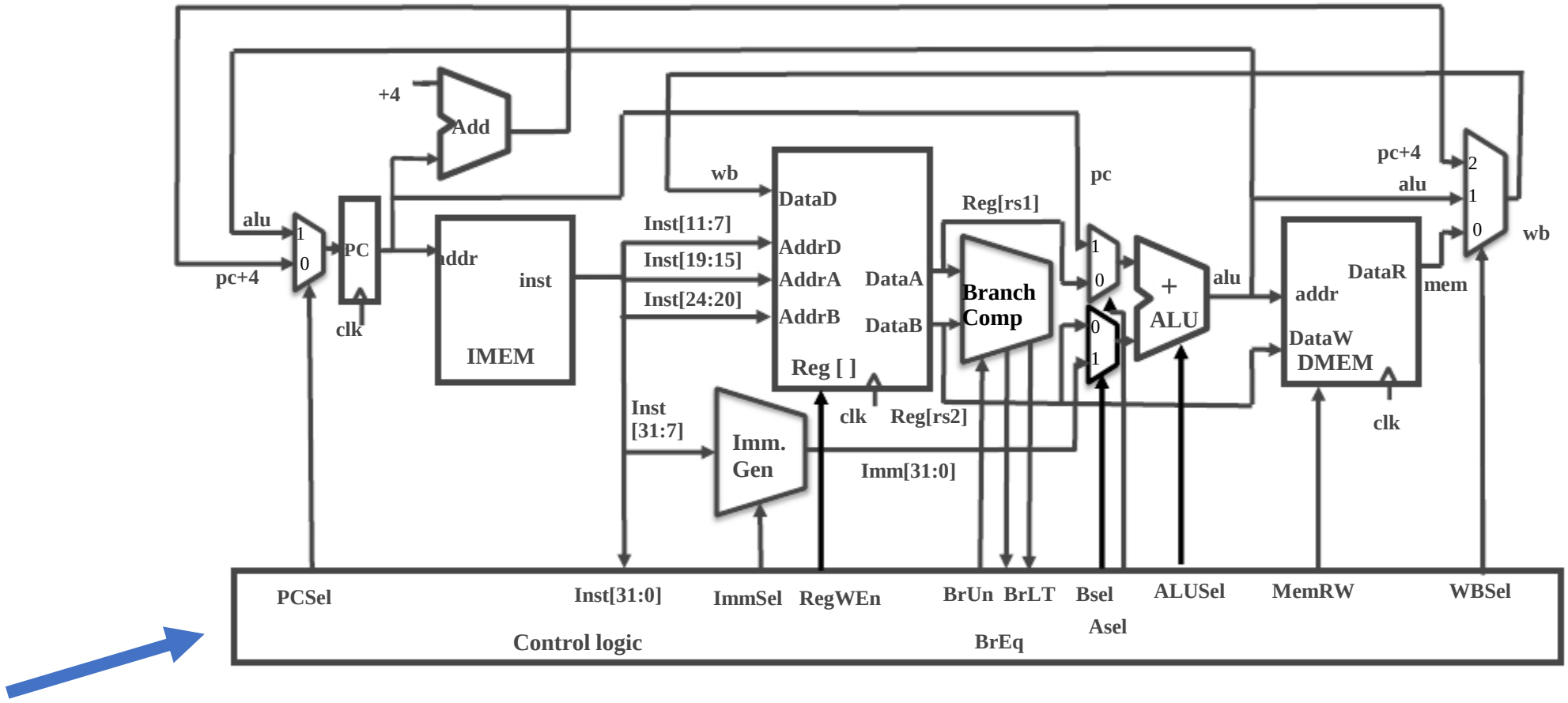
Discussion 8

Control Logic & FSM

Yida Zhao

zhaoyd1@shanghaitech.edu.cn

Complete RV32I Datapath!



Control Logic Truth Table

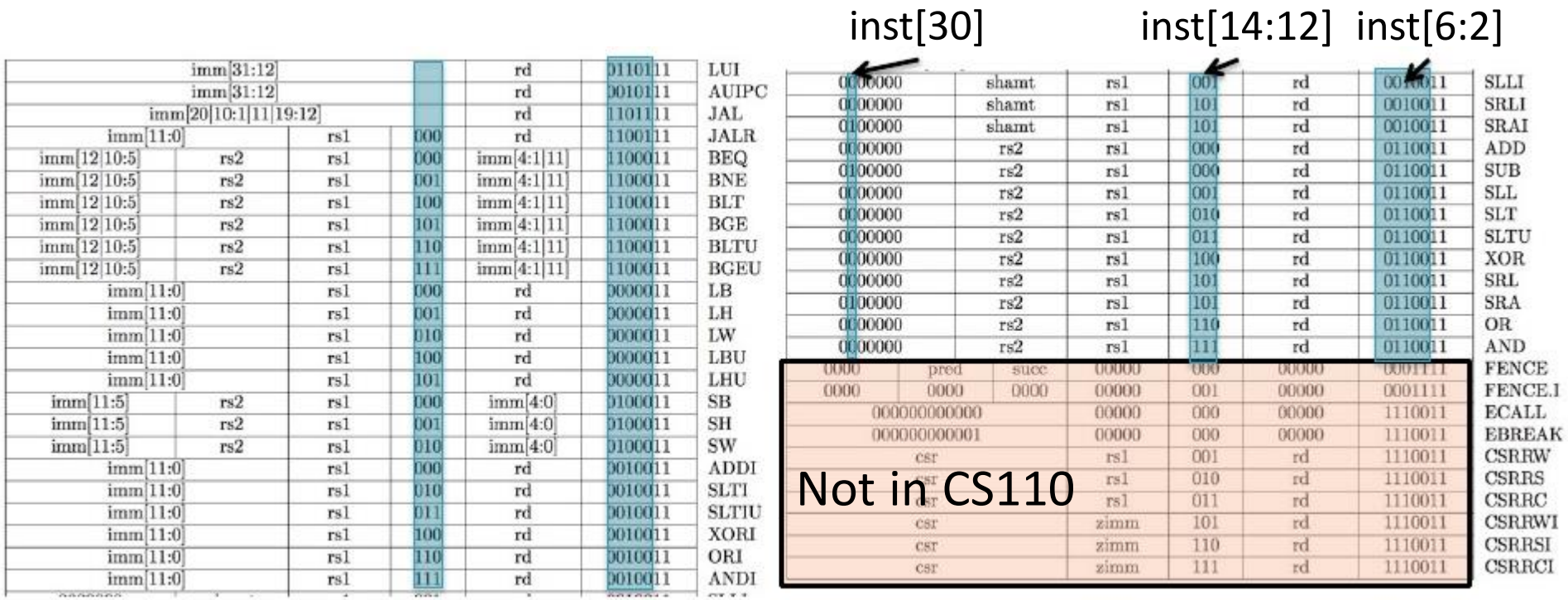
Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
`add`	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
`sub`	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	*(Op)*	Read	1	ALU
`addi`	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
`lw`	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
`sw`	*	*	+4	S	*	Reg	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

Implementation: ROM or Combinational Logic

Combinational Logic: faster

ROM: easy to design and reprogram

RV32I, a nine-bit ISA!

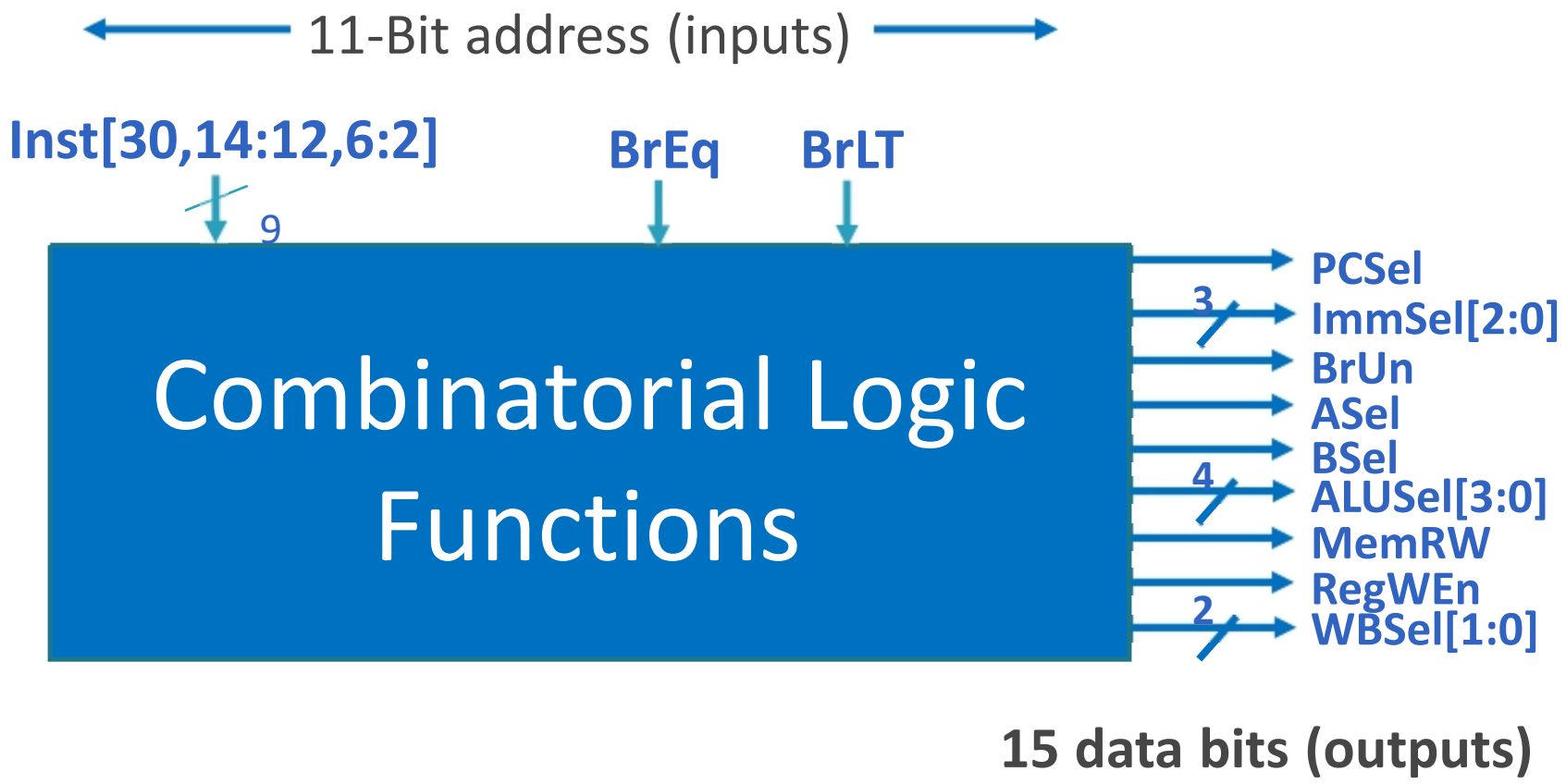


imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	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

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]

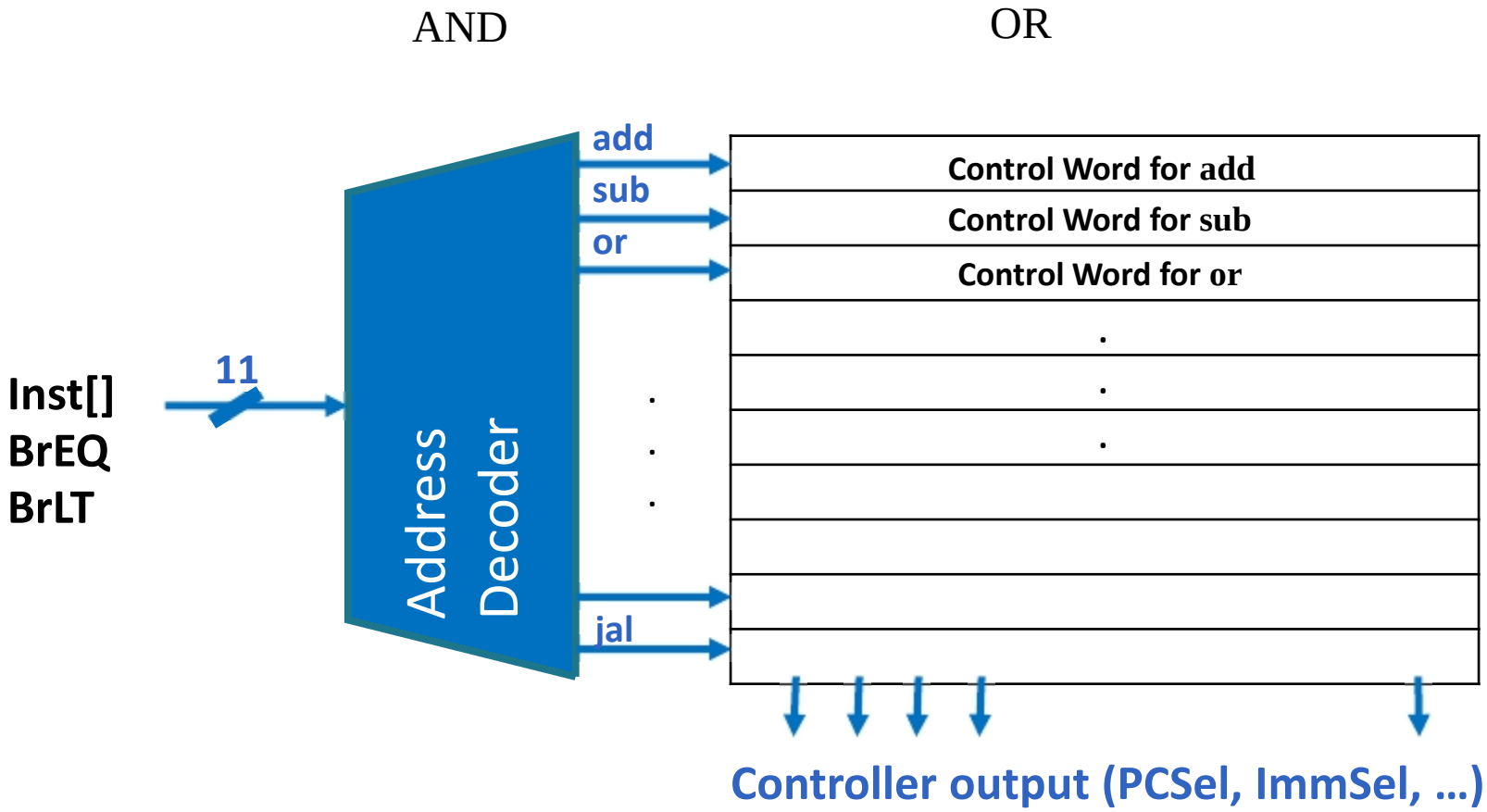
Control Block Design



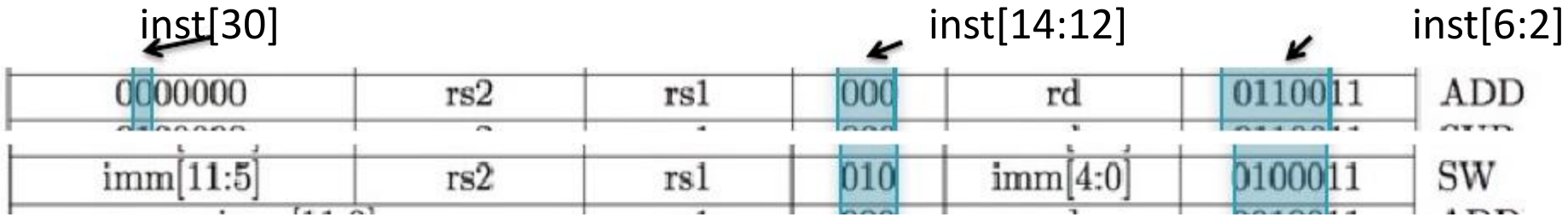
Control Logic Truth Table

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
(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

ROM Controller Implementation



Controller: AND logic (add, sw)



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

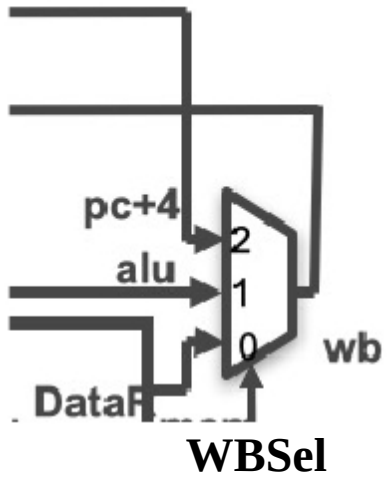
AND Controller Logic

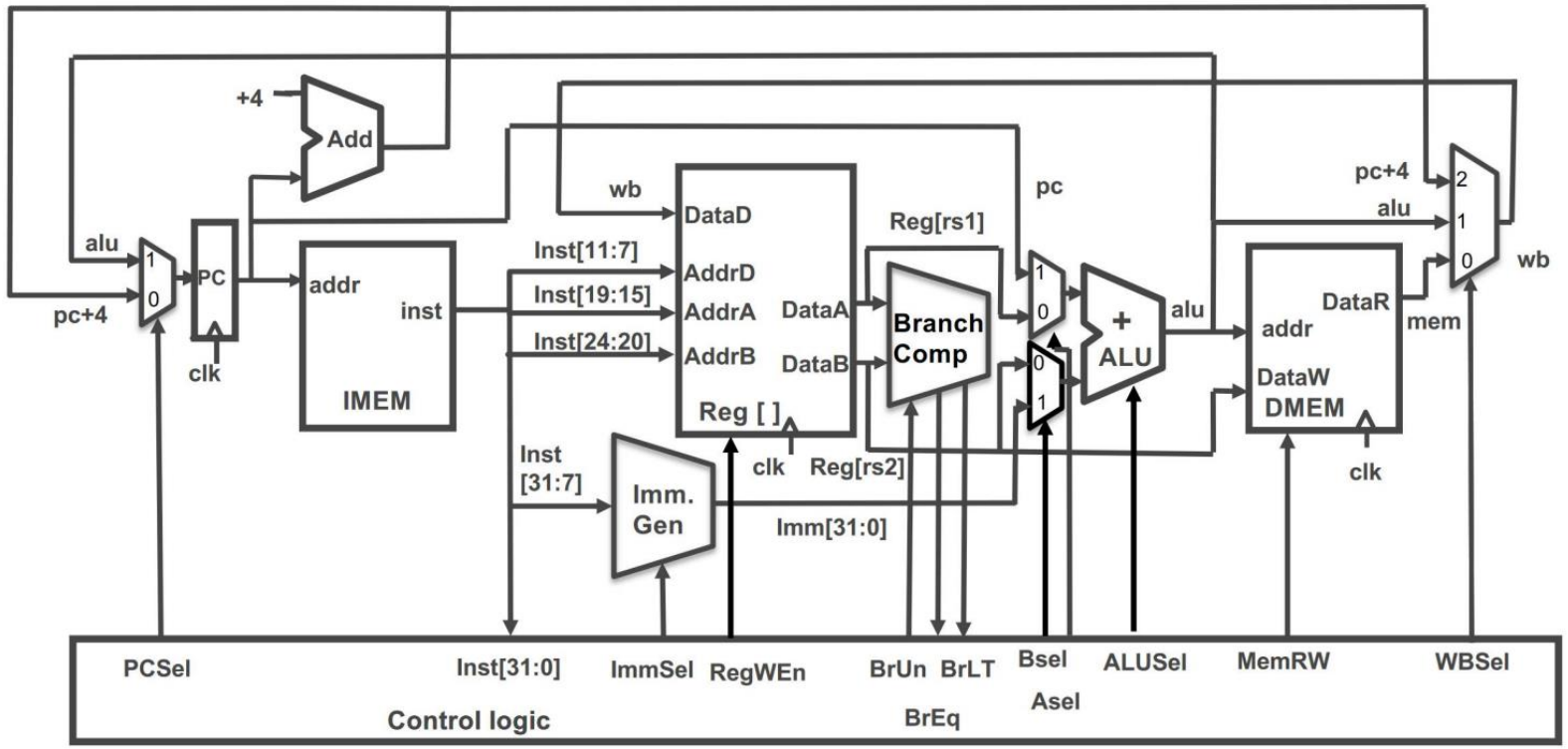
		2		3		4		5		6		12		13		14		30		unused	
add	xor(i[2],	1	) • xor(i[3],	1	) • xor(i[4],	0	) • xor(i[5],	0	) • xor(i[6],	1	) • xor(i[12],	1	) • xor(i[13],	1	) • xor(i[14],	1	) • (xor(i[30],	1	) +	0	)
sw	xor(i[2],	1	) • xor(i[3],	1	) • xor(i[4],	1	) • xor(i[5],	0	) • xor(i[6],	1	) • xor(i[12],	1	) • xor(i[13],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

- bne_t = bne • $\overline{\textbf{BrEQ}}$
- bne_f = bne • **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
- …





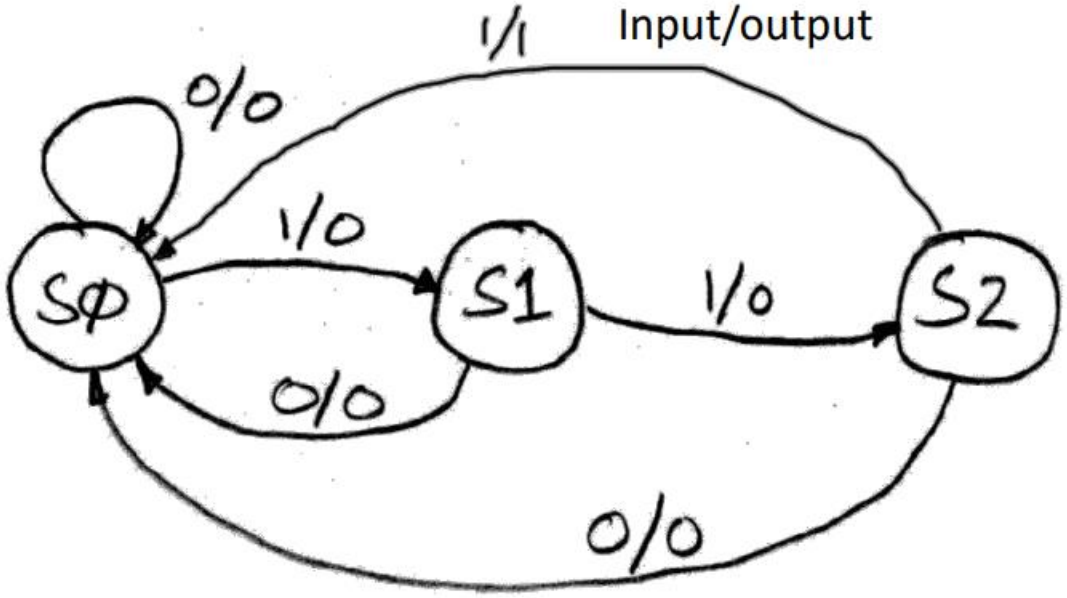
(c) Assume t3 = 0x8fffffff, t4 = 0x0fffffff. Write down control signals for **blt t3, t4, label**. Please use * to indicate that what this signal is does not matter.

PCSel	ImmSel	RegWEn	BrUn	BrEq	BrLT	ASel	BSel	ALUSel	MemRW	WBSel

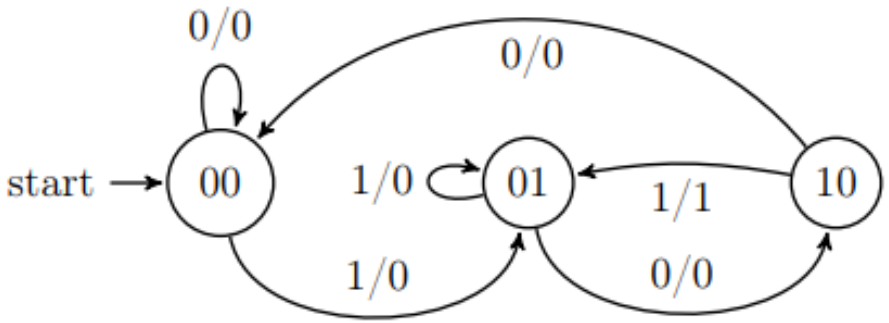
Solution: PCSel = 1 ImmSel = B RegWEn = 0 BrUn = 0 BrEq = 0 BrLT = 1 ASel = 1 BSel = 1 ALUSel = Add MemRW = Read WBSel = *

Finite State Machines

- A convenient way to conceptualize computation over time
- Start at a state, given an input, follow some edge to another
- With combinational logic and registers, any FSM can be implemented in hardware
- edge: input/output

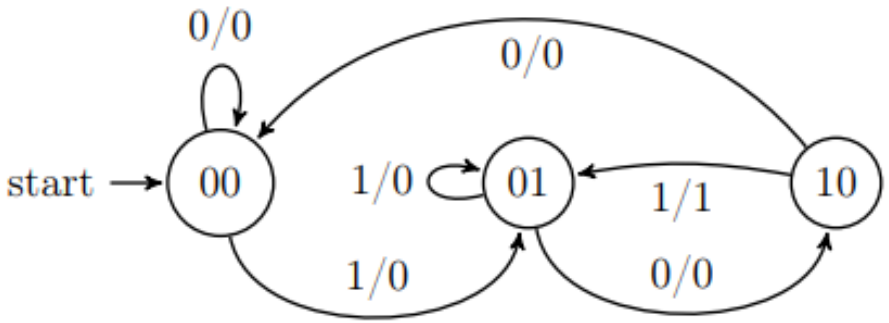


Example



state bit1	state bit0	Input	next state bit1	next state bit0	Output
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	0	1	1

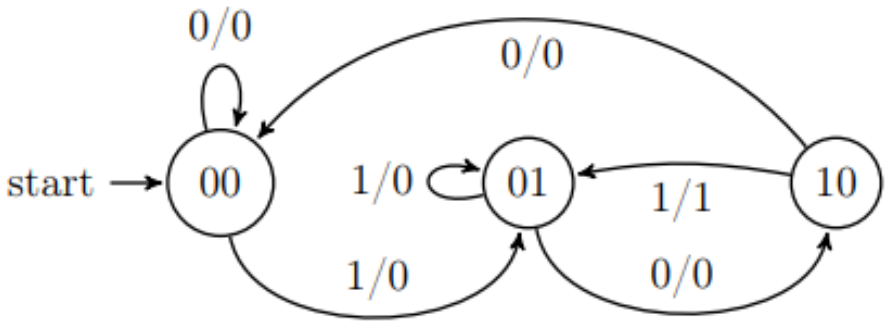
Example



What does the given FSM output with the input bit string '0100101010'?

Solution: It outputs '0000001010'.

Example



What does the given FSM implement (Describe when the FSM will output 1)?

Solution: When the FSM receives '101', it outputs 1.

(d) Draw a FSM that outputs 1 when it receives two or more successive '0'.

Solution: