

CS 110
Computer Architecture
Lecture 13:
*Single-Cycle CPU
Control*

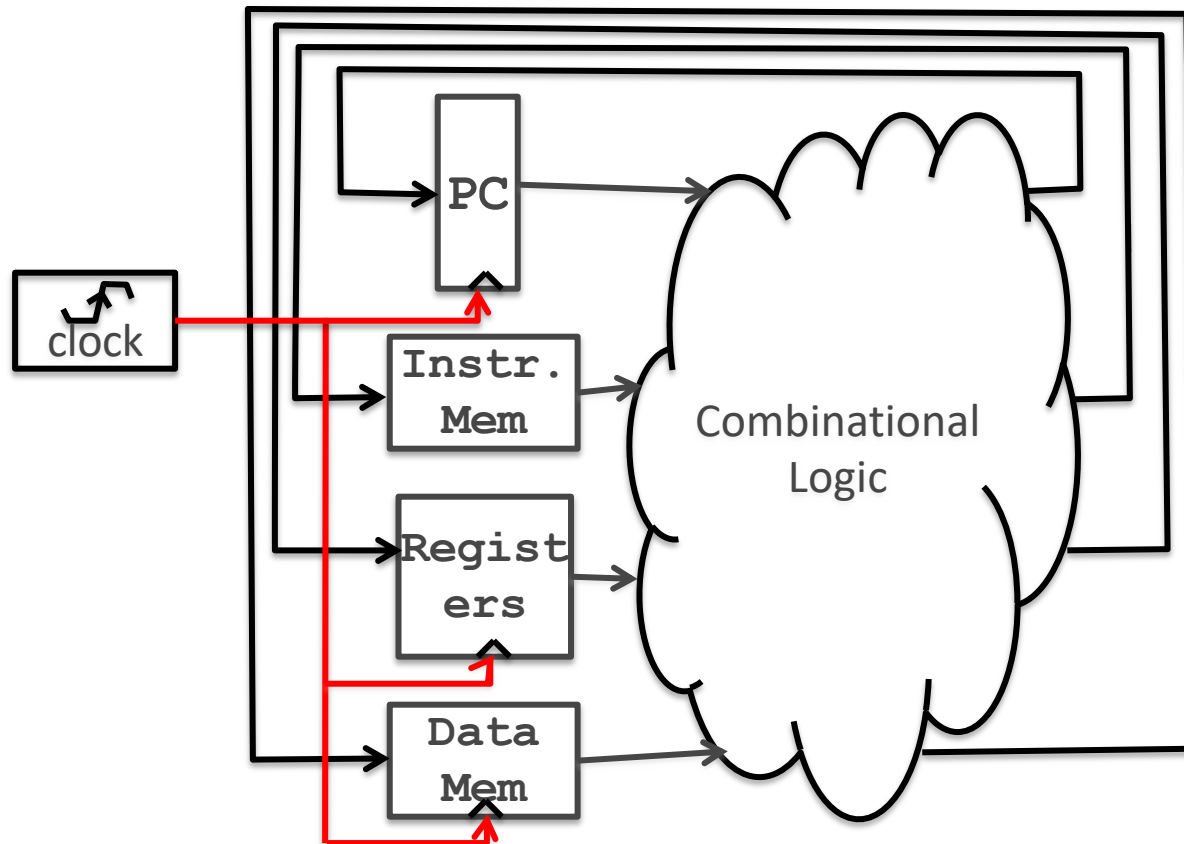
Instructors:
Sören Schwertfeger & Chundong Wang

School of Information Science and Technology SIST

ShanghaiTech University

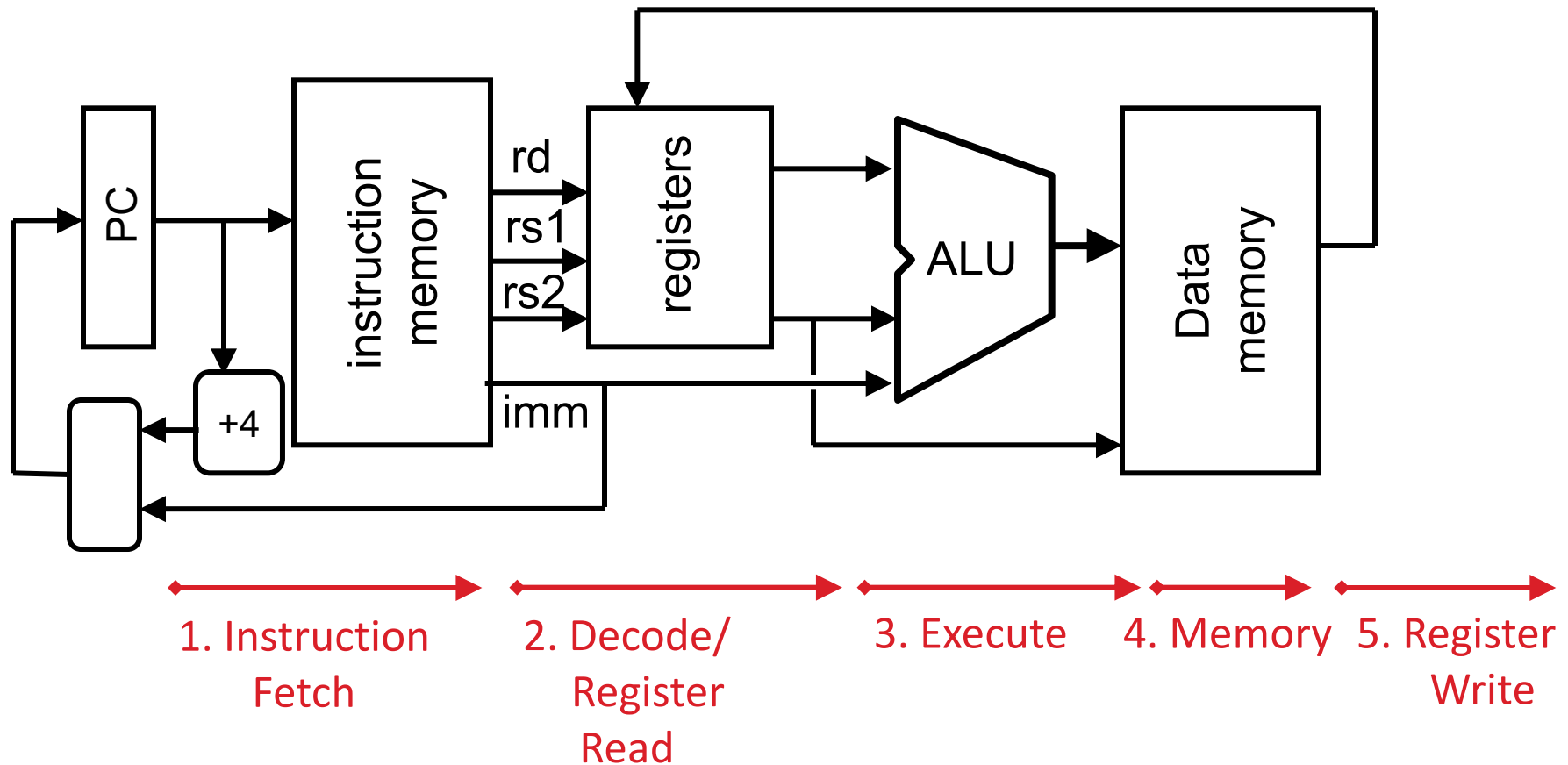
Slides based on UC Berkley's CS61C

One-Instruction-Per-Cycle RISC-V Machine



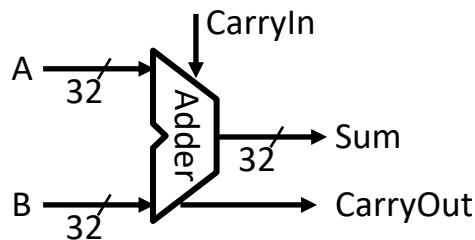
- One clock tick => one instruction
- Current state outputs => inputs to combinational logic => outputs settle at the values of state before next clock edge
- Rising clock edge:
 - all state elements are updated with combinational logic outputs
 - execution moves to next clock cycle

Stages of Execution on Datapath

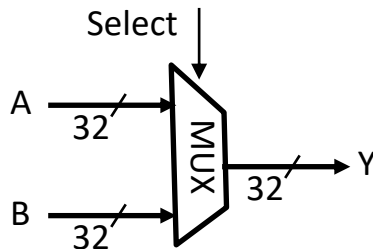


Datapath Components: Combinational

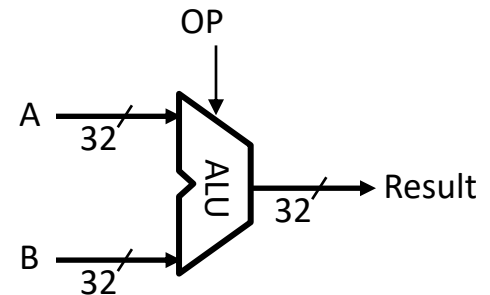
- Combinational Elements



Adder



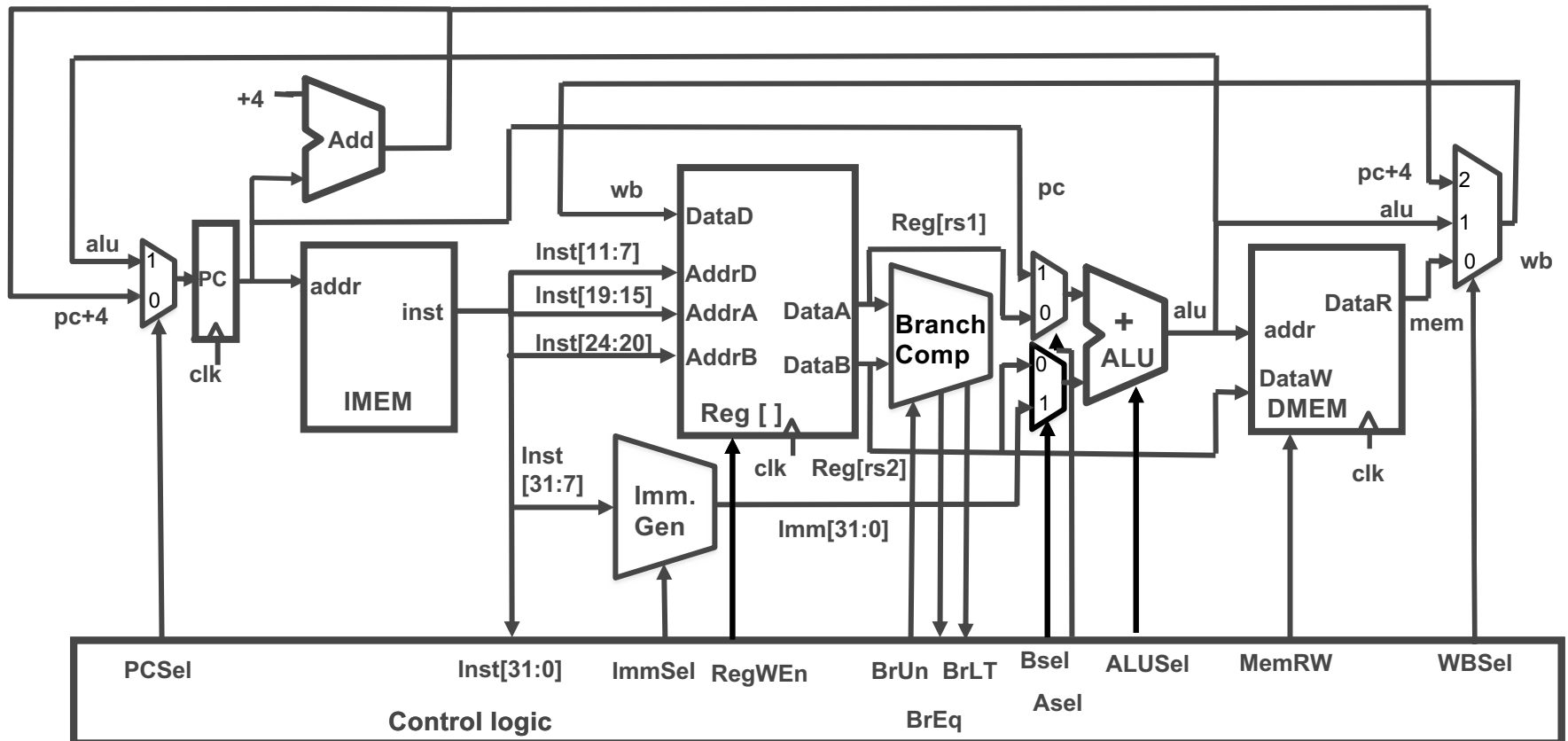
Multiplexer



ALU

- Storage Elements + Clocking Methodology
 - Registers
 - Register File (General Purpose Registers x0-x31)

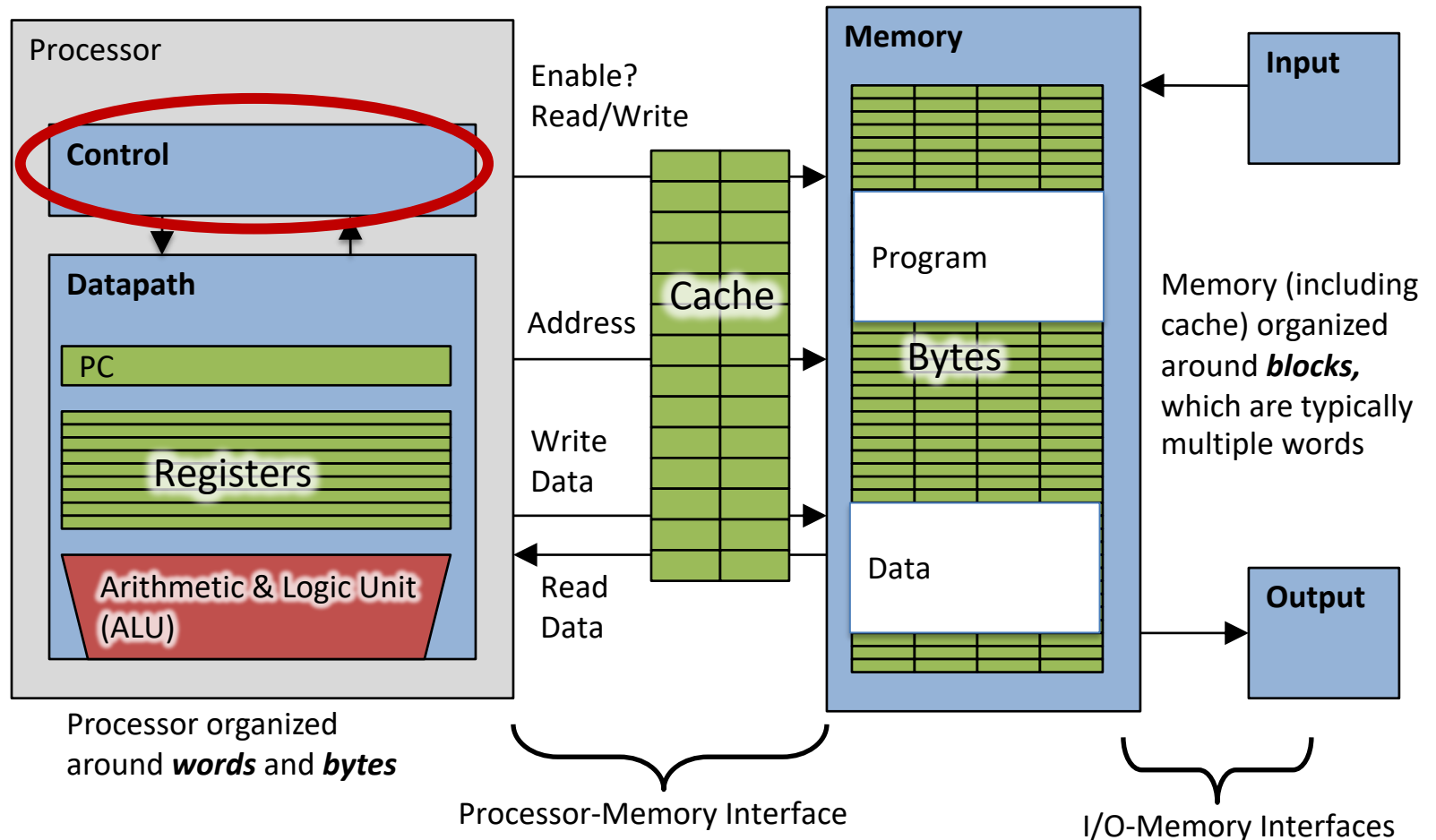
Complete RV32I Datapath!



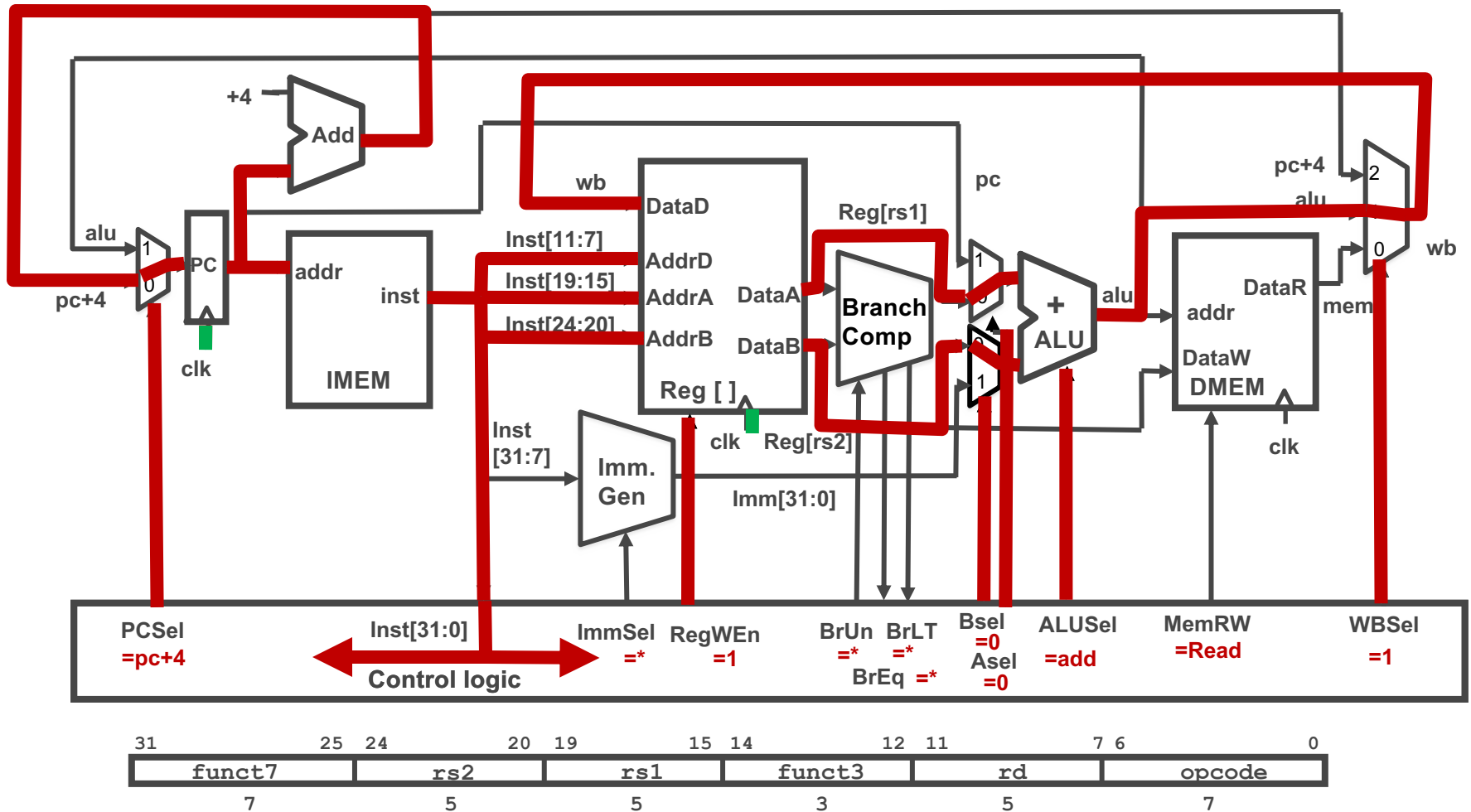
Control

- Set the signals for the datapath
- Based on the instruction to be executed

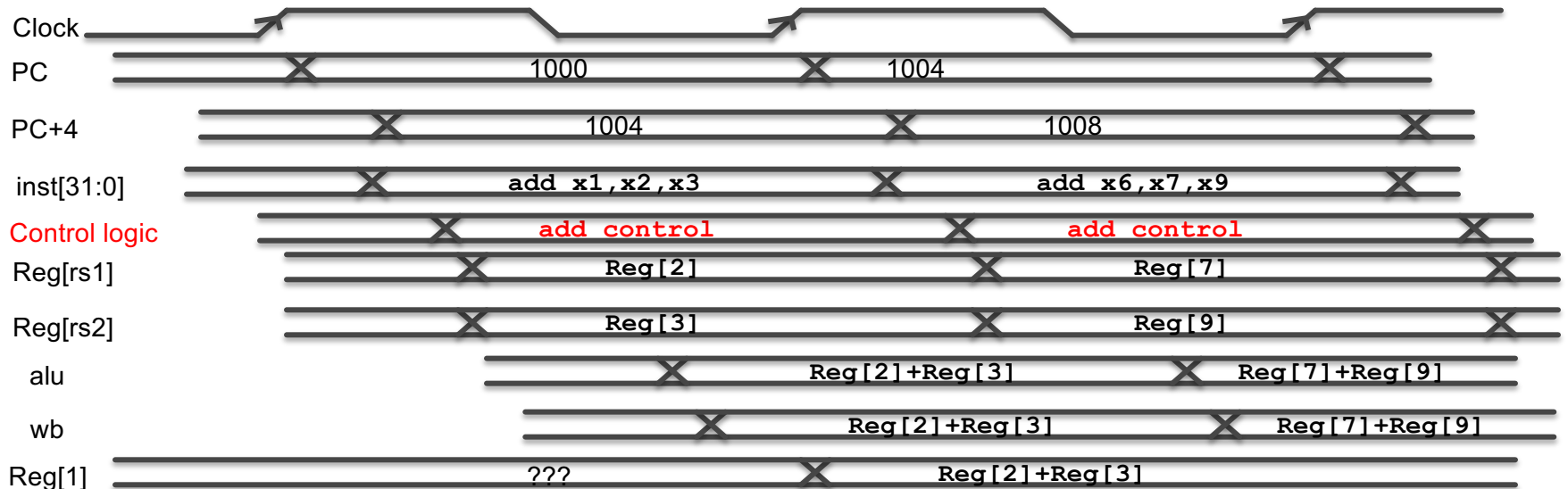
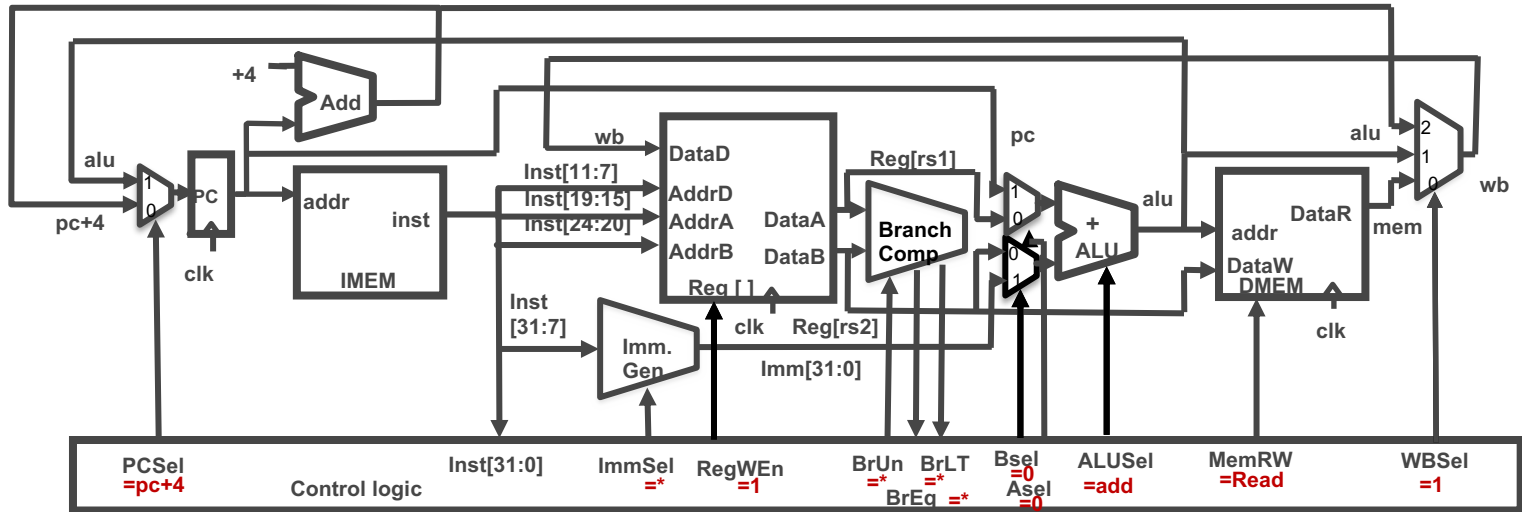
Our Single-Core Computer (without FPU, SIMD)



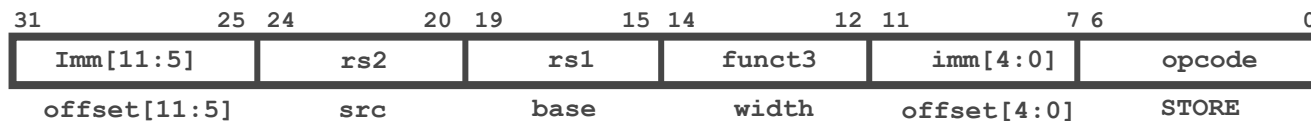
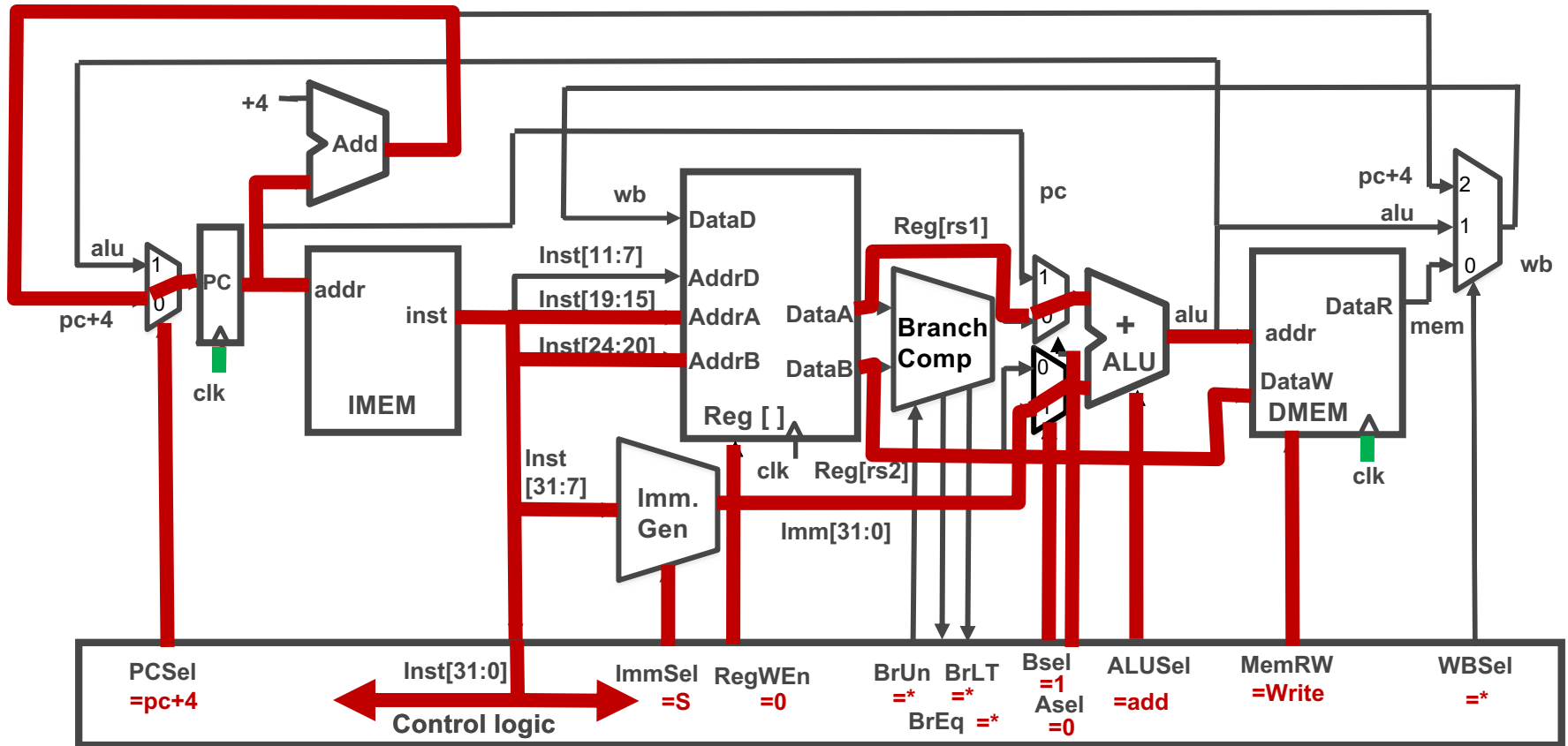
Example: add



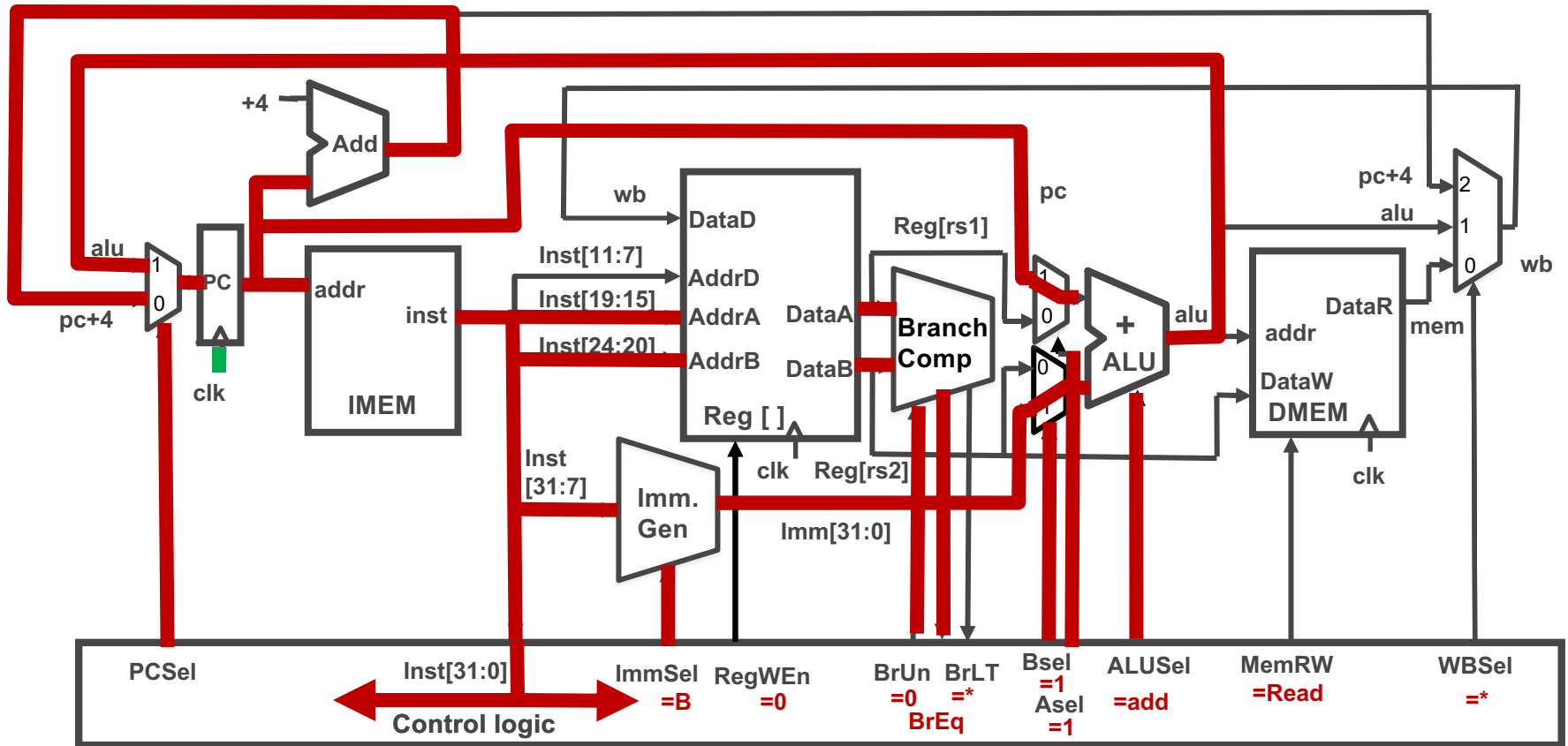
add Execution



Example: sw



Example: beq



Control Logic Truth Table

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
<i>(R-R Op)</i>	*	*	+4	*	*	Reg	Reg	<i>(Op)</i>	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

Control Realization Options

- ROM
 - “Read-Only Memory”
 - During “normal” operation it is only being read
 - 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	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]	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

inst[30]

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

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
0000000000000			00000	000	00000	1110011	ECALL
0000000000001			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 CS110

Not in CS110

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

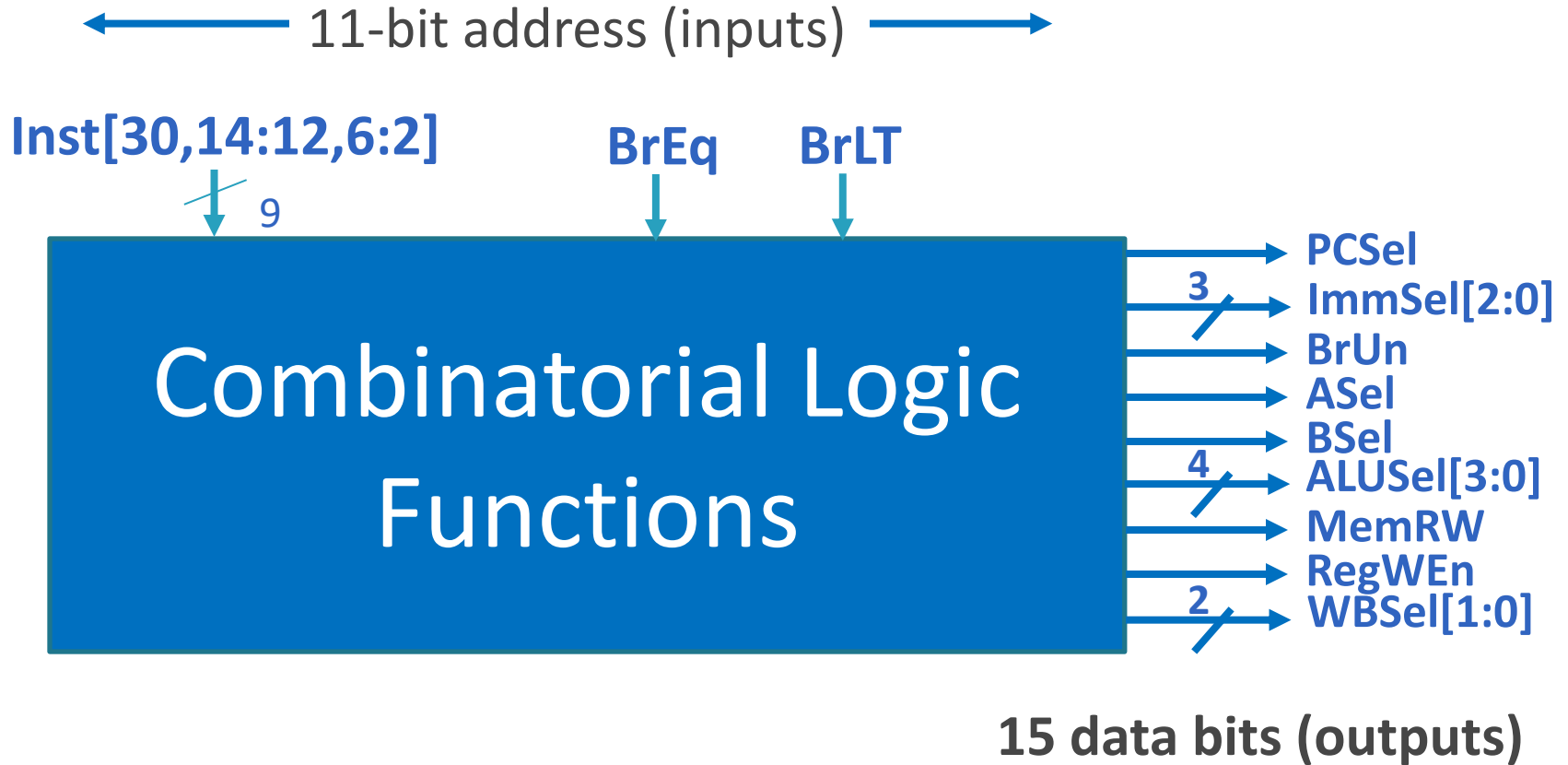
Combinational Logic Control

- Simplest example: BrUn

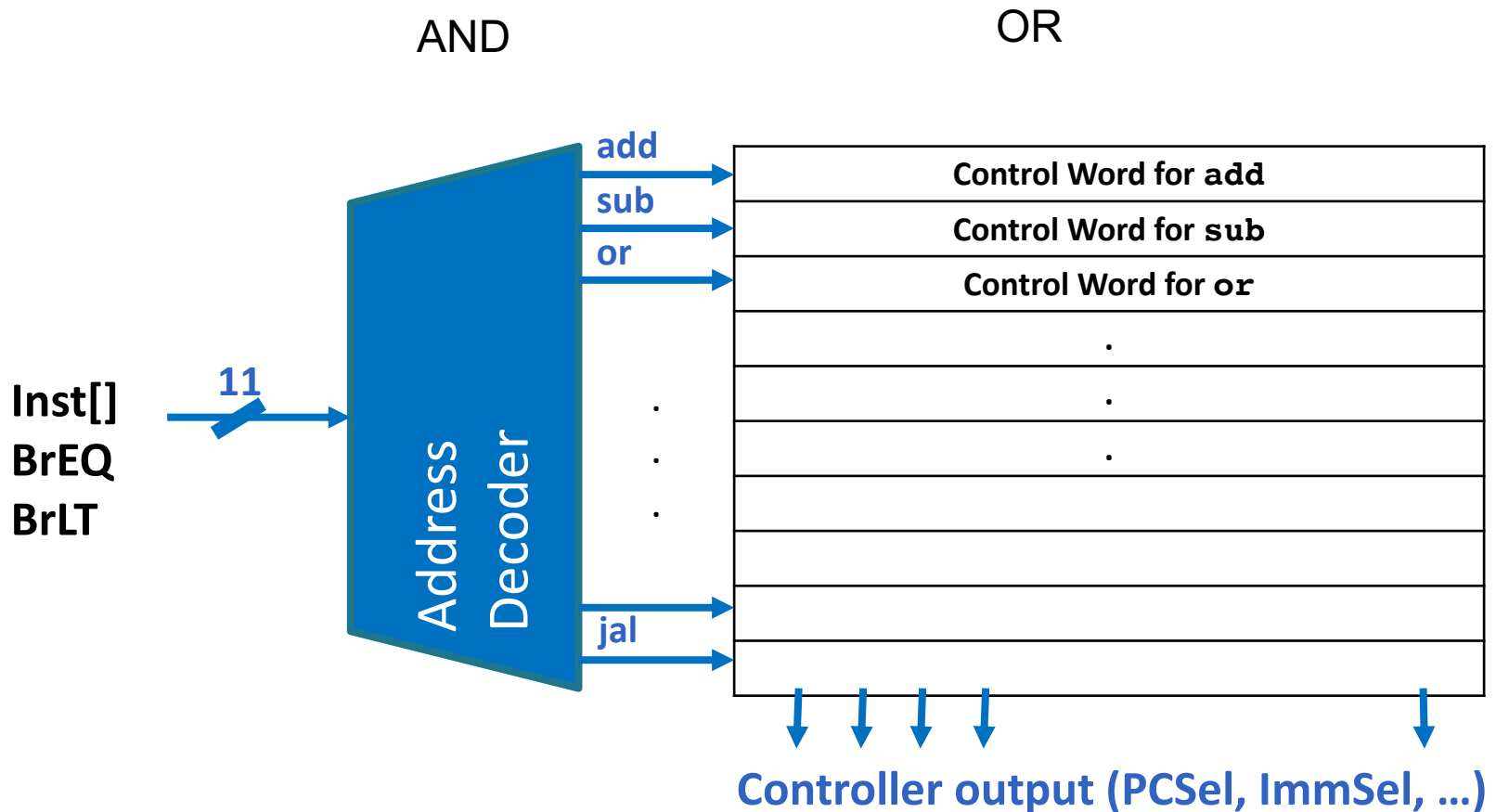
				inst[14:12]			inst[6:2] = Branch	
imm[12:10:5]	rs2	rs1	000	imm[4:1:11]	11000	11	BEQ	
imm[12:10:5]	rs2	rs1	001	imm[4:1:11]	11000	11	BNE	
imm[12:10:5]	rs2	rs1	100	imm[4:1:11]	11000	11	BLT	
imm[12:10:5]	rs2	rs1	101	imm[4:1:11]	11000	11	BGE	
imm[12:10:5]	rs2	rs1	110	imm[4:1:11]	11000	11	BLTU	
imm[12:10:5]	rs2	rs1	111	imm[4:1:11]	11000	11	BGEU	

- How to decode whether BrUn is 1?
 - BrUn = Inst [13] • Branch

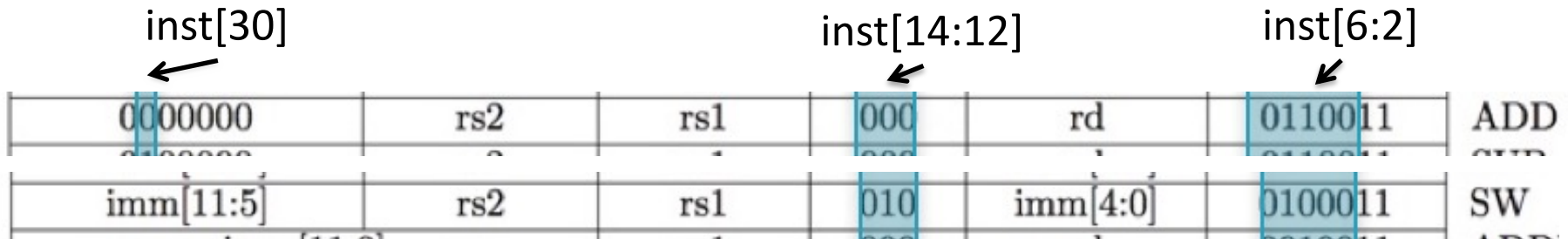
Control Block Design



ROM Controller Implementation



Controller: AND logic (add, sw)



- $\text{add} = \overline{\text{inst}[2]} \cdot \overline{\text{inst}[3]} \cdot \text{inst}[4] \cdot \text{inst}[5] \cdot \overline{\text{inst}[6]} \cdot \overline{\text{inst}[12]} \cdot \overline{\text{inst}[13]} \cdot \overline{\text{inst}[14]} \cdot \overline{\text{inst}[30]}$
- $\text{sw} = \overline{\text{inst}[2]} \cdot \overline{\text{inst}[3]} \cdot \overline{\text{inst}[4]} \cdot \text{inst}[5] \cdot \overline{\text{inst}[6]} \cdot \overline{\text{inst}[12]} \cdot \text{inst}[13] \cdot \overline{\text{inst}[14]}$

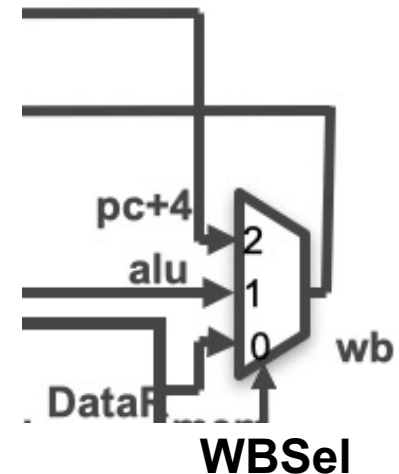
AND Controller Logic															
	2	3	4	5	6	12	13	14	30	unused					
add	xor(i[2], 1)	xor(i[3], 1)	xor(i[4], 0)	xor(i[5], 0)	xor(i[6], 1)	xor(i[12], 1)	xor(i[13], 1)	xor(i[14], 1)	xor(i[30], 1)	0					
sw	xor(i[2], 1)	xor(i[3], 1)	xor(i[4], 1)	xor(i[5], 0)	xor(i[6], 1)	xor(i[12], 1)	xor(i[13], 1)	xor(i[14], 1)	xor(i[30], 1)	1					
...															

- $\text{bne_t} = \text{bne} \cdot \overline{\text{BrEQ}}$
- $\text{bne_f} = \text{bne} \cdot \text{BrEQ}$

Controller: OR logic



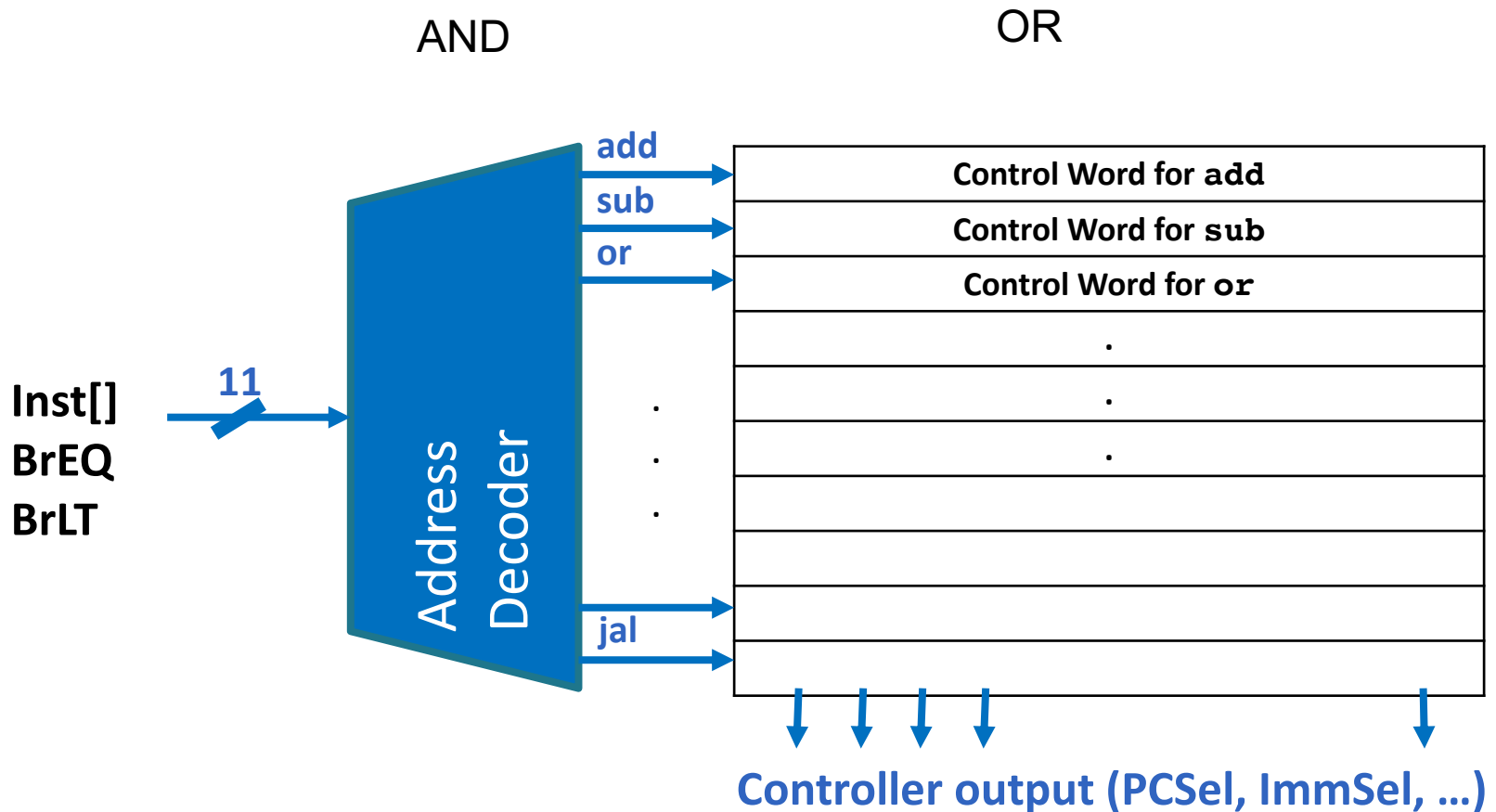
- MemRW = sw + sh + sb
- WBSel[0] = add + (all reg. to reg.) + AUIPC + ...
- WBSel[1] = JALR + JAL
- PCSel = beq_t + bne_t + blt_t + bltu_t + jal + jalr
- ...



Control Logic Truth Table

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
<i>(R-R Op)</i>	*	*	+4	*	*	Reg	Reg	<i>(Op)</i>	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

ROM Controller Implementation



Question

- Select the statements that are TRUE:
 - A. We should design the control logic as fast as possible because it adds to the critical path in the CPU.
 - B. The control needs all 32 bits of the instruction to decide on the control signals.
 - C. The value of all control signals is known from the instruction bits.
 - D. It is possible to add certain new instructions to the ISA by just modifying the control (e.g. subi).

Register Transfer Level (RTL)

- RTL describes instructions in figures or text
- Can use C (or Verilog) to describe RTL
- RISC-V green card show RTL for all instructions in Verilog
- RTL is a subset of a Hardware Description Language

Inst Register Transfers

add $R[rd] \leftarrow R[rs1] + R[rs2]; PC \leftarrow PC + 4$

sub $R[rd] \leftarrow R[rs1] - R[rs2]; PC \leftarrow PC + 4$

ori $R[rd] \leftarrow R[rs1] \mid Imm; PC \leftarrow PC + 4$

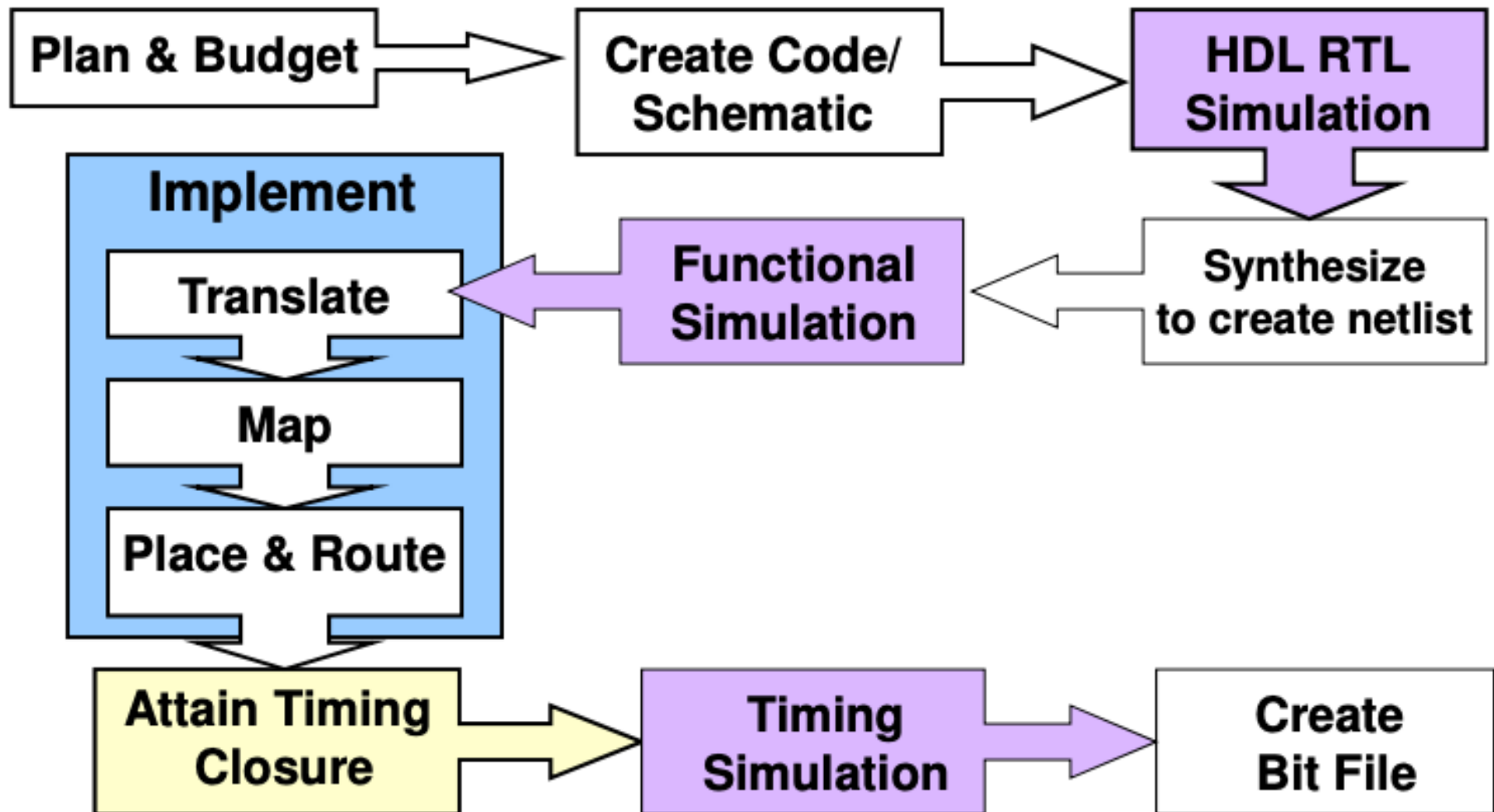
jal $R[rd] \leftarrow PC + 4; PC \leftarrow R[rs1] + Imm$

beq $\text{if } (R[rs1] == R[rs2])$
 $PC \leftarrow PC + Imm$
 $\text{else } PC \leftarrow PC + 4$

Hardware Description Languages

- Example HDL's : ABEL, VERILOG, VHDL
- Advantages:
 - Documentation
 - Flexibility (easier to make design changes or mods)
 - Portability (if HDL is standard)
 - One language for modeling, simulation (test benches), and synthesis
 - Let synthesis worry about gate generation
 - Engineer productivity
- However: A different way of approaching design
 - engineers are used to thinking and designing using graphics (schematics) instead of text.

Simulate Circuits (e.g. for FPGA)



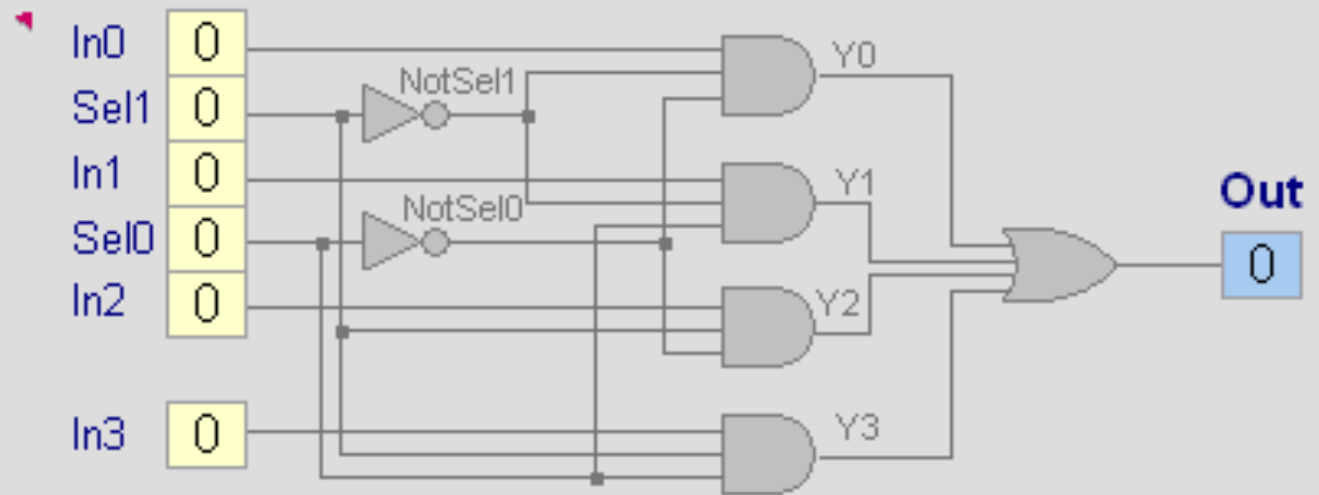
Synthesis

- Use a HDL to create:
 - VLSI (Very Large Scale Integration): create an IC (Integrated Circuit) – a chip – for example a CMOS CPU, Memory
 - ASIC (Application-specific integrated circuit): CMOS chip for a specific application in a specific device
 - a program for an FPGA (Field-programmable gate array): Programmable logic blocks that can perform combinatorial logic
 - Include state elements (memory)
 - E.g. Xilinx VU19P: 9million logic gates -> can simulate 16 Arm Cortex A9 cores at the same time!

Verilog and VHDL

- VHDL - like Pascal and Ada programming languages
 - VHDL (VHSIC-HDL) (Very High Speed Integrated Circuit Hardware Description Language)
- Verilog - more like 'C' programming language
- VHDL is strongly typed; Verilog weakly typed
- Verilog is easier to learn compared to VHDL
- VHDL with library management, more complex datatypes

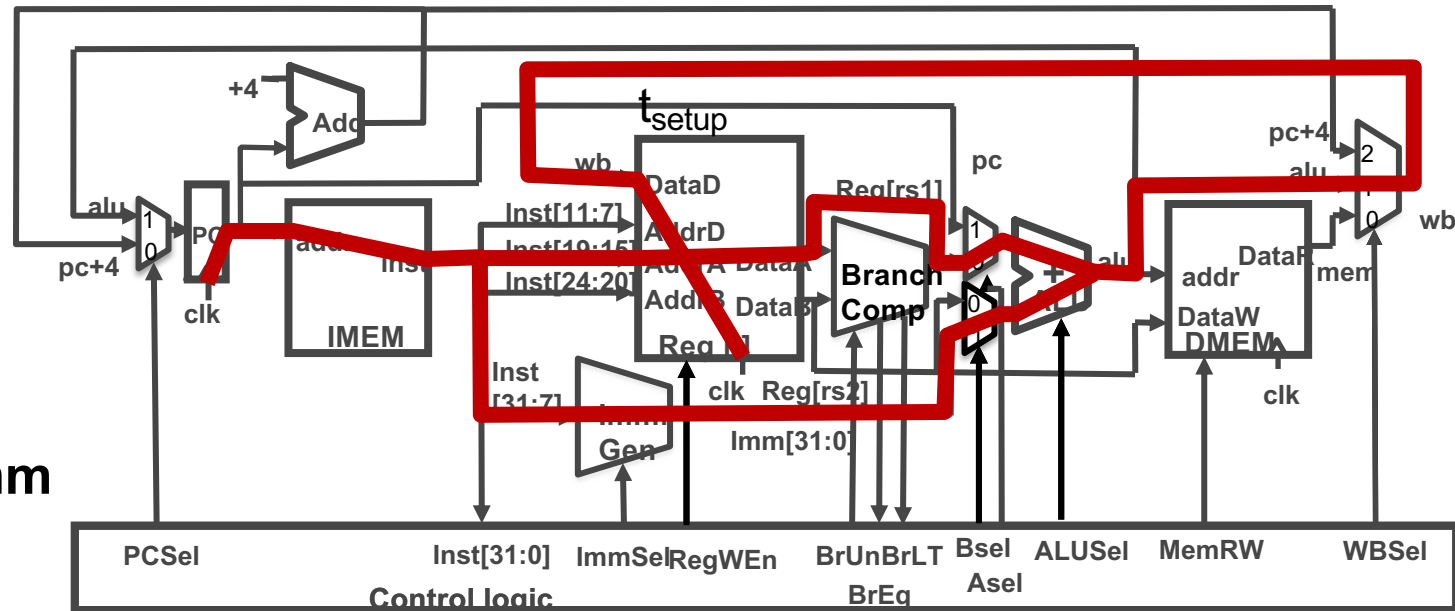
Verilog Example:



```
module mux_4_to_1 (Out, In0, In1, In2, In3, Sel1, Sel0);  
output Out;  
input In0, In1, In2, In3, Sel0, Sel1;  
  
wire NotSel0, NotSel1;  
wire Y0, Y1, Y2, Y3;  
  
not (NotSel0, Sel0);  
not (NotSel1, Sel1);  
and (Y0, In0, NotSel1, NotSel0);  
and (Y1, In1, NotSel1, Sel0);  
and (Y2, In2, Sel1, NotSel0);  
and (Y3, In3, Sel1, Sel0);  
or (Out, Y0, Y1, Y2, Y3);  
  
endmodule
```

How to make a CMOS chip?

Question: Critical Path

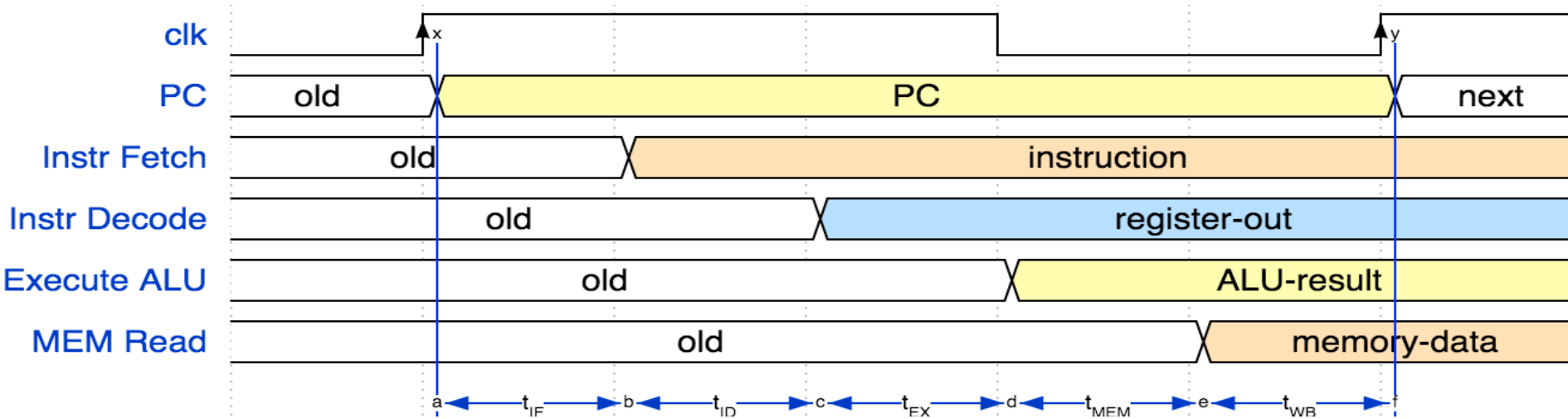


Critical path for xori

$$R[rd] = R[rs1] + imm$$

- A. $t_{clk-q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 2 * t_{mux} + t_{Setup}$
- B. $t_{clk-q} + t_{Add} + t_{IMEM} + t_{Reg} + t_{BComp} + t_{ALU} + t_{DMEM} + t_{mux} + t_{Setup}$
- C. $t_{clk-q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 3 * t_{mux} + t_{DMEM} + t_{Setup}$
- D. None of the above

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		X	600ps
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. $f_{\max, \text{ALU}} = 1/200\text{ps} = 5 \text{ GHz!}$

Performance

- “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	250 km/h	100 km/h
Fuel consumption	20 l/100km	20 l/100km

Schwerin => Berlin trip: 200 km



=>



Transportation Analogy



	Sports Car	Bus
Passenger Capacity	2	50
Travel Speed	250 km/h	100 km/h
Fuel consumption	20 l/100km	20 l/100km

Schwerin => Berlin trip: 200 km

	Sports Car	Bus
Travel Time	48 min	120 min
Time for 100 passengers	40 h	4 h
Fuel per passenger	2000 l	80 l

Computer Analogy

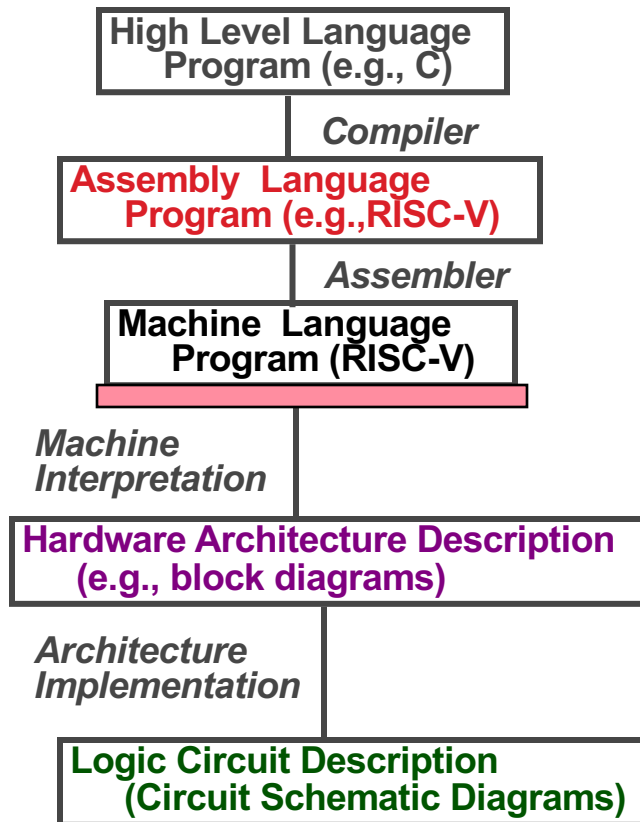
Transportation	Computer
Travel Time	Program execution time (latency) e.g. time to update display
Time for 100 passengers	Throughput: e.g. number of server requests handled per hour
Fuel per passenger	Energy per task*: e.g.: - how many movies can you watch per battery charge - energy bill for datacenter

* 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

Next Lecture:

- Improve performance through pipelining:
 - One stage per clock cycle!
 - Need 5 clock cycles to complete one instruction
 - Clock cycles much shorter now
- => Higher throughput 😊
- => Higher latency ☹️

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



“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