CSE140: Components and Design Techniques for Digital Systems

Register Transfer Level (RTL) Design

Instructor: Mohsen Imani

Slides from Tajana Simunic Rosing

CAPE

CAPEs are out!!!

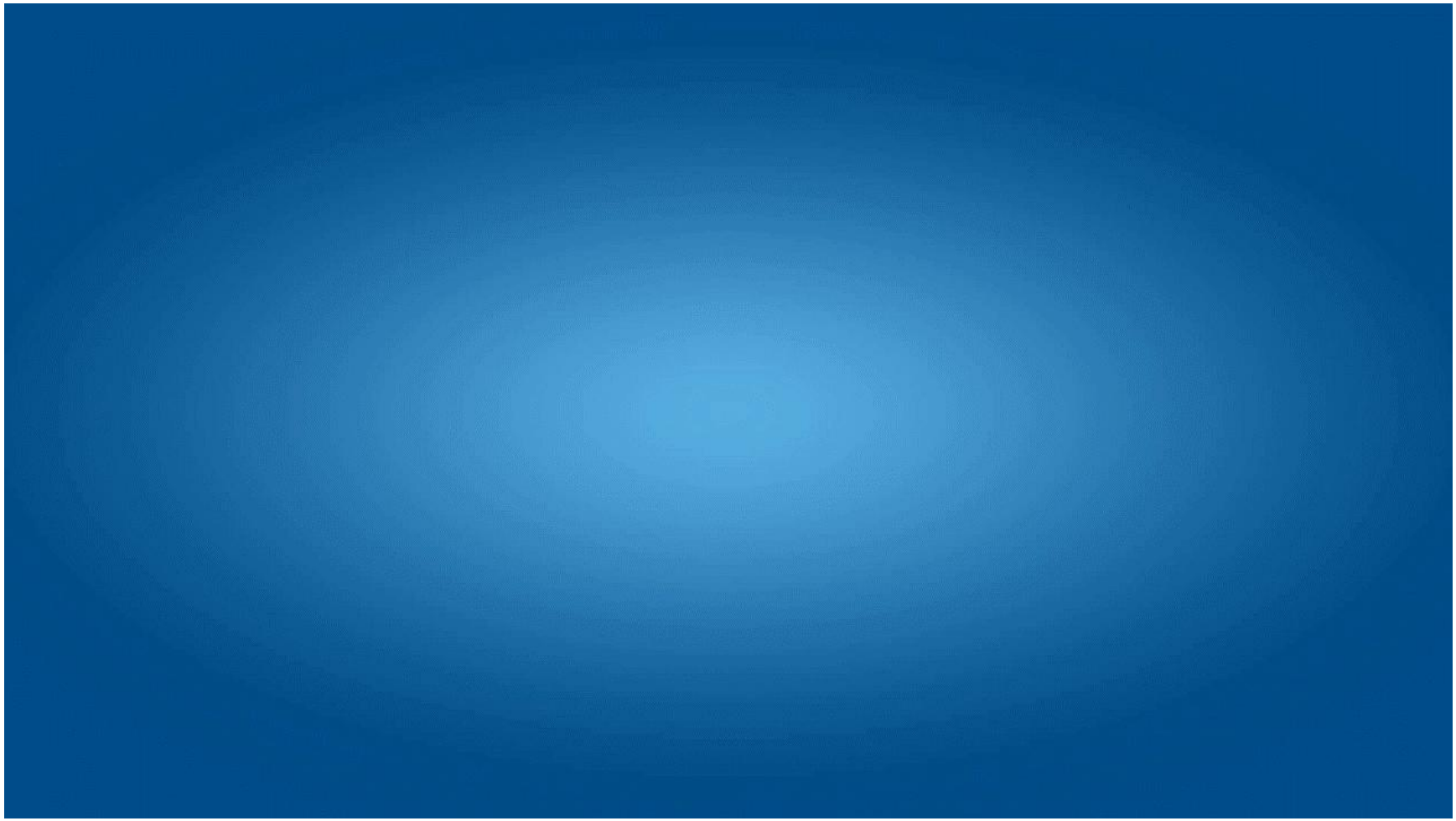
https://cape.ucsd.edu/students/

Please submit your evaluations !!!!

Your feedback is very important, please take the time to fill out the survey.

Sources: TSR, Katz, Boriello, Vahid, Perkowski

3D Xpoint Memory



Sources: TSR, Katz, Boriello, Vahid, Perkowski

About the final

~150 minutes Final Exam

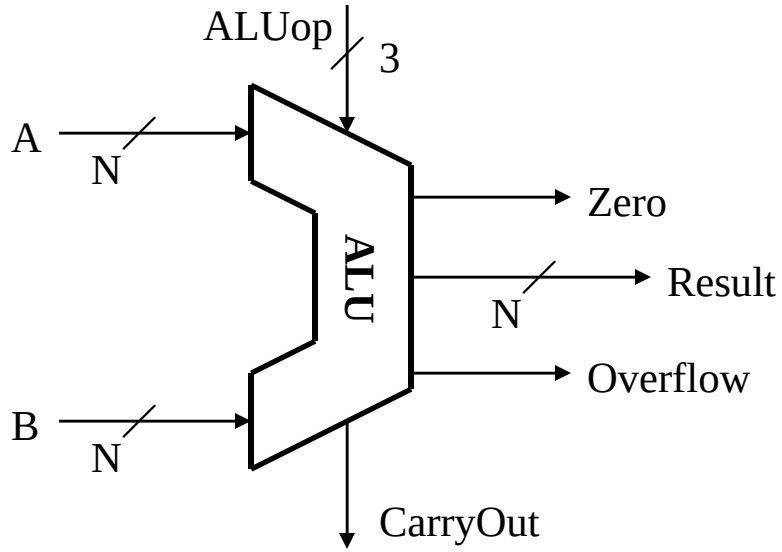
Bring one 8 ½ x 11” paper with handwritten notes, but nothing else

Problems including (but not limited to):

- ALU design
- ALU components (shifters, adders, etc.)
- FSM
  - Moore, Mealy, design of FSM using state table, excitation table
- Timing constraints
- RTL design and HLSM

ALU: Arithmetic Logic Unit

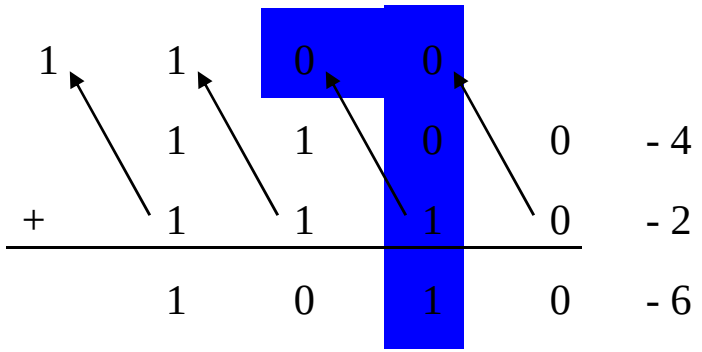
Designing an Arithmetic Logic Unit

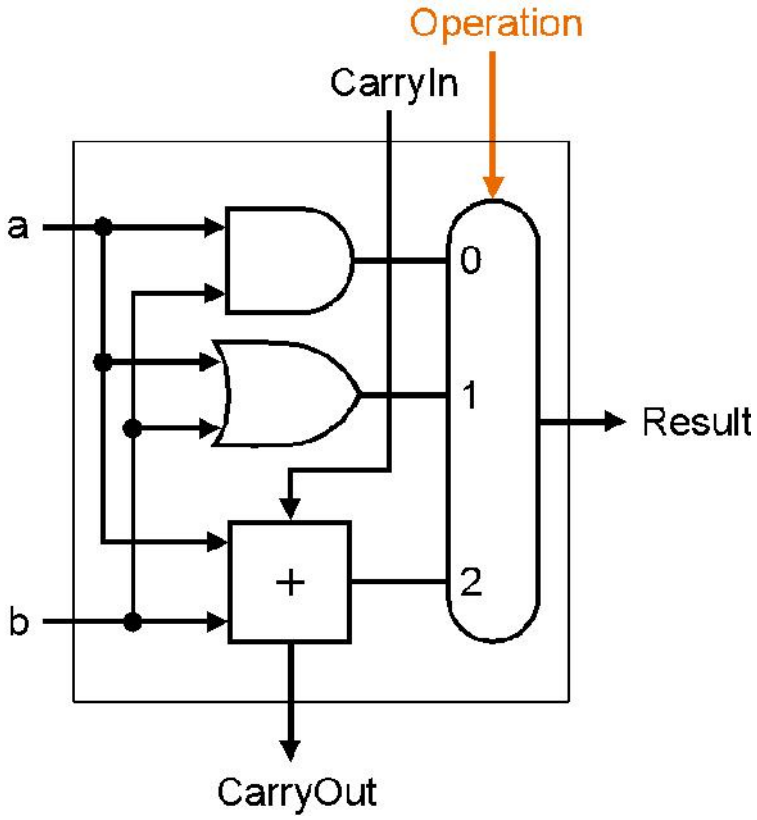


- **ALU Control Lines (ALUop) — Function**
  - **000 — And**
  - **001 — Or**
  - **010 — Add**
  - **110 — Subtract**
  - **111 — Set-on-less-than**

A One Bit ALU

- This 1-bit ALU performs AND, OR, and ADD

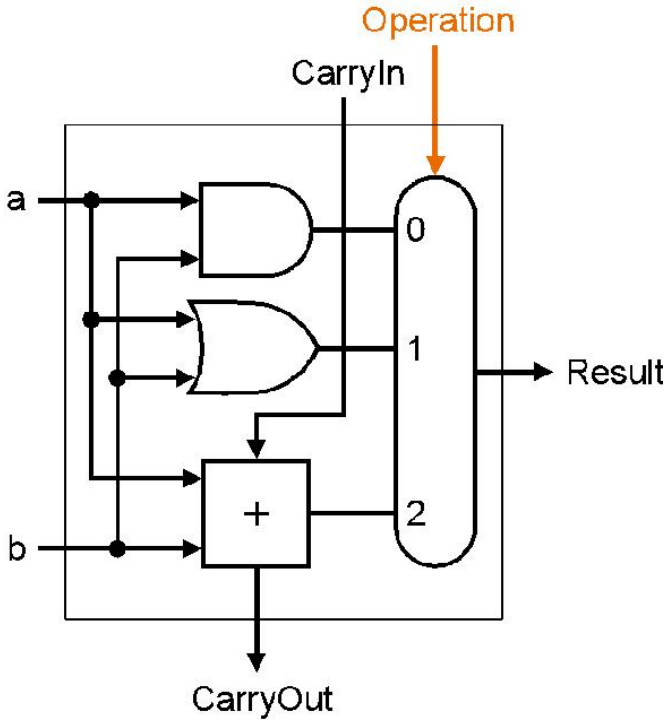




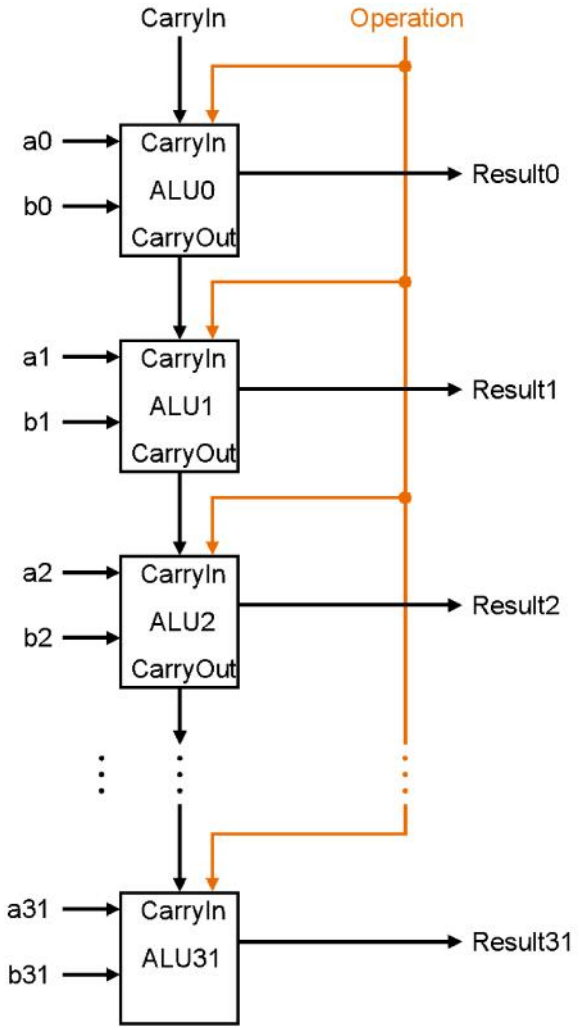
Sources: TSR, Katz, Boriello, Vahid, Perkowski

A 32-bit ALU

1-bit ALU



32-bit ALU

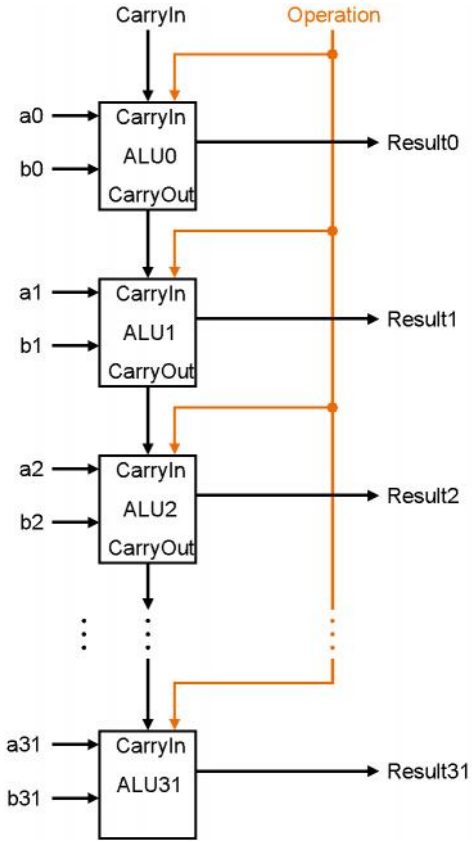


Sources: TSR, Katz, Boriello, Vahid, Perkowski

Subtract – We'd like to implement a means of doing A-B (subtract) but with only minor changes to our hardware. How?

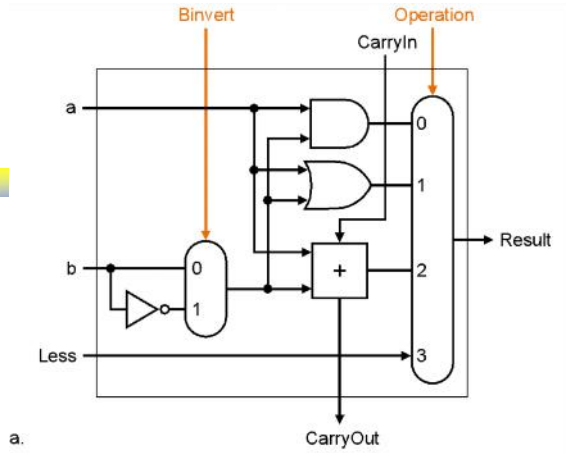
1. Provide an option to use bitwise NOT A
2. Provide an option to use bitwise NOT B
3. Provide an option to use bitwise A XOR B
4. Provide an option to use 0 instead of the first CarryIn
5. Provide an option to use 1 instead of the first CarryIn

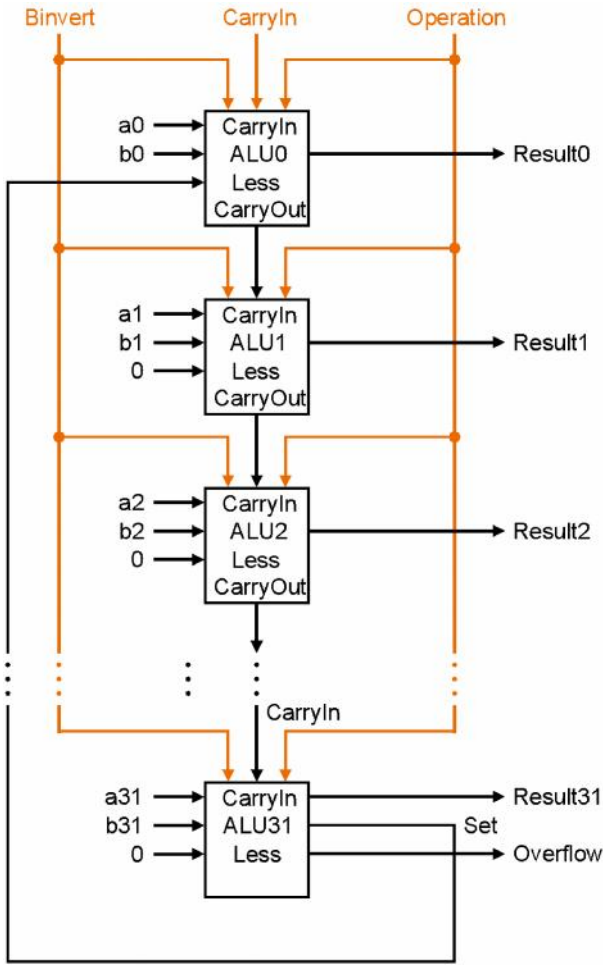
Selection	Choices
A	1 alone
B	Both 1 and 2
C	Both 3 and 4
D	Both 2 and 5
E	None of the above



Sources: TSR, Katz, Boriello, Vahid, Perkowski

Full 32-bit ALU





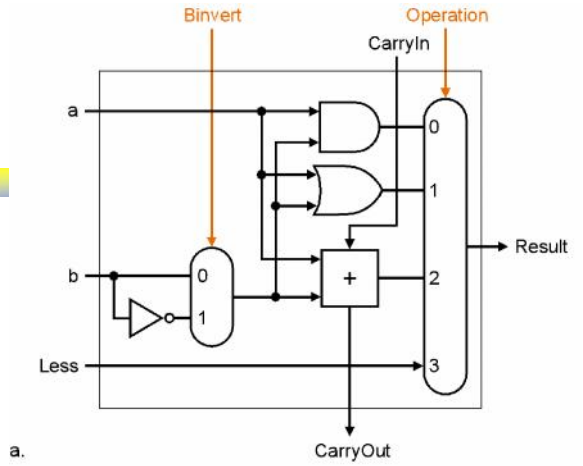
sign bit (adder output from bit 31)

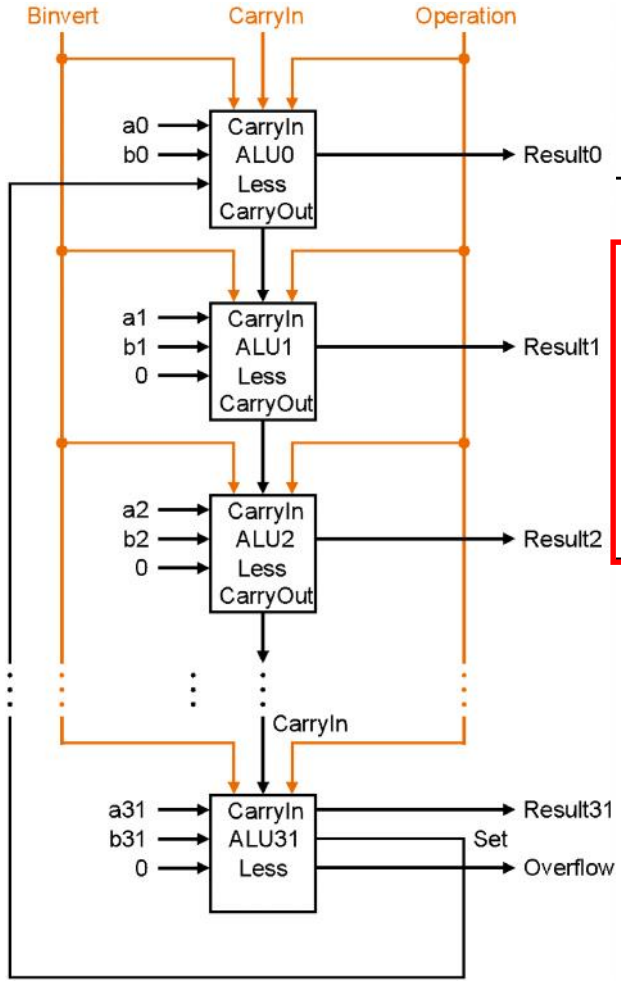
what signals accomplish ADD?

	Binvert	CIn	Oper
A	1	0	2
B	0	1	2
C	1	1	2
D	0	0	2
E	NONE OF THE ABOVE		

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Full 32-bit ALU





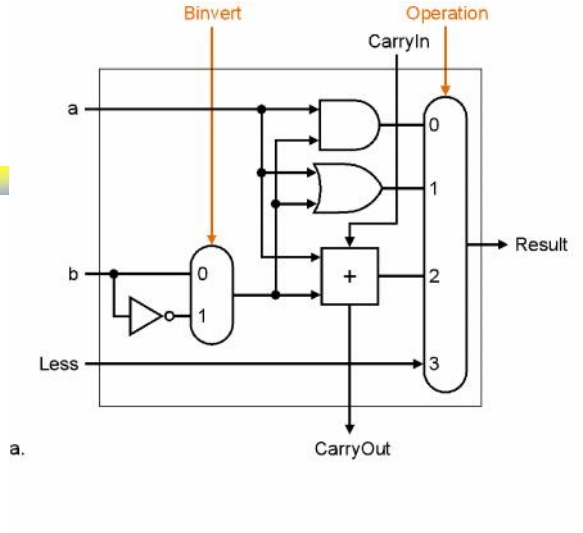
sign bit (adder output from bit 31)

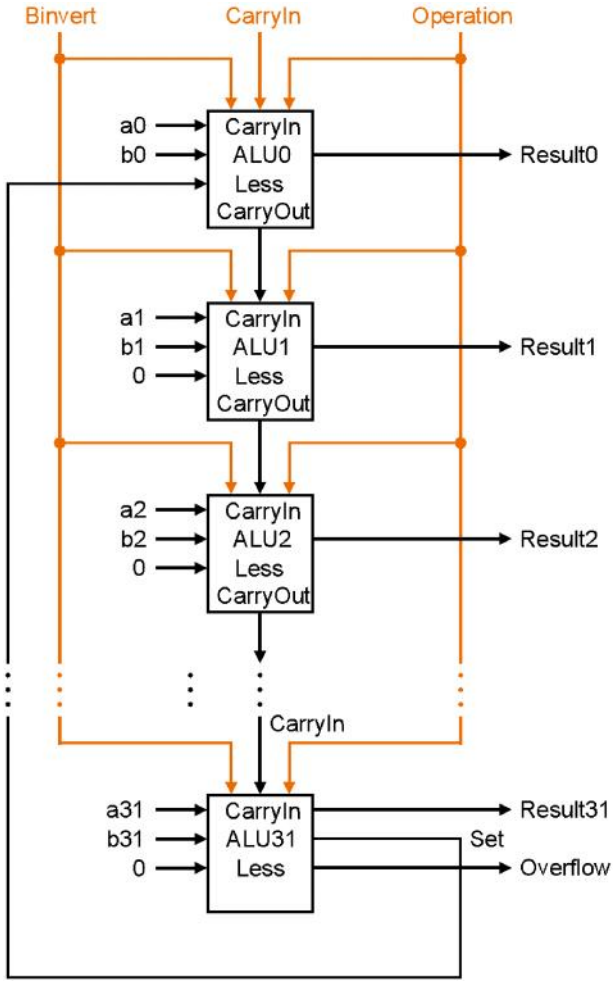
what signals accomplish OR?

	Binvert	CIn	Oper
A	1	0	0
B	0	1	1
C	1	1	0
D	1	0	1
E	NONE OF THE ABOVE		

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Full 32-bit ALU





sign bit (adder output from bit 31)

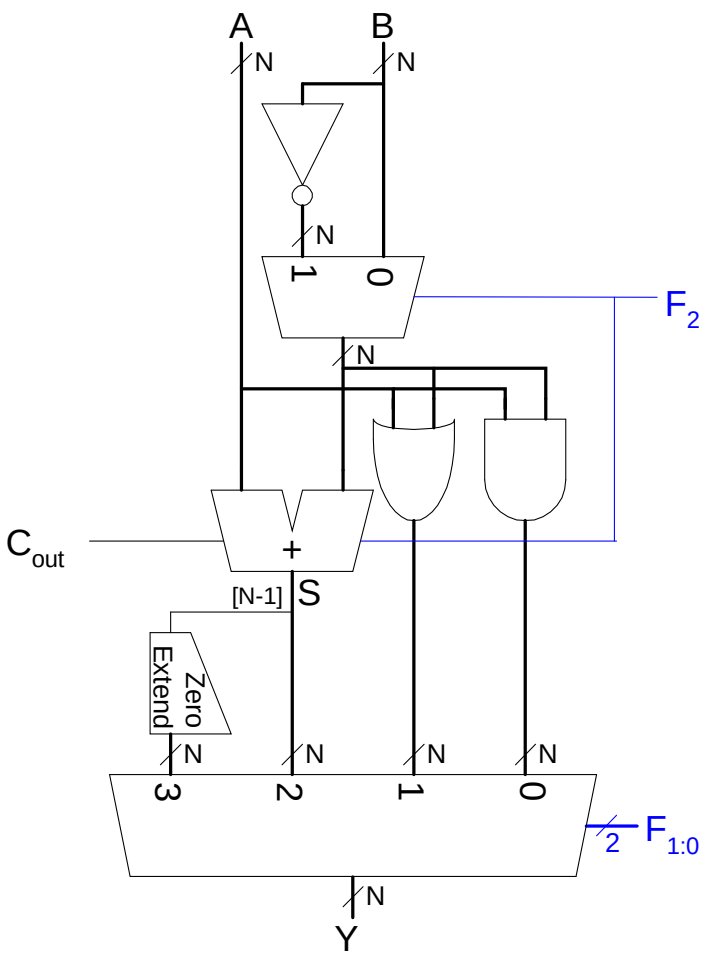
Little more intense – can you get this?

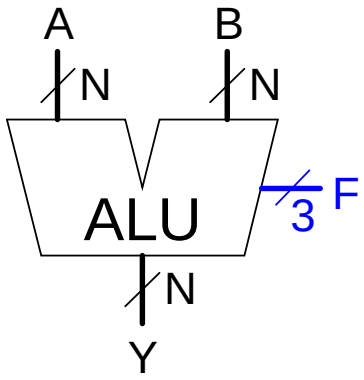
what signals accomplish SUB?

	Binvert	CIn	Oper
A	1	0	2
B	0	1	2
C	1	1	2
D	0	0	2
E	NONE OF THE ABOVE		

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Arithmetic Logic Unit – Example 2





$F_{2:0}$	Function
000	A & B
001	A | B
010	A + B
011	Not used
100	A & ~B
101	A | ~B
110	A - B
111	Not used

Sources: TSR, Katz, Boriello, Vahid, Perkowski

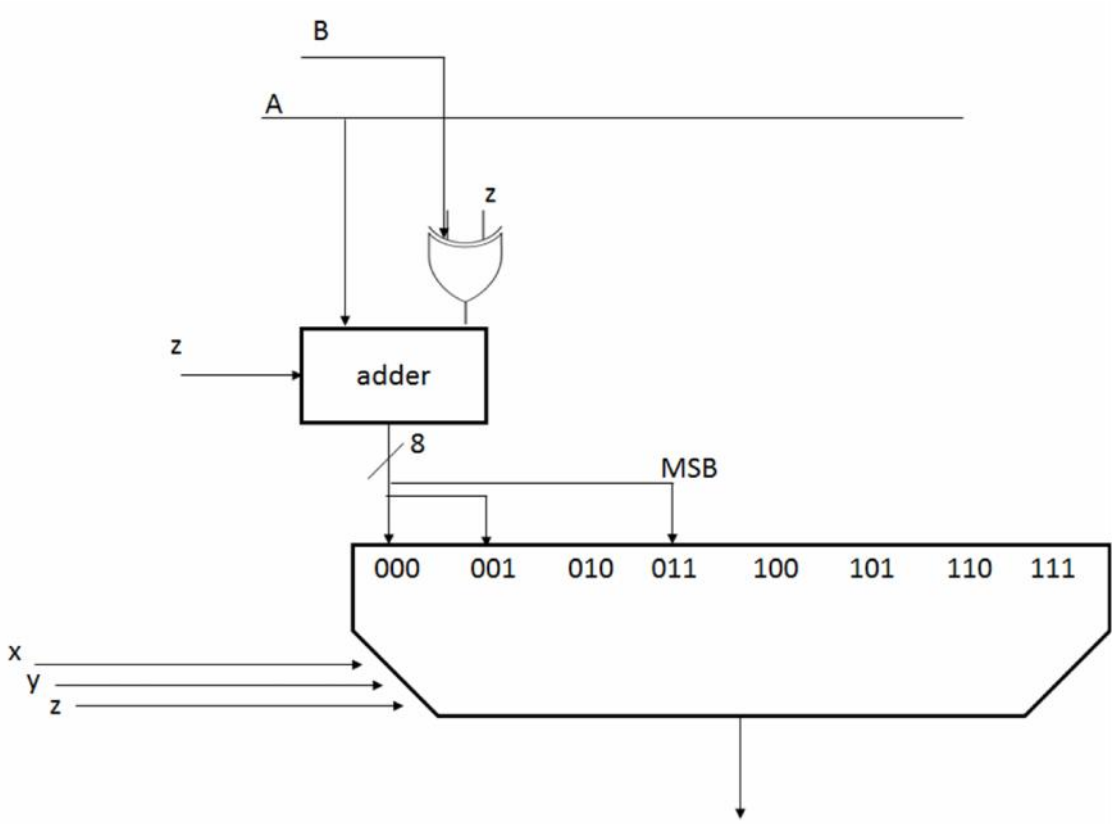
ALU Problem

x	y	z	OP
0	0	0	A + B
0	0	1	A - B
0	1	0	A + A
0	1	1	A < B
1	0	0	A * 4
1	0	1	A / 4
1	1	0	Reverse (A)
1	1	1	2-complement (B)

Design an ALU with two 8-bit inputs A and B that has three control bits xyz and implements the operations described in the table

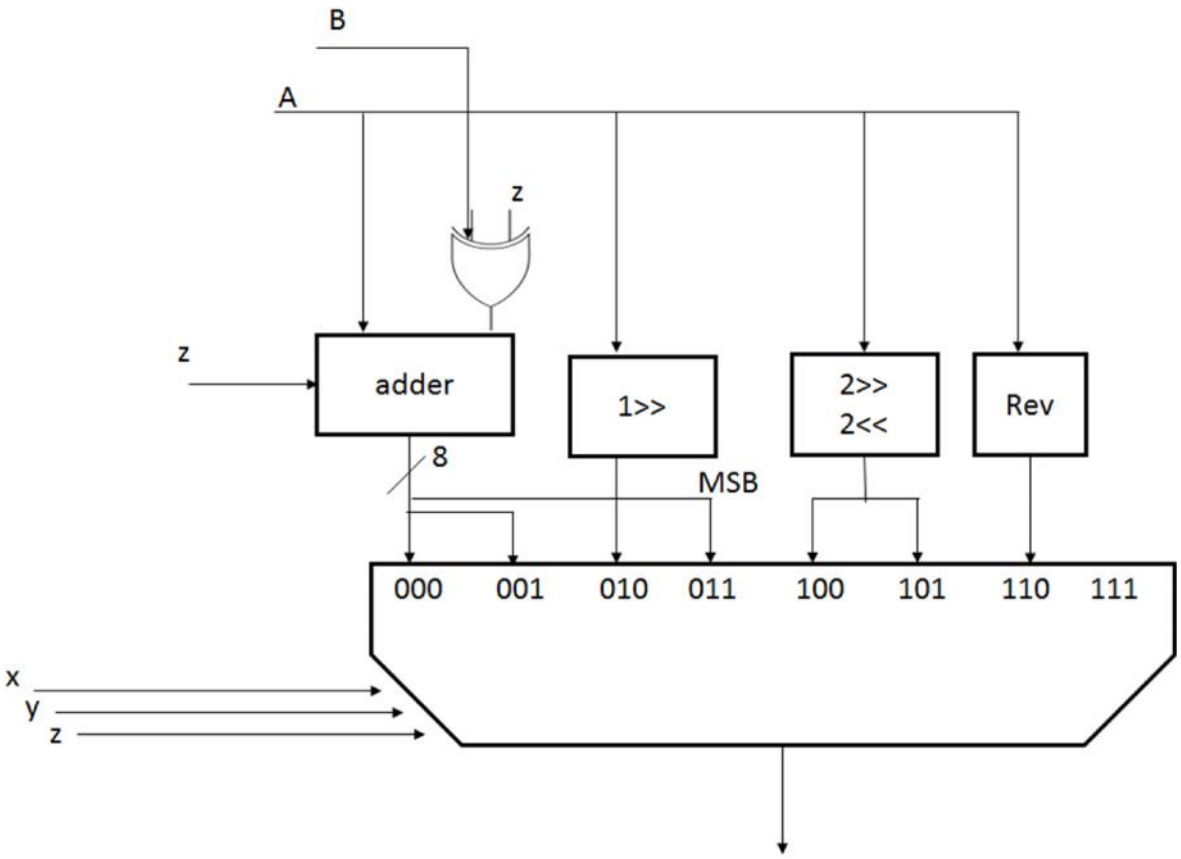
ALU Problem

x	y	z	OP
0	0	0	A + B
0	0	1	A - B
0	1	0	A + A
0	1	1	A < B
1	0	0	A * 4
1	0	1	A / 4
1	1	0	Reverse (A)
1	1	1	2-complement (B)



ALU Problem

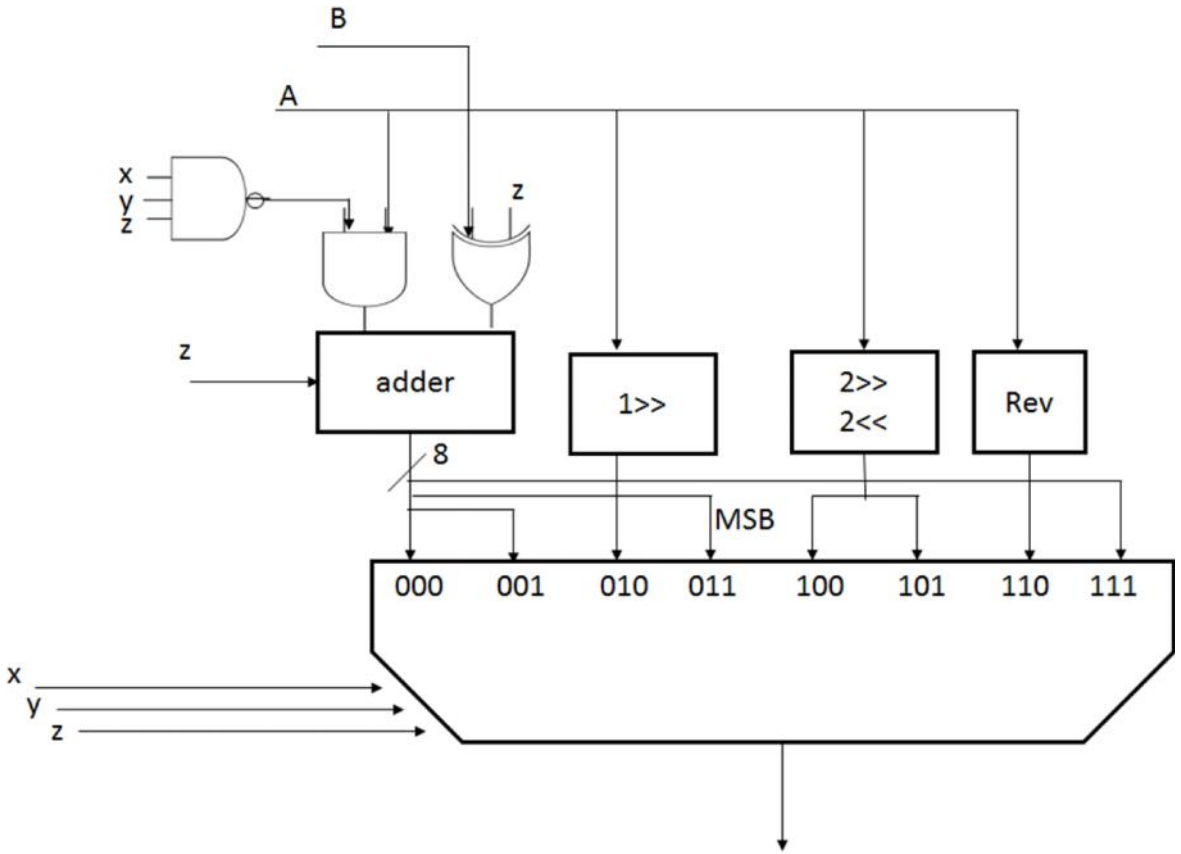
x	y	z	OP
0	0	0	A + B
0	0	1	A - B
0	1	0	A + A
0	1	1	A < B
1	0	0	A * 4
1	0	1	A / 4
1	1	0	Reverse (A)
1	1	1	2-complement (B)



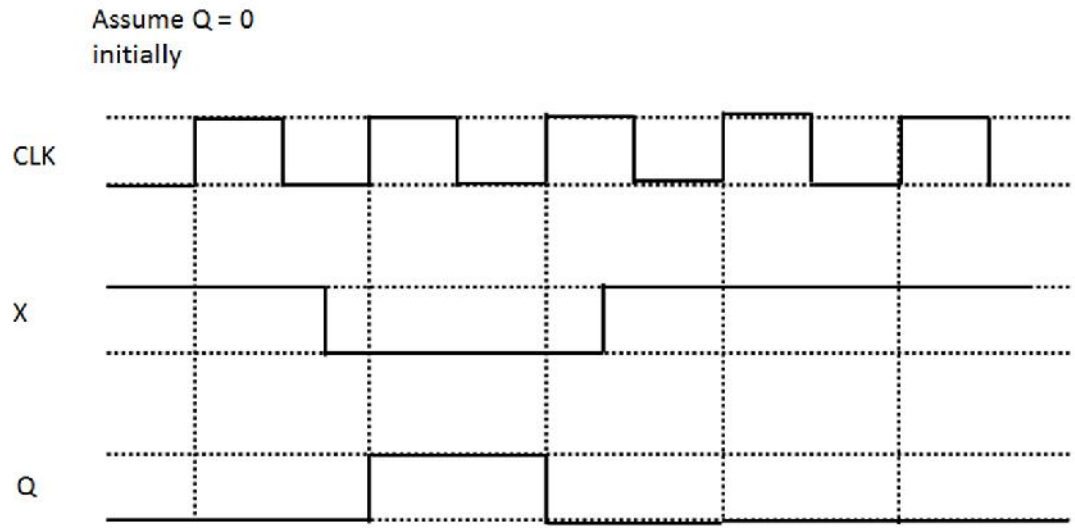
Sources: TSR, Katz, Boriello, Vahid, Perkowski

ALU Problem

x	y	z	OP
0	0	0	A + B
0	0	1	A - B
0	1	0	A + A
0	1	1	A < B
1	0	0	A * 4
1	0	1	A / 4
1	1	0	Reverse (A)
1	1	1	2-complement (B)



Design of a new FF (waveforms)



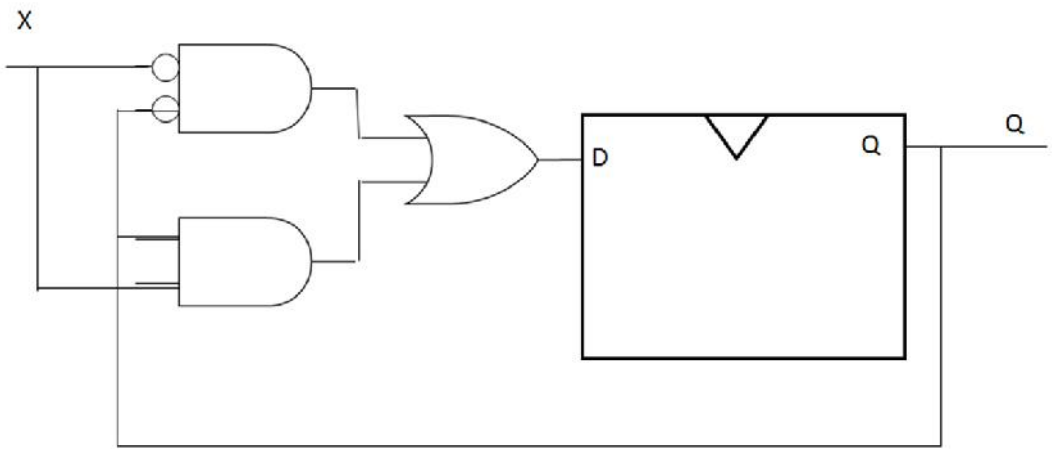
Q(t)	x	Q(t+1)
0	0	
0	1	
1	0	
1	1	

Sources: TSR, Katz, Boriello, Vahid, Perkowski

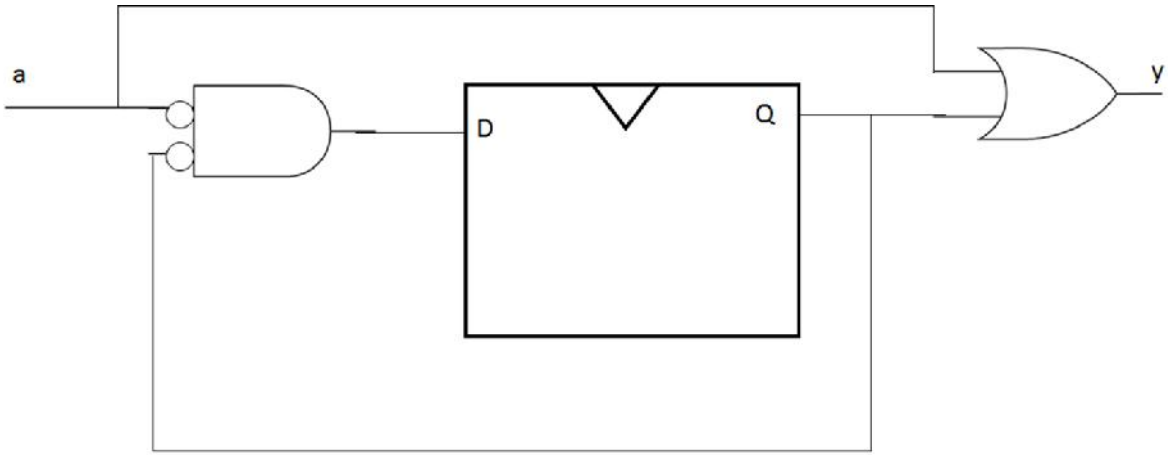
Design of a new FF

$$Q(t+1) = Q(t)'x' + Q(t)x$$

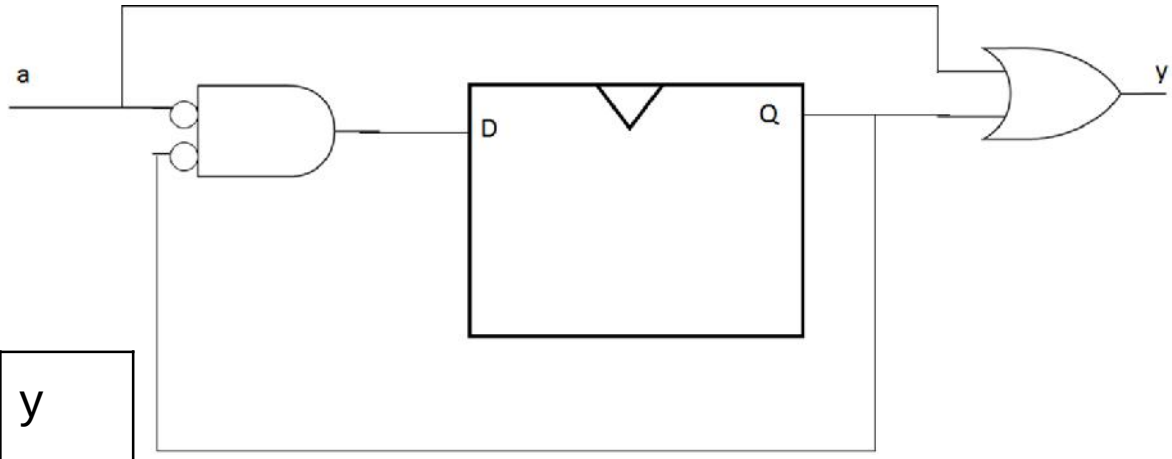
Q(t)	x	Q(t+1)
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**



Problem: FSM from circuit to state diagram



Problem: FSM from circuit to state diagram



D0 =

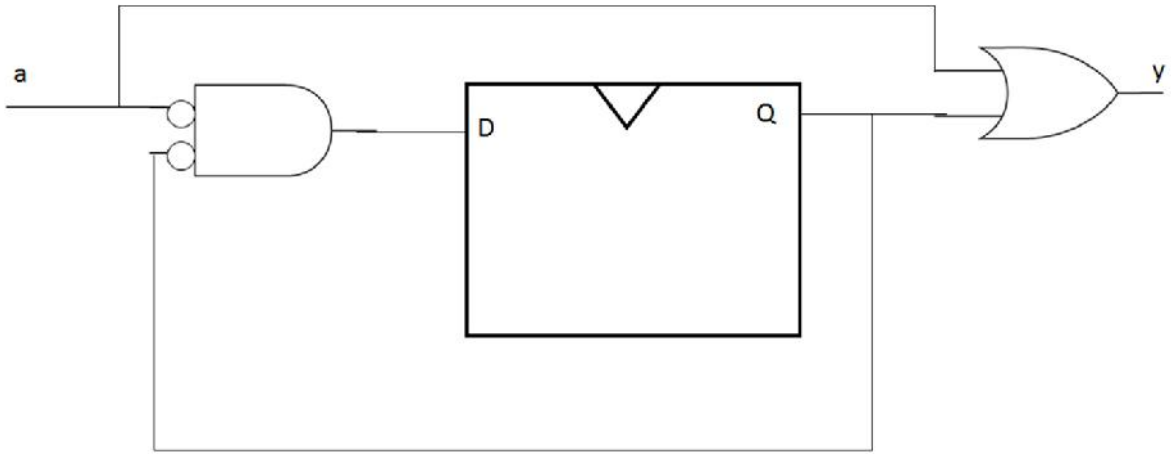
y =

Q0 a	D0	y
00		
01		
10		
11		

Qo \ a	0	1
0		
1		

Sources: TSR, Katz, Boriello, Vahid, Perkowski

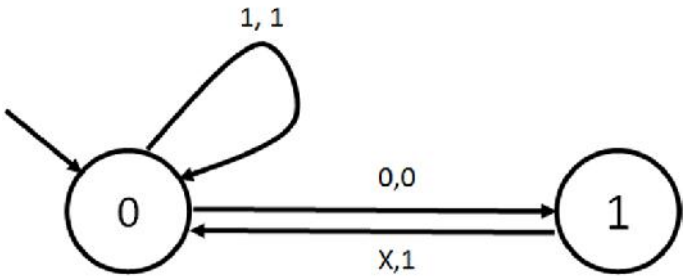
Problem: FSM from circuit to state diagram



D0 = Q0'*a'
y = Qo + a

Q0 a	D0	y
0 0	1	0
0 1	0	1
1 0	0	1
1 1	0	1

Qo \ a	0	1
0	1,0	0,1
1	0,1	0,1



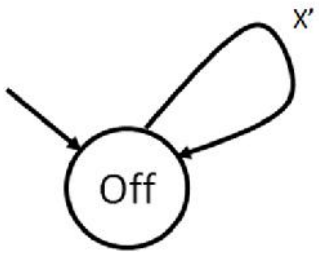
Sources: TSR, Katz, Boriello, Vahid, Perkowski

Problem: FSM from word description to state diagram

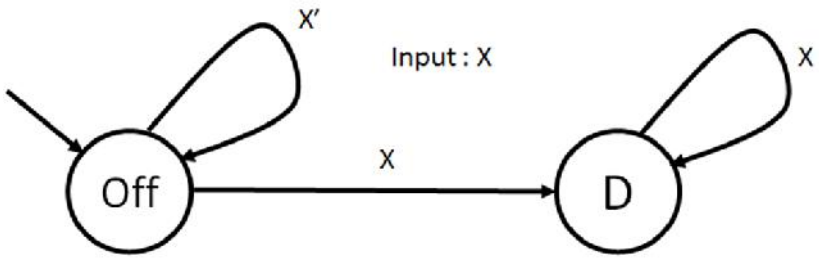
A microwave oven has 3 modes of operation when it is on: High temperature (H), Low temperature (L) and Defrost (D). A 1-bit input signal X is controlled by a button. Whenever the button is pressed, X changes its value. When the oven is off, X = 1 will make it switch on to D mode. After that, every time X changes its value, the oven goes to L, then H the off again.

a. Is it a Moore or a Mealy machine ?
b. How many states does it have? How many bits are required to represent them?
c. Draw the state diagram

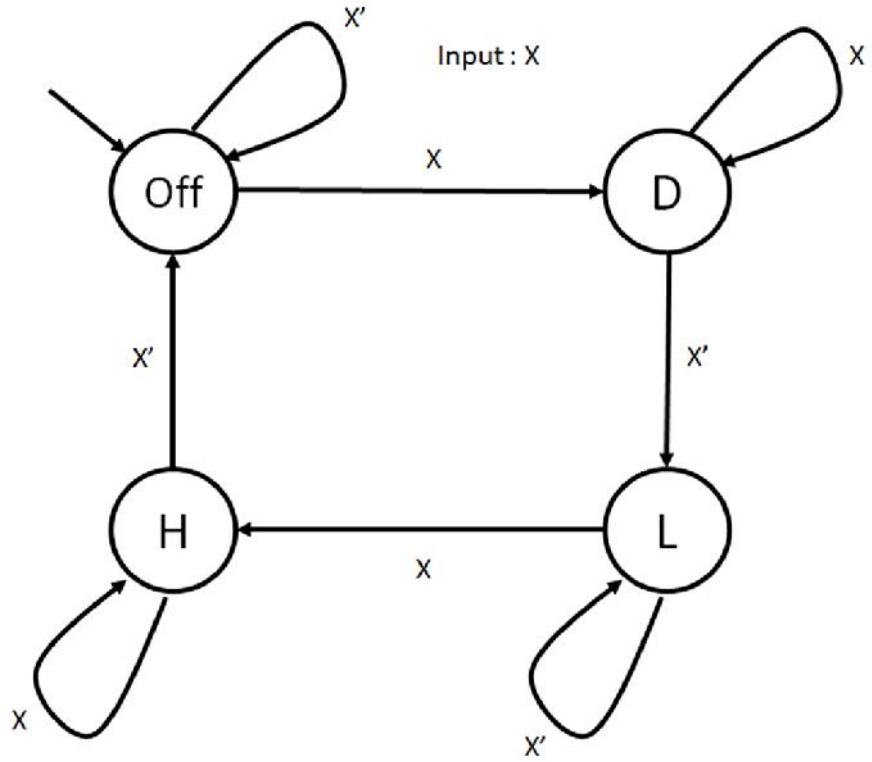
Solution



Solution

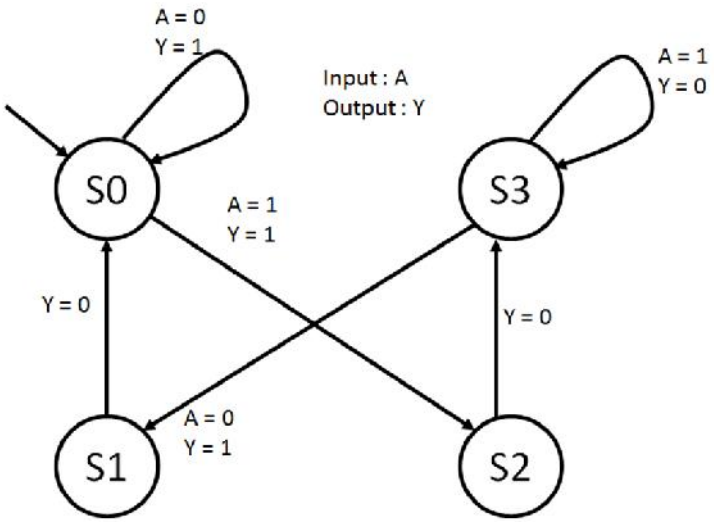


Solution



Sources: TSR, Katz, Boriello, Vahid, Perkowski

Problem: FSM from state diagram to circuit

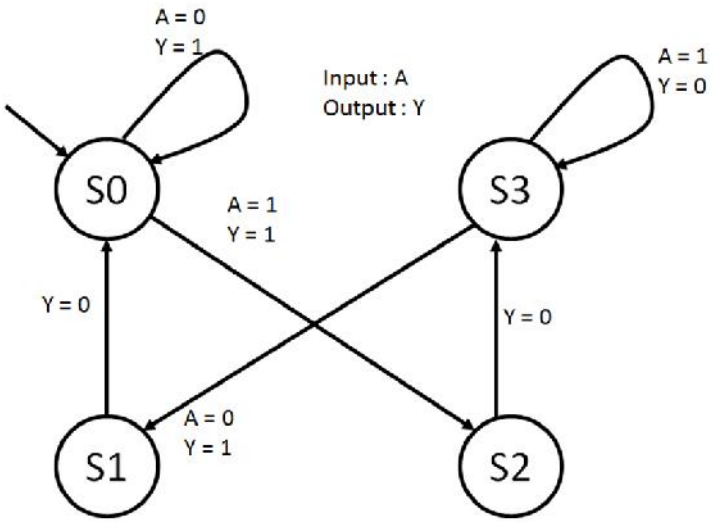


Q1Q0\A	0	1
00	00, 1	10, 1
01	00, 0	00, 0
10	11, 0	11, 0
11	01, 1	11, 0

Q1Q0 A	D1	D0	Y
000	0	0	1
001	1	0	1
010	0	0	0
011	0	0	0
100	1	1	0
101	1	1	0
110	0	1	1
111	1	1	0

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Solution



Q1Q0\A	0	1
00	00, 1	10, 1
01	00, 0	00, 0
10	11, 0	11, 0
11	01, 1	11, 0

Q1Q0 A	D1	D0	Y
000	0	0	1
001	1	0	1
010	0	0	0
011	0	0	0
100	1	1	0
101	1	1	0
110	0	1	1
111	1	1	0

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Solution

D0 = Q1

Q1 \ Q0A	00	01	11	10
0	0	0	0	0
1	1	1	1	1

D1 = Q1Q0’ + Q1A + Q0’A

Q1 \ Q0A	00	01	11	10
0	0	1	0	0
1	1	1	1	0

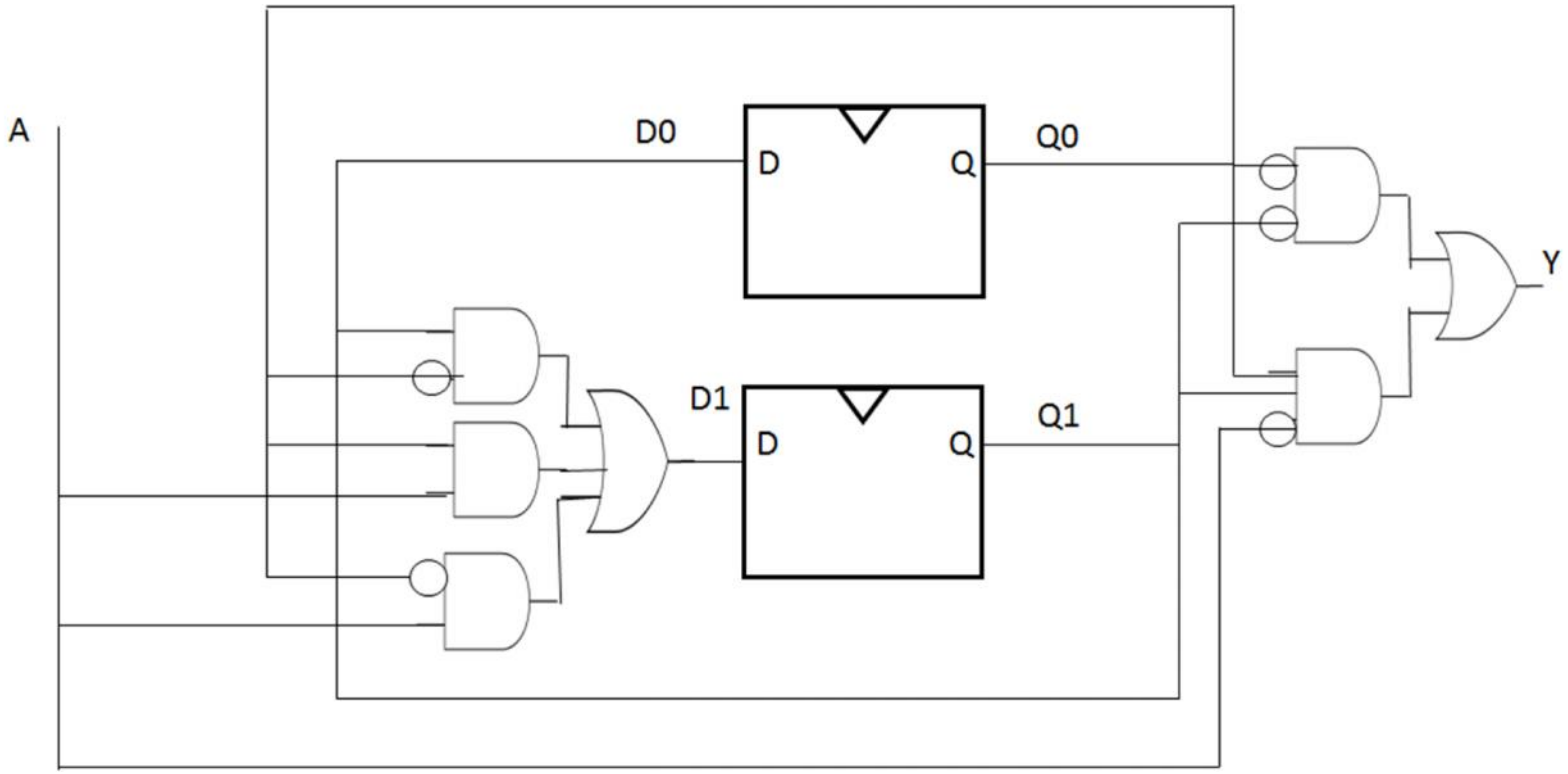
Y = Q1’Q0’ + Q1Q0A’

Q1 \ Q0A	00	01	11	10
0	1	1	0	0
1	0	0	0	1

Q1Q0A	D1	D0	Y
000	0	0	1
001	1	0	1
010	0	0	0
011	0	0	0
100	1	1	0
101	1	1	0
110	0	1	1
111	1	1	0

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Solution



Circuit timing recap

Gate:

- minimum (contamination) delay : t_cd
- maximum (propagation) delay : t_pd

D-FF:

- Input:
  - Setup : t_setup
  - Hold : t_hold
- Output:
  - Minimum (contamination) delay : t_ccq
  - Maximum (propagation) delay : t_pcq

Time constraints

Setup time constraints:
$T_c \geq t_{pcq} + t_{pd} + t_{setup}$

Hold time constraints:
$t_{hold} < t_{ccq} + t_{cd}$

Time constraints with skew

Setup time constraints:

$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$

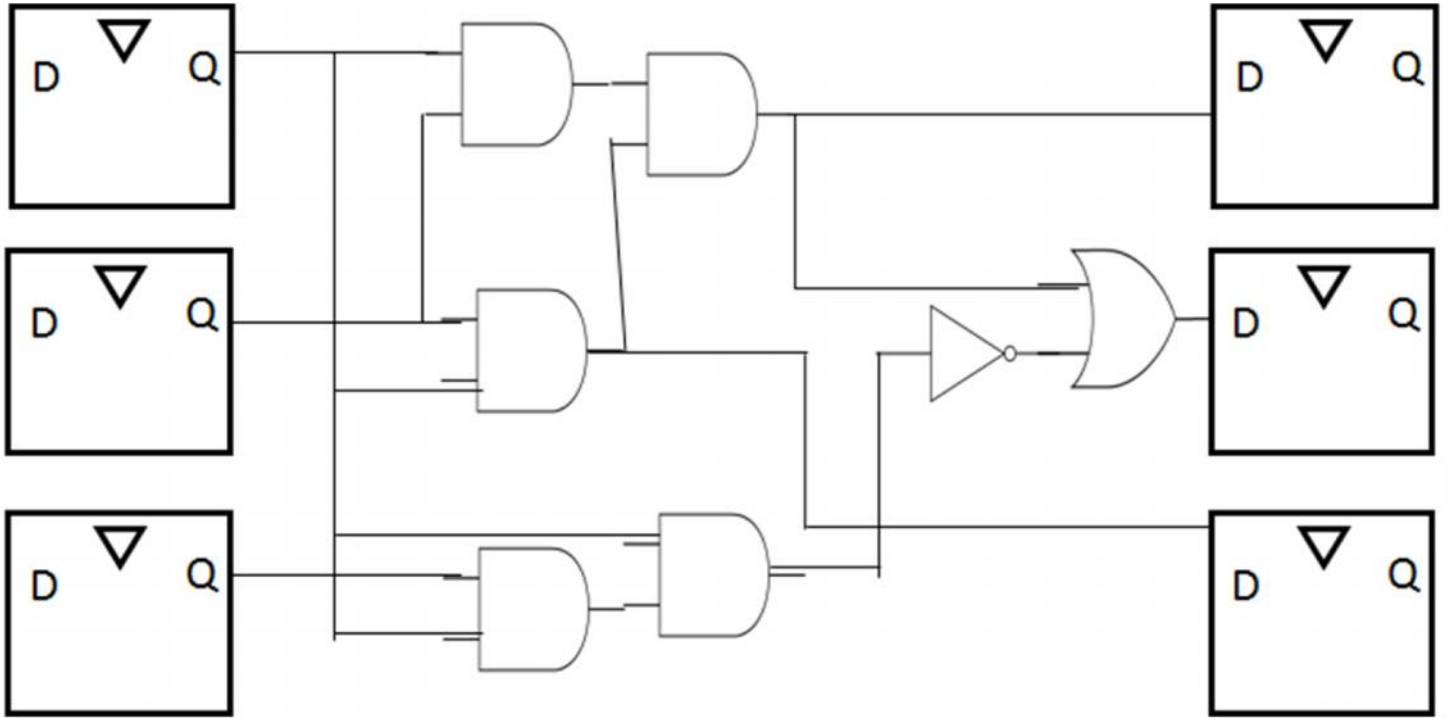
$t_{pd} \leq T_c - (t_{pcq} + t_{setup} + t_{skew})$

Hold time constraints:

$t_{ccq} + t_{cd} > t_{hold} + t_{skew}$

$t_{cd} > t_{hold} + t_{skew} - t_{ccq}$

Timing constraints example



D-FF:

- t_ccq = 40
- t_pcq = 60
- t_setup = 70
- t_hold = 80

Gates:

AND:

- t_pd = 40
- t_cd = 30

OR:

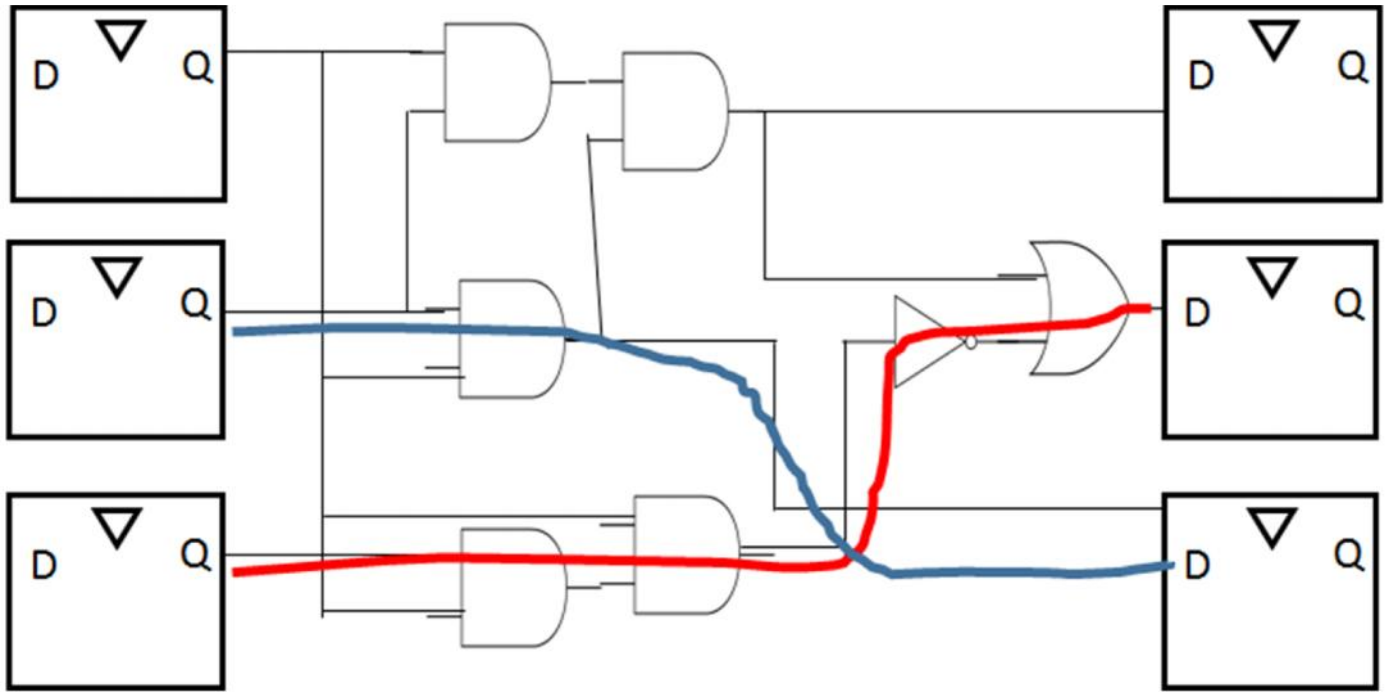
- t_pd = 40
- t_cd = 30

NOT:

- t_pd = 30
- t_cd = 20

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Timing constraints example



D-FF:

- t_ccq = 40
- t_pcq = 60
- t_setup = 70
- t_hold = 80

Gates:

AND:

- t_pd = 40
- t_cd = 30

OR:

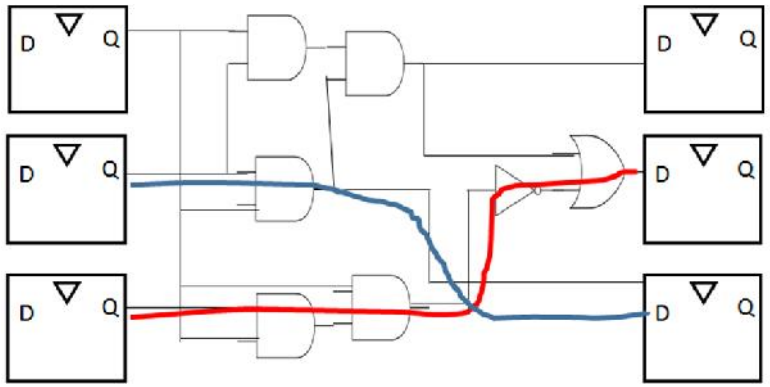
- t_pd = 40
- t_cd = 30

NOT:

- t_pd = 30
- t_cd = 20

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Timing constraints example



Setup time constraints:

$T_c \geq t_{pcq} + t_{pd} + t_{setup}$

$T_c \geq (60) + (3*40 + 30) + 70$

$T_c \geq 280ps$

Hold time constraints:

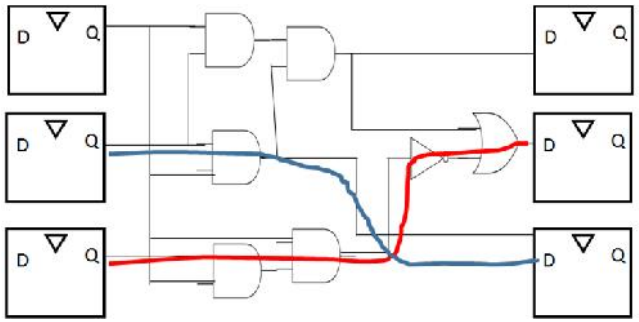
$t_{hold} < t_{ccq} + t_{cd}$

$80 < 40 + 30$ ----> NO !!

----> add buffer !

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Timing constraints example



Setup time constraints with skew:
What is the maximum t_skew that can be tolerated before having a setup timing error with a frequency of 3 GHz?
T_c = 1 / f = 1 / 3e9 = 333ps

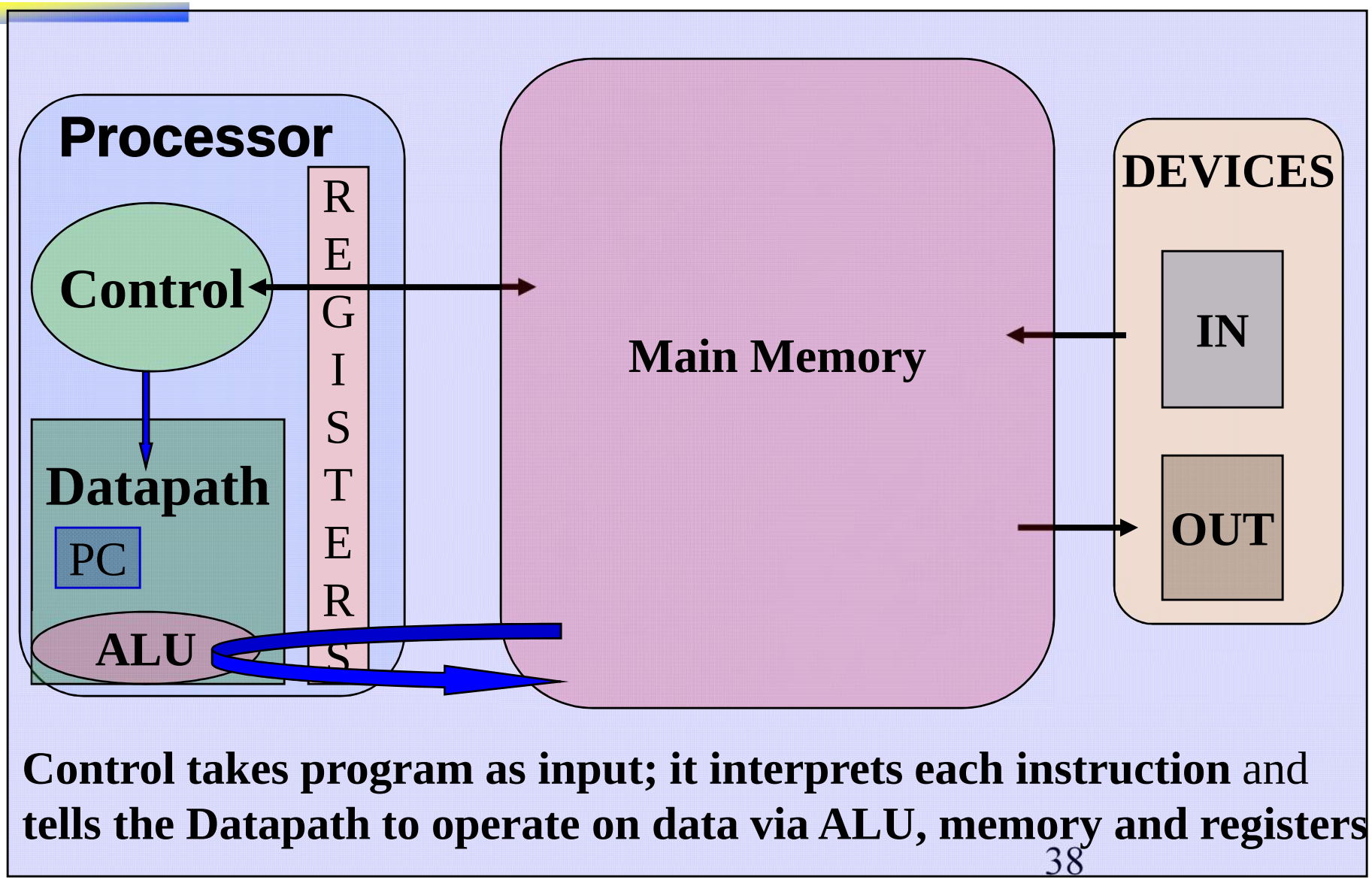
$T_c \geq t_{pcq} + t_{pd} + t_{setup} +$ **t_{skew}**

$333 - (60) - (3*40 + 30) - 70 \geq t_{skew}$

$53ps \geq t_{skew}$

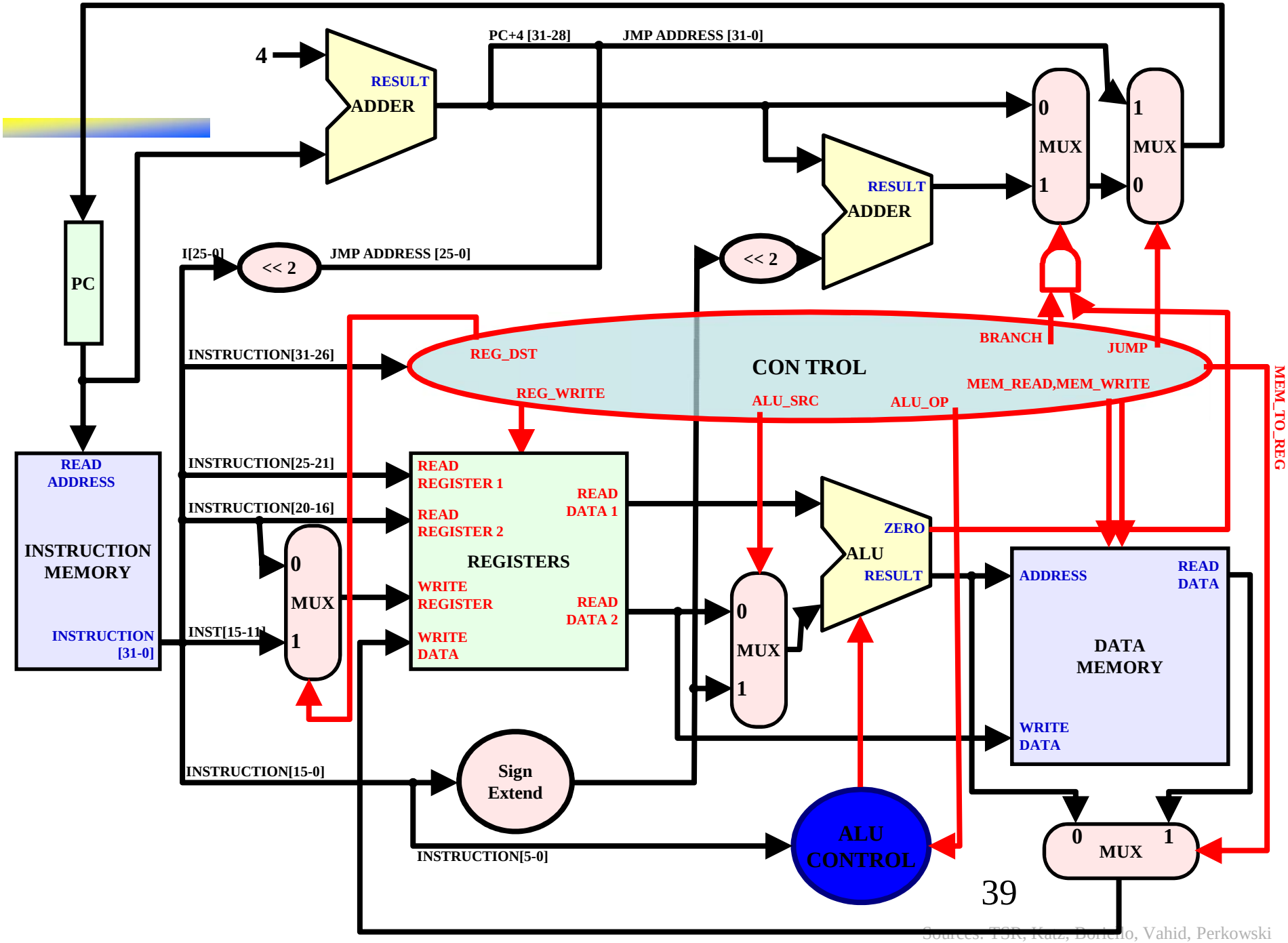
Sources: TSR, Katz, Boriello, Vahid, Perkowski

CPU Control and Datapath Execute Instruction Set



Control takes program as input; it interprets each instruction and **tells the Datapath to operate on data via ALU, memory and registers**

Sources: TSR, Katz, Boriello, Vahid, Perkowski



Sources: TSR, Katz, Borriello, Vahid, Perkowski

CPU Components – Single Cycle Execution

Assumptions:

– Every machine language instruction happens in 1 Clock Cycle

– MIPS architecture

- ***M*icroprocessor without *i*nterlocked *p*ipeline stages**
- reg-reg architecture: all operands must be in registers (total 24)
- 3 Instruction Types; each instruction 32 bits long
  1. R-type: all data in registers (most arithmetic and logical)
     e.g. add $s1, $s2, $s3
  2. I-type: branches, memory transfers, constants
     e.g. beq $s1, $s2, Label; lw $s1, 32($s2)
  3. J-type: jumps and calls
     e.g. j Label;

add $s0, $s1, $s2

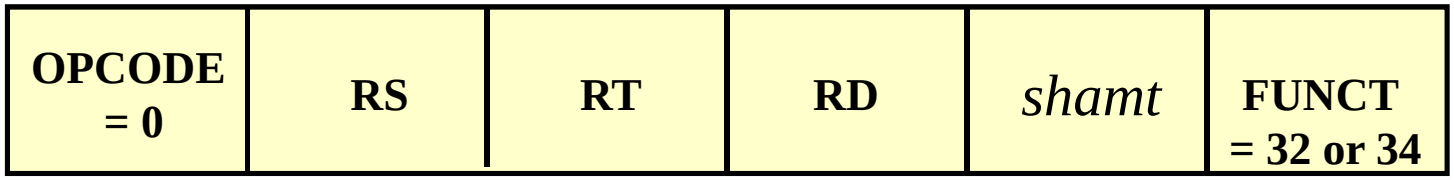
000000	10001	10010	10000	00000	100000
0	17	18	16	0	32

R-type Instruction: reg-reg ALU ops (e.g. add, and)

Tells operation to be performed

Tells specific variant of operation (e.g. add/sub have same opcode)

R-type Instruction



ADD $S1, $S2, $S3

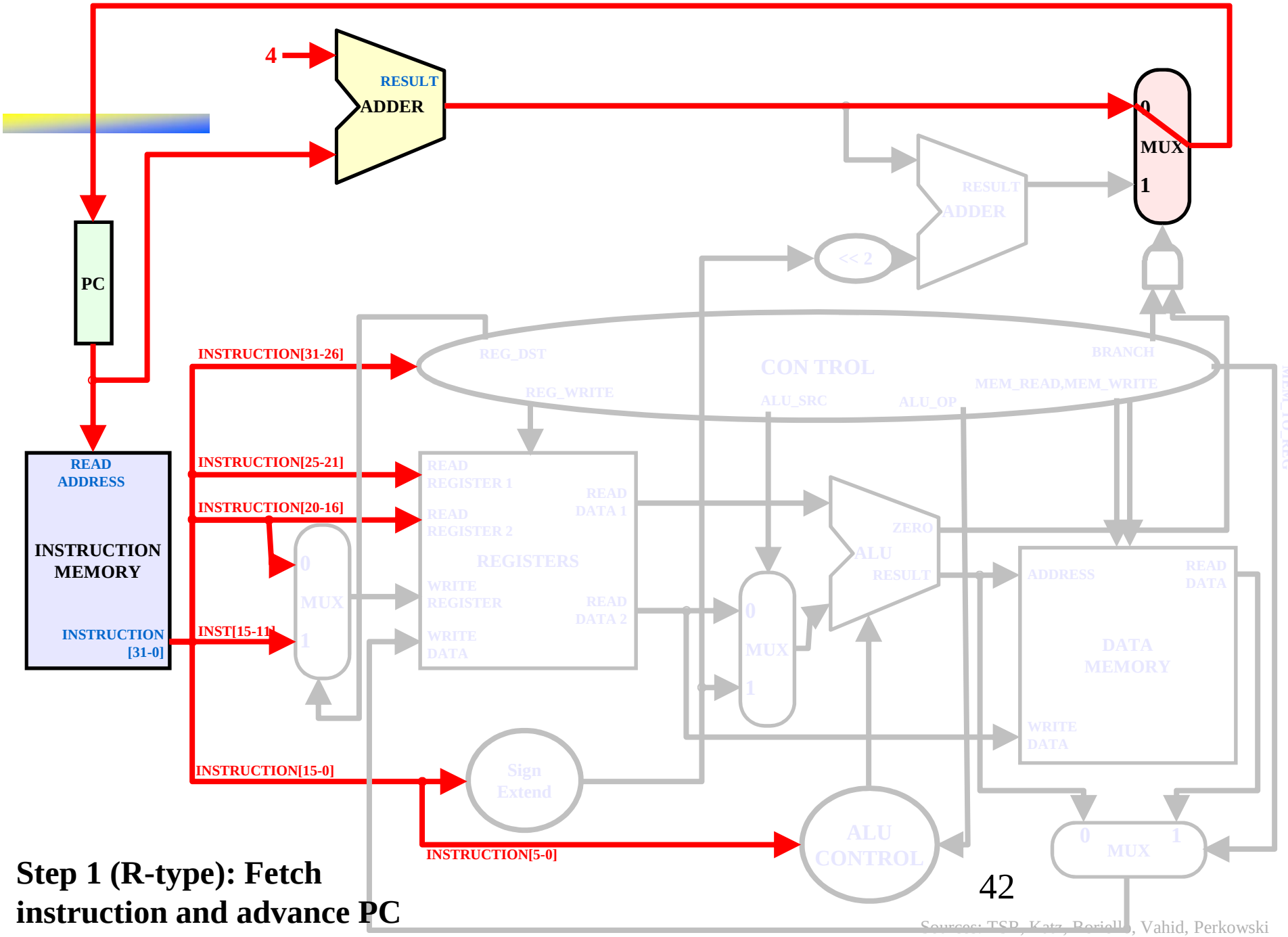
ADD RD, RS, RT

Shift amount (for sll, srl etc.)

Source Register 2 (attached to "Read Register 2" input)

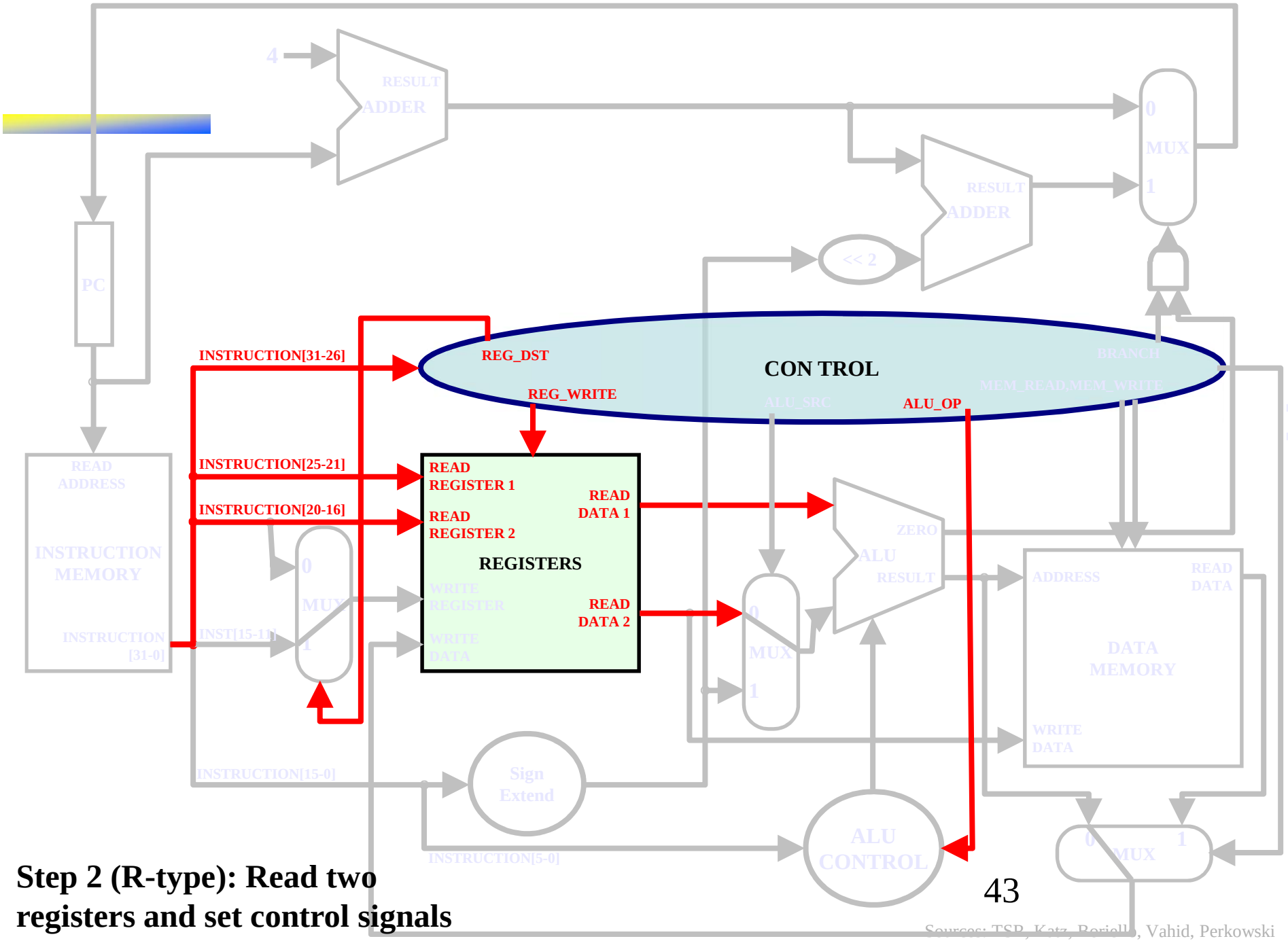
Destination Register (attached to "Write Register" input)

Source Register 1 (attached to "Read Register 1" input)



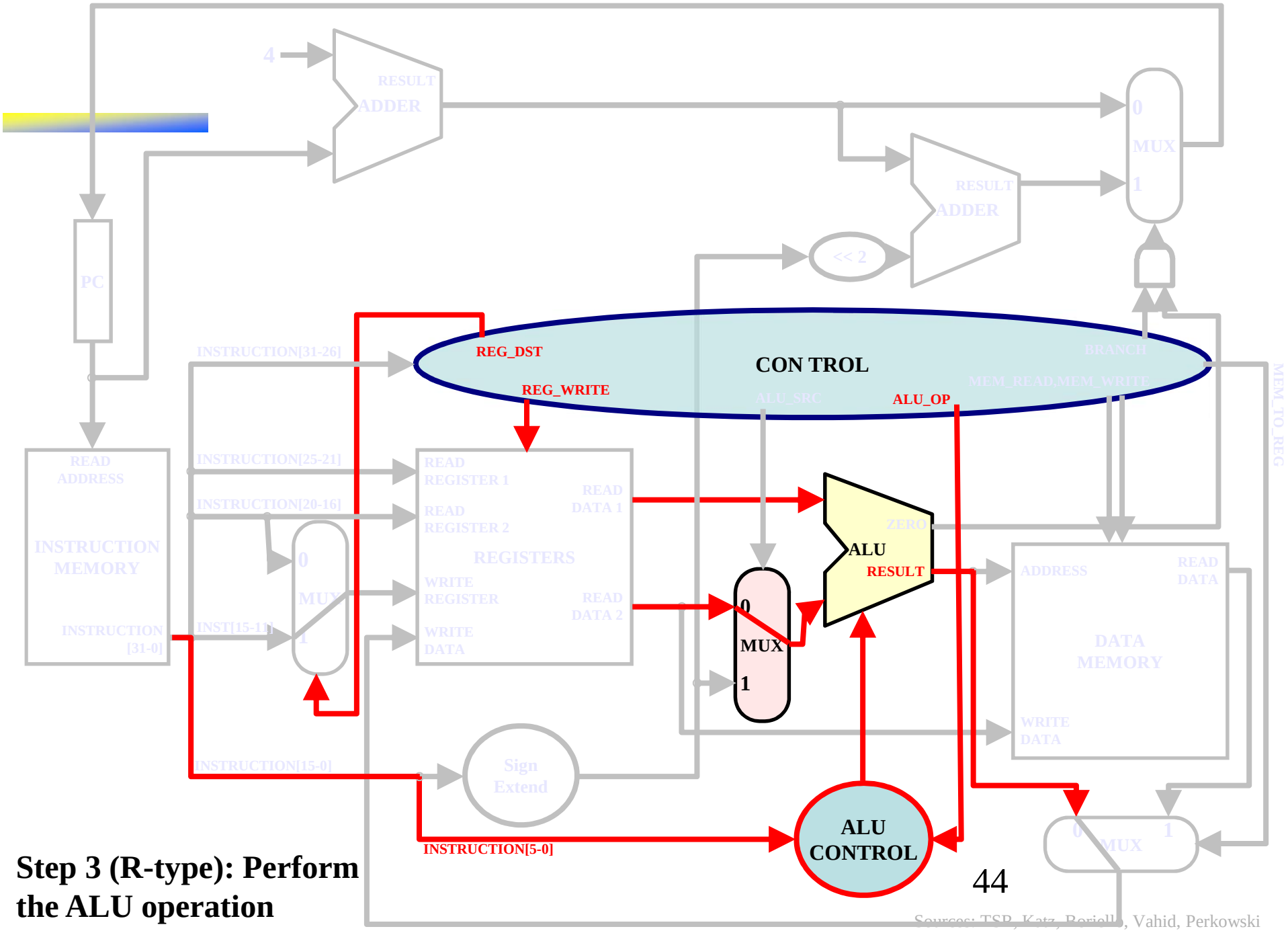
Step 1 (R-type): Fetch instruction and advance PC

Sources: TSP, Katz, Borriello, Vahid, Perkowski



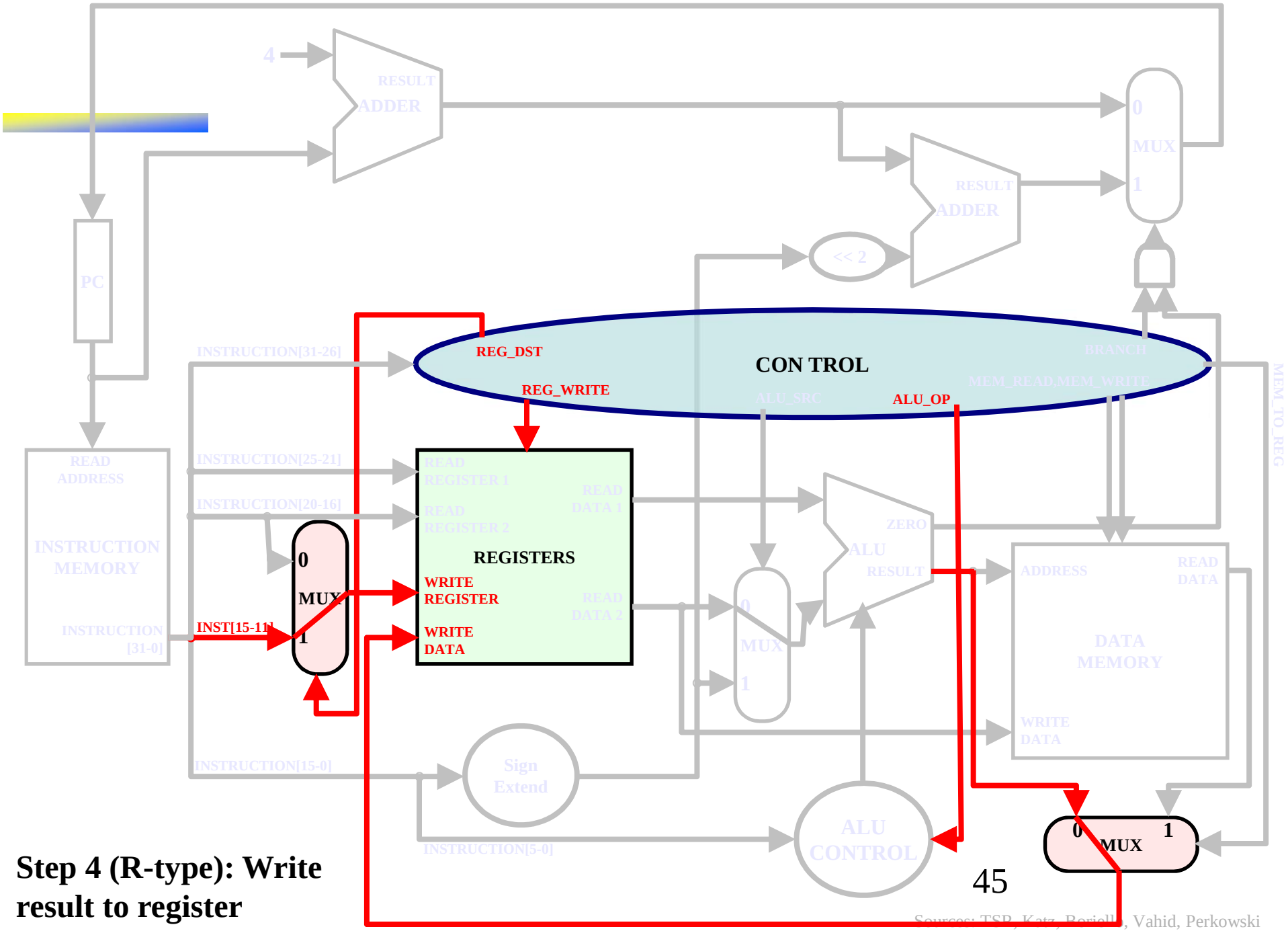
Step 2 (R-type): Read two registers and set control signals

Sources: TSP, Katz, Borriello, Vahid, Perkowski



Step 3 (R-type): Perform the ALU operation

Sources: TSP, Katz, Borriello, Vahid, Perkowski

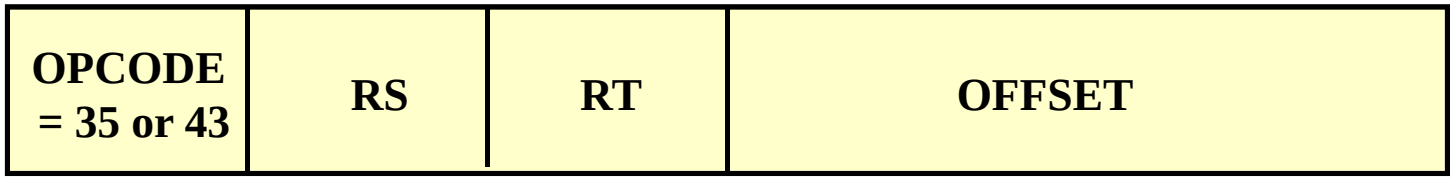


Step 4 (R-type): Write result to register

I-Type: Store Instruction

Tells operation to be performed

Store Instruction



31-26 **25-21** **20-16** **15-0**

Base Address Register (attached to "Read Register 1" input)

Source register whose value will be stored to memory (attached to "Read Register 2" input)

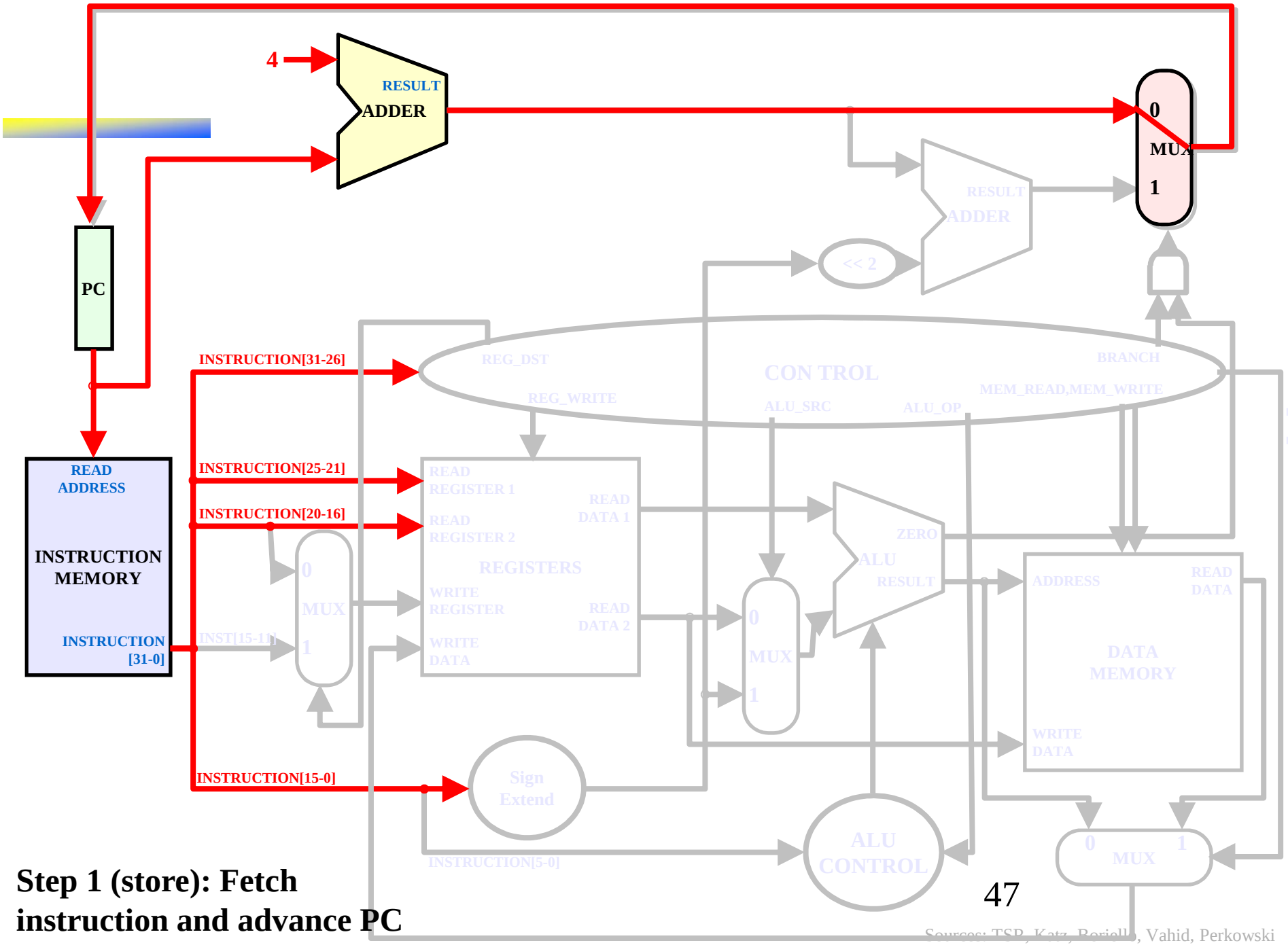
Constant offset (added to the base address in RS)

SW $S1, 32($S2)

SW RT, #(RS)

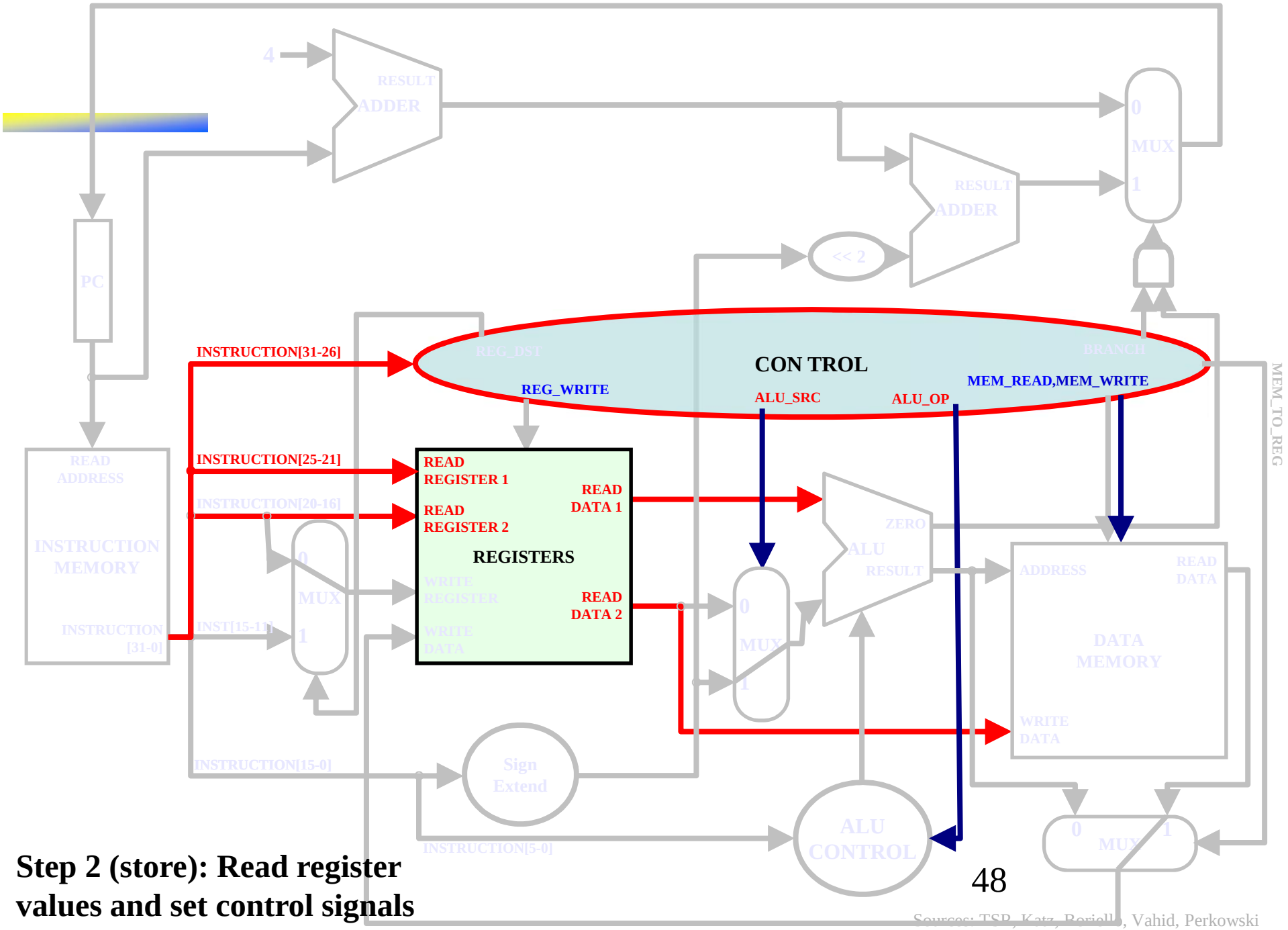
Note: same as x86
MOV [ebx+32], eax

Sources: TSR, Katz, Boriello, Vahid, Perkowski



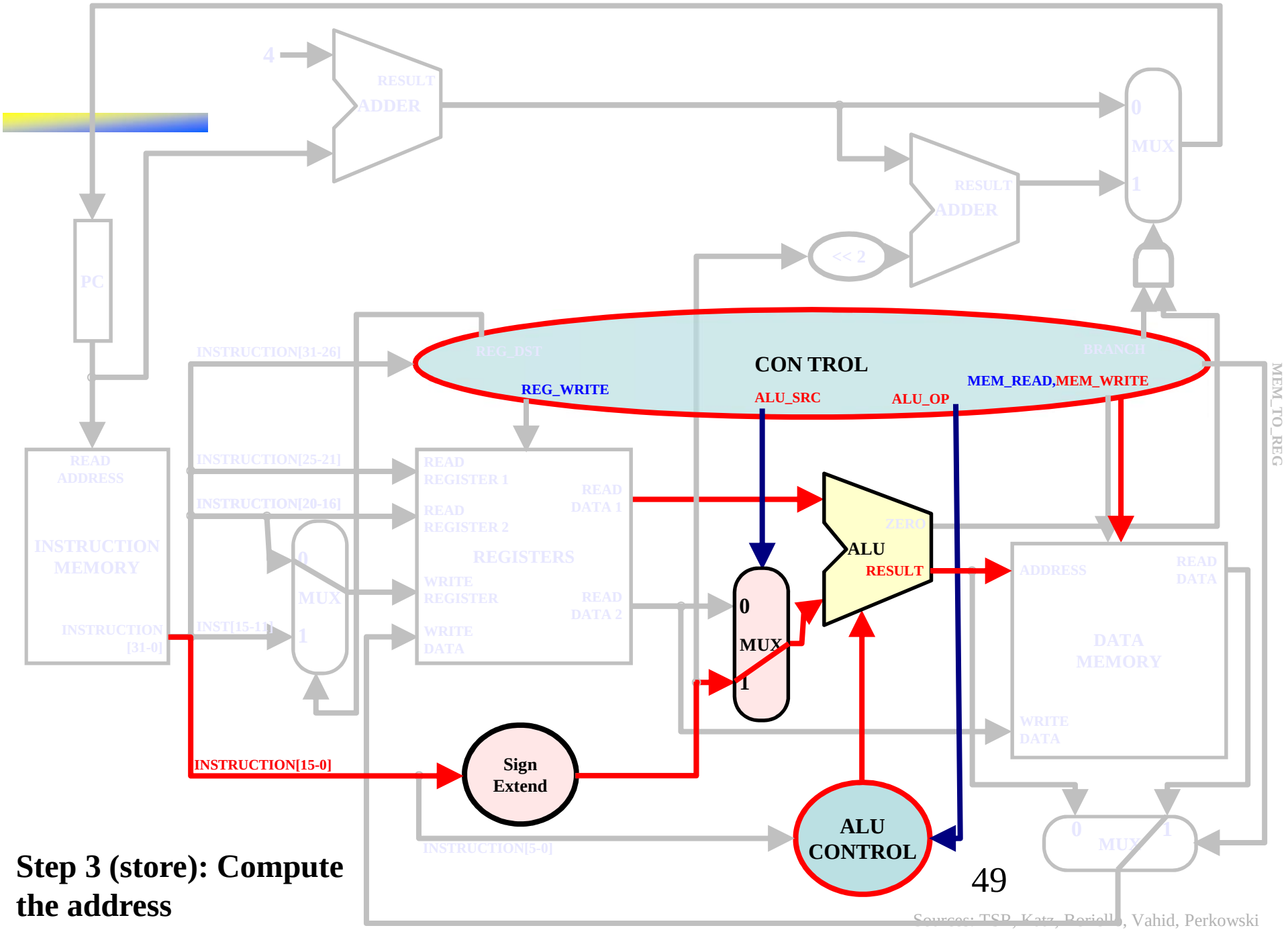
Step 1 (store): Fetch instruction and advance PC

Sources: TSP, Katz, Borriello, Vahid, Perkowski



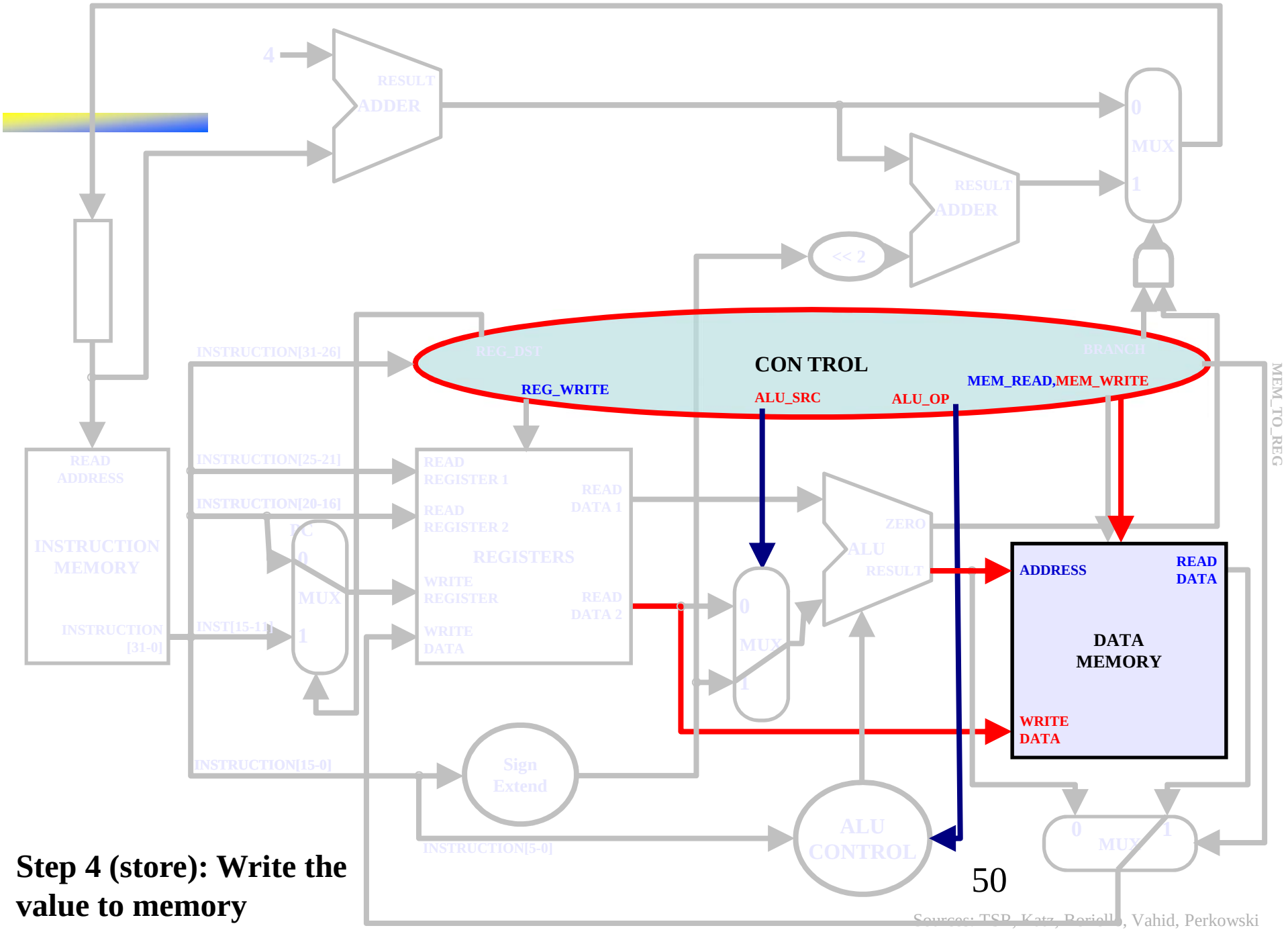
Step 2 (store): Read register values and set control signals

Sources: TSP, Katz, Borriello, Vahid, Perkowski



Step 3 (store): Compute the address

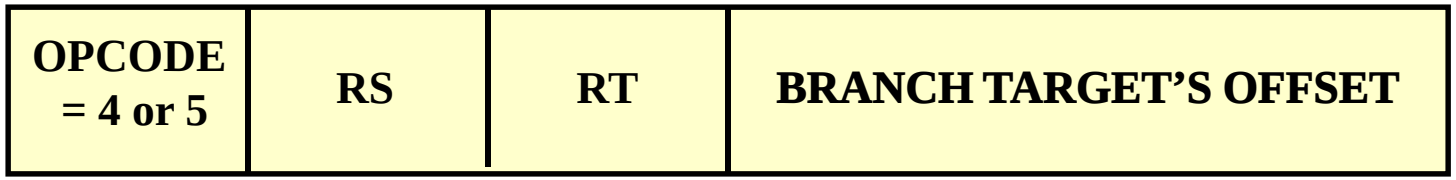
Sources: TSP, Katz, Borriello, Vahid, Perkowski



Step 4 (store): Write the value to memory

Sources: TSP, Katz, Borriello, Vahid, Perkowski

I-Type: Conditional Branch



BEQ/BNE Instruction

Source Register 1 (attached to "Read Register 1" input)

Source register 2 (attached to "Read Register 2" input)

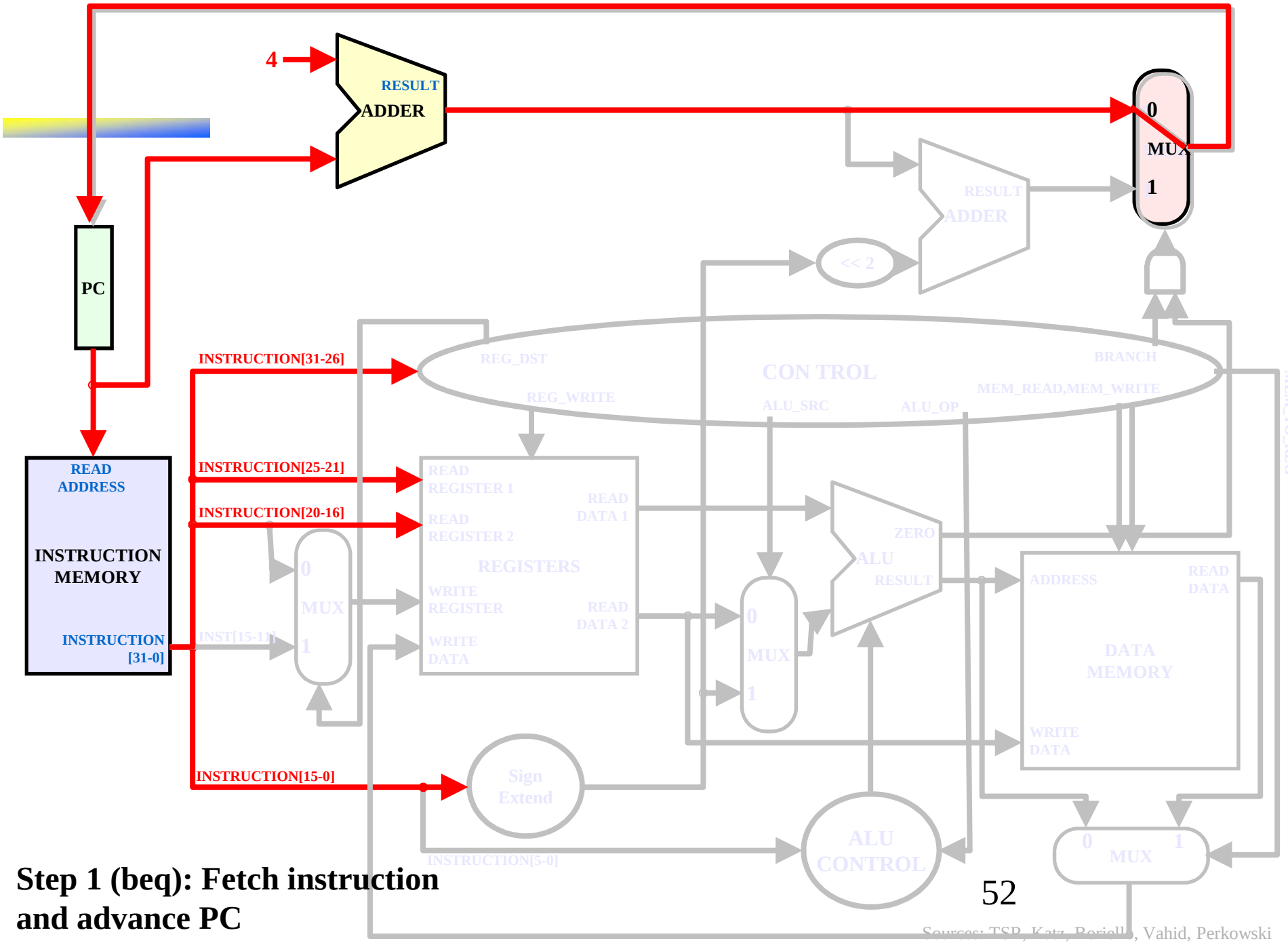
Word Offset, which we multiply by 4 (via <<2) to get Bit Offset, then add to PC+4 to get the address of the instruction to which we branch if RS = RT) "PC-relative address"

BEQ Source1, Source2, Offset

BEQ $S1, $S2, 100 = AL

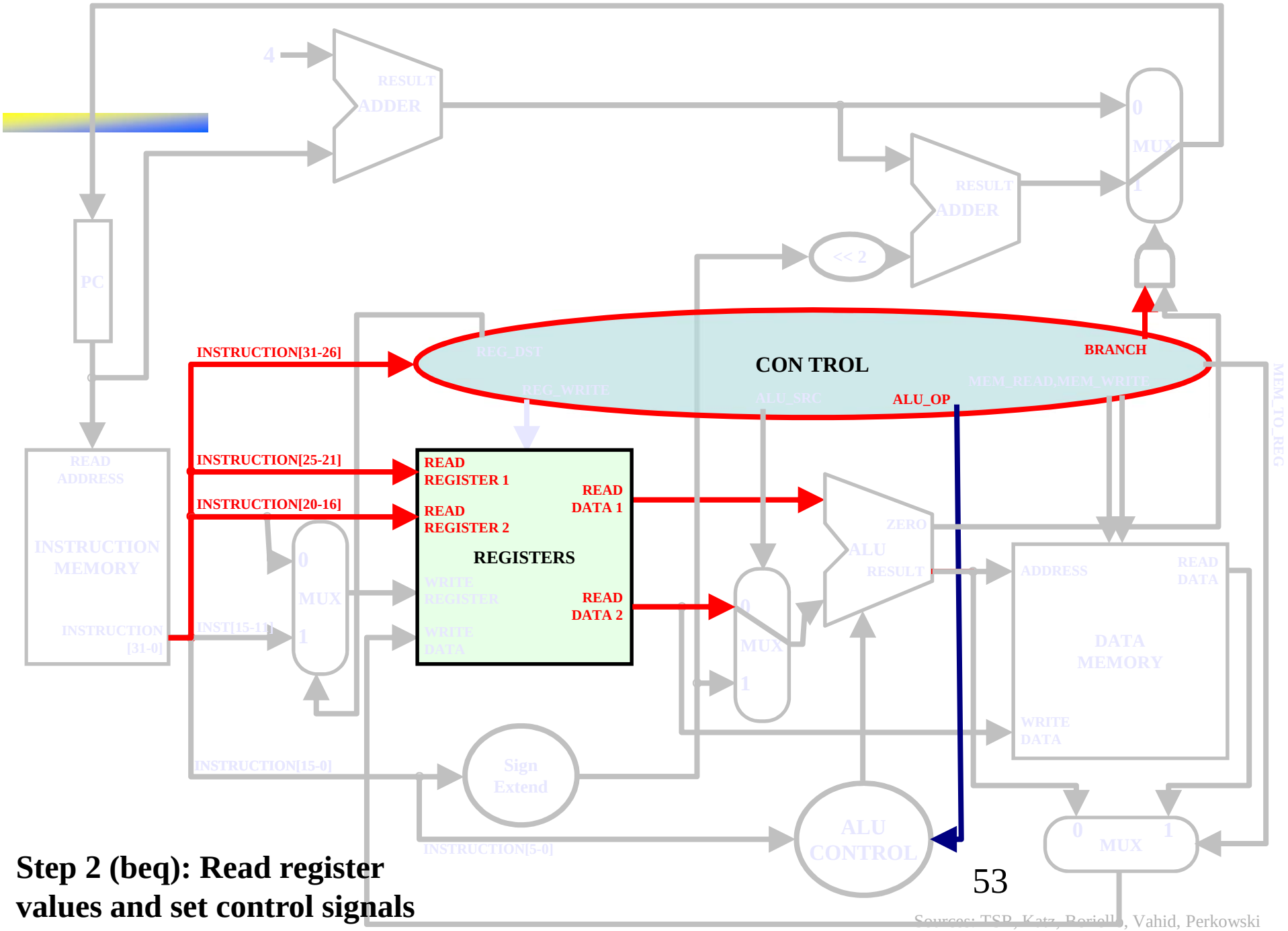
4 17 18 25 = ML (in binary)

Sources: TSR, Katz, Boriello, Vahid, Perkowski



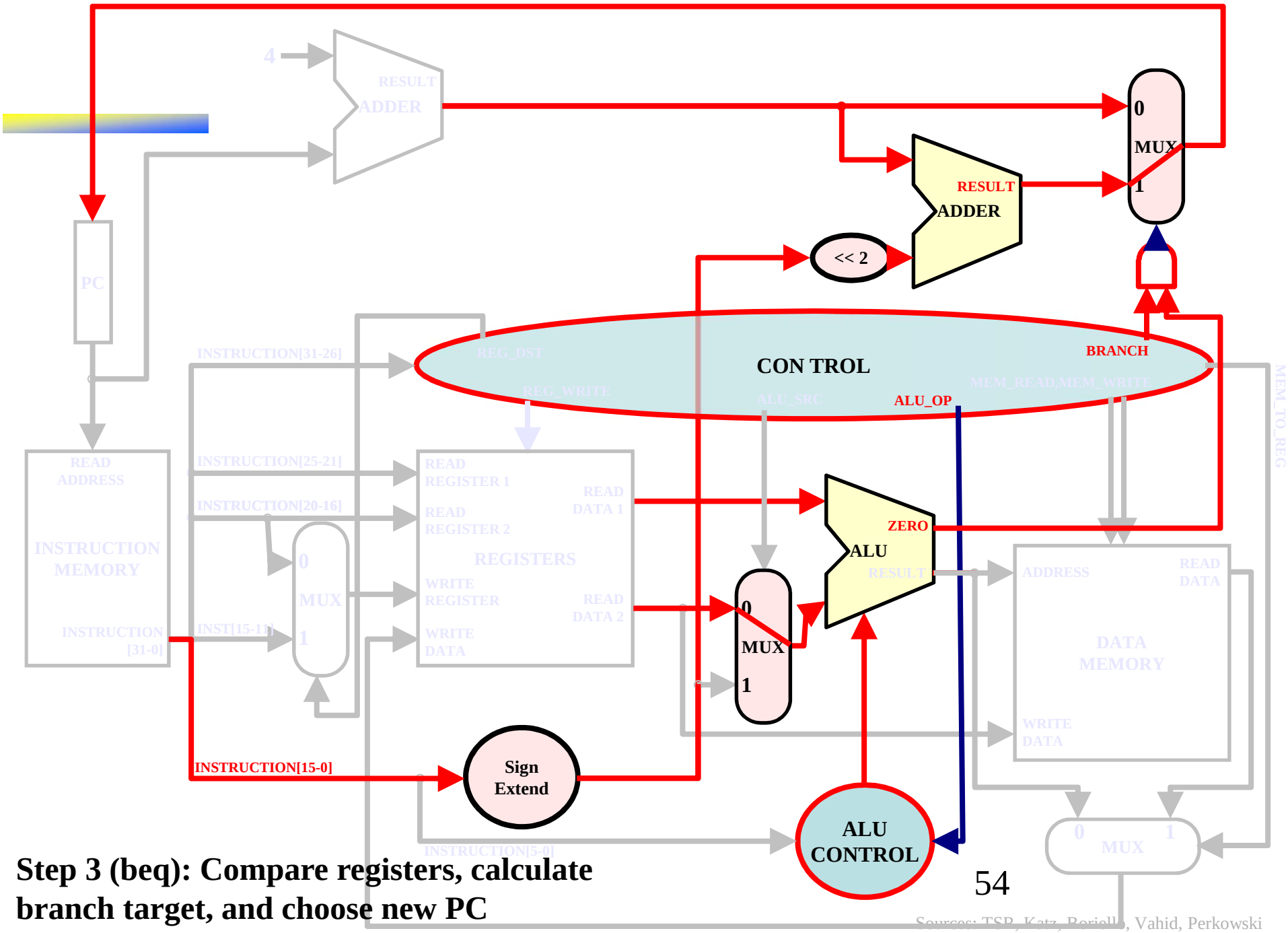
Step 1 (beq): Fetch instruction and advance PC

Sources: TSP, Katz, Borriello, Vahid, Perkowski



Step 2 (beq): Read register values and set control signals

Sources: TSP, Katz, Borriello, Vahid, Perkowski



Step 3 (beq): Compare registers, calculate branch target, and choose new PC

Sources: TSP, Katz, Borriello, Vahid, Perkowski

J-Type: Unconditional Branch

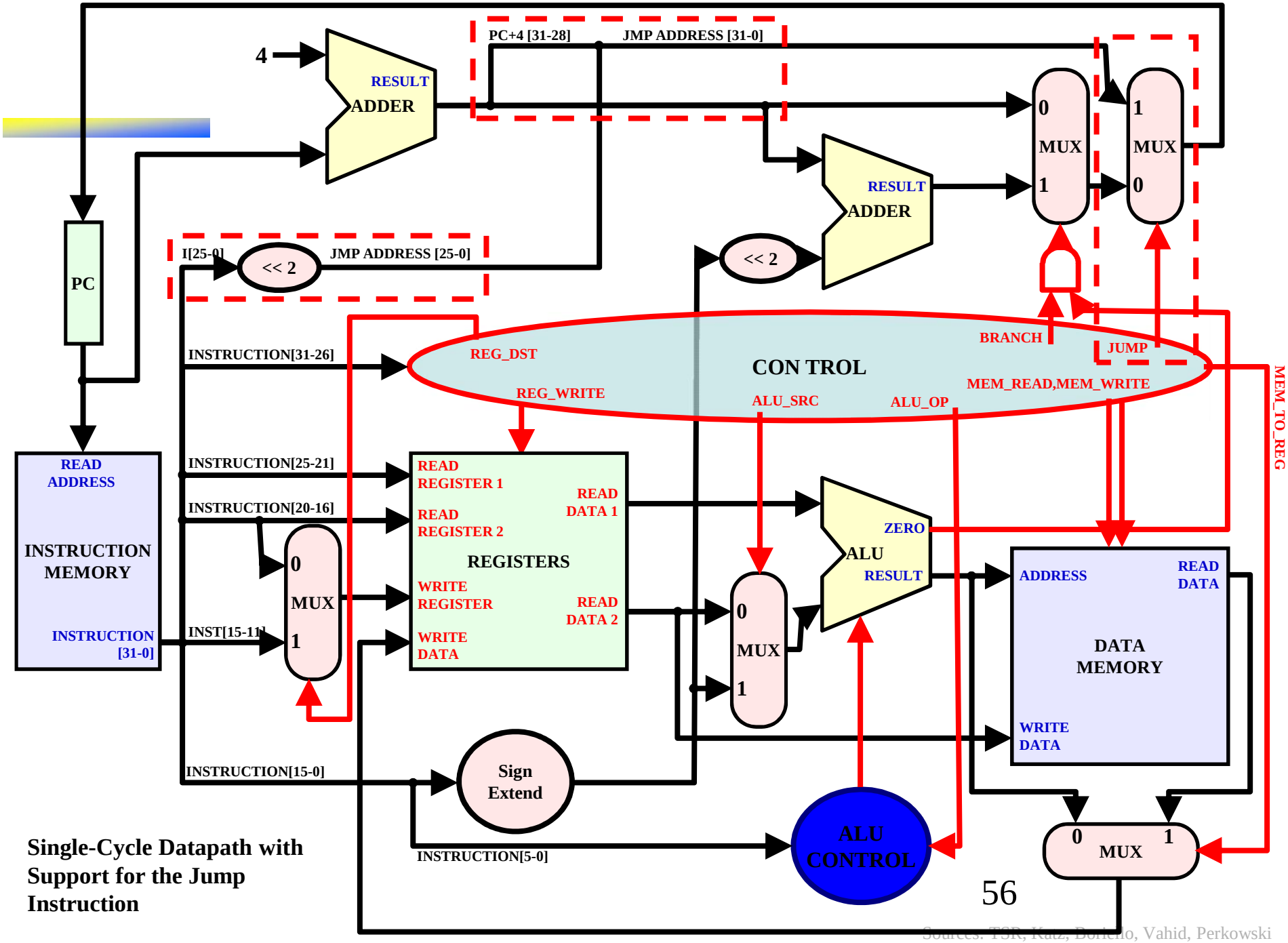
JMP/JAL Instruction

J Offset

J 10000 = AL

2 2500 = ML (in binary)

Actual Address (in words) which we multiply by 4 (<<2) to get 28-Bit Address, then concatenate to upper 4 bits of PC+4 to get the 32-bit addresss of instruction to which we branch unconditionally



Single-Cycle Datapath with Support for the Jump Instruction

Sources: TSR, Katz, Boriello, Vahid, Perkowski