Simple Processor Design

Single Cycle Implementation

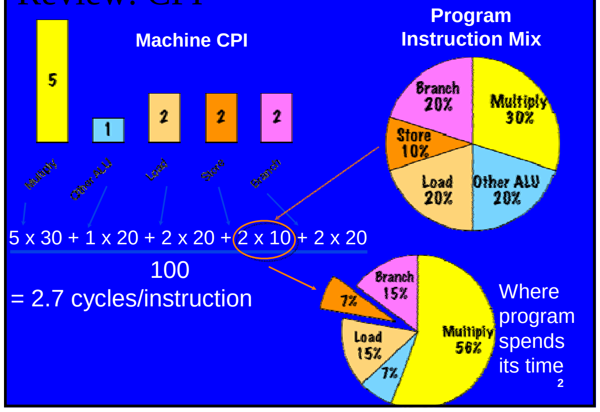
Chapter 5.1-5.4

EEC170 FQ 2005

“How to eat an elephant? One byte at a time”

1

Review: CPI



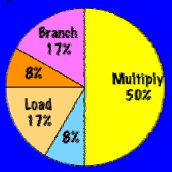
$$\frac{5 \times 30 + 1 \times 20 + 2 \times 20 + 2 \times 10 + 2 \times 20}{100}$$

= 2.7 cycles/instruction

2

Review: Amdahl’s Law (of Diminishing Returns)

Where program spends its time



If enhancement “E” speeds up multiply, but other instructions are unchanged, what is the maximum speedup S?

$$S_{max} = \frac{1}{1 - (\%\ \text{affected} / 100\ \%)} = \frac{1}{1 - (50/100)} = 2$$

Attributed to Gene Amdahl -- “Amdahl’s Law”

3

How to Design a Simple Processor: st

- 1. Analyze instruction set => datapath requirements
  - the meaning of each instruction is given by the *register transfers*
  - datapath must include storage element for ISA registers
    - possibly more
  - datapath must support each register transfer
- 2. Select set of datapath components
  - establish clocking methodology
- 3. Assemble datapath meeting the requirements
- 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
- 5. Assemble the control logic

4

Review: R-Format Instructions

- Define “fields” of the following number of bits each: 6 + 5 + 5 + 5 + 5 + 6 = 32

6	5	5	5	5	6

- For simplicity, each field has a name:

opcode	rs	rt	rd	shamt	funct

5

Review: I-Format Instructions

- Define “fields” of the following number of bits each:

6 bits	5 bits	5 bits	16 bits

- Each field has a name:

opcode	rs	rt	immediate

- Key Concepts
  - Keep opcode field identical to R-format and J-format for consistency.
  - Can specify jumps and address displacement within (roughly) $\pm 2^{15}$ range.

6

Review: J-Format Instructions

- Define "fields" of the following number of bits each:

6 bits	26 bits

- As usual, each field has a name:

opcode	target address

- Key Concepts
  - Keep opcode field identical to R-format and I-format for consistency.
  - Combine all other fields to make room for large target address.

7

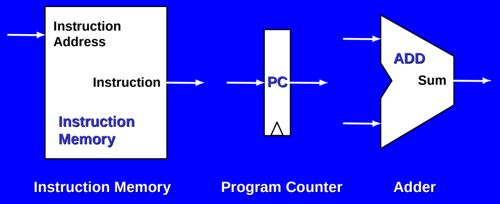
J-Format Instructions (2/2)

- Summary:
  - New PC = { PC[31..28], target address, 00 }
- Understand where each part came from!
- Note: In Verilog,
  { , , } means concatenation
  { 4 bits , 26 bits , 2 bits } = 32 bit address
  - { 1010, 11111111111111111111111111, 00 }
    = 10101111111111111111111111111100

8

Instruction Path Components

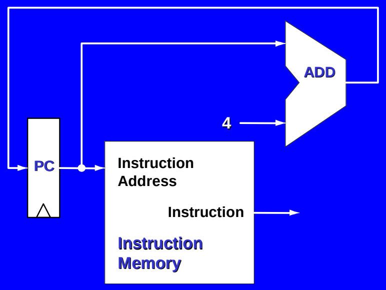
- Requires three components



9

Instruction Path

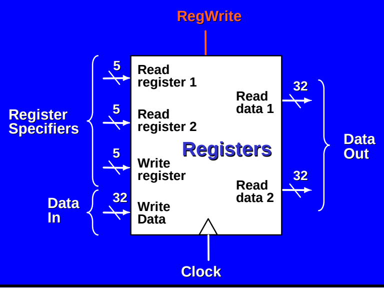
- Increments to next word in instruction memory



10

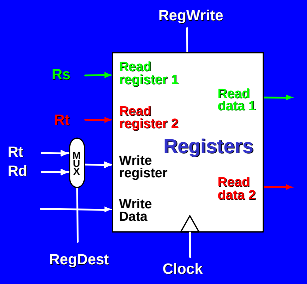
Register File

- Thirty-two 32-bit registers
  - Register specifiers from instruction fields Rs, Rt and Rd



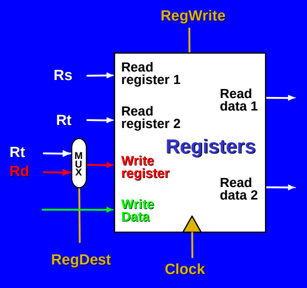
11

Register File Read



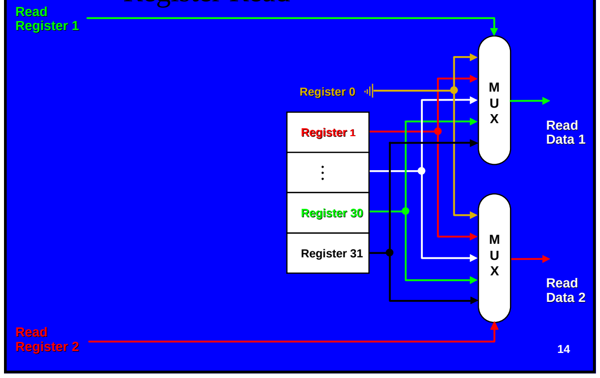
12

Register File Write: R-Type



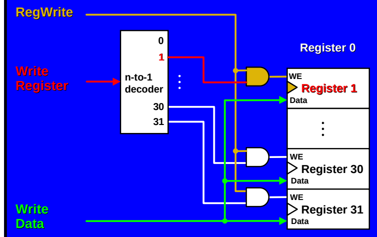
Register File Design

Register Read

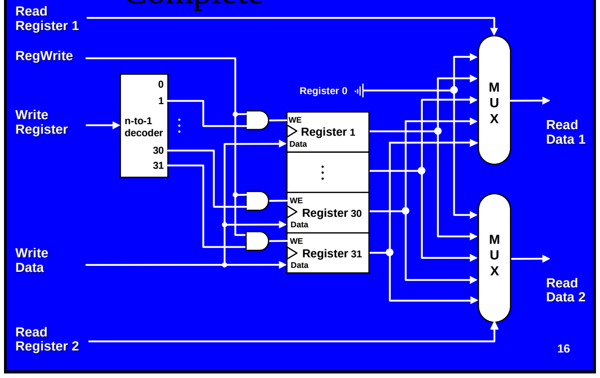


Register File Design

Register Write

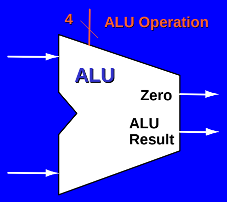


Register File Design: Complete



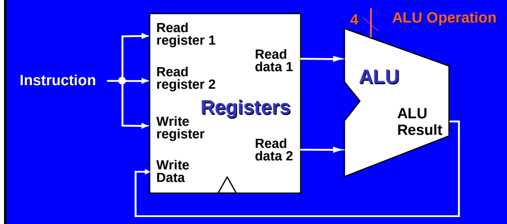
ALU

- We already know internal ALU design
  - Will see ALU Operation control-line specification later



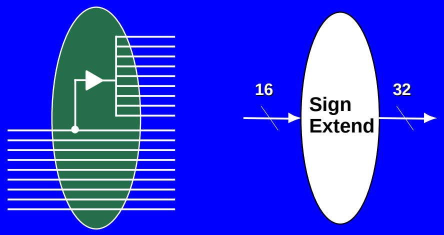
R-Type Instruction Datapath

- Register-file output goes directly to ALU



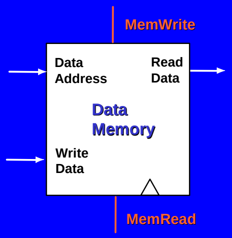
Sign Extension 'Unit'

- Converts a 16-bit signed integer to a 32-bit signed integer to make ALU input uniform
  - Only need to replicate (fan out) MSB of 16-bit int



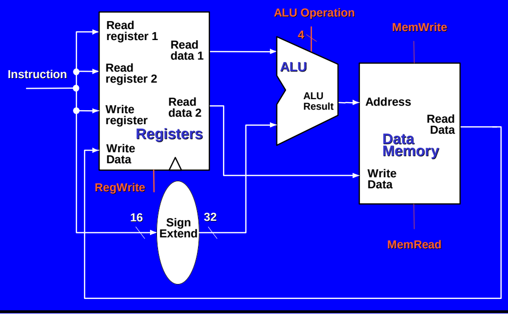
Data Memory Unit

- Separate memory units for Instructions and Data so they can be accessed during same cycle



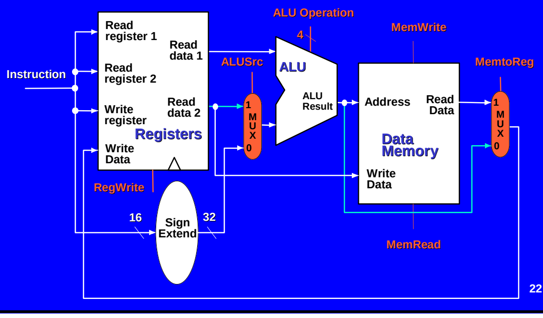
Data Path for Load & Store

- ALU is used to compute Address = Rs + offset



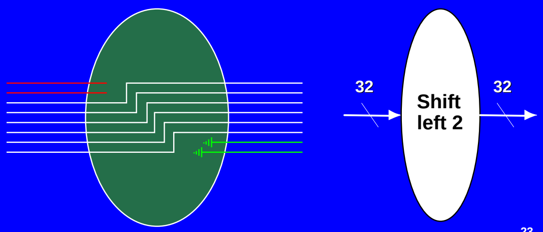
Data Path for R-Type, Load & Store

- ALU B input and Reg Write Data are selected based on instruction type



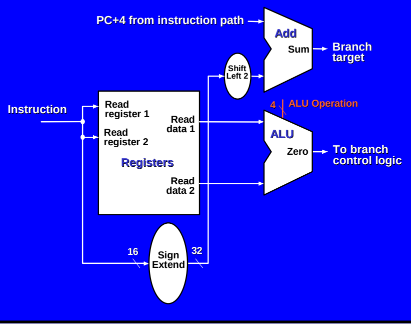
Shift Left 2 'Unit'

- Not a barrel shifter as we saw in the ALU design, rather simply a shift of the wiring
  - Delete two MSBs (wires), insert two zero LSBs (wires)



Branch Computation

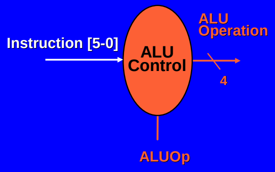
- Must compute PC + 4 + word offset using dedicated Adder, Equal (Zero) using ALU



ALU Operation

- Four ALU Operation lines are used to encode our MIPS instruction subset

ALU Operation	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set less than
1100	NOR



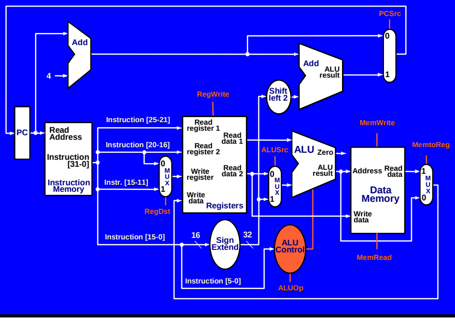
ALU Operation Specification

- ALU Operation is based on ALUOp and Funct Field inputs

Instruction Opcode	ALUOp	Instruction Operation	Funct Field	ALU Action	ALU Operation
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 less than	101010	set less than	0111

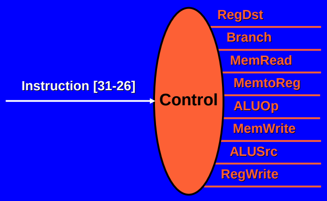
Complete Data Path

- Allows R-type, I-type, LW/SW and Branch



Main Control Unit

- Control unit takes opcode as input, produces control lines as output



Control Unit Specification

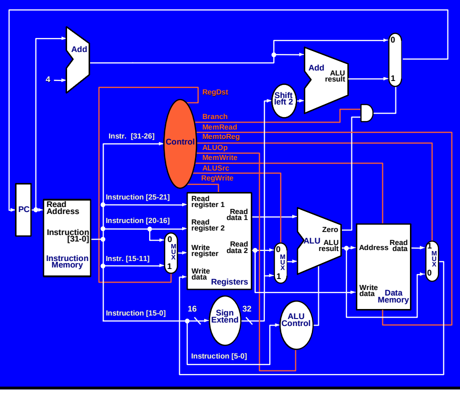
- Opcode in, various control lines out

Instr	Reg Dst	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp2
R-Type	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

Announcements!

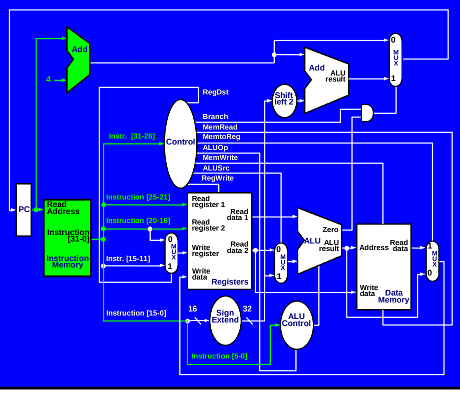
- HW #3 is online
  - Due Friday November 4th at 5pm.
- 30~45 min Quiz
  - On Monday
    - yes, this coming Monday, i.e., 10/31
  - Chapter 1, 2 and 3.
  - You have finished homework on these chapters by Monday => should be straight forward.
- Midterm
  - In two weeks (wed 11/9)
  - Chapters 1-5

Datapath with Control Unit



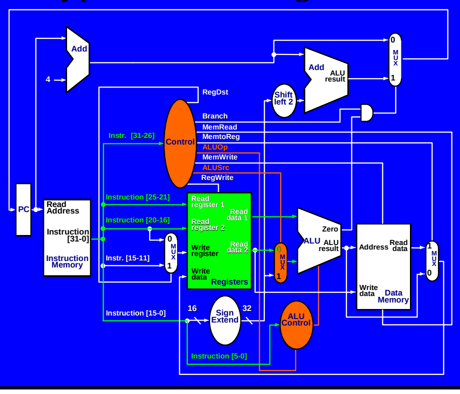
31

R-Type Phase 1: Fetch



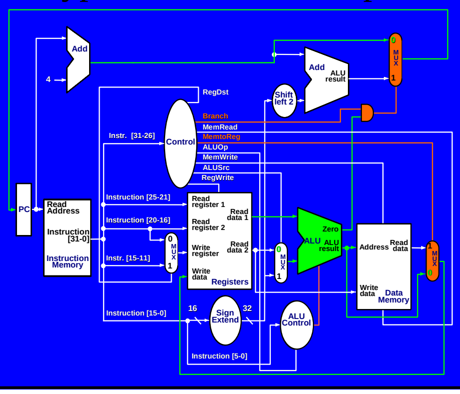
32

R-Type Phase 2: Reg Read



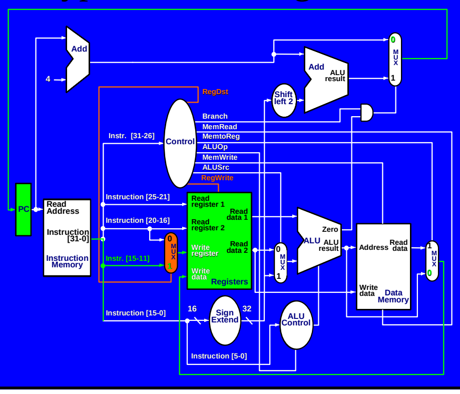
33

R-Type Phase 3: ALU Op



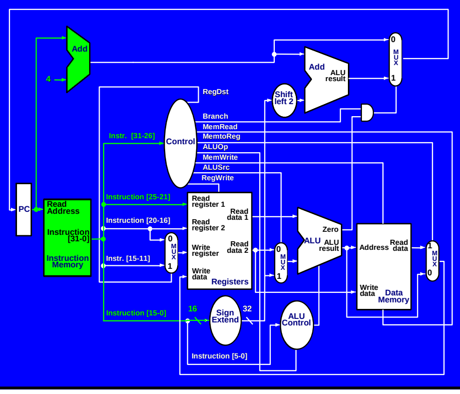
34

R-Type Phase 4: Reg Write



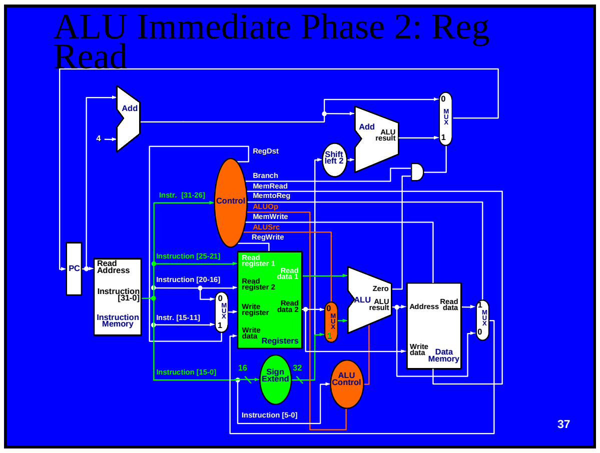
35

ALU Immediate Phase 1: Fetch

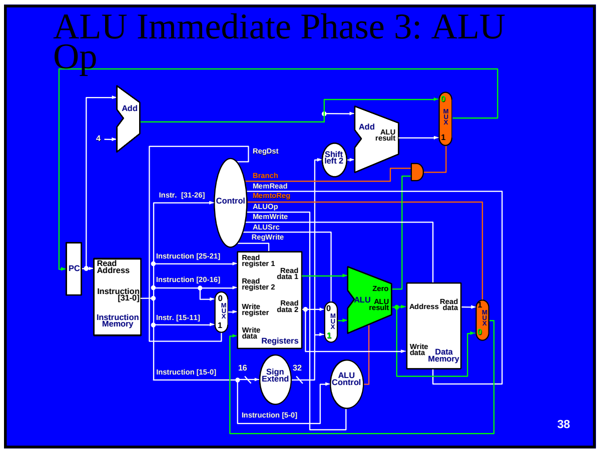


36

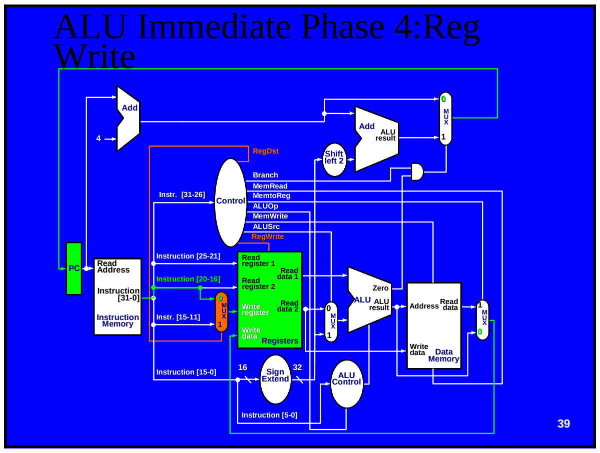
ALU Immediate Phase 2: Reg Read



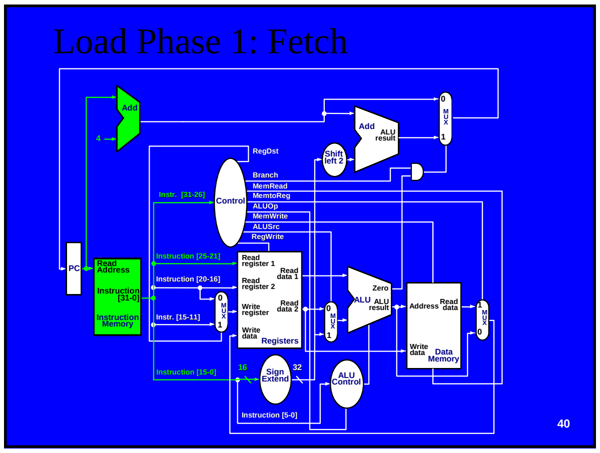
ALU Immediate Phase 3: ALU Op



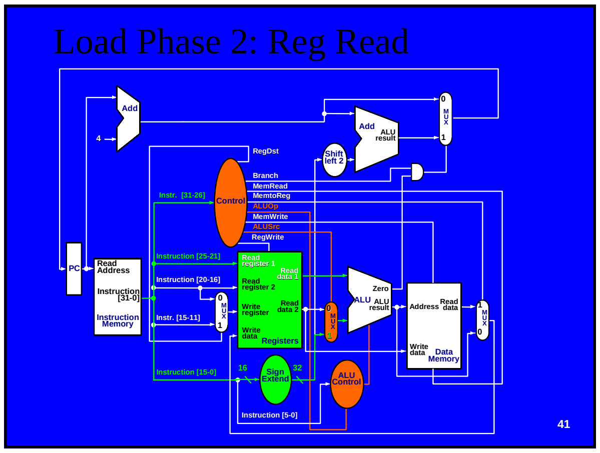
ALU Immediate Phase 4:Reg Write



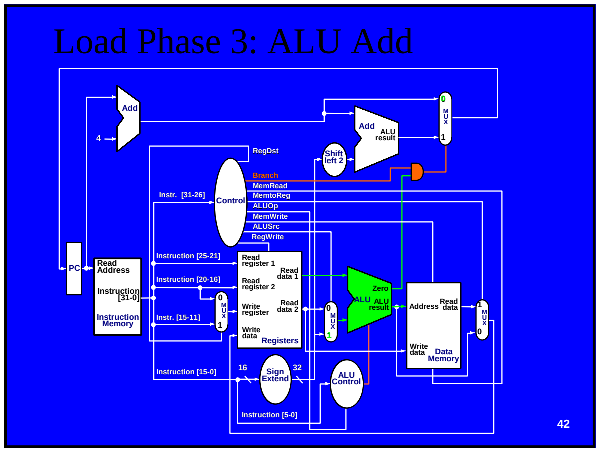
Load Phase 1: Fetch



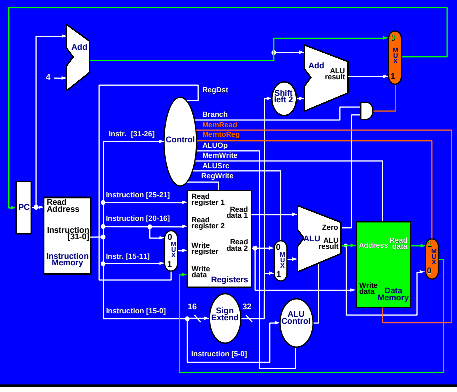
Load Phase 2: Reg Read



Load Phase 3: ALU Add

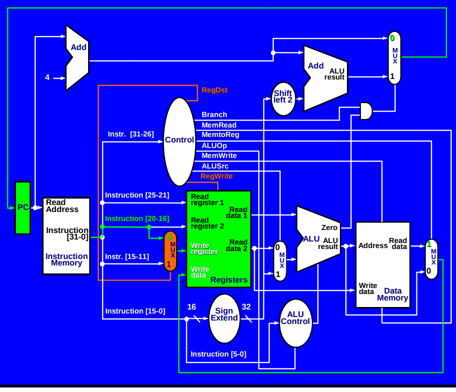


Load Phase 4: Memory Read



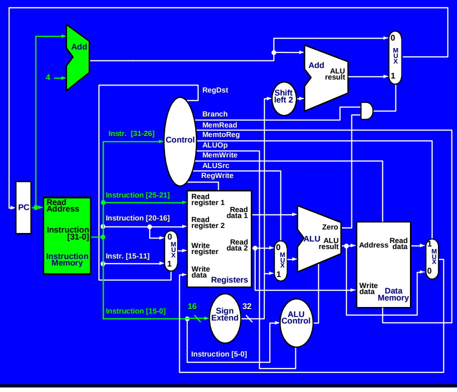
43

Load Phase 5: Reg Write



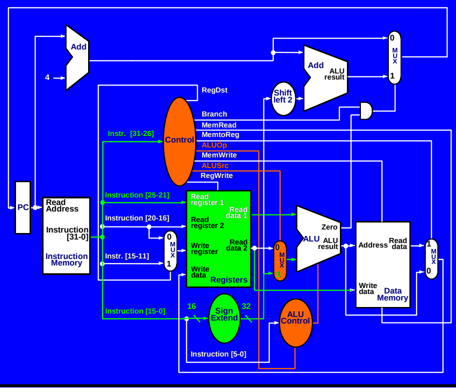
44

Store Phase 1: Fetch



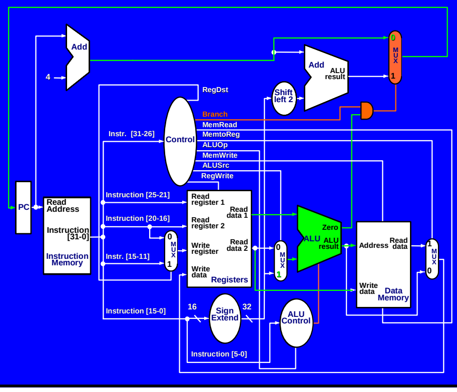
45

Store Phase 2: Reg Read



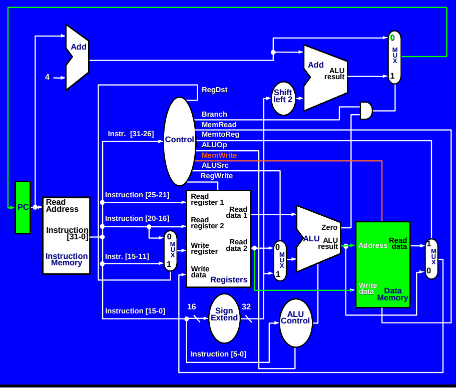
46

Store Phase 3: ALU Add



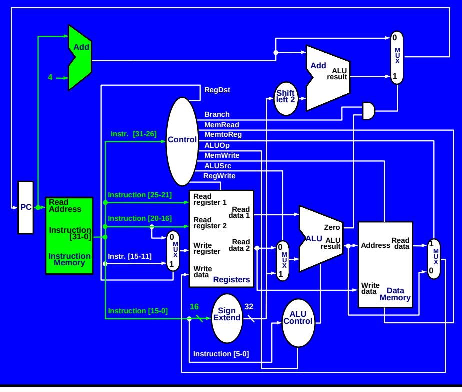
47

Store Phase 4: Memory Store



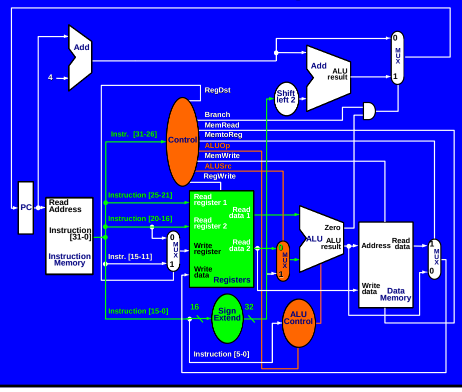
48

Branch Phase 1:Fetch



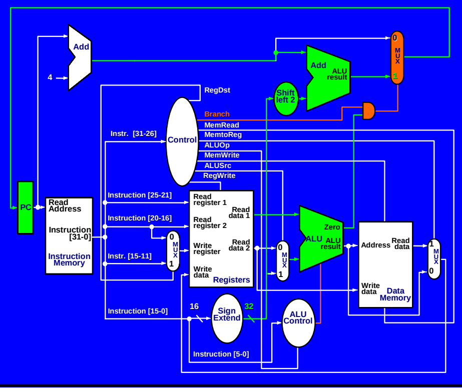
49

Branch Phase 2: Reg Read



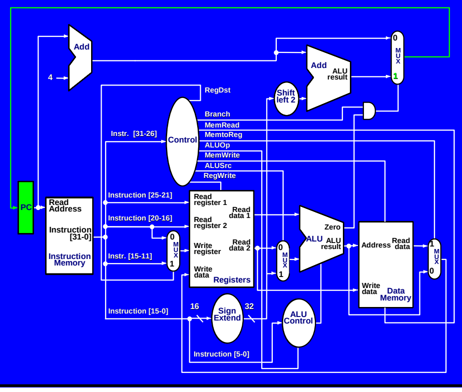
50

Branch Phase 3: ALU Subtract



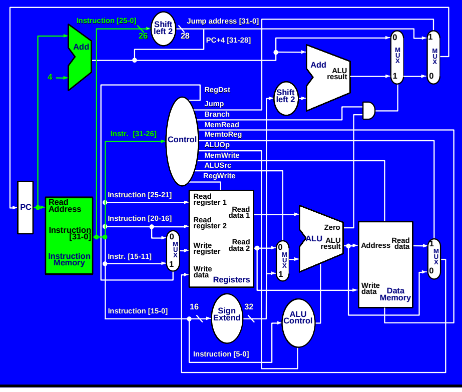
51

Branch Phase 4: PC Write



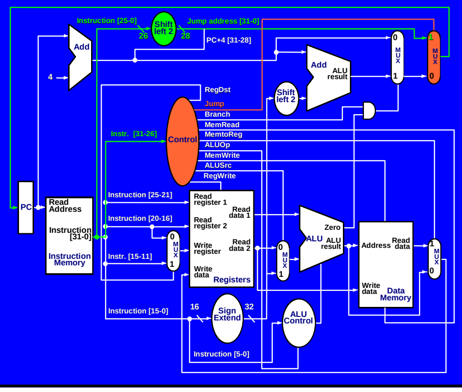
52

Jump Phase 1: Fetch



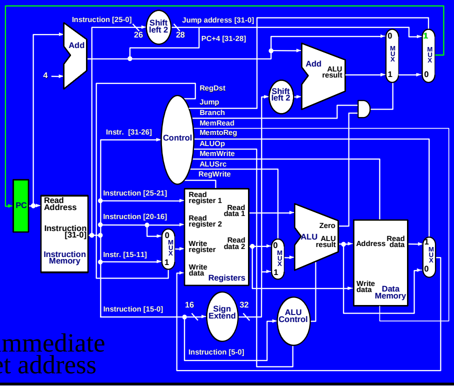
53

Jump Phase 2: Decode



54

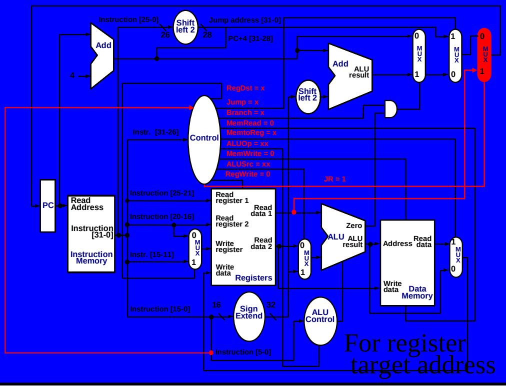
Jump Phase 3: PC Write



For immediate target address

55

Jump Phase 3: PC Write



For register target address

56

Single-Cycle Performance

- By nature of design CPI = 1
- CCT is determined by the longest instruction, Load, which uses all executions units

Instruction Class	Functional Units used by instruction class				
R-type	I Fecth	Reg Read	ALU	Reg Write	
Load	I Fecth	Reg Read	ALU	Mem Read	Reg Write
Store	I Fecth	Reg Read	ALU	Mem Write	
Branch	I Fecth	Reg Read	ALU		
Jump	I Fecth				

57

Functional Units Timing

- Example timing for functional units
  - Register file: 50ps
  - ALU and adders: 100ps
  - Memory: 200ps

58

Clock Cycle Time

- Clock cycle time is determined by Load instruction

Instruction Class	IFetch	Reg Read	ALU	Memory Access	Reg Write	Total
R-type	200ps	50ps	100ps		50ps	400ps
Load	200ps	50ps	100ps	200ps	50ps	600ps
Store	200ps	50ps	100ps	200ps		550ps
Branch	200ps	50ps	100ps			350ps
Jump	200ps					200ps

59

Single Cycle Design Summary

- Good news: CPI = 1 is excellent
- Bad news:
  - CCT = 600ps is poor
  - Redundant hardware
    - Multiple adders
    - Two memory units
- Figure of merit: average instruction execution time (IET) = CPI x CCT = 1 x 600ps = 600ps
- Can we do better? Next attempt Multicycle design

60