

## The Control Unit

---

- Decodes instruction to determine what segments will be active in the datapath
- Generates signals to
  - Set muxes to correct input
  - Operation code to ALU
  - Read and write to register file
  - Read and write to memory (load/store)
  - Update of program counter (branches)
  - Branch target address computation
- Two parts: **ALU control** and **Main control** (muxes, etc)

46

## ALU Control

---

- ALU control: specifies what operation ALU performs
  - I.e., *ALU operation* control signals
  - Eight input combinations (3 input control signals)
  - Five combinations used to select operation

<u>ALU control input</u>	<u>Function</u>
000	AND
001	OR
010	add
110	subtract
111	set on less than

- Based on instruction class, one of these will be done

47

## ALU Control - Selecting Operation

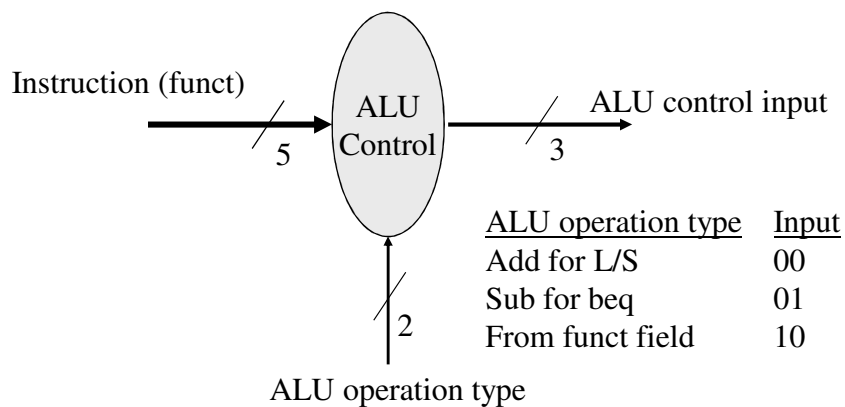
<u>Class (Opcode)</u>	<u>Operation</u>	<u>Control</u>
Load/store	Addition (memory address)	010
Branch	Subtraction (comparison)	110
Arithmetic	Depends on <b>funct</b> field:	
	100000      add	010
	100010      subtract	110
	100100      and	000
	100101      or	001
	101010      set on less than	111

- Generate ALU control based on **opcode** and **funct** field

48

## ALU Control Unit

- Small control unit associated with ALU
  - Generates appropriate control signals to ALU



49

## Building the ALU Control Unit

---

- Use truth table to determine how output will be generated based on the inputs

<u>Operation</u>	<u>ALUOp</u>	<u>Funct</u>	<u>Output</u>
Load/Store	00	XXXXXX	010 (add)
Beq	X1	XXXXXX	110 (sub)
Arithmetic	1X	XX0000	010 (add)
	1X	XX0010	110 (sub)
	1X	XX0100	000 (and)
	1X	XX0101	001 (or)
	1X	XX1010	111 (slt)

- From truth table, we can derive the control circuit 50

## Main Control Unit

---

- Use fields from instruction to generate control
  - We will “connect” the fields of the instruction to the datapath via the main control unit

R-type instruction	0	rs	rt	rd	shamt	funct
	31-26	25-21	20-16	15-11	10-6	5-0
Load/Store	35 / 43	rs	rt	address		
	31-26	25-21	20-16	15-0		
Branch (beq)	4	rs	rt	address		
	31-26	25-21	20-16	15-0		

## Main Control Unit

- Use fields from instruction to generate control
  - We will “connect” the fields of the instruction to the datapath via the main control unit

R-type instruction	0	rs	rt	rd	shamt	funct
	31-26	25-21	20-16	15-11	10-6	5-0
Load/Store	35 / 43	rs	rt	address		
	31-26	25-21	20-16	15-0		
Branch (beq)	4	rs	rt	address		
	31-26	25-21	20-16	15-0		

*Opcode is always in same position (31-26), called “Op[5-0]”*

52

## Main Control Unit

- Use fields from instruction to generate control
  - We will “connect” the fields of the instruction to the datapath via the main control unit

R-type instruction	0	rs	rt	rd	shamt	funct
	31-26	25-21	20-16	15-11	10-6	5-0
Load/Store	35 / 43	rs	rt	address		
	31-26	25-21	20-16	15-0		
Branch (beq)	4	rs	rt	address		
	31-26	25-21	20-16	15-0		

*Registers to be read are always rs and rt (always in fixed place)*

53

## Main Control Unit

---

- Use fields from instruction to generate control
  - We will “connect” the fields of the instruction to the datapath via the main control unit

R-type instruction	0	rs	rt	rd	shamt	funct
	31-26	25-21	20-16	15-11	10-6	5-0

Load/Store	35 / 43	rs	rt	address
	31-26	25-21	20-16	15-0

Branch (beq)	4	rs	rt	address
	31-26	25-21	20-16	15-0

*Base register for load/store is always rs in position 25-21*

54

## Main Control Unit

---

- Use fields from instruction to generate control
  - We will “connect” the fields of the instruction to the datapath via the main control unit

R-type instruction	0	rs	rt	rd	shamt	funct
	31-26	25-21	20-16	15-11	10-6	5-0

Load/Store	35 / 43	rs	rt	address
	31-26	25-21	20-16	15-0

Branch (beq)	4	rs	rt	address
	31-26	25-21	20-16	15-0

*16-bit offset for branch equal, load, and store always in 15-0*

55

## Main Control Unit

- Use fields from instruction to generate control
  - We will “connect” the fields of the instruction to the datapath via the main control unit

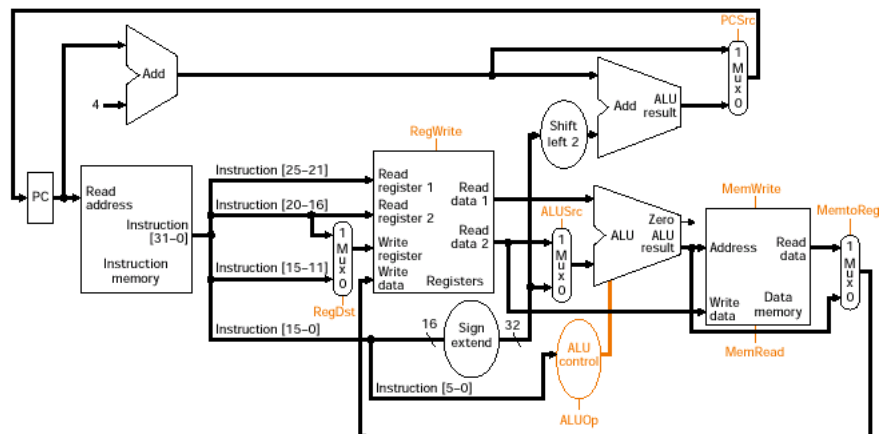
R-type instruction	0	rs	rt	rd	shamt	funct
	31-26	25-21	20-16	15-11	10-6	5-0
Load/Store	35 / 43	rs	rt	address		
	31-26	25-21	20-16	15-0		
Branch (beq)	4	rs	rt	address		
	31-26	25-21	20-16	15-0		

Destination register in one of two places: 15-11 for arithmetic and 20-16 for load; need multiplexor on write register address

56

## Full Datapath and Control Signals

- Control includes four muxes, ALU control unit, and control to register file and data memory



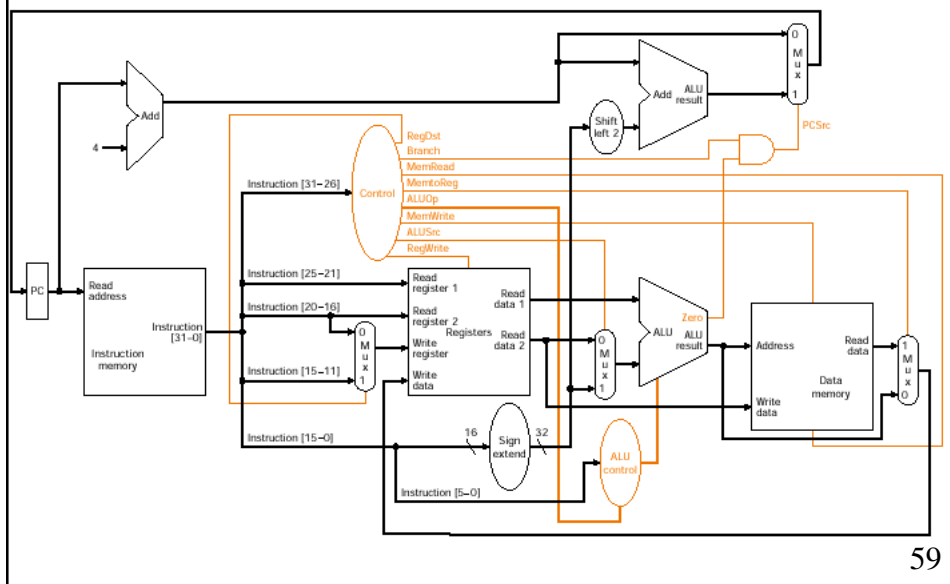
57

## The Control Signals

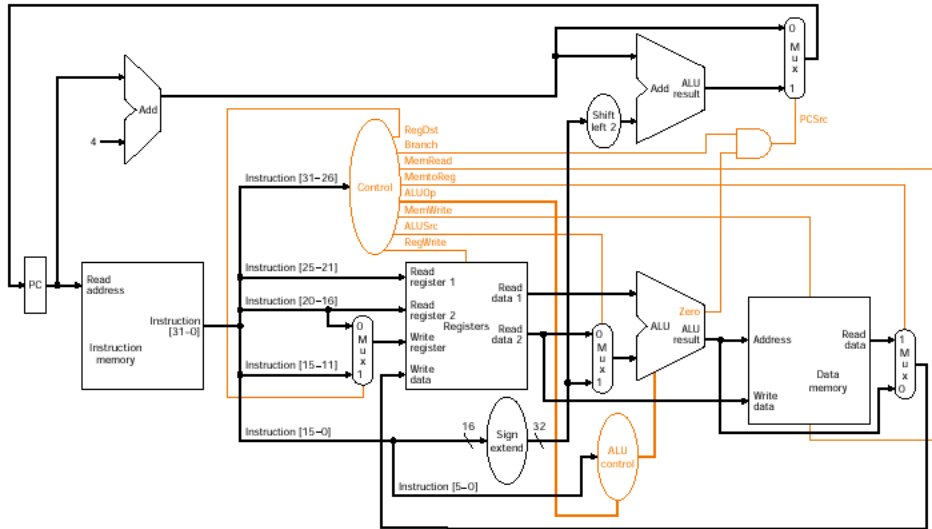
- Two ALUOp signals
- Seven other signals
  - **RegDst** - which field for write register
  - **RegWrite** - write to register file
  - **ALUSrc** - source for second ALU input
  - **PCSrc** - source for PC (PC + 4 or target address)
  - **MemRead** - read input address from memory
  - **MemWrite** - write input address/data to memory
  - **MemToReg** - source of write register port data input
- Branch control signal (set when instruction is branch)

58

## Full Datapath and Full Control



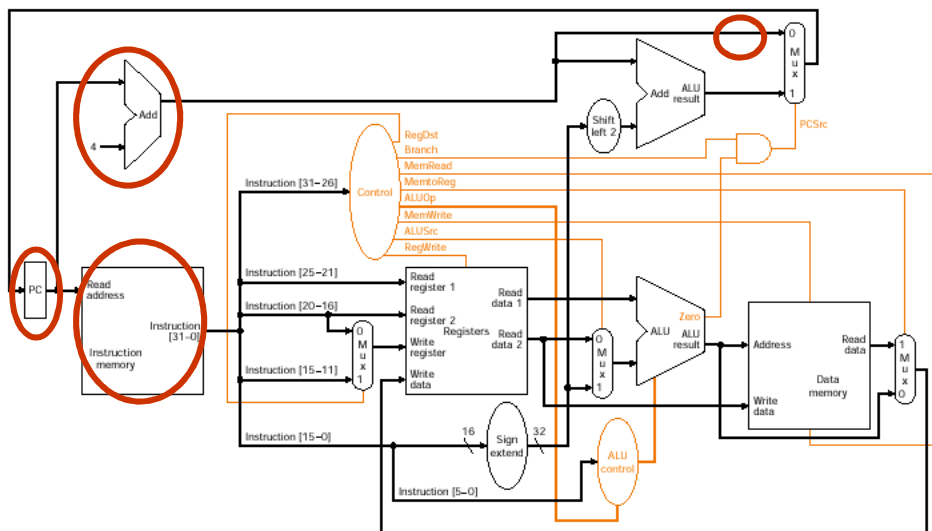
## Operation - R-type instructions



Let's do a `add $t1, $t2, $t3`

60

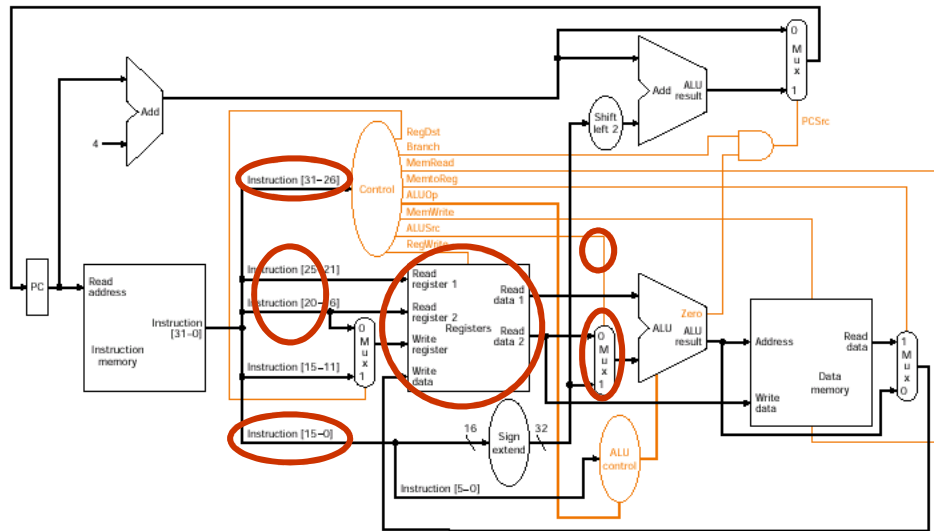
## FETCH - add \$t1,\$t2,\$t3



On fetch - PC, increment PC by 4, read from instruction memory

61

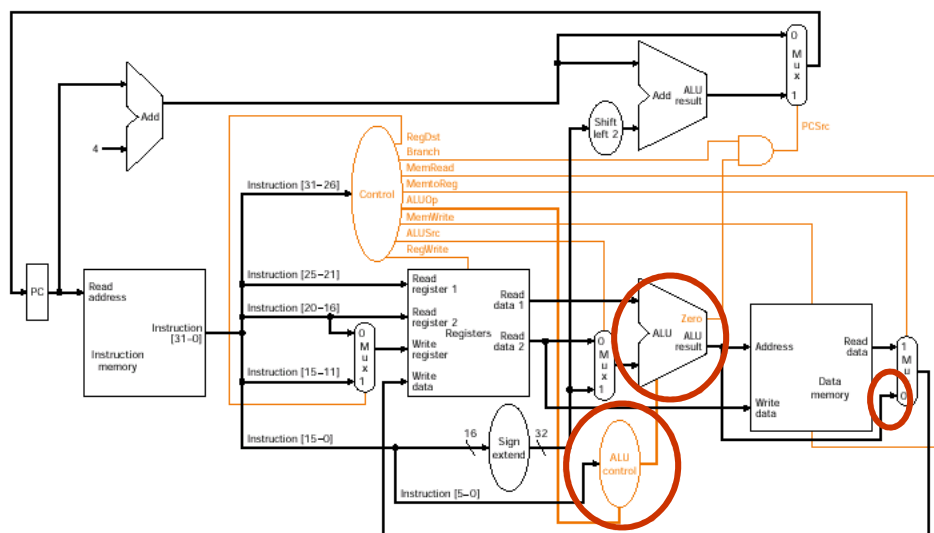
## READ REGISTERS (t1,t2) - add \$t1,\$t2,\$t3



Read source registers, main control unit computed all settings

62

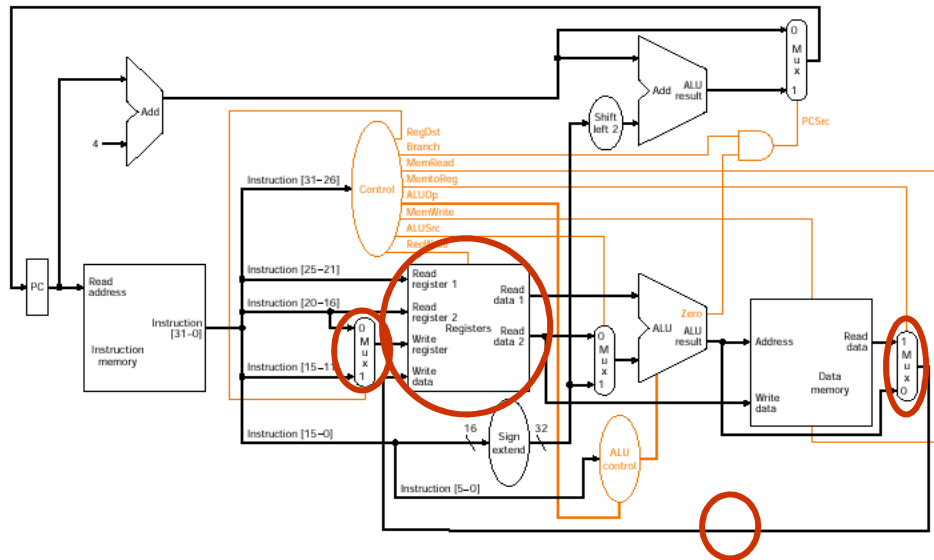
## EXECUTE - add \$t1,\$t2,\$t3



ALU operates on data from register file, ALU control determined

63

## WRITE DESTINATION (t3) - add \$t1,\$t2,\$t3



*Result from ALU is written to register file*

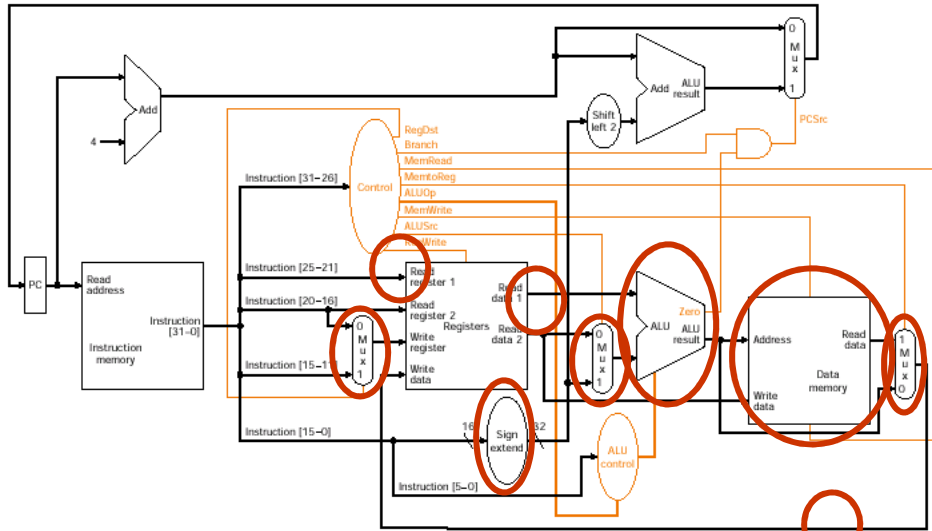
64

## Remember...Combinational Single Cycle

- Distinct steps shown for clarity
- Reality - information flows in those steps but it's all combinational logic
- Signals within the datapath vary and stabilize roughly in the flow of steps given
- All units and paths as marked during each step are active throughout the process!

65

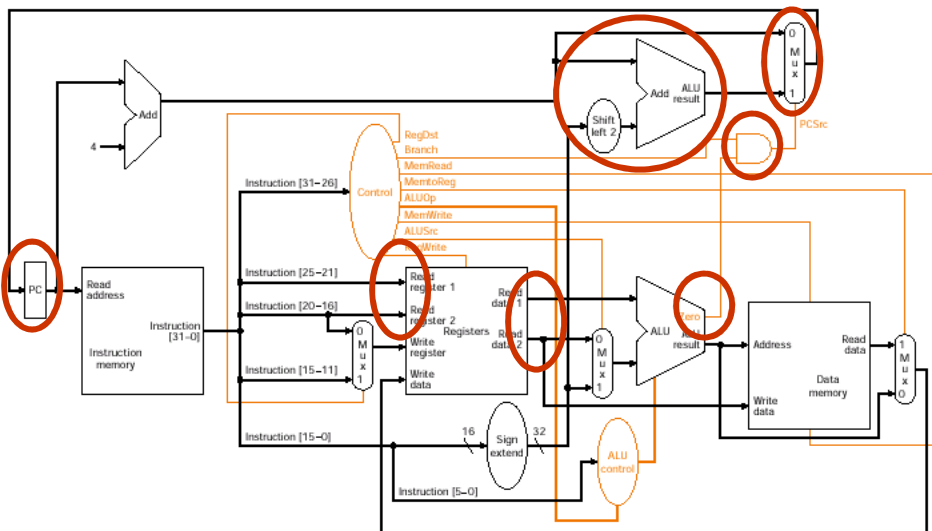
## Load Word - lw \$t1, offset(\$t2)



Only one source reg to read, sign extend, form address, write to destination in different position in instruction (bits 20-16)

66

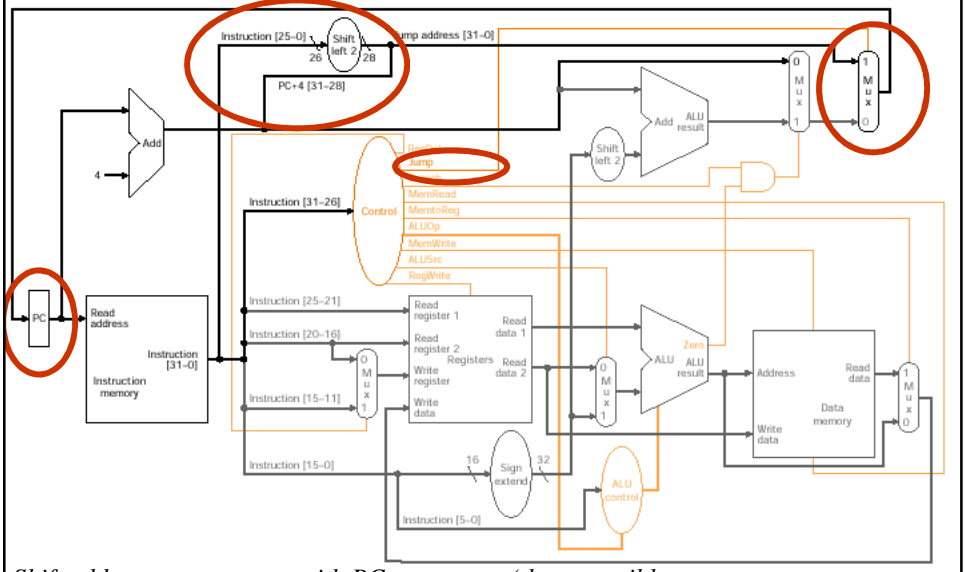
## Branches - beq \$t1,\$t2,offset



Read and compare sources, generate zero & branch control, compute target address, update PC if zero and branch

67

# Jumps - j label



Shift address, concatenate with PC, extra mux (three possible sources now for program counter)