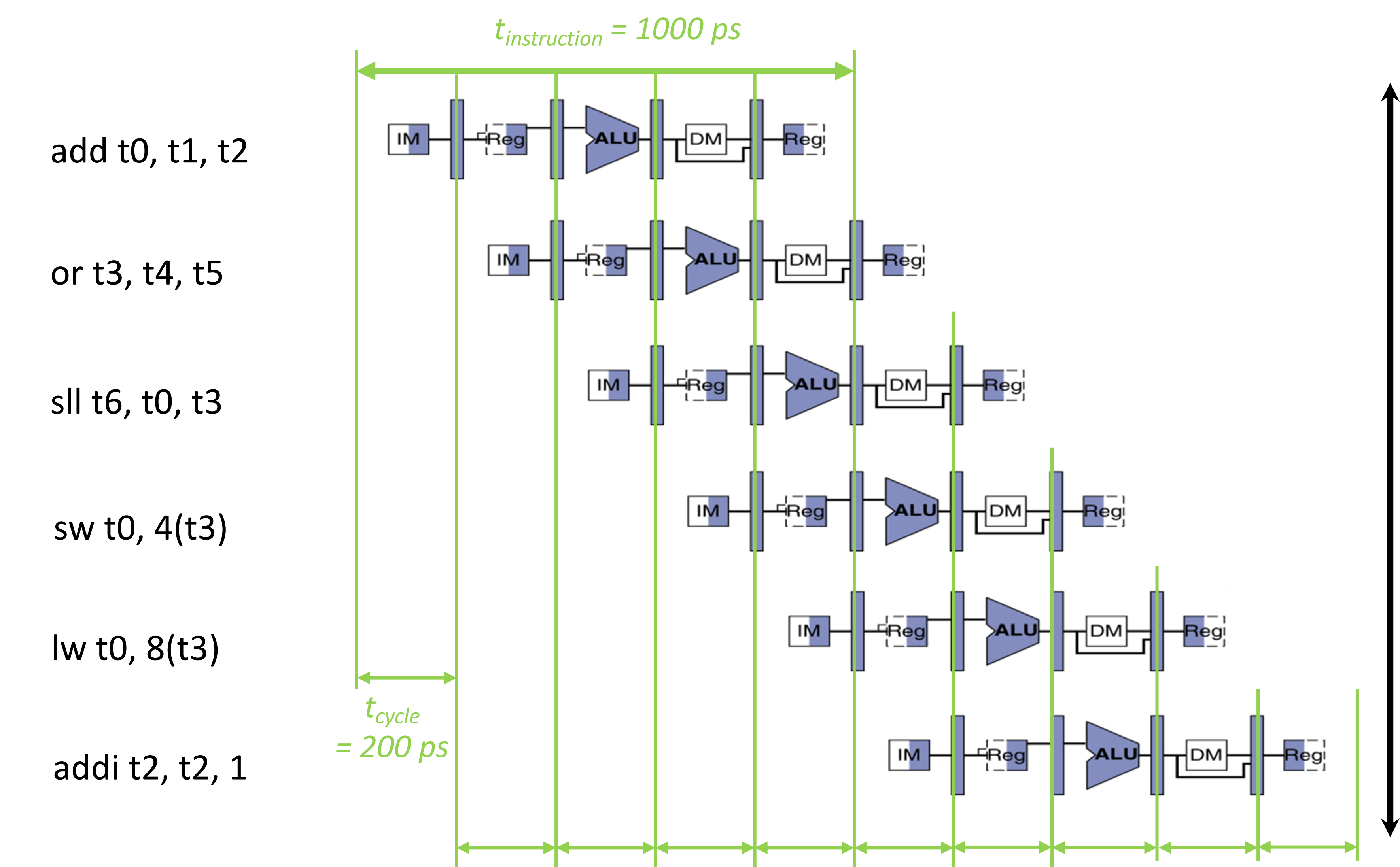


	Single Cycle	Pipelining
Timing	t_{step} = 100 … 200 ps	t_{cycle} = 200 ps
	Register access only 100 ps	All cycles same length
Instruction time, $t_{instruction}$	= t_{cycle} = 800 ps	1000 ps
Clock rate, f_s	1/800 ps = 1.25 GHz	1/200 ps = 5 GHz

UNIVERSITY of WASHINGTON

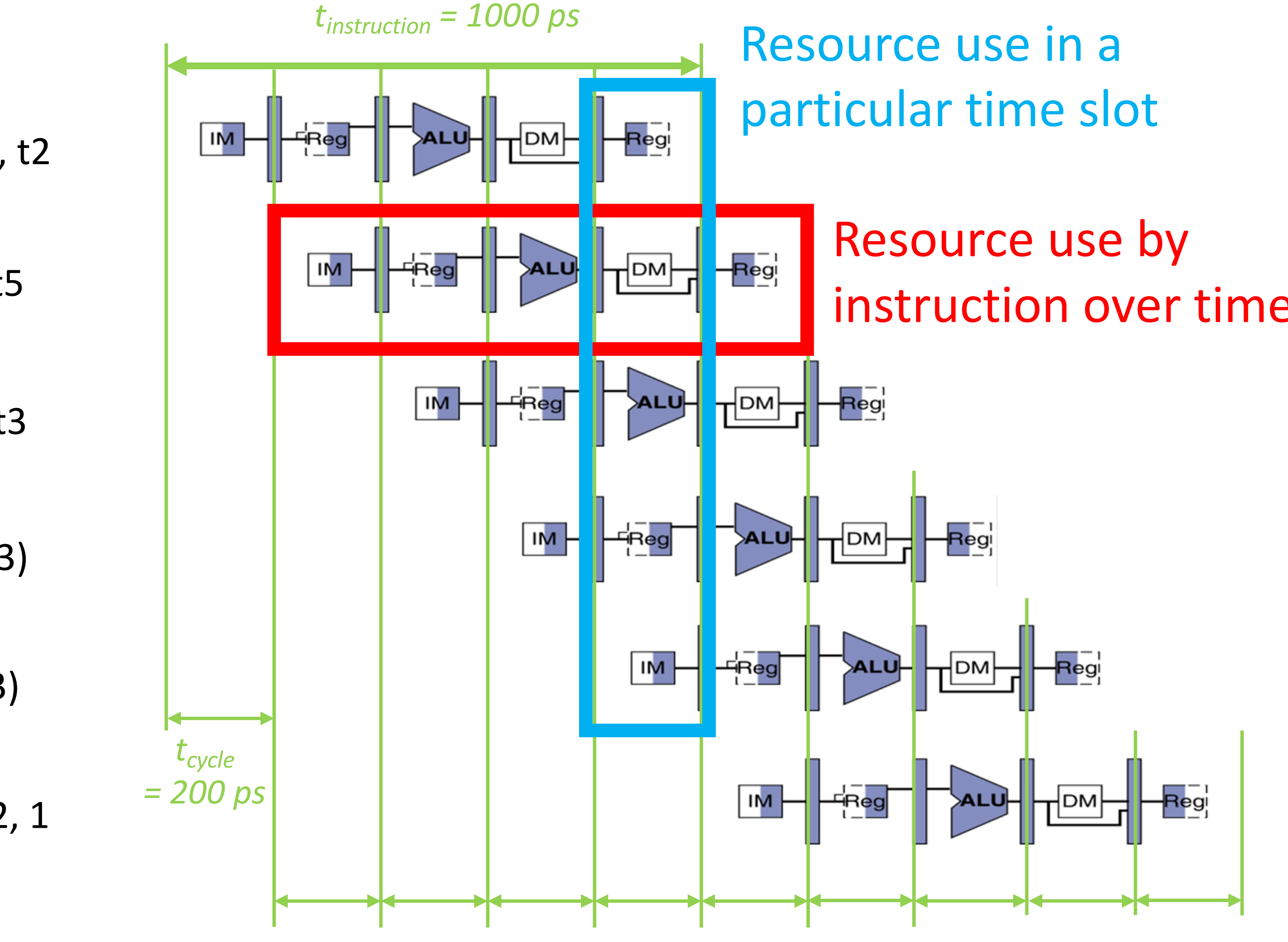
Sequential vs Simultaneous

What happens <u>sequentially</u>, what happens <u>simultaneously</u>?



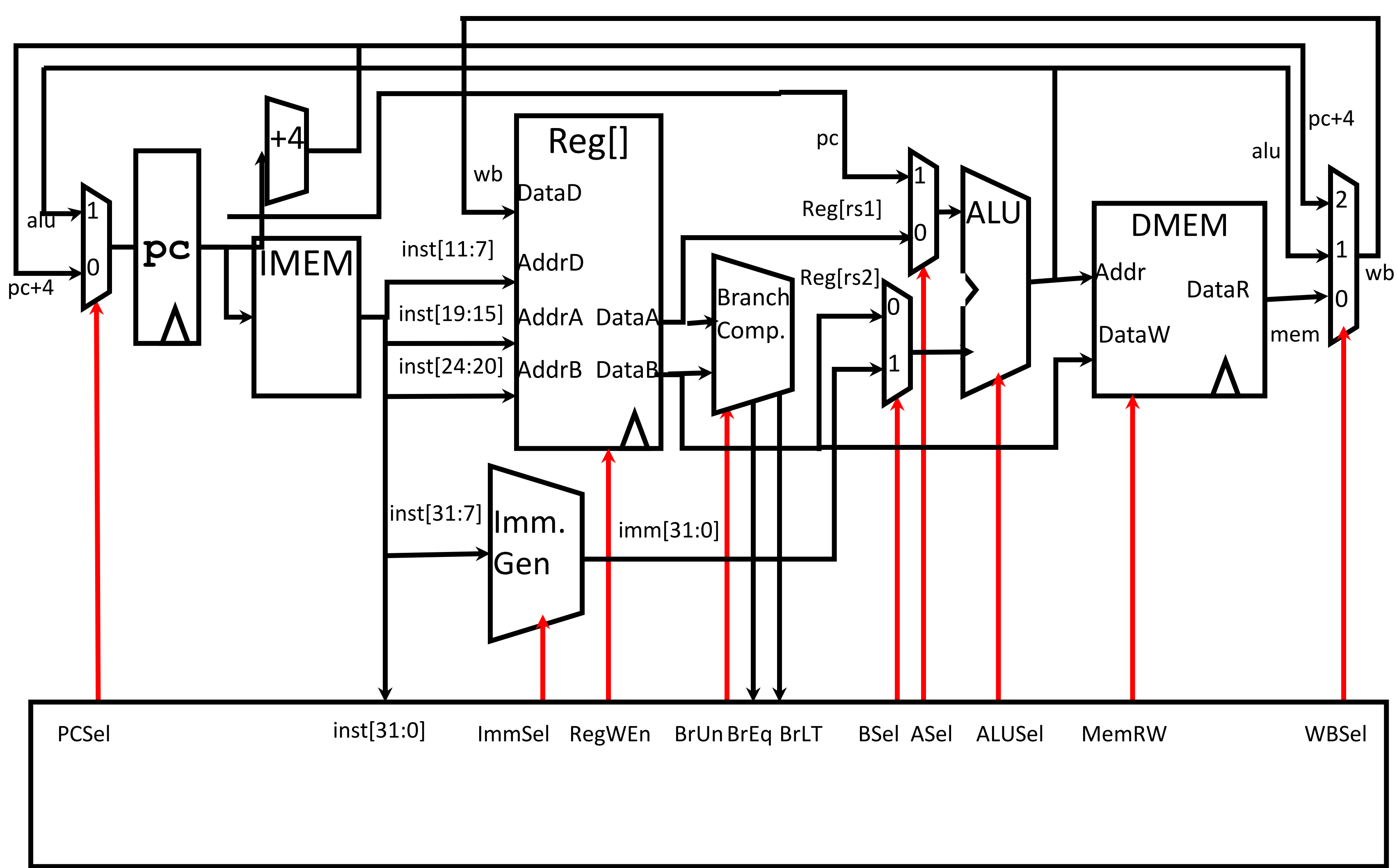
UNIVERSITY of WASHINGTON

RISC-V Pipeline



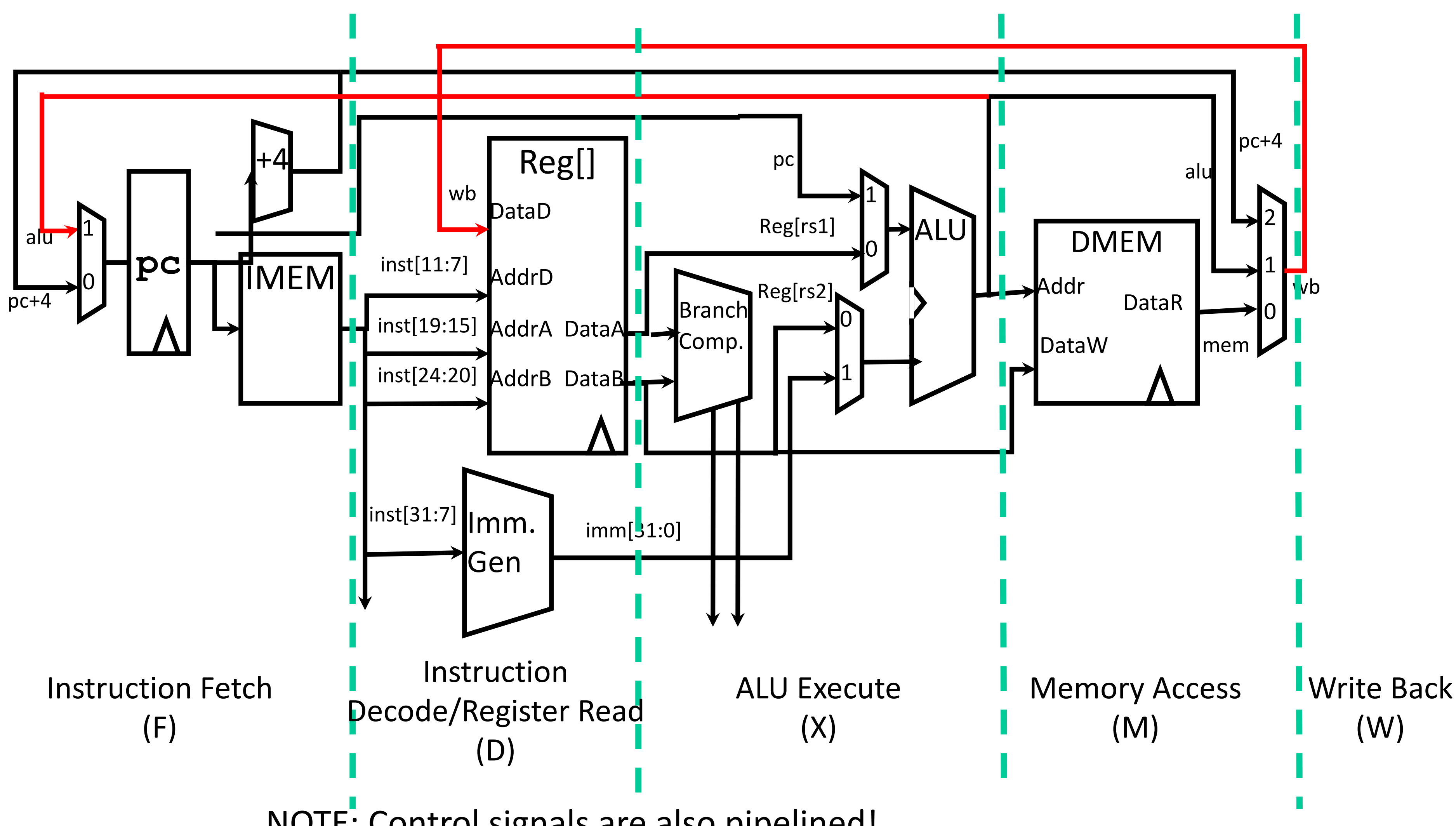
UNIVERSITY of WASHINGTON

Single-Cycle RISC-V RV32I Datapath



UNIVERSITY of WASHINGTON

Pipelining RISC-V RV32I Datapath

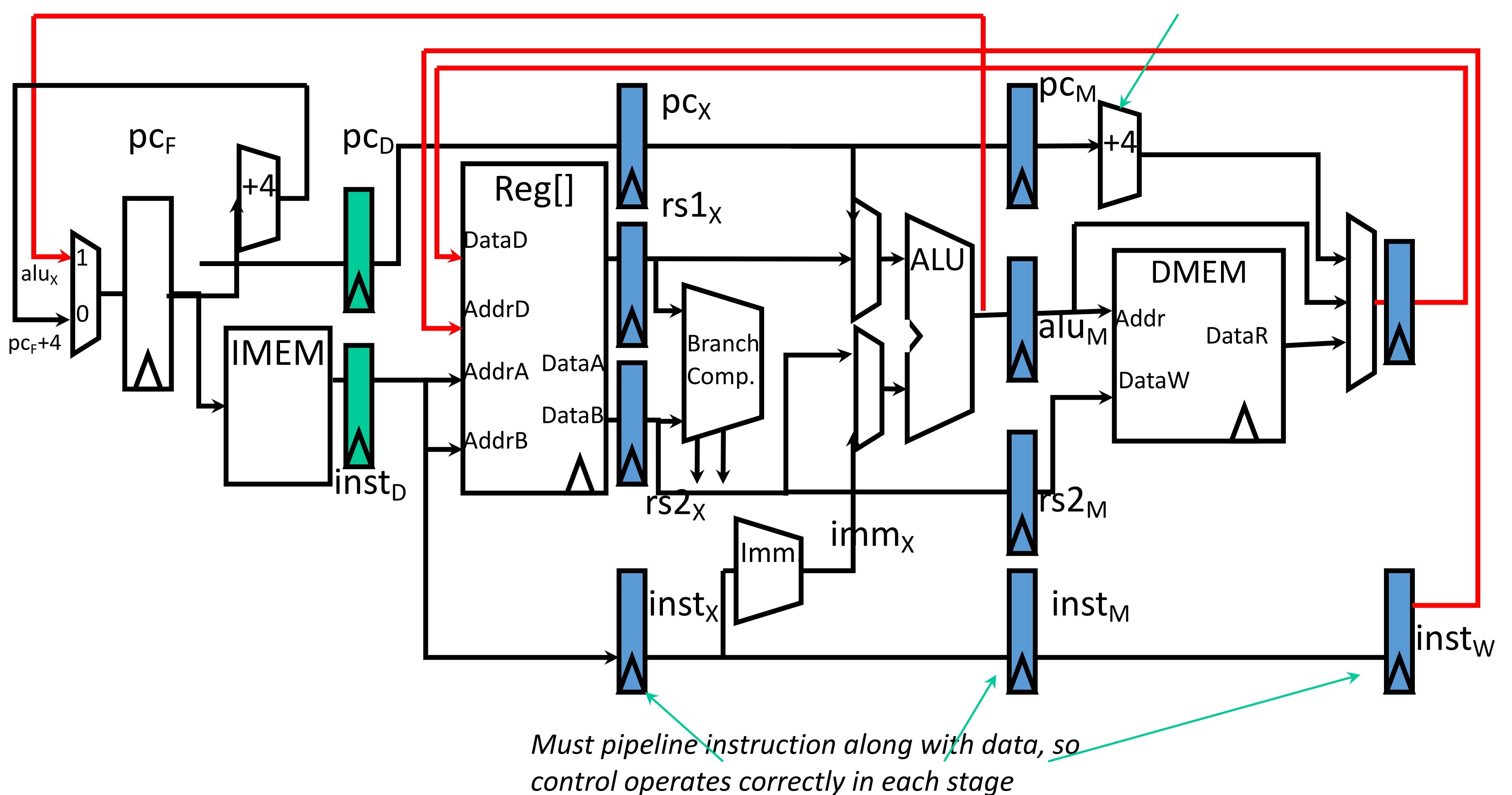


NOTE: Control signals are also pipelined!

UNIVERSITY of WASHINGTON

Pipelined RISC-V RV32I Datapath

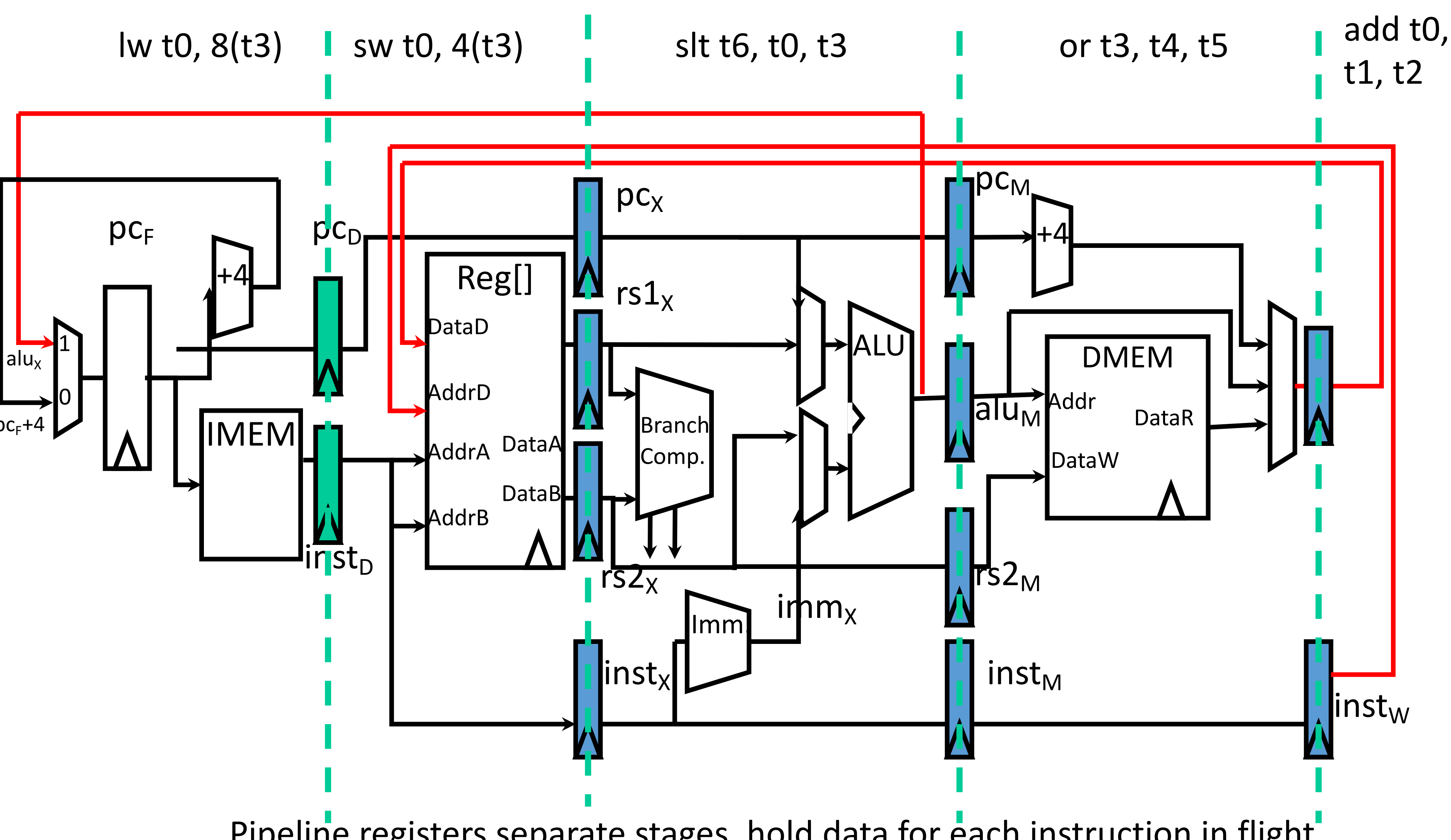
Recalculate PC+4 in M stage to avoid sending both PC and PC+4 down pipeline



Must pipeline instruction along with data, so control operates correctly in each stage

UNIVERSITY of WASHINGTON

Each stage operates on different instruction



Pipeline registers separate stages, hold data for each instruction in flight

UNIVERSITY of WASHINGTON

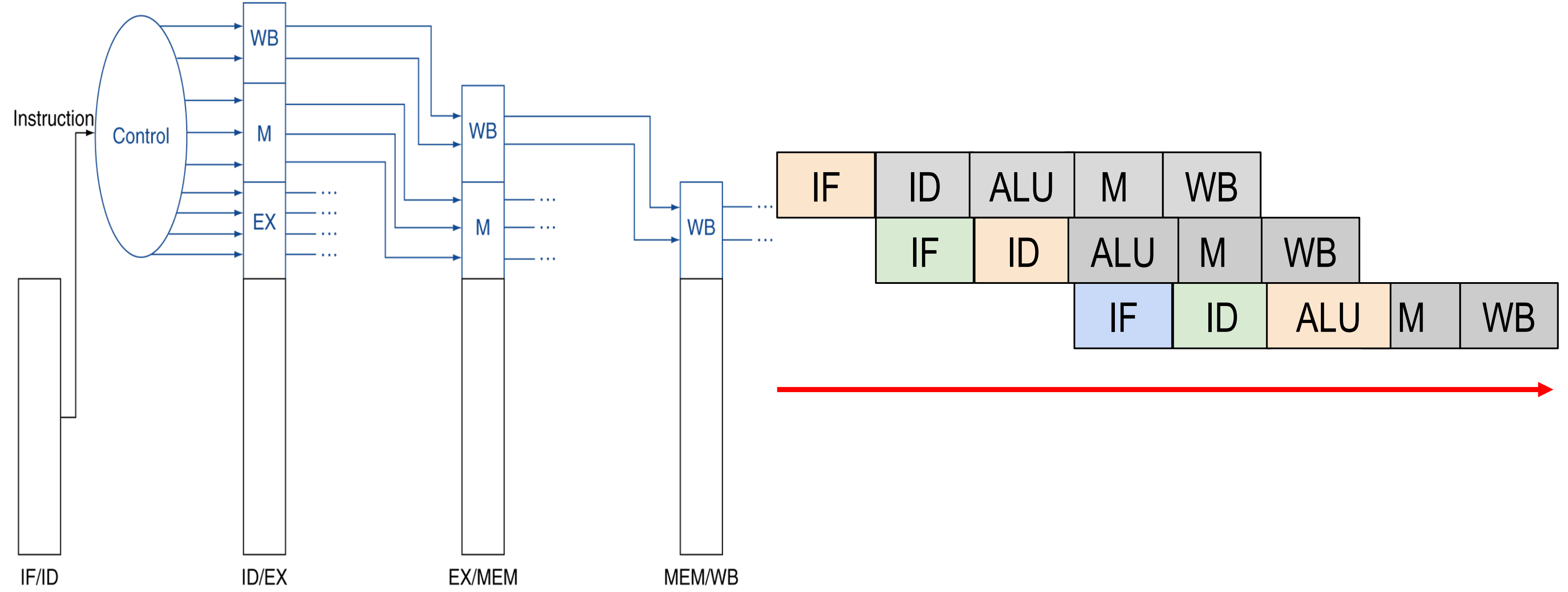
Instruction Level Parallelism (ILP)

- 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)

UNIVERSITY of WASHINGTON

Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation
  - Information is stored in pipeline registers for use by later stages
- At any given point, there are up to 5 different instructions in the datapath! We must keep track of 5 different sets of control bits!



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:

(IF+ID+EX+WB) = 600 →

(4*max_stage)/4 = 200

old/new = 600/200 = 3x 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.

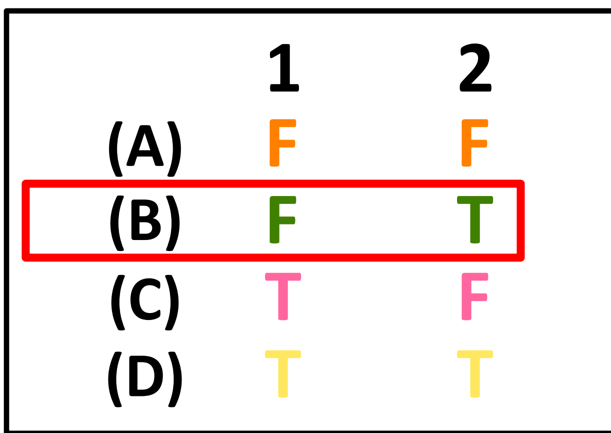
	1	2
(A)	F	F
(B)	F	T
(C)	T	F
(D)	T	T

No mem access! Latency:
IF+ID+EX+WB = 600 →
4*max_stage = 800
old/new = 600/800 = negative speedup!
800/600 = 1.33x 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.



UNIVERSITY of WASHINGTON

Summary

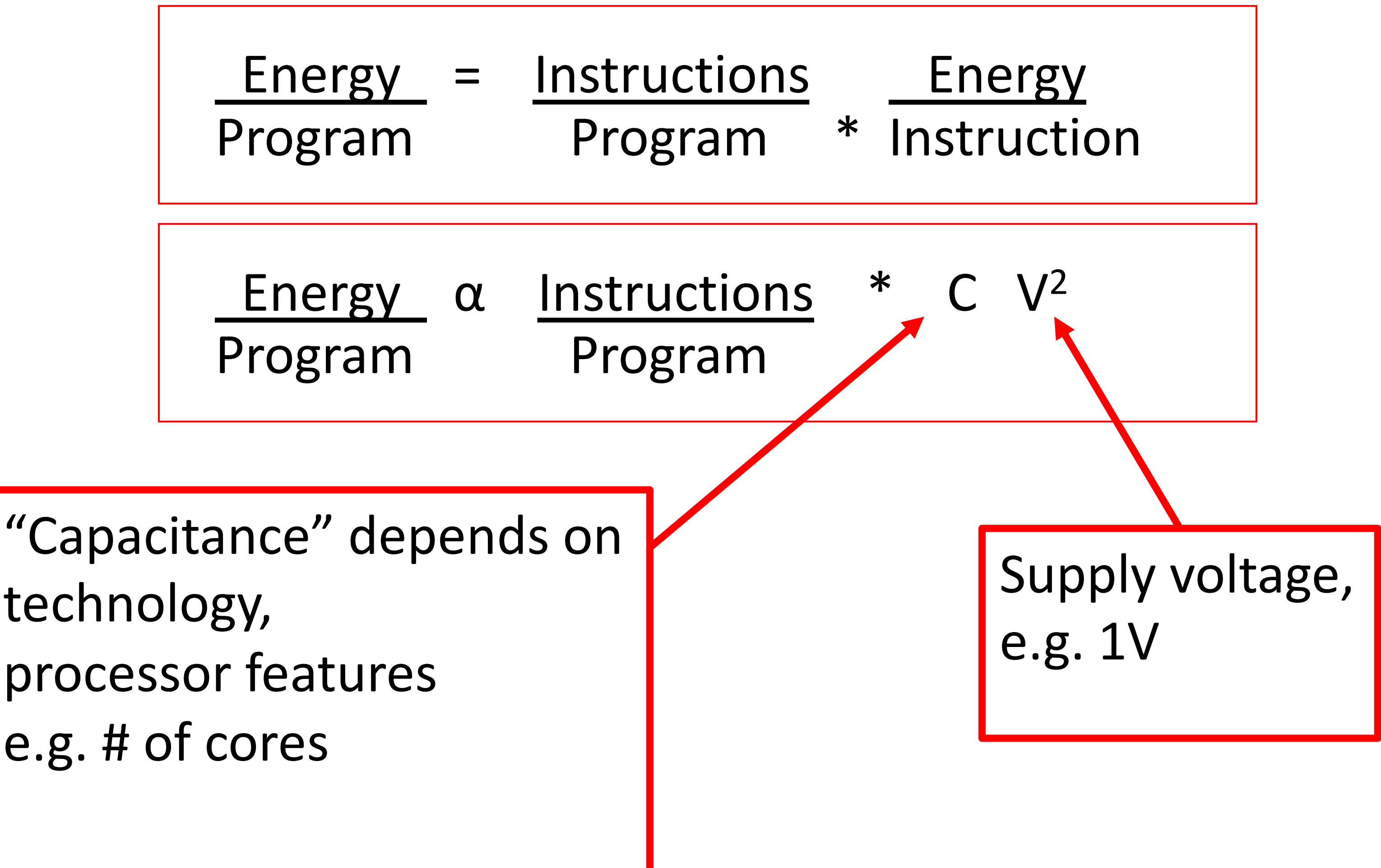
- Implementing controller for your datapath
  - Ask yourself the questions on the beginning slides!
  - Work in stages, put everything together at the end!
- Pipelining improves performance by exploiting Instruction Level Parallelism
  - 5-stage pipeline for RV32I: IF, ID, EX, MEM, WB
  - Executes multiple instructions in parallel
  - Each instruction has the same latency, but there's better throughput
  - Think: what problems does pipelining introduce? (*more on this next lecture*)

UNIVERSITY of WASHINGTON

Bonus slides (non testable)

UNIVERSITY of WASHINGTON

Energy per Task



Want to reduce capacitance and voltage to reduce energy/task

UNIVERSITY of WASHINGTON

Energy Trade-off Example

- "Next-generation" processor
  - C (Moore's Law): -15 %
  - Supply voltage, V_{sup}: -15 %
  - Energy consumption: $1 - (1-0.85)^3$ = -39 %

- Significantly improved energy efficiency thanks to
  - Moore's Law AND
  - Reduced supply voltage

UNIVERSITY of WASHINGTON

Energy "Iron Law"

Performance = Power * 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

UNIVERSITY of WASHINGTON

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