

UC Berkeley  
Teaching Professor  
Dan Garcia

# CS61C

## Great Ideas in Computer Architecture (a.k.a. Machine Structures)



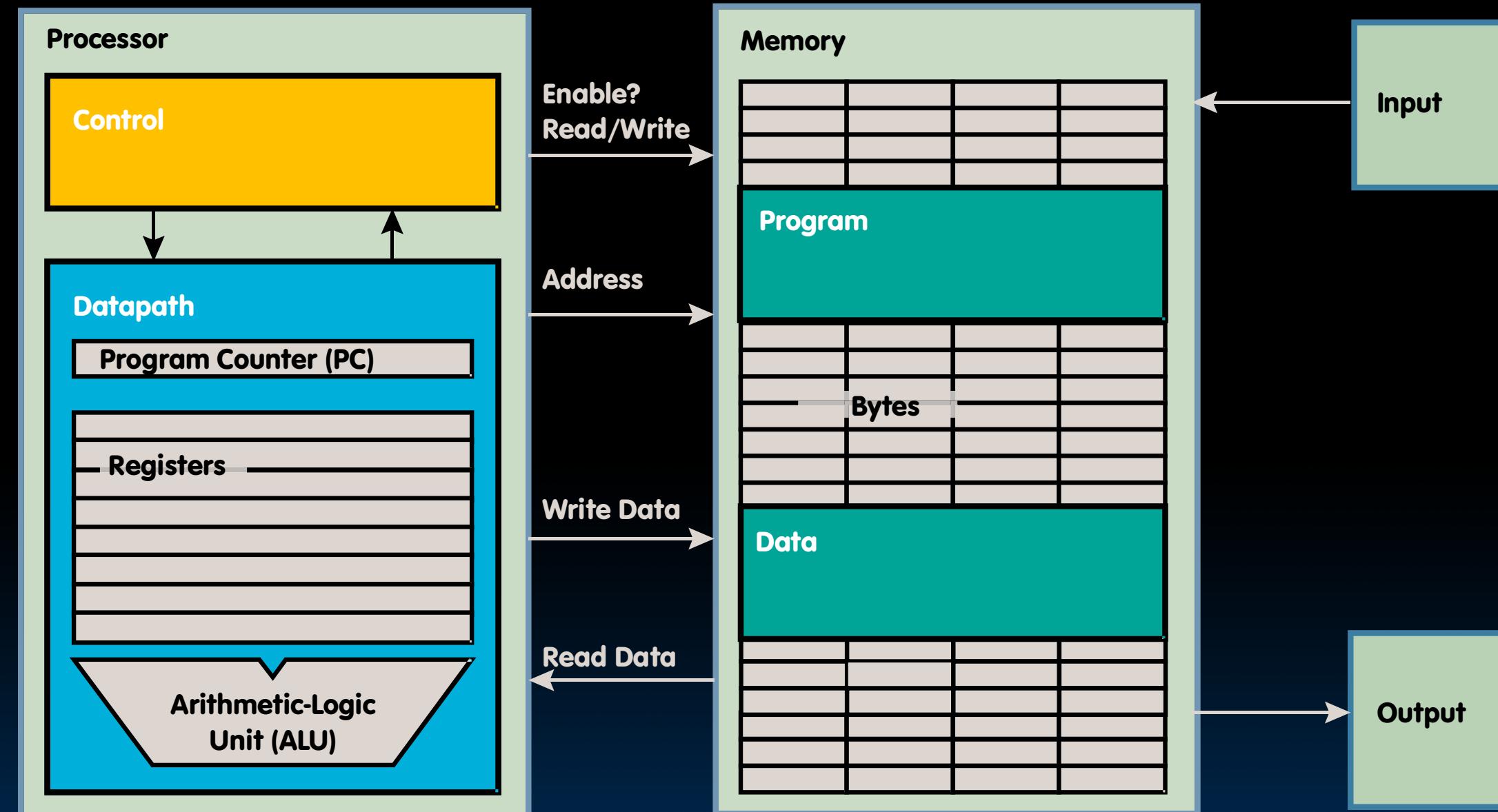
UC Berkeley  
Teaching Professor  
Lisa Yan

## RISC-V Processor Design Control

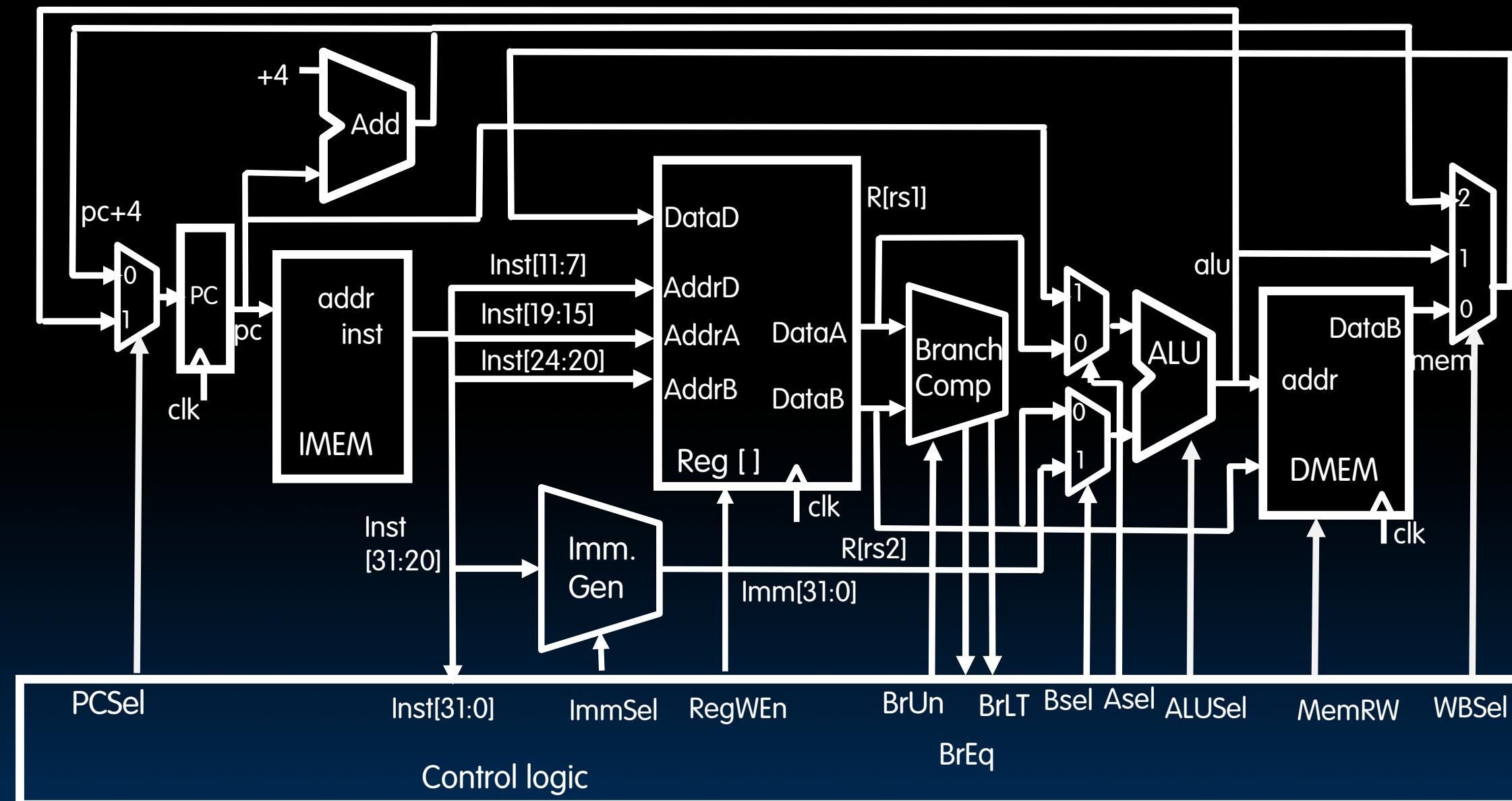
# Datapath Control

- We have designed a complete datapath
  - Capable of executing all RISC-V instructions in one cycle each
  - 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
  - We still need to design it

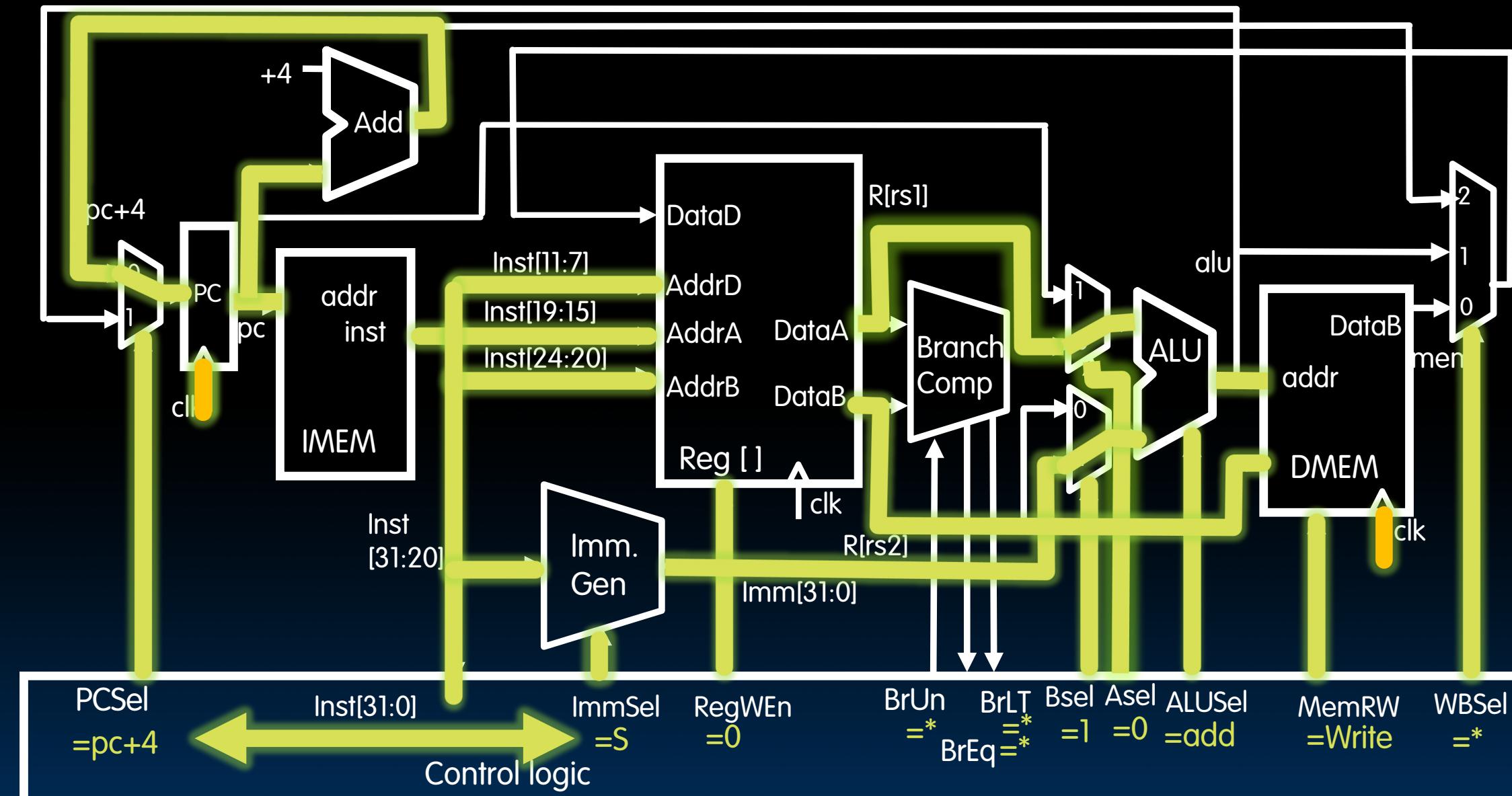
# Our Single-Core Processor



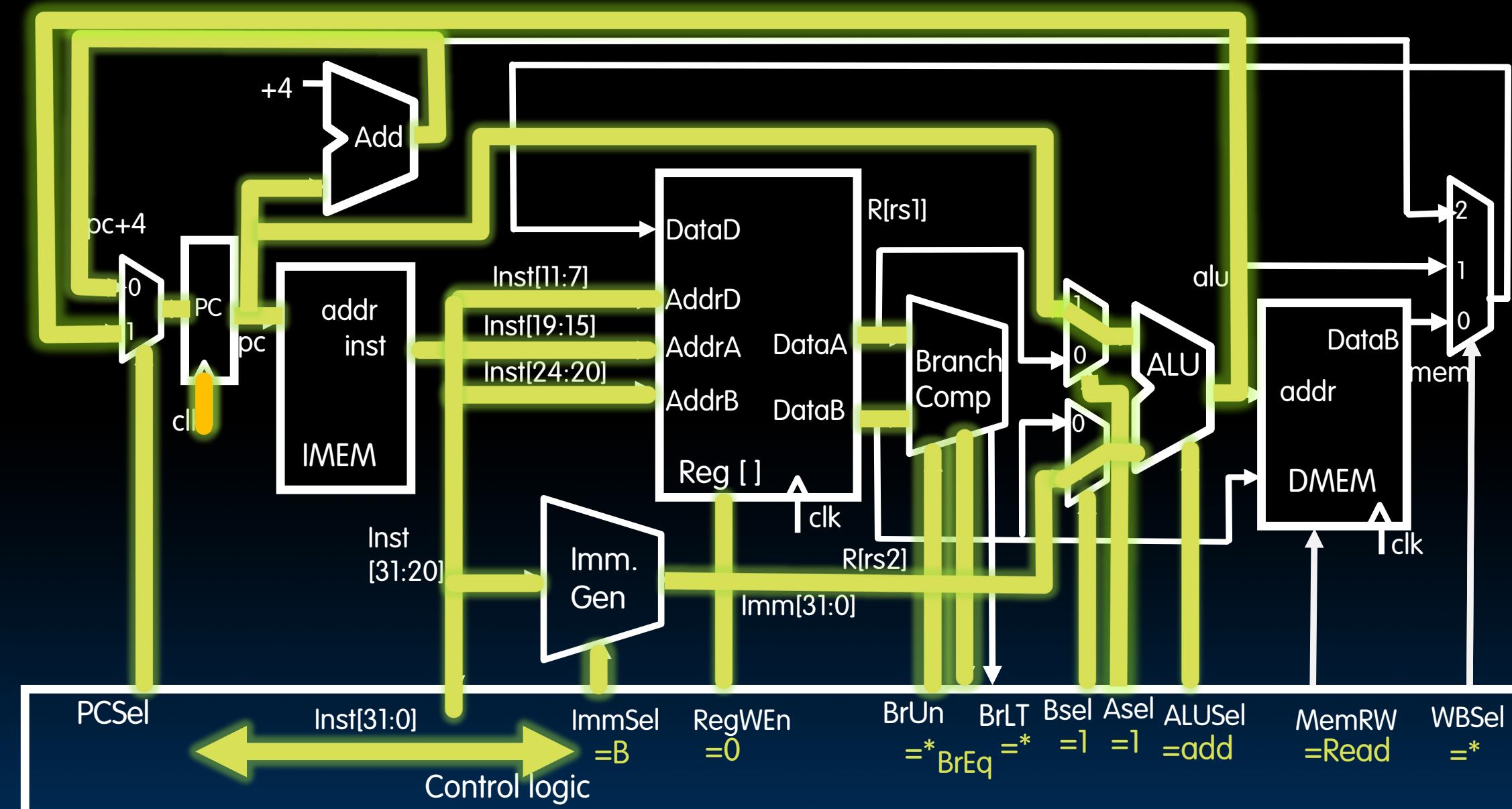
# Single-Cycle RV32I Datapath and Control



# Example: sw reg, offset (regbaseptr)

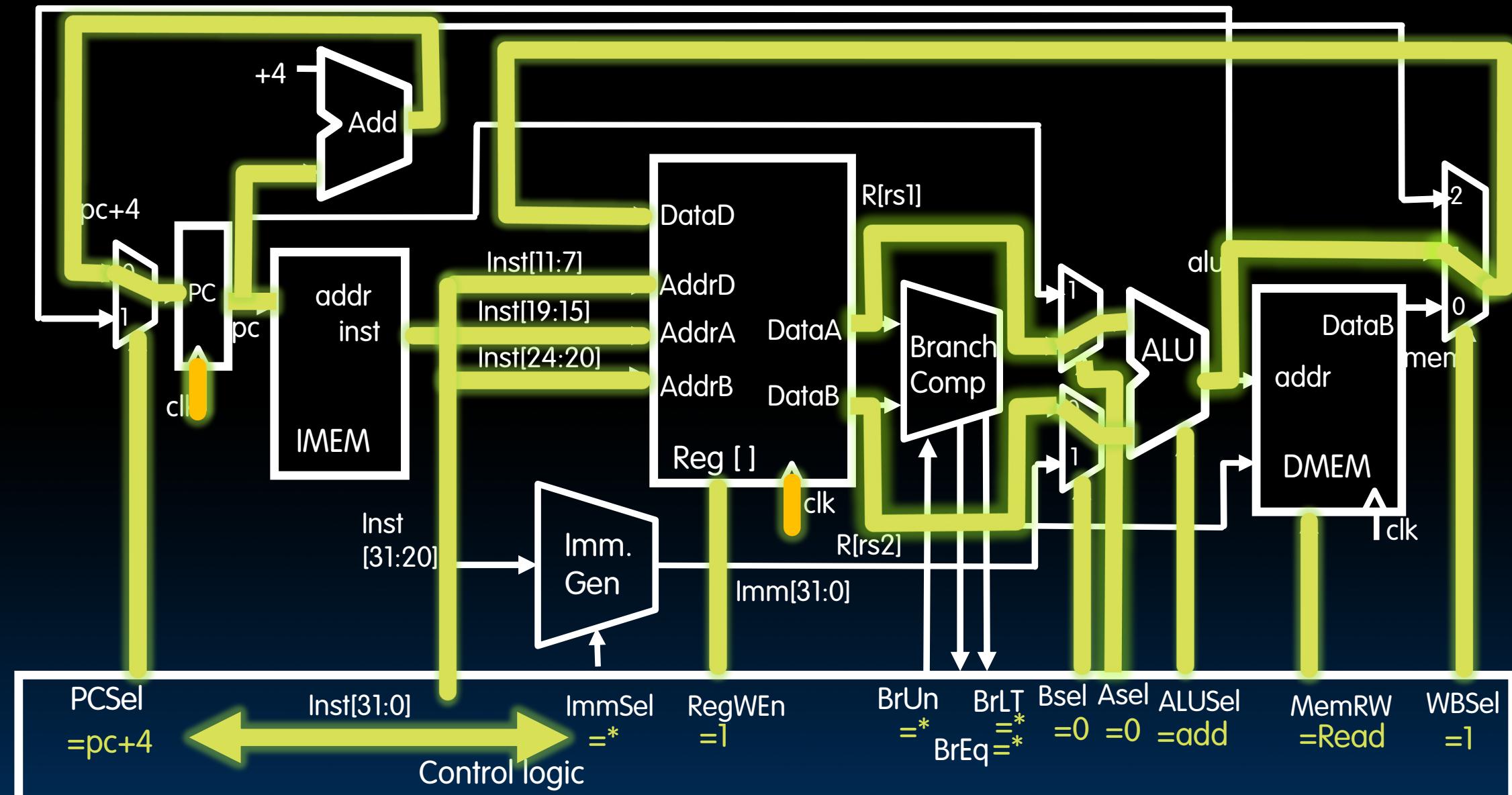


# Example: beq reg1, reg2, Label

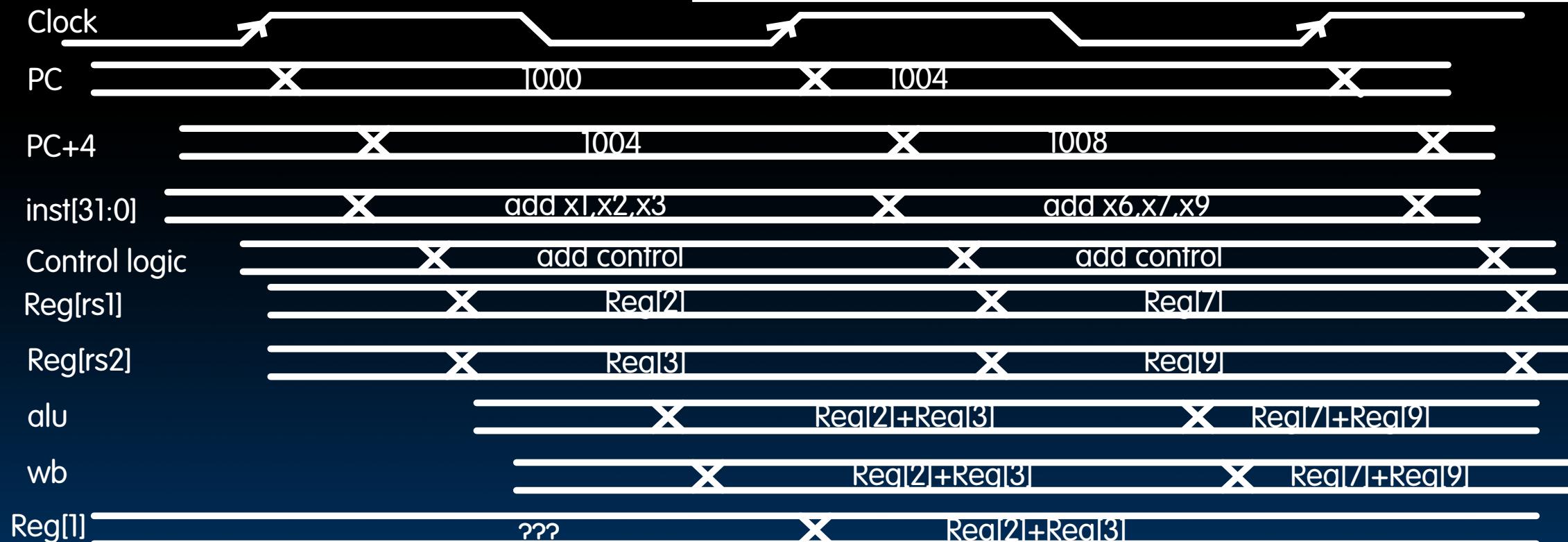
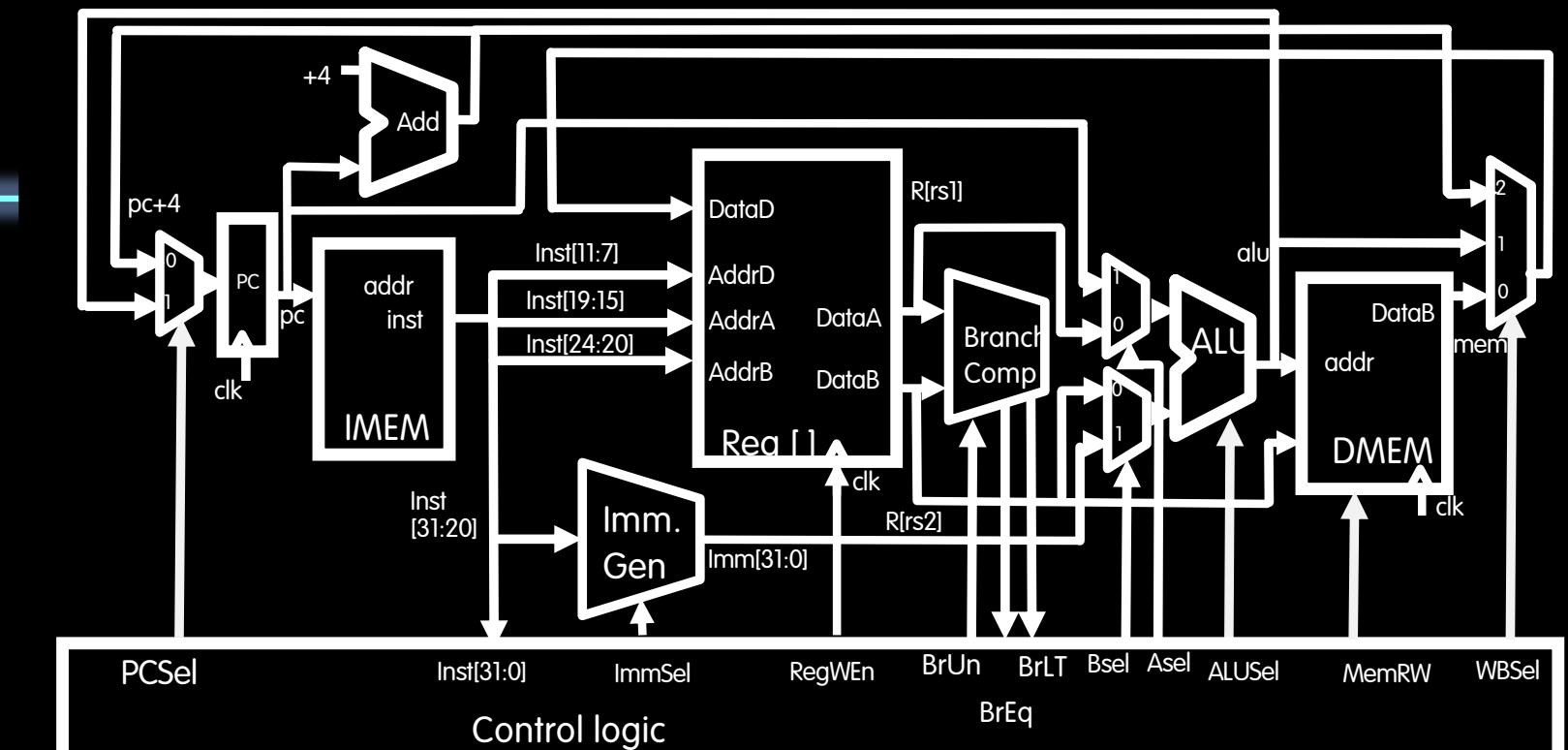


# Instruction Timing

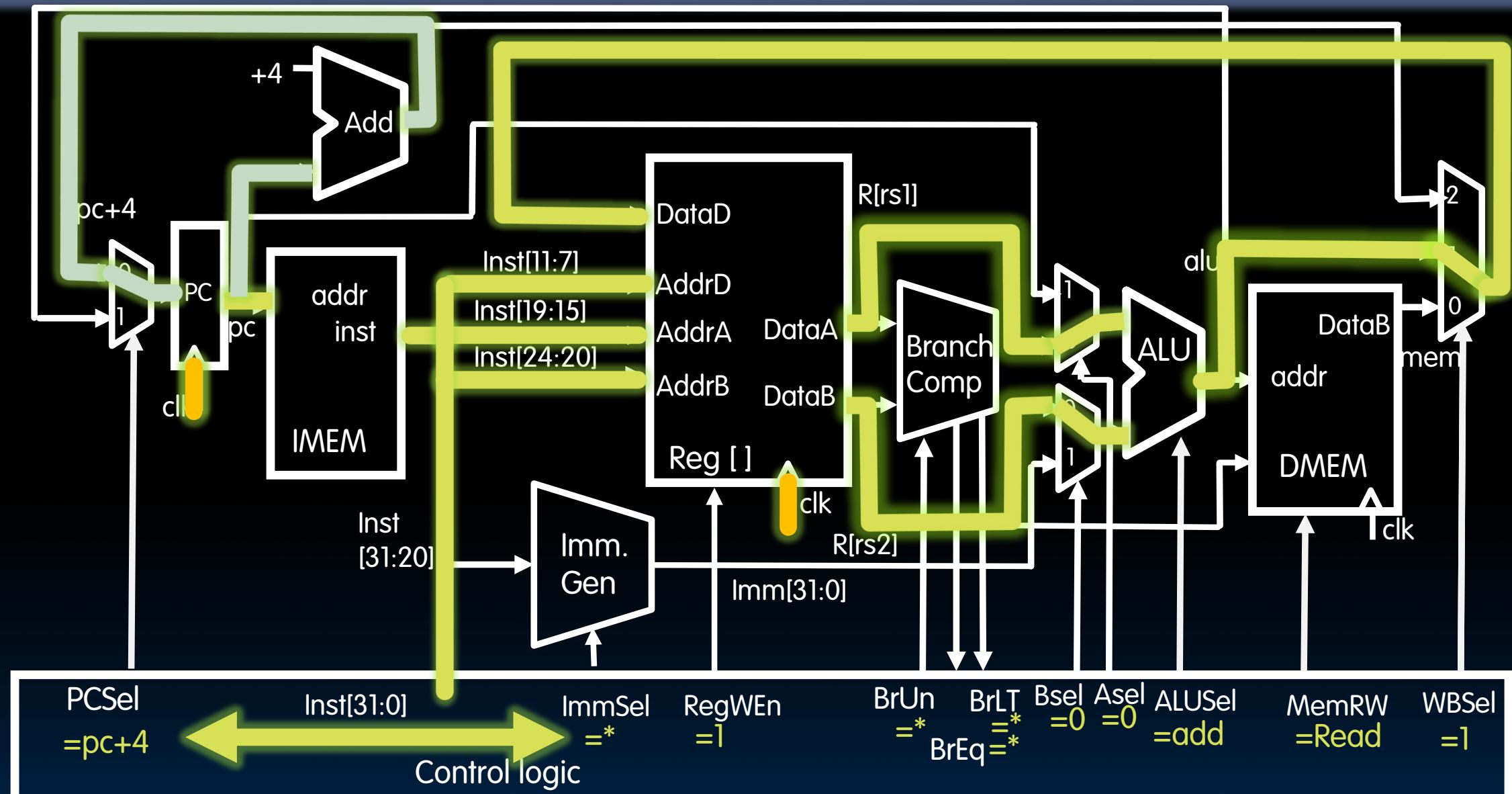
# Example: add rd, reg1, reg2



# add Execution

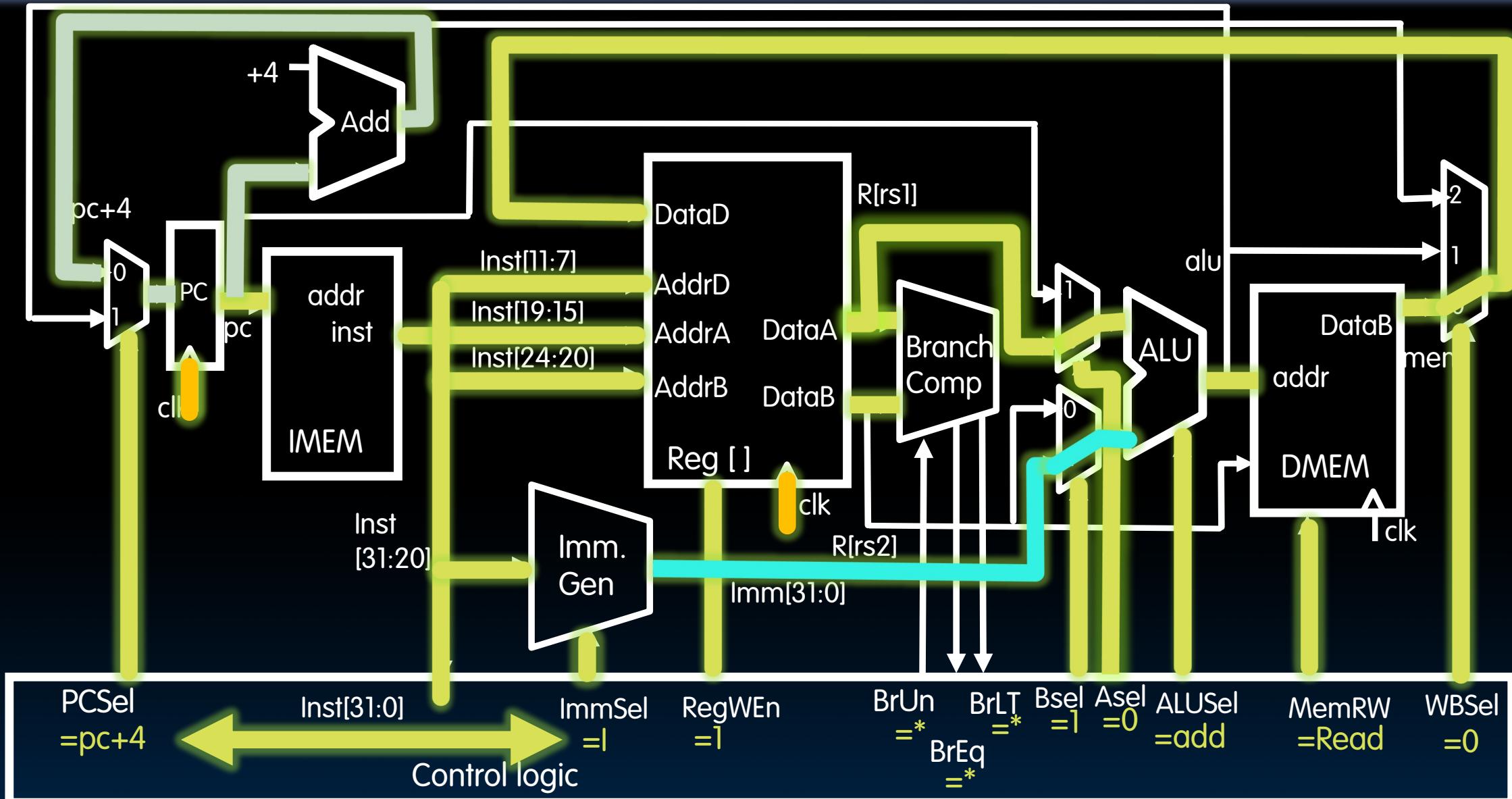


# Example: add timing



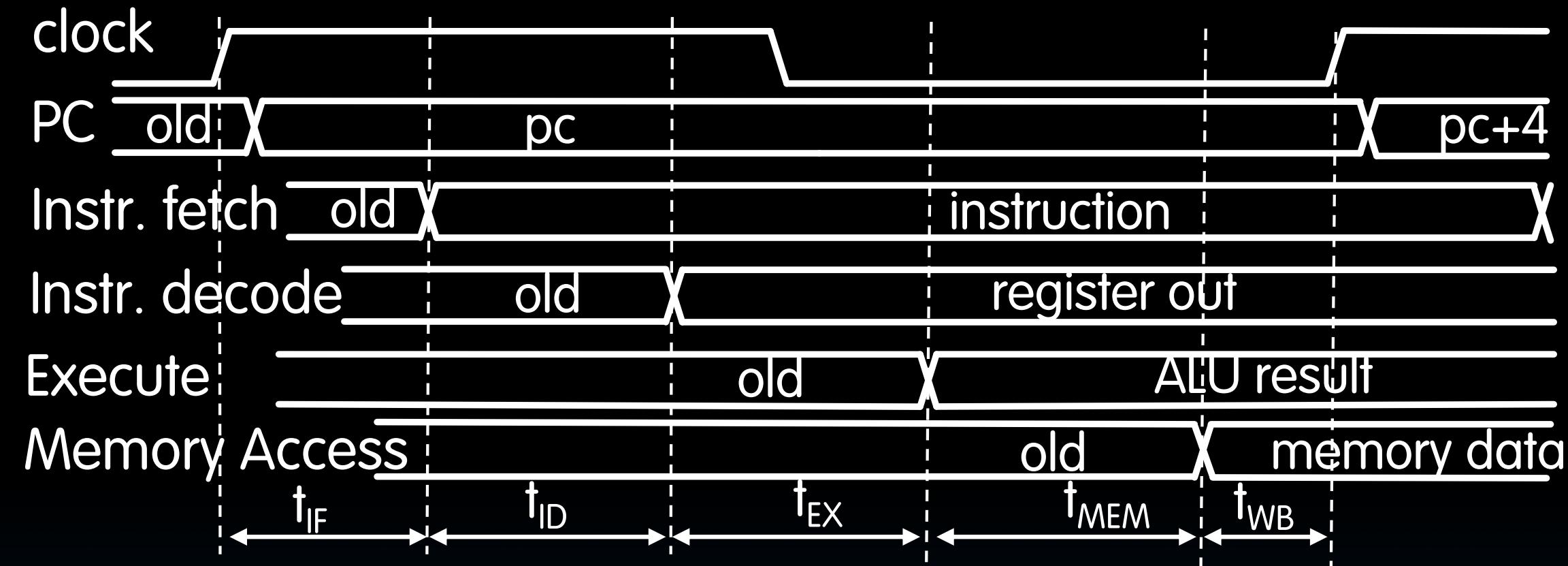
$$\begin{aligned}
 \text{Critical path} &= t_{\text{clk-q}} + \max \{ t_{\text{Add}} + t_{\text{mux}}, t_{\text{IMEM}} + t_{\text{Reg}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{mux}} \} + t_{\text{setup}} \\
 &= t_{\text{clk-q}} + t_{\text{IMEM}} + t_{\text{Reg}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{mux}} + t_{\text{setup}}
 \end{aligned}$$

# Example: lw reg, offset (regbaseptr)



$$\text{Critical path} = t_{\text{clk-q}} + \max \{ t_{\text{Add}} + t_{\text{mux}}, t_{\text{IMEM}} + t_{\text{Imm}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{DMEM}} + t_{\text{mux}}, \\ t_{\text{IMEM}} + t_{\text{Reg}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{DMEM}} + t_{\text{mux}} \} + t_{\text{setup}}$$

# 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
<b>add</b>	X	X	X		X	600ps
<b>beq</b>	X	X	X			500ps
<b>jal</b>	X	X	X			500ps
<b>lw</b>	X	X	X	X	X	800ps
<b>sw</b>	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.  $f_{\max, \text{ALU}} = 1/200\text{ps} = 5 \text{ GHz!}$

# Control Logic Design

# Control Logic Truth Table

Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
<b>add</b>	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
<b>sub</b>	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
<i>(R-R Op)</i>	*	*	+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

# 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

# RV32I, A Nine-Bit ISA!

imm[31:12]			rd	0110111
imm[31:12]			rd	0010111
imm[20:10:1 11 19:12]			rd	1101111
imm[11:0]	rs1	000	rd	1100111
imm[12:10:5]	rs2	000	imm[4:1 11]	1100011
imm[12:10:5]	rs2	001	imm[4:1 11]	1100011
imm[12:10:5]	rs2	100	imm[4:1 11]	1100011
imm[12:10:5]	rs2	101	imm[4:1 11]	1100011
imm[12:10:5]	rs2	110	imm[4:1 11]	1100011
imm[12:10:5]	rs2	111	imm[4:1 11]	1100011
imm[11:0]	rs1	000	rd	0000011
imm[11:0]	rs1	001	rd	0000011
imm[11:0]	rs1	010	rd	0000011
imm[11:0]	rs1	100	rd	0000011
imm[11:0]	rs1	101	rd	0000011
imm[11:5]	rs2	000	imm[4:0]	0100011
imm[11:5]	rs2	001	imm[4:0]	0100011
imm[11:5]	rs2	010	imm[4:0]	0100011
imm[11:0]	rs1	000	rd	0010011
imm[11:0]	rs1	010	rd	0010011
imm[11:0]	rs1	011	rd	0010011
imm[11:0]	rs1	100	rd	0010011
imm[11:0]	rs1	110	rd	0010011
imm[11:0]	rs1	111	rd	0010011
0000000	shamt	001	rd	0010011
0000000	shamt	101	rd	0010011
0100000	shamt	101	rd	0010011
0000000	rs2	000	rd	0110011
0100000	rs2	000	rd	0110011
0000000	rs2	001	rd	0110011
0000000	rs2	010	rd	0110011
0000000	rs2	011	rd	0110011
0000000	rs2	100	rd	0110011
0000000	rs2	101	rd	0110011
0100000	rs2	101	rd	0110011
0000000	rs2	110	rd	0110011
0000000	rs2	111	rd	0110011
fm	pred	succ	rs1	000
00000000000000			00000	00000
00000000000001			00000	00000

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

**inst[6:2]**

**inst[14:12]**

**inst[30]**

# Combinational Logic Control

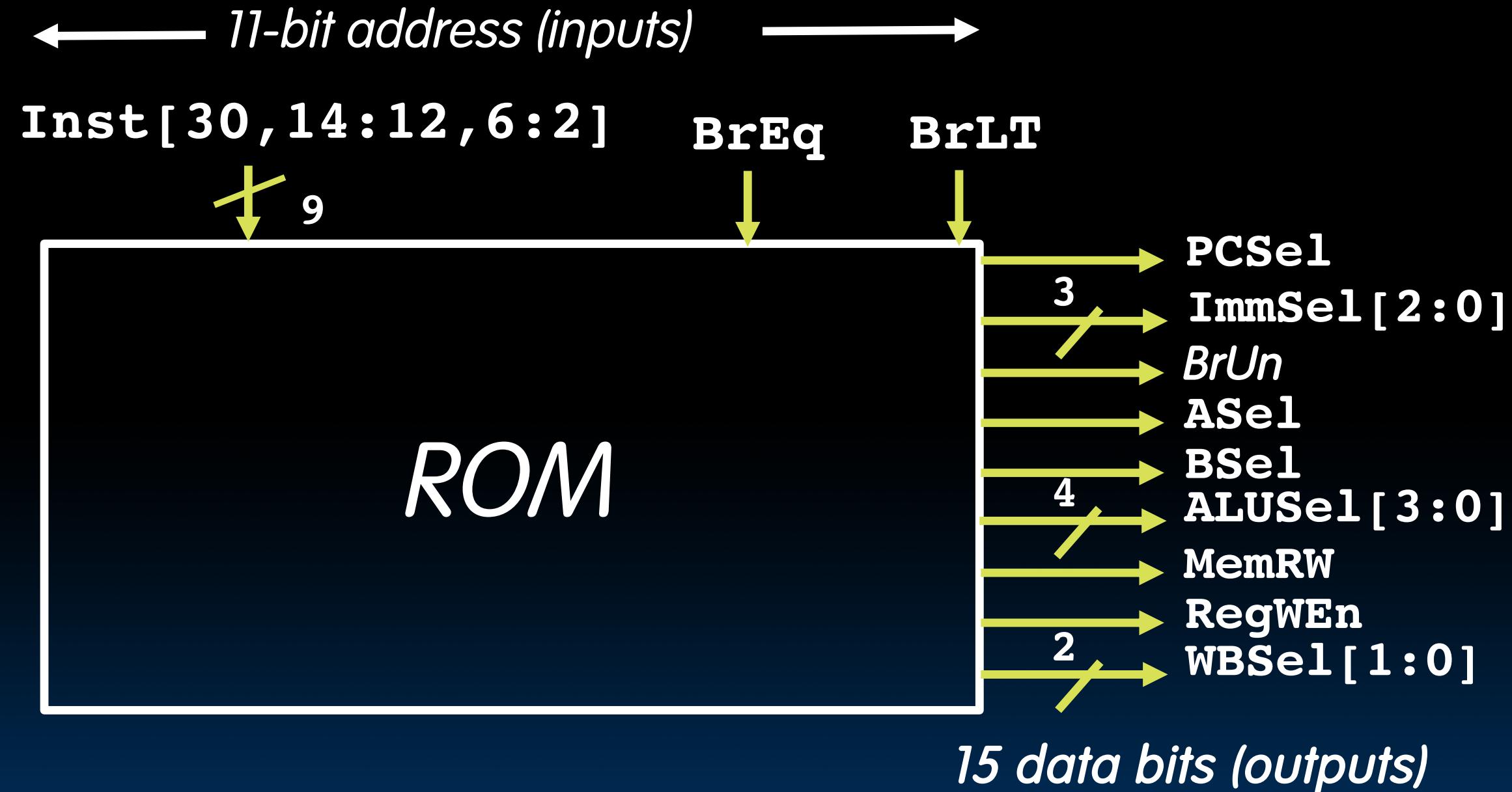
- Simplest example: **BrUn**

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

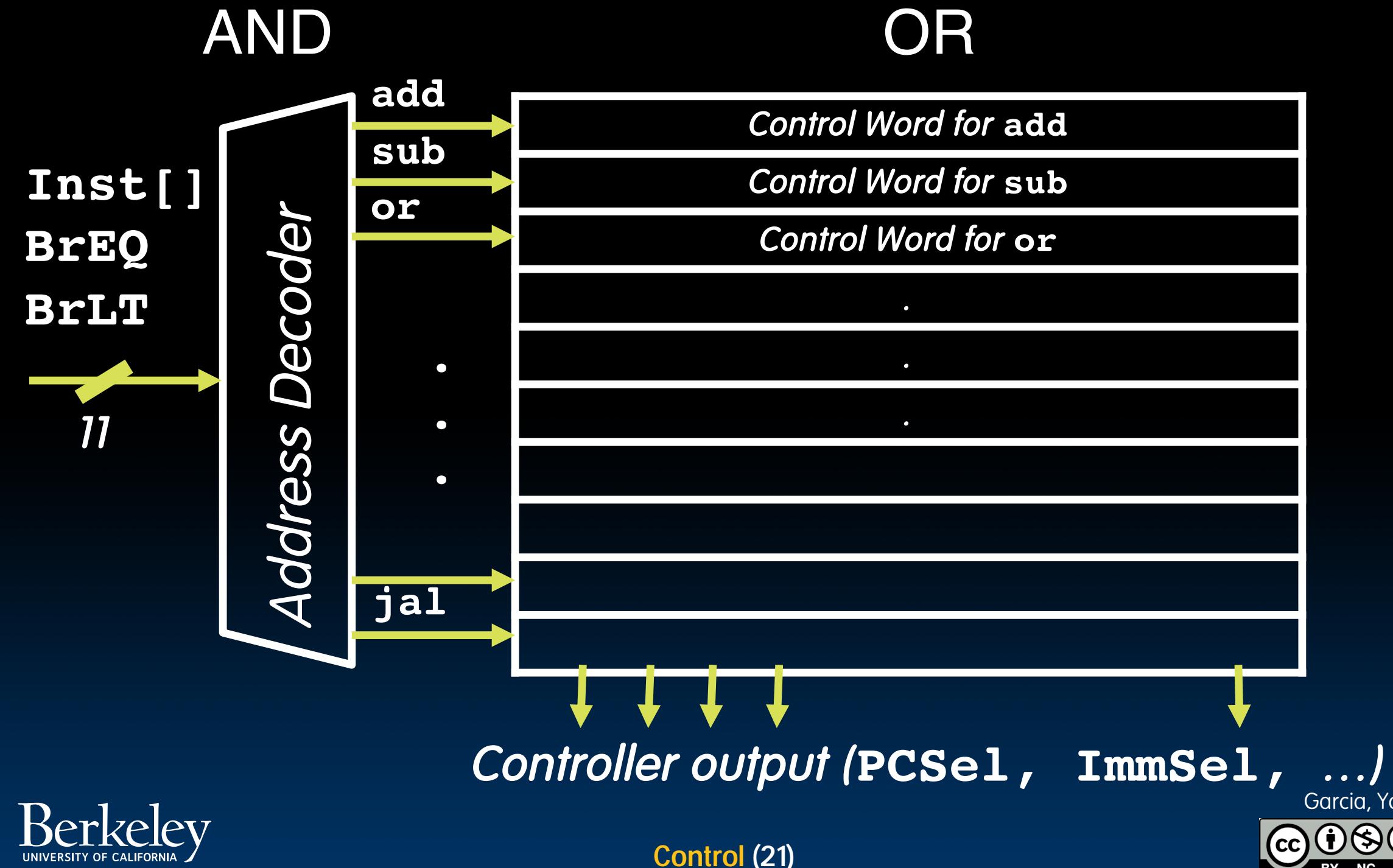
*How to decode whether BrUn is 1?*

$$\text{BrUn} = \text{inst}[13] \bullet \text{Branch}$$

# ROM-based Control



# ROM Controller Implementation



# Control Logic to Decode add

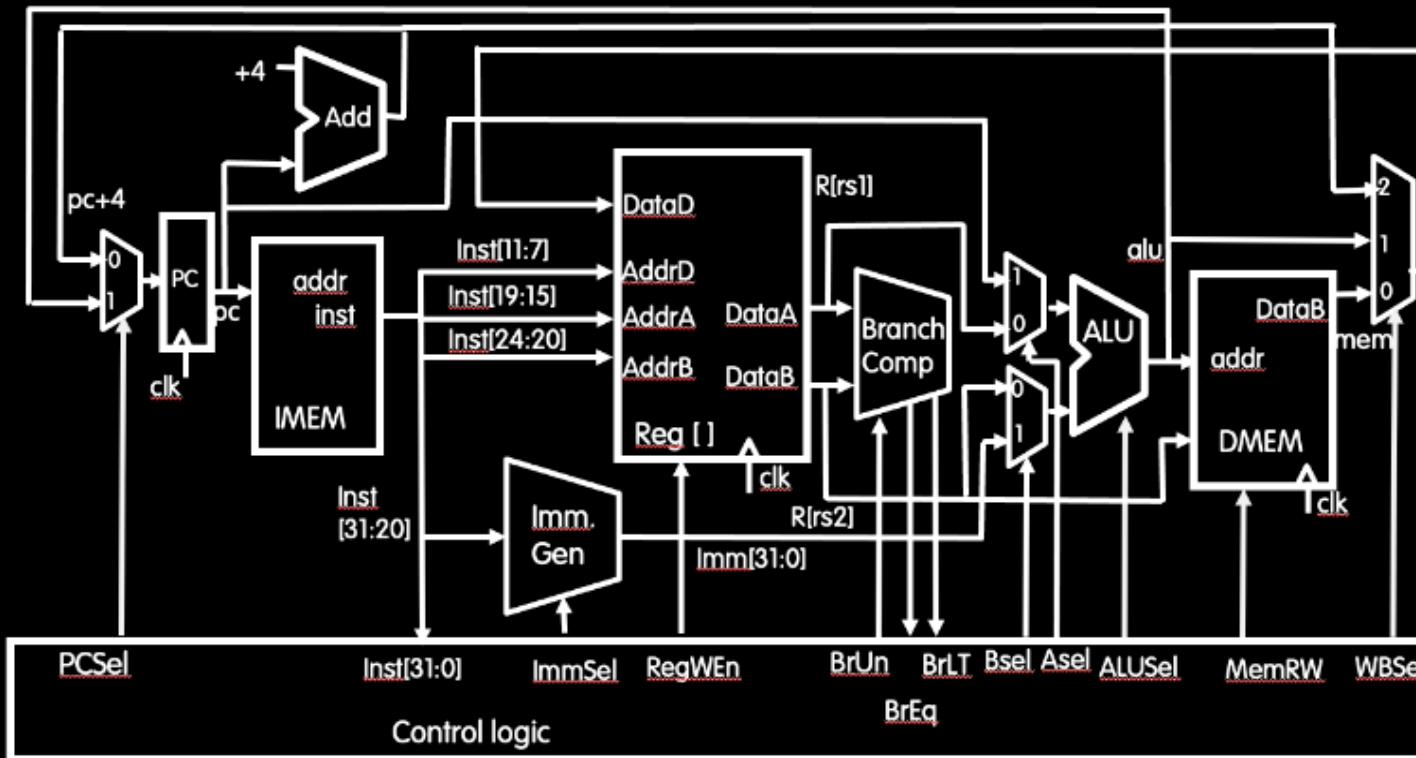
inst[30]	inst[14:12]	inst[6:2]	
0000000	shamt	rs1	001 SLLI
0000000	shamt	rs1	101 SR LI
0100000	shamt	rs1	101 SRAI
0000000	rs2	rs1	000 ADD
0100000	rs2	rs1	000 SUB
0000000	rs2	rs1	001 SLL
0000000	rs2	rs1	010 SLT
0000000	rs2	rs1	011 SLTU
0000000	rs2	rs1	100 XOR
0000000	rs2	rs1	101 SRL
0100000	rs2	rs1	101 SRA
0000000	rs2	rs1	110 OR
0000000	rs2	rs1	111 AND

$$\text{add} = \overline{i[30]} \cdot \overline{i[14]} \cdot \overline{i[13]} \cdot \overline{i[12]} \cdot \text{R-type}$$

$$\text{R-type} = \overline{i[6]} \cdot \overline{i[5]} \cdot \overline{i[4]} \cdot \overline{i[3]} \cdot \overline{i[2]} \cdot \text{RV32I}$$

$$\text{RV32I} = i[1] \cdot i[0]$$

# L20 Critical path for addi ( $R[rd] = R[rs1] + imm$ )



1: tclk-q + tAdd + tIMEM + tReg + tBComp + tALU + tDMEM + tmux + tSetup

2: tclk-q + tIMEM + max{tReg, tImm} + tALU + 3tmux + tSetup

3: tclk-q + tIMEM + max{tReg, tImm} + tALU + 3tmux + tDMEM + tSetup

None of the above

**"And In  
Conclusion..."  
(drum roll)**

# Call home, we've made HW/SW contact!

High Level Language  
Program (e.g., C)

```
temp = v[k];  
v[k] = v[k+1]
```

| Compiler

Assembly Language  
Program (e.g., RISC-V)

```
lw x3, 0(x10)  
lw x4, 4(x10)  
sw x4, 0(x10)  
sw x3, 4(x10)
```

| Assembler

Machine Language  
Program (RISC-V)

```
1000 1101 1110 0010 0000 0000 0000 0000  
1000 1110 0001 0000 00 0000 0000 0100  
1010 1110 0001 0010 0000 0000 0000 0000  
1010 1101 1110 0010 00 0000 0000 0100
```

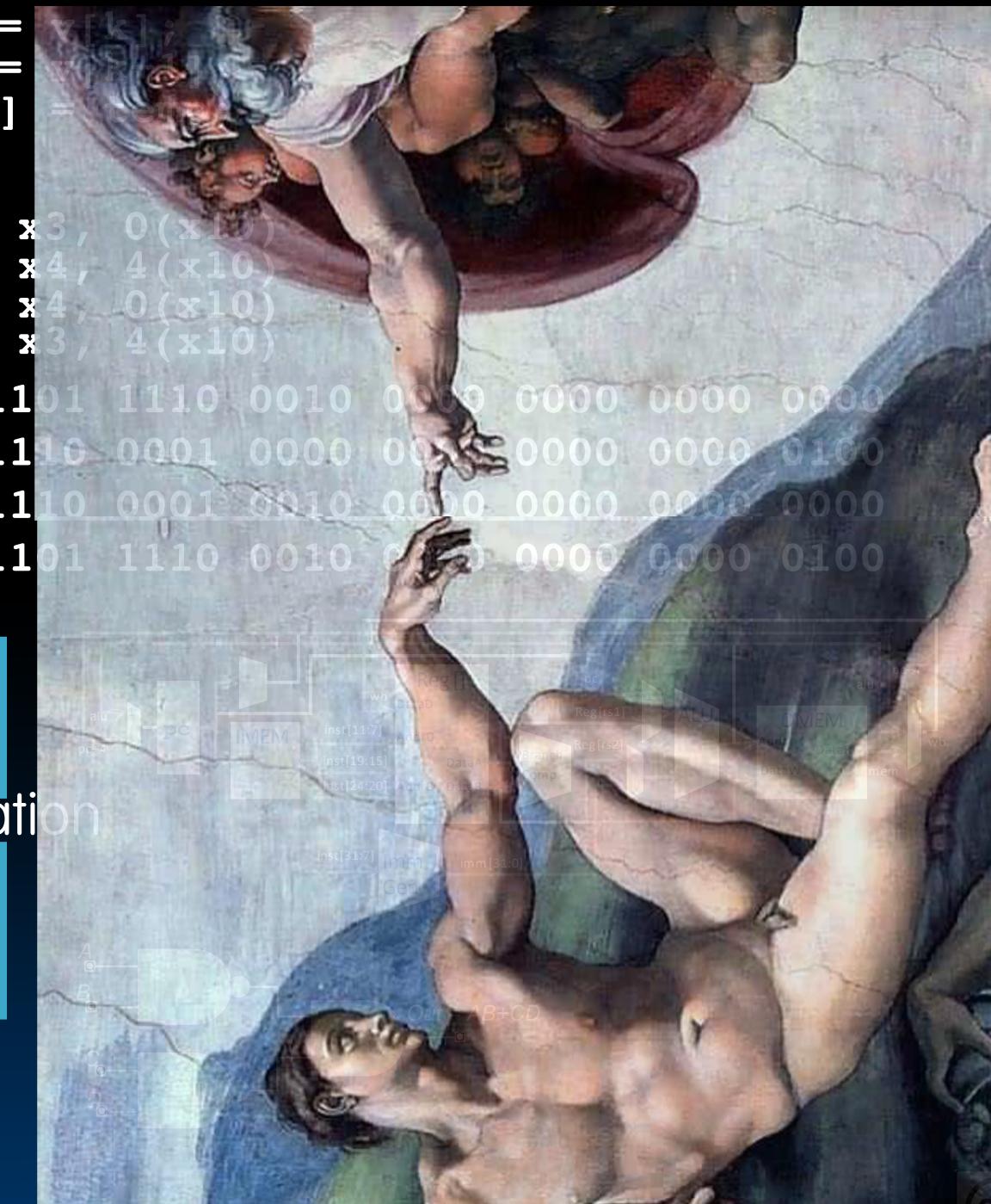
|

Hardware Architecture Description  
(e.g., block diagrams)



| Architecture Implementation

Logic Circuit Description  
(Circuit Schematic Diagrams)



# “And In conclusion...”

- We have built a processor!
  - Capable of executing all RISC-V instructions in one cycle each
  - Not all units (hardware) used by all instructions
  - Critical path changes
- 5 Phases of execution
  - IF, ID, EX, MEM, WB
  - Not all instructions are active in all phases
- Controller specifies how to execute instructions
  - Implemented as ROM or logic