INTRODUCTION TO COMPUTER ORGANIZATION

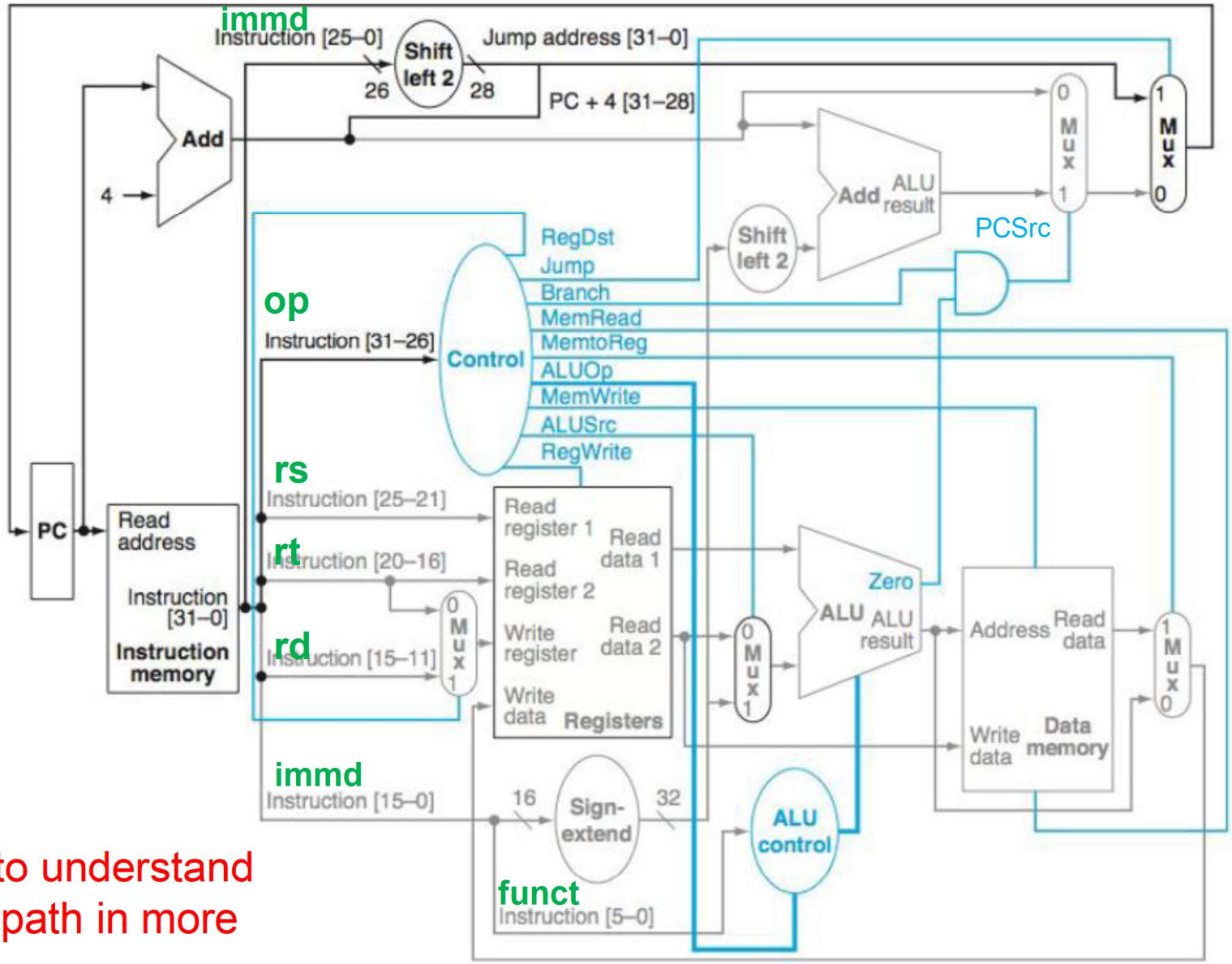
CCIT4026

Chapter 8: Single Cycle Process (Part 2)

R-type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Single Cycle Processor

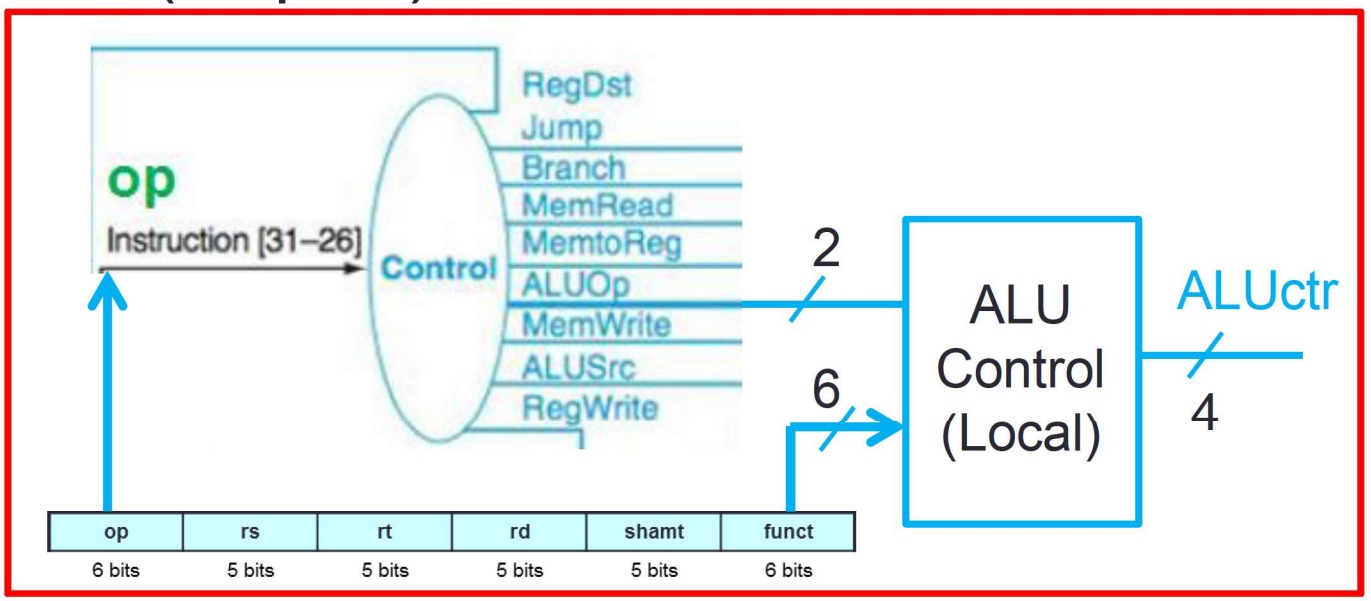


Let's try to understand
The datapath in more
detail !!

Single Cycle Processor

Main Control:

- 6-bit opcode (input) – instr[31:26] = The most significant 6 bits
  - ⇒ 9 control signals (outputs)
    - RegDst
    - Branch
    - MemRead
    - MemToReg
    - ALUOp (2 bits)
    - MemWrite
    - ALUSrc
    - RegWrite

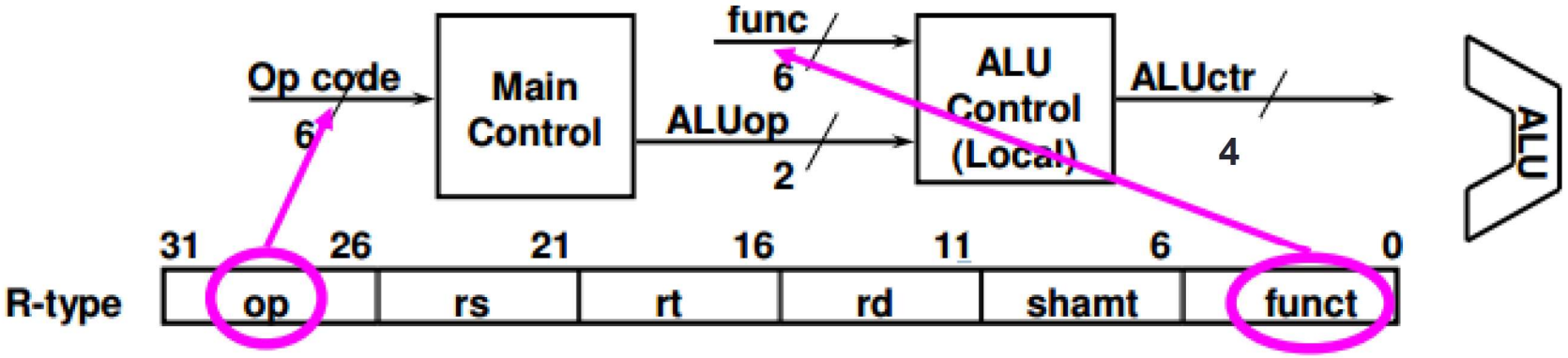


ALU Control (Local):

- 6-bit function (inputs) – instr[5:0] = The least significant 6 bits
- + 2-bit ALUop (inputs)
- ⇒ 4-bit "ALU Control" operation, ALUctr (outputs)

Each control signal should be 0, 1 or don't care (X)

Control Signal



2-Level Decoding

1st level: main control

6-bit opcode (instr[31:26])

→ RegDst, Branch, MemRead, MemtoReg, ALUOp[1:0], MemWrite, ALUSrc, RegWrite

2nd level: ALU control

6-bit funct code (instr[5:0]), ALUop[1:0]

→ ALUctr[3:0] = ALU operation

ALU Control (2nd level)

Calculate offset+$register

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

"slt" instruction

Calculate ($rs-$rt)

Remarks:

ALUop is 2-bit to represent:

00: add for load/store

01: sub for beq

10: R-type (need to reference funct field)

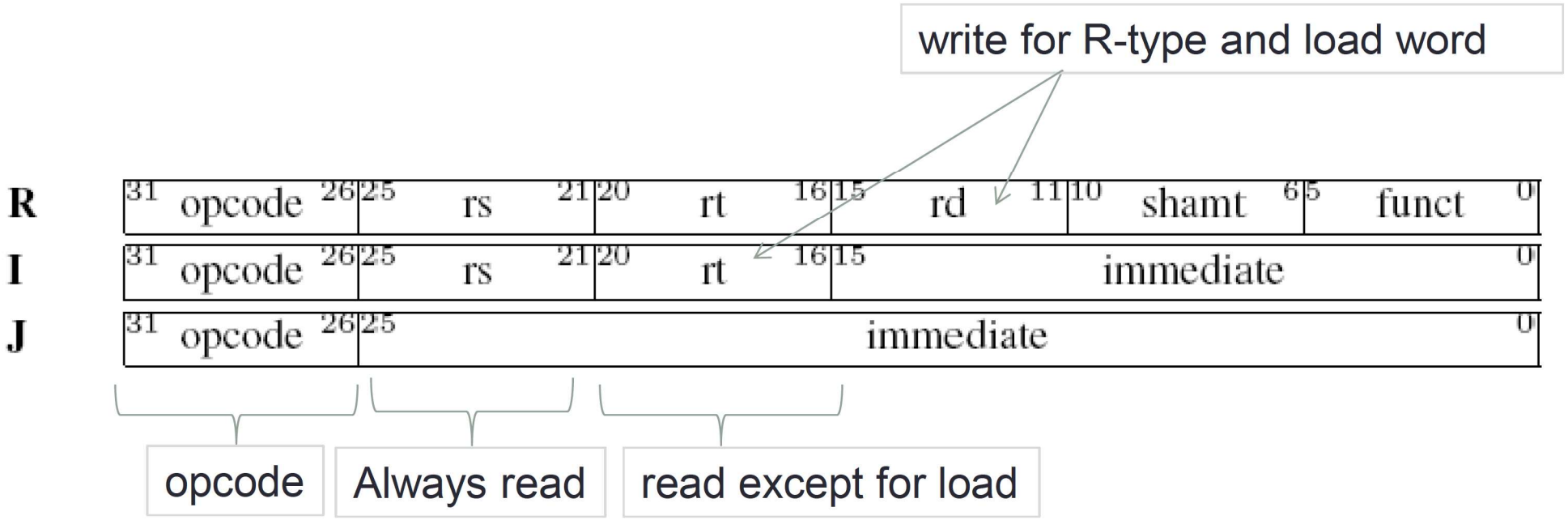
ALU Control (2nd level)

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

- Since there is no 11 in ALUOp, we can use 1X and X1
- Some don't-care entries are added.
- Once the truth table is constructed, it can be optimized and turned into gates.

Main Control (1st level)

- To design the main control unit, we need to understand:



Main Control (1st level)

Signal name	0 Effect when deasserted	1 Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

A 1-bit control is asserted means 1; otherwise, the control is deasserted means 0.

Main Control (1st level)

- Inputs to main control unit – opcode

Instruction	Opcode in decimal	Opcode in binary					
		Op5	Op4	Op3	Op2	Op1	Op0
R-format	0	0	0	0	0	0	0
`lw`	35 (23h)	1	0	0	0	1	1
`sw`	43 (2Bh)	1	0	1	0	1	1
`beq`	4	0	0	0	1	0	0

- Outputs of the main control unit (first level)

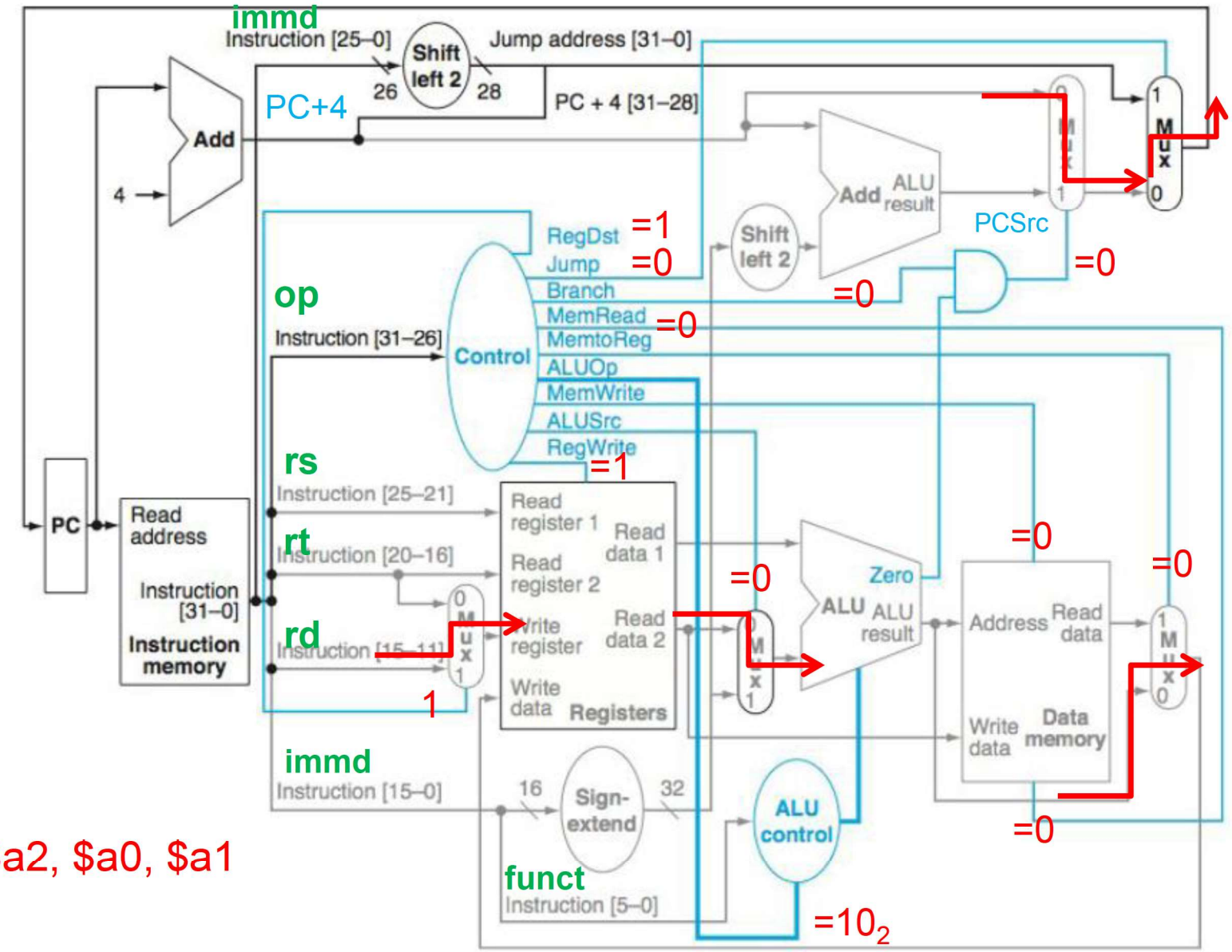
Instruction	Reg-Dst	ALU-Src	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

Main Control (1st level)

Instruction	Reg-Dst	ALU-Src	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0

- For all R-format instructions (add, sub, AND, OR, slt),
  - Destination register field is rd, so **RegDst** =1
  - Source register fields are rs and rt, i.e. the second operand to the ALU is retrieved from register, so **ALUSrc** = 0
  - It takes the ALU output instead of memory output, so **MemtoReg** = 0
  - It writes a register (**RegWrite** = 1), but neither reads nor writes data memory (**MemRead** = 0, **MemWrite** = 0)
  - There is no branch target (**Branch** = 0), and PC+4
  - **ALUop** field is set to 10 to indicate that ALU control should be generated from funct field

Instruction	Reg-Dst	ALU-Src	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0



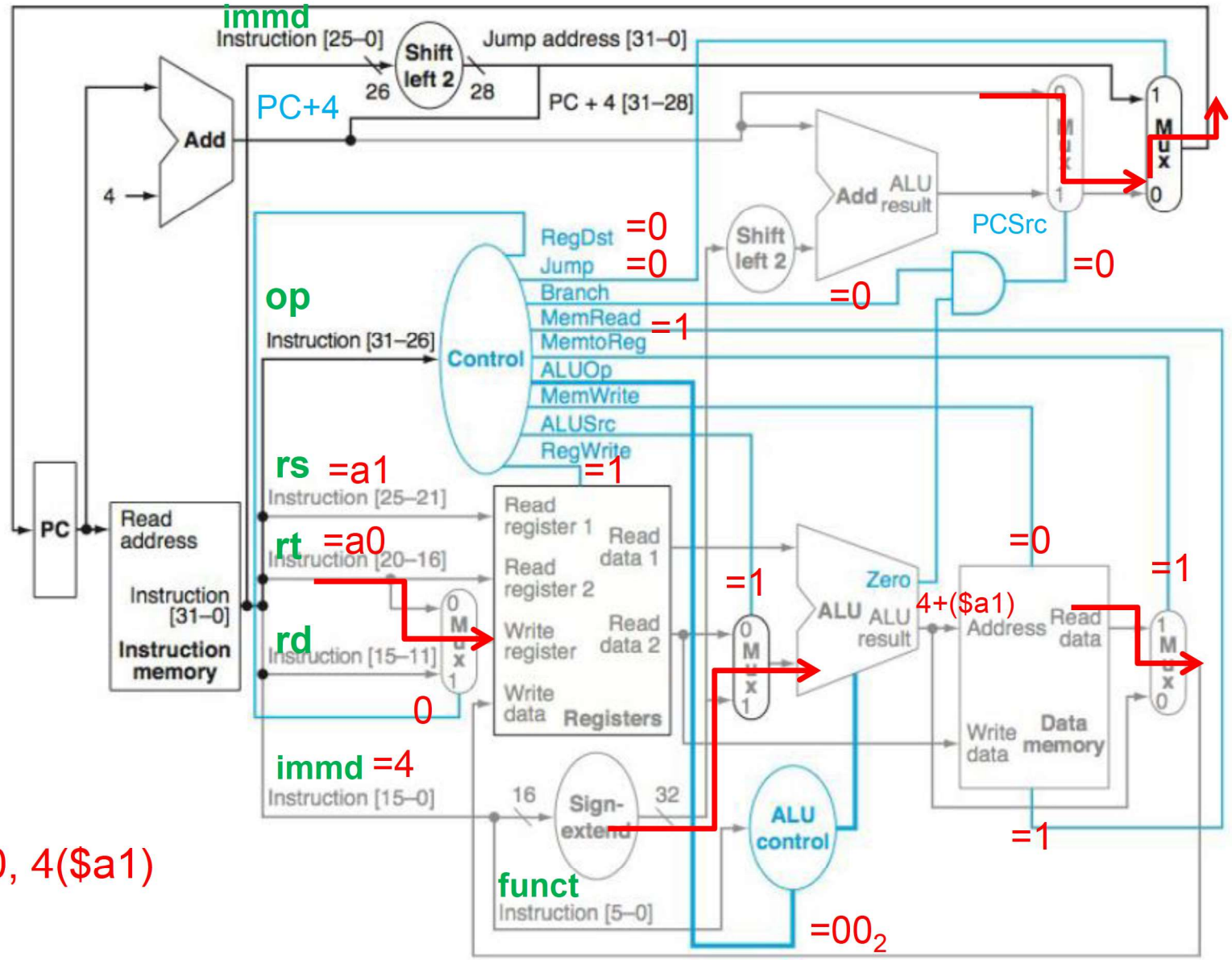
add $a2, $a0, $a1

Main Control (1st level)

Instruction	Reg-Dst	ALU-Src	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
`lw`	0	1	1	1	1	0	0	0	0

- For lw,
  - **RegDst** and **RegWrite** are set for load, and result is stored into rt
    - Destination register field is rt, so **RegDst** = 0
  - The second operand to the ALU is retrieved from the instruction (or immediate), so **ALUSrc** = 1
  - **ALUOp** field is set to 00 to indicate lw/sw instructions
  - It takes memory output, so **MemtoReg** = 1
  - Involve memory read for load (**MemRead** = 1), but not write (**MemWrite** = 0)

Instruction	Reg-Dst	ALU-Src	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
`lw`	0	1	1	1	1	0	0	0	0



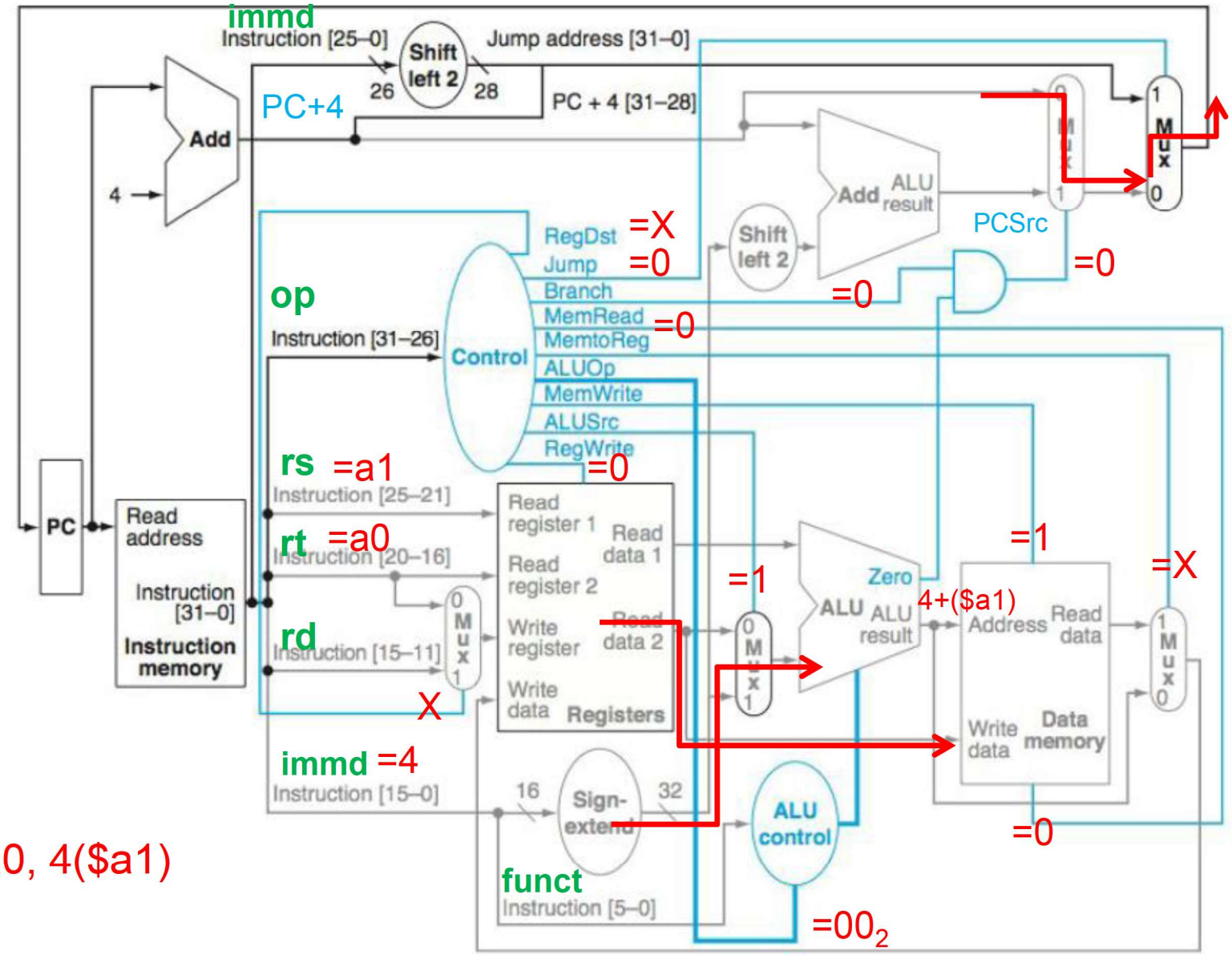
lw $a0, 4($a1)

Main Control (1st level)

Instruction	Reg-Dst	ALU-Src	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0

- For sw,
  - It doesn't write register, so **RegWrite** = 0,
    - **RegDst** and **MemToReg** become irrelevant when RegWrite = 0, so they are set to don't care (X).
  - The second operand to the ALU is retrieved from the instruction (or immediate), so **ALUSrc** = 1 (same as lw)
  - **ALUOp** field is set to 00 to indicate lw/sw instructions (same as lw)
  - Involve memory write for store (**MemWrite** = 1), but not read (**MemRead** = 0)

Instruction	Reg-Dst	ALU-Src	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0



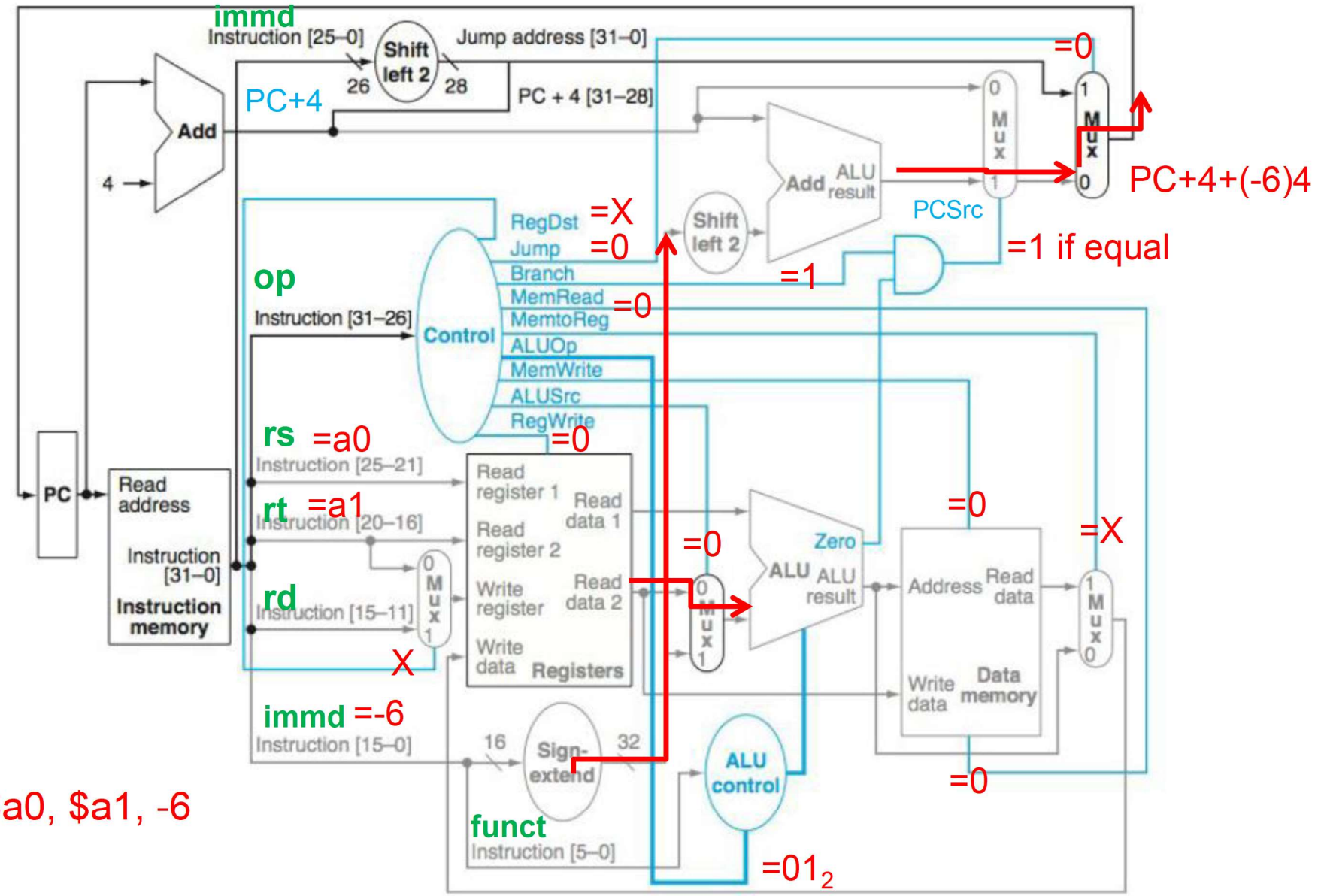
sw $a0, 4($a1)

Main Control (1st level)

Instruction	Reg-Dst	ALU-Src	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
beq	X	0	X	0	0	0	1	0	1

- For branch instruction,
  - Source register fields are rs and rt, i.e. the second operand to the ALU is retrieved from register, so **ALUSrc** = 0 (same as R-format)
  - It doesn't write register, so **RegWrite** = 0, (same as sw)
    - **RegDst** and **MemToReg** become irrelevant when RegWrite = 0, so they are set to don't care (X).
  - **ALUOp** field is set to 01 to indicate branch instruction
  - It doesn't involve memory read/write
  - It is a branch instruction

Instruction	Reg-Dst	ALU-Src	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
beq	X	0	X	0	0	0	1	0	1

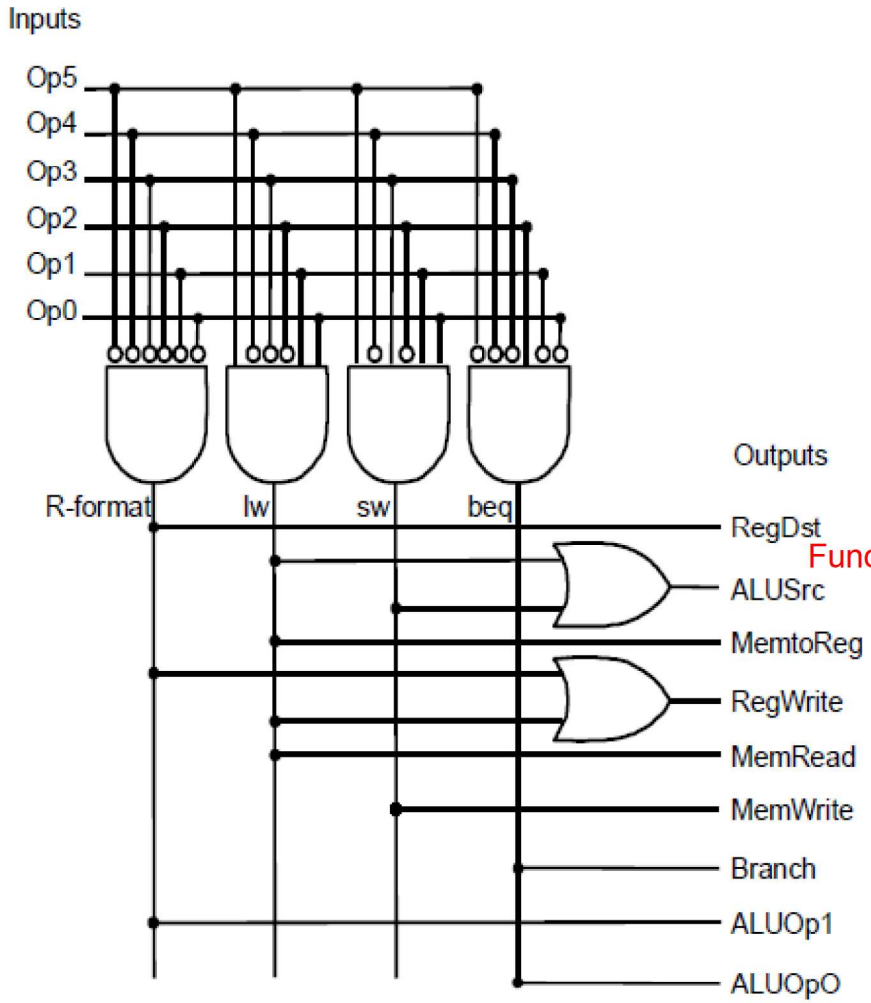


Beq $a0, $a1, -6

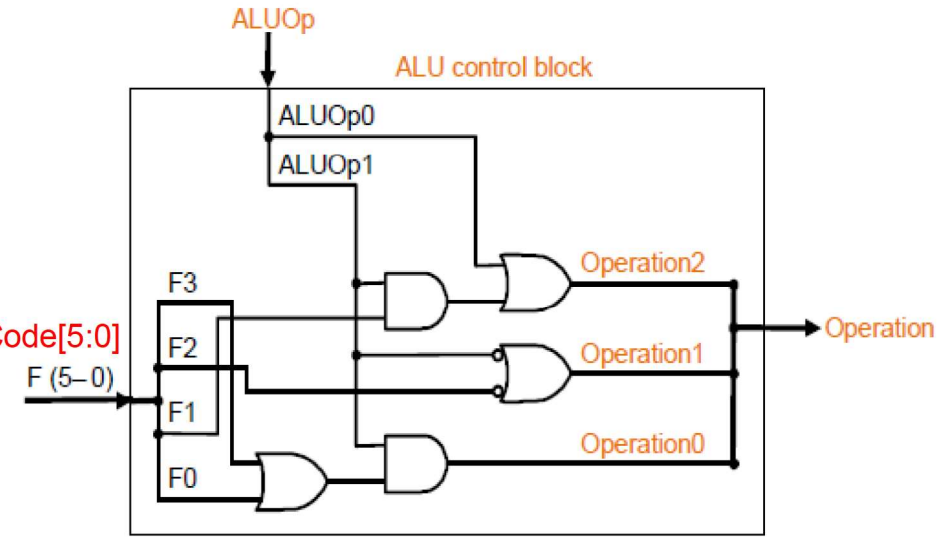
Main Control (1st level)

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Control Unit (Hardware implementation)



First level decode



Second level decode

Example

The following instruction sequence runs on the single-cycle. Please find the data on every bus.

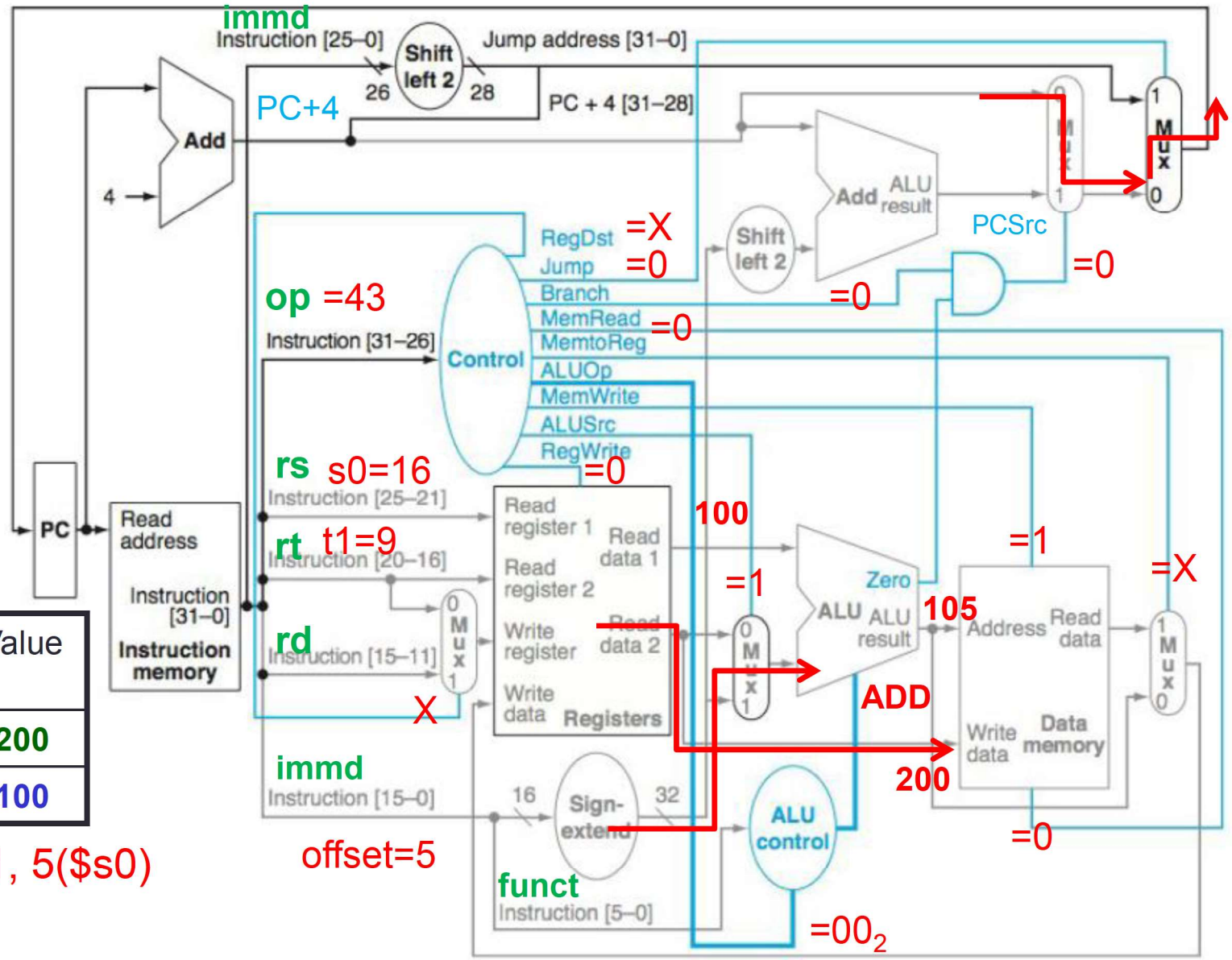
Instruction sequence:

L1:	sw	$t1, 5($s0)	101011 10000 01001 0000 0000 0000 0101
	sub	$t0, $s1, $s2	000000 10001 10010 01000 00000 100010
	beq	$t0, $t1, L1	000100 01000 01001 1111 1111 1111 1101

Assume the registers and memory have following contents before execution:

$s0=100, $s1=500, $s2=300, $t1=200, L1=1000, mem[105] = 6

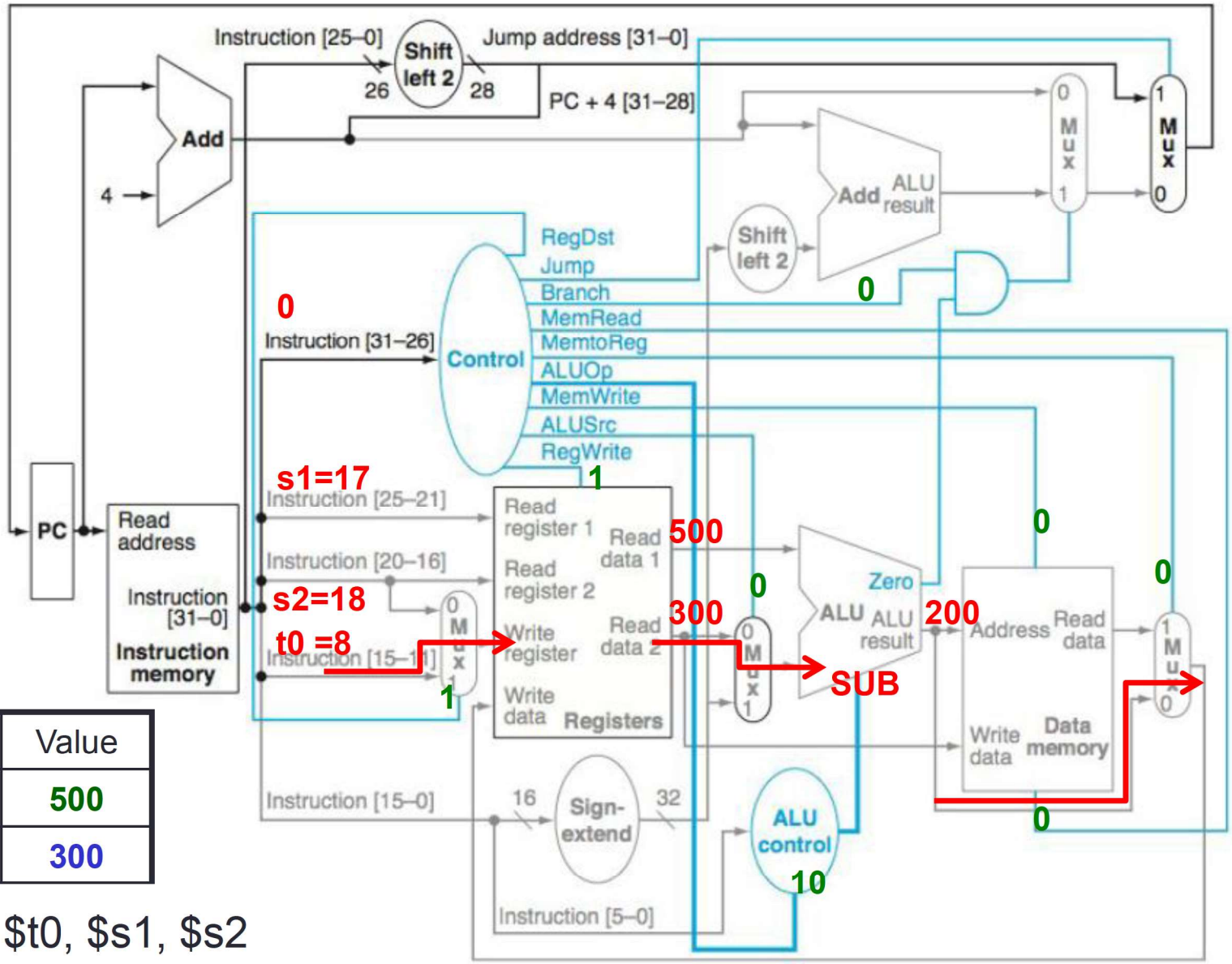
sw	$t1	$s0	offset
101011 (43d)	9	16	5



Register	Value
$t1	200
$s0	100

sw $t1, 5($s0)

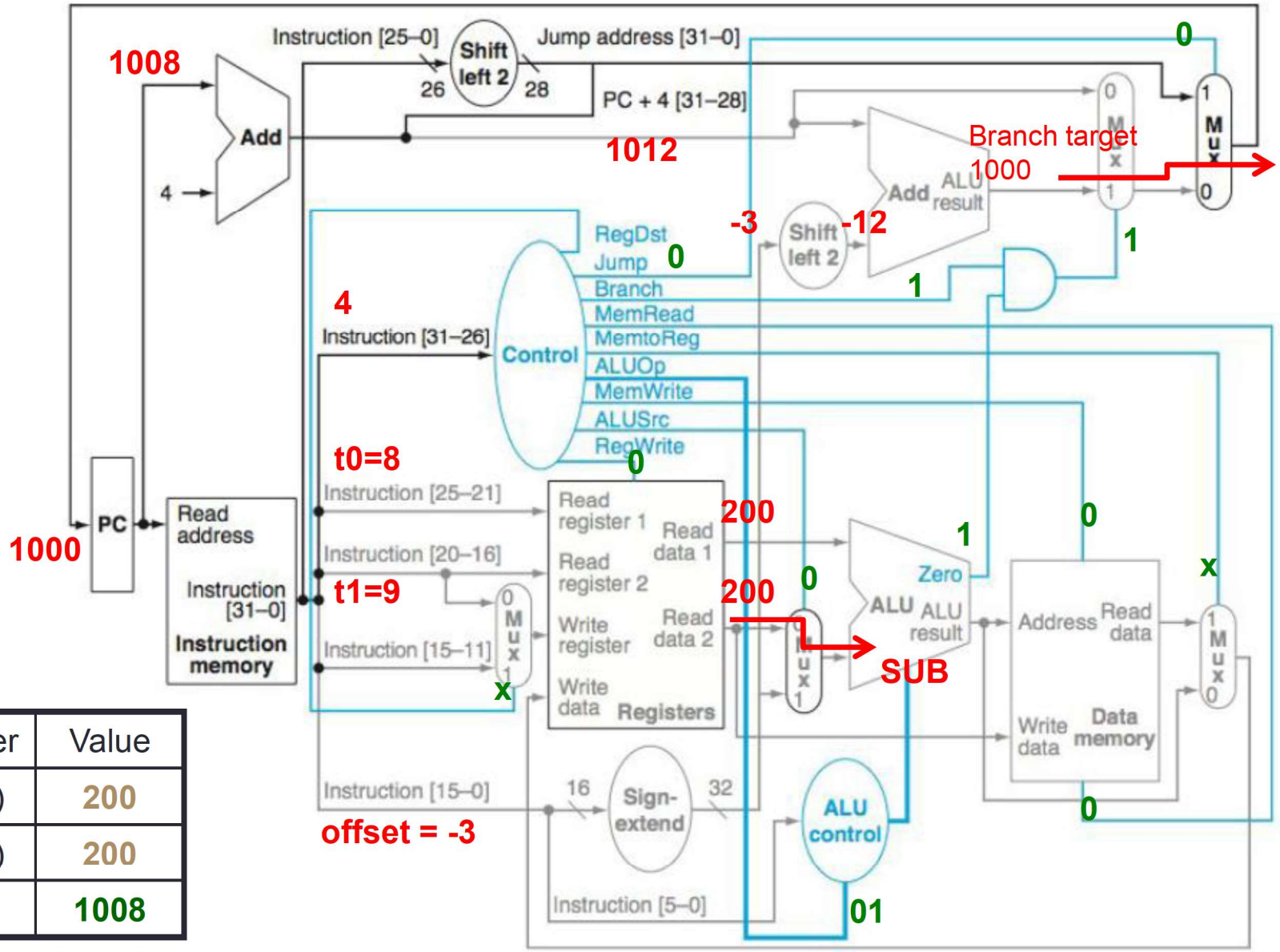
sub	$s1	$s2	$t0	shamt	function
000000 (0d)	10001 (17)	10010 (18)	01000 (8)	00000	100010 (SUB)



Register	Value
$s1	500
$s2	300

sub $t0, $s1, $s2

Beq	$t0	$t1	Offset
000100 (4)	01000 (8)	01001 (9)	1111 1111 1111 1101 (-3)



Register	Value
t0 (=8)	200
t1 (=9)	200
PC	1008

beq $t0, $t1, L1

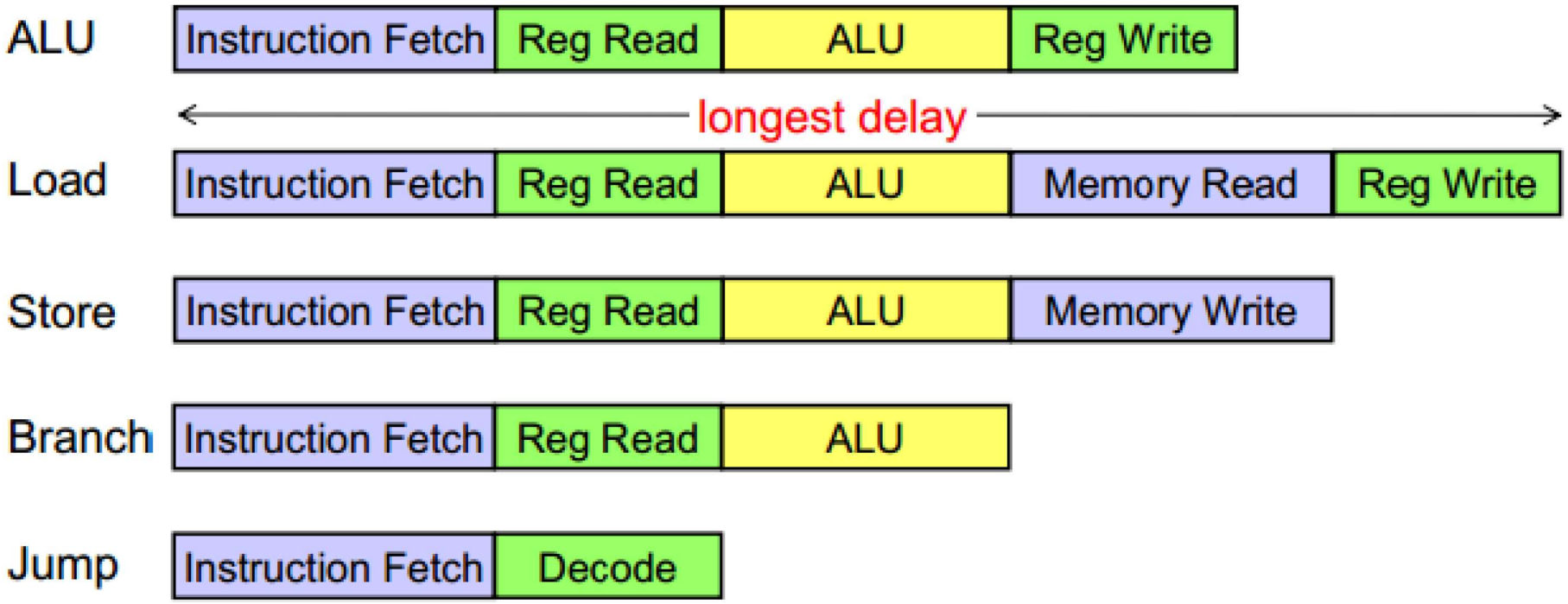
Single Cycle Datapath

- Combine all into a single architecture
  - Using mux with control signals

- Five steps finishes in a single clock cycle
  - Step 1: Instruction Fetch
  - Step 2: Instruction Decode and Register Read
  - Step 3: Execution, Memory Address Computation, or Branch Completion
  - Step 4: Memory Access or R-type instruction completion
  - Step 5: Write-back step

- No functional unit can be used twice in one clock cycle
- The clock period is decided by the slowest instruction

Single Cycle Datapath



Properties:

- One clock cycle per instruction
- Long cycle time