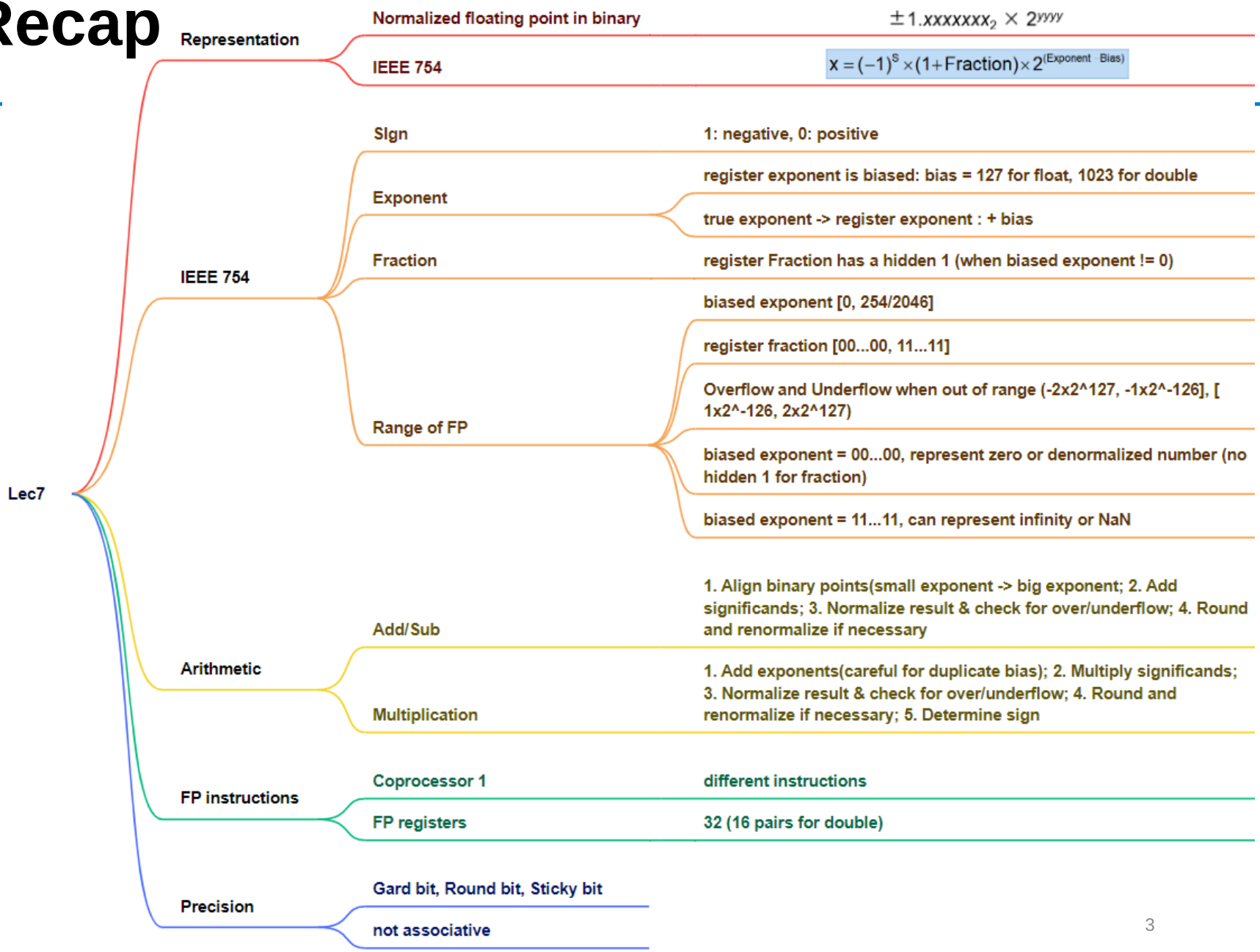
Lecture 8
The Processor

CS202 2023 Spring

Today's Agenda

- Recap
- Context
  - Introduction to designing a processor
  - Analyzing the instruction set
  - Building the datapath
  - A single-cycle implementation
  - Control for the single-cycle CPU
- Reading: Textbook 4.1-4.4
  - Midterm content: Section 1.1-4.4

Recap


Lec7
Representation
Normalized floating point in binary
$\pm 1.xxxxxxx_2 \times 2^{yyyy}$
IEEE 754
$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$
IEEE 754
Sign
1: negative, 0: positive
Exponent
register exponent is biased: bias = 127 for float, 1023 for double
true exponent -> register exponent : + bias
Fraction
register Fraction has a hidden 1 (when biased exponent != 0)
Range of FP
biased exponent [0, 254/2046]
register fraction [00...00, 11...11]
Overflow and Underflow when out of range (-2x2^127, -1x2^-126], [1x2^-126, 2x2^127)
biased exponent = 00...00, represent zero or denormalized number (no hidden 1 for fraction)
biased exponent = 11...11, can represent infinity or NaN
Arithmetic
Add/Sub
1. Align binary points(small exponent -> big exponent; 2. Add significands; 3. Normalize result & check for over/underflow; 4. Round and renormalize if necessary
Multiplication
1. Add exponents(careful for duplicate bias); 2. Multiply significands; 3. Normalize result & check for over/underflow; 4. Round and renormalize if necessary; 5. Determine sign
FP instructions
Coprocessor 1
different instructions
FP registers
32 (16 pairs for double)
Precision
Gard bit, Round bit, Sticky bit
not associative

Introduction

- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware

$$\text{CPU Time} = \text{Instructions} \times \text{CPI} \times \text{Clock Period} = \frac{\text{Instructions} \times \text{CPI}}{\text{Clock Rate}}$$

- We will examine two MIPS implementations
  - A simplified version
  - A more realistic pipelined version
- Simple subset, shows most aspects
  - Arithmetic/logical: add, sub, and, or, slt
  - Memory reference: lw, sw
  - Control transfer: beq, j

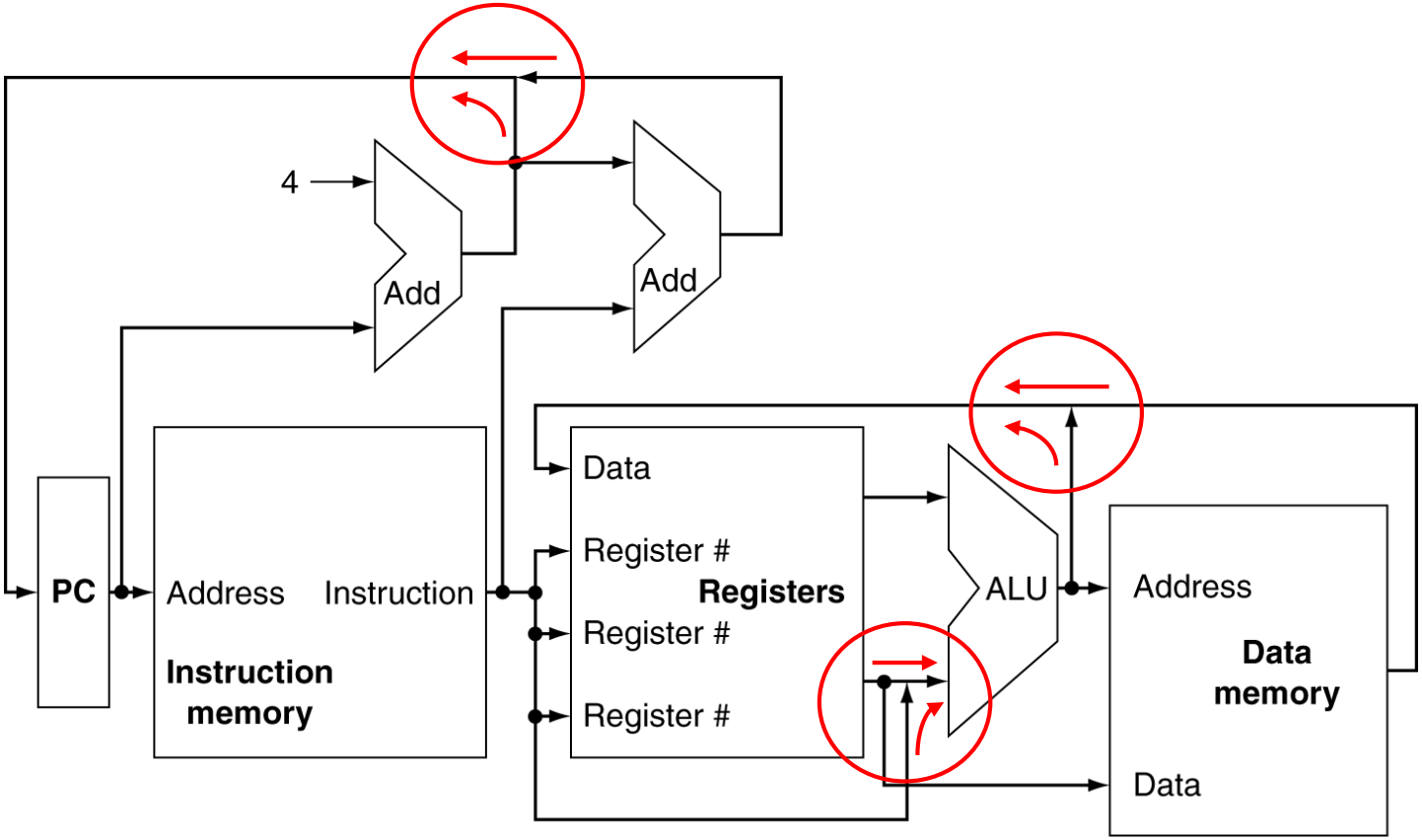
Instruction Execution

- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
  - Use ALU to calculate
    - Arithmetic result
    - Memory address for load/store
    - Branch target address
  - Access data memory for load/store
  - PC ← target address or PC + 4

CPU Overview

- We need Memory, Registers, ALU and control logics
  - The role of each unit?
  - The inputs to the Memory / ALU / Registers?

指令是控制信号

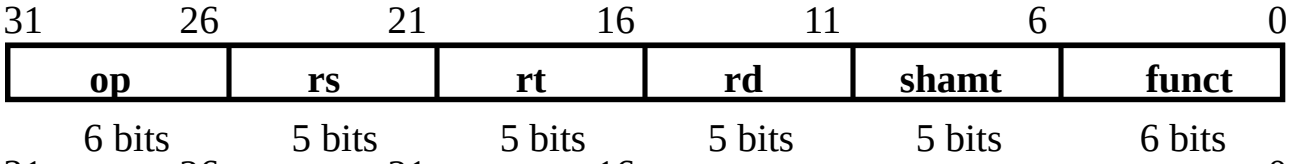


How to Design a Processor?

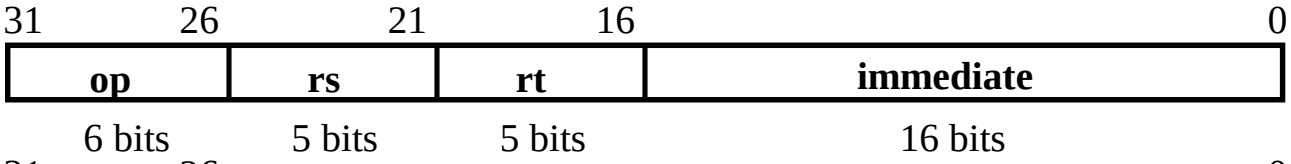
1. Analyze instruction set (datapath requirements)
   - The meaning of each instruction is given by the register transfers
   - Datapath must include storage element
   - Datapath must support each register transfer
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting the requirements
4. Assemble the control logic
   - by analyzing implementation of each instruction to determine setting of control points effecting register transfer

Step 1: Analyze Instruction Set

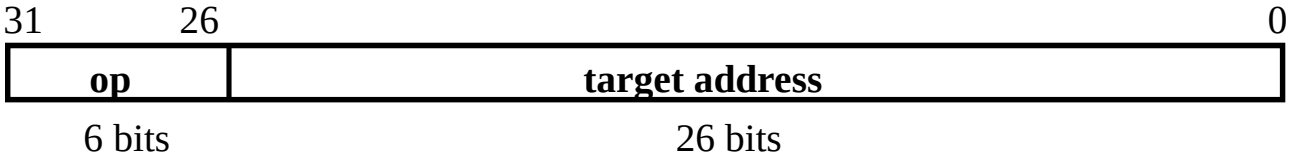
- All MIPS instructions are 32 bits long with 3 formats:
  - R-type:



  - I-type:



  - J-type:

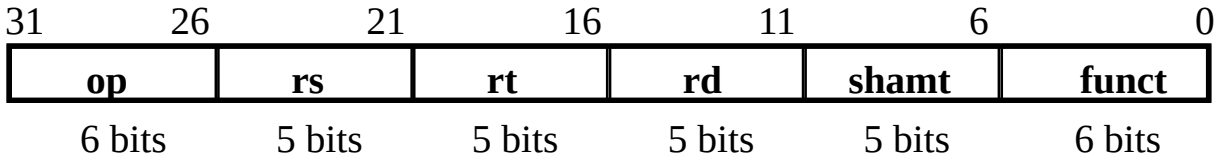


- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: source and destination register
  - shamt: shift amount
  - funct: selects variant of the “op” field
  - address / immediate
  - target address: target address of jump

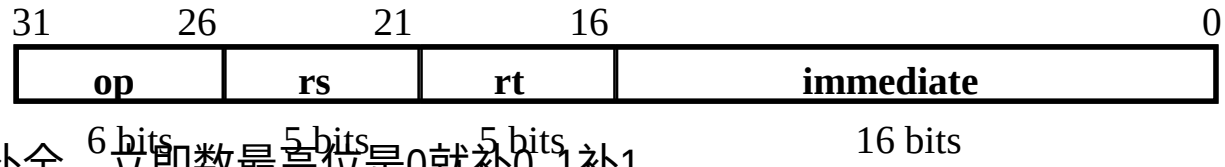
Instruction examples of a MIPS Subset

- We focus on 9 typical instructions
- R-Type:
  - add rd, rs, rt
  - sub rd, rs, rt
  - and rd, rs, rt
  - or rd, rs, rt
  - slt rd, rs, rt

读，写



- Load/Store:
  - lw rt,rs,imm16
  - sw rt,rs,imm16

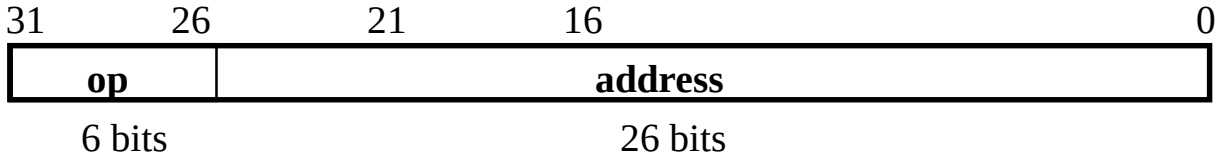


补全，立即数最高位是0就补0，1补1

- Branch:
  - beq rs,rt,imm16

和pc相加，先做补全，补全之后相加，然后末尾加两个0（左移两位）

- Jump:
  - j target



Requirements of Instruction Set

- Datapath needs the followings:
  - Memory
    - store instructions and data
  - Registers (32 x 32)
    - read RS 读
    - read RT 有时读有时写
    - Write RT or RD
  - PC
  - Extender for zero- or sign-extension
  - Add and sub register or extended immediate (ALU)
  - Add 4 or extended immediate to PC

How to Design a Processor?

1. Analyze instruction set (datapath requirements)
   - The meaning of each instruction is given by the register transfers
   - Datapath must include storage element
   - Datapath must support each register transfer
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting the requirements
4. Assemble the control logic
   - by analyzing implementation of each instruction to determine setting of control points effecting register transfer

Step 2: Select Datapath Building Units

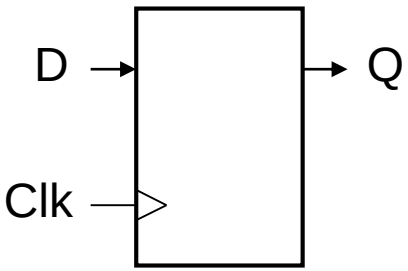
- Datapath
  - Elements that process data and addresses in the CPU
    - Registers, ALUs, MUXs, Memories, …
- We will build a MIPS datapath incrementally
  - Refining the overview design

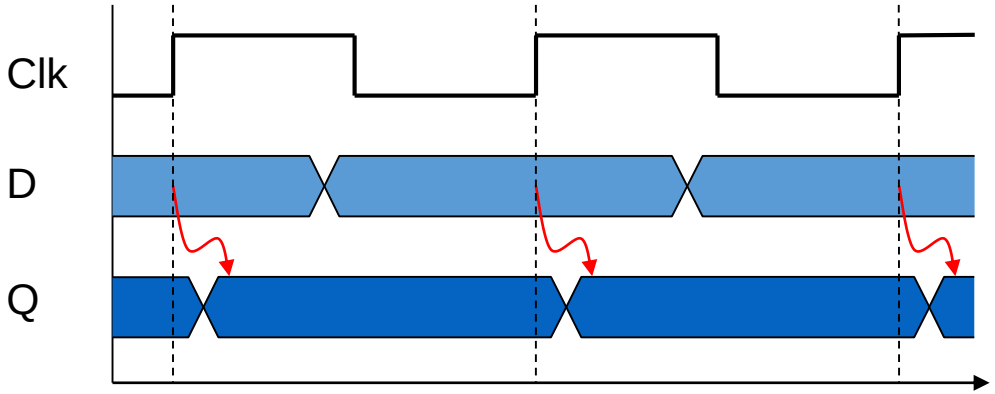
Logic Design Basics

- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  - Operate on data
  - Output is a function of input
- State (sequential) elements
  - Store information

State Elements (sequential)

- State element
  - The state element has a pre-stored state
  - It has some internal storage
  - Has at least two inputs and one output (e.g. D-type flip-flop):
    - The data to be written into the element
    - The clock which determines when the data is written
    - The output: the value that was written in earlier cycle
  - Examples: register and memory

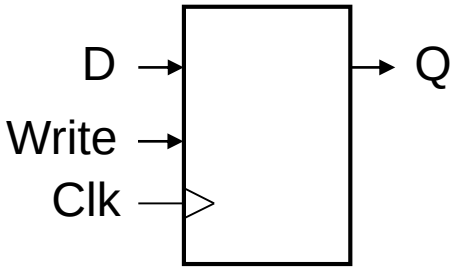


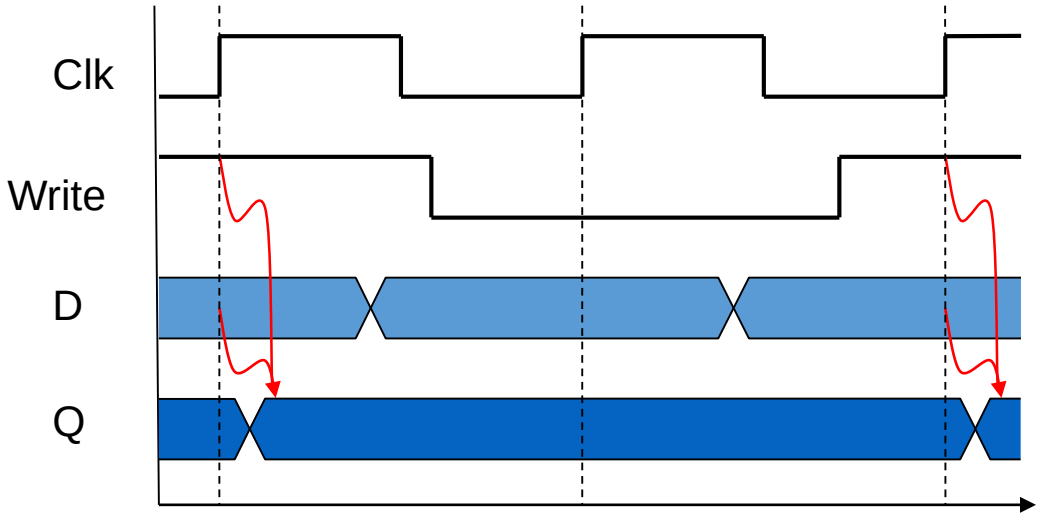


State Elements (sequential)

- Register without write control (e.g. program counter)
  - Uses a clock signal to determine when to update
  - Edge-triggered: update when Clk changes from 0 to 1
- Register with write control (e.g. data memory/register)
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later

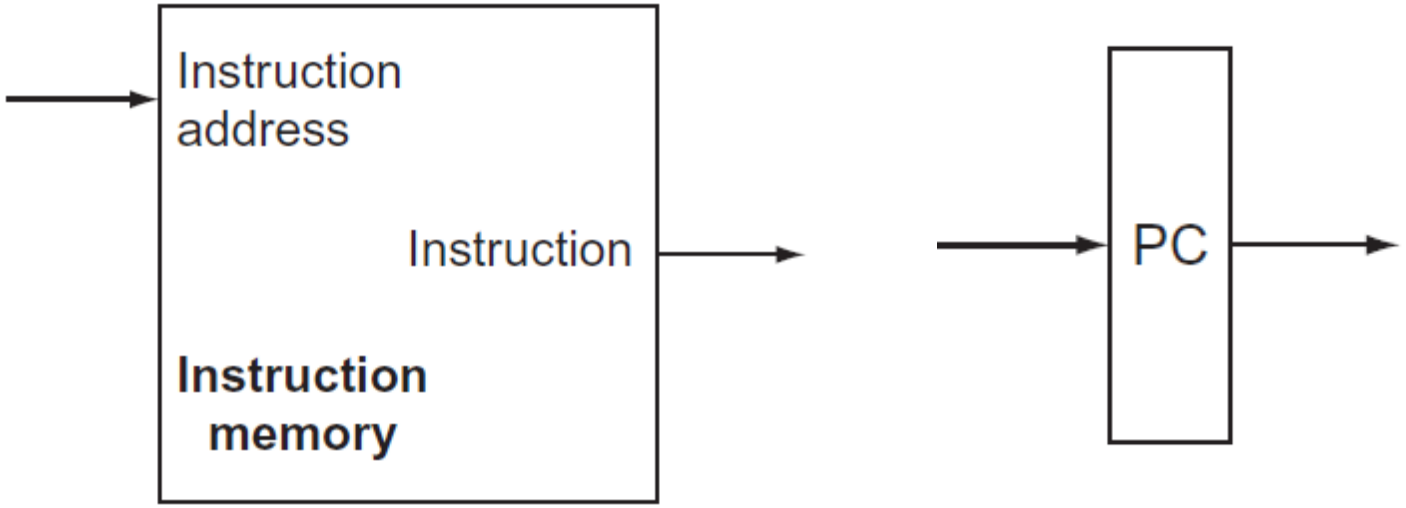
控制输入的





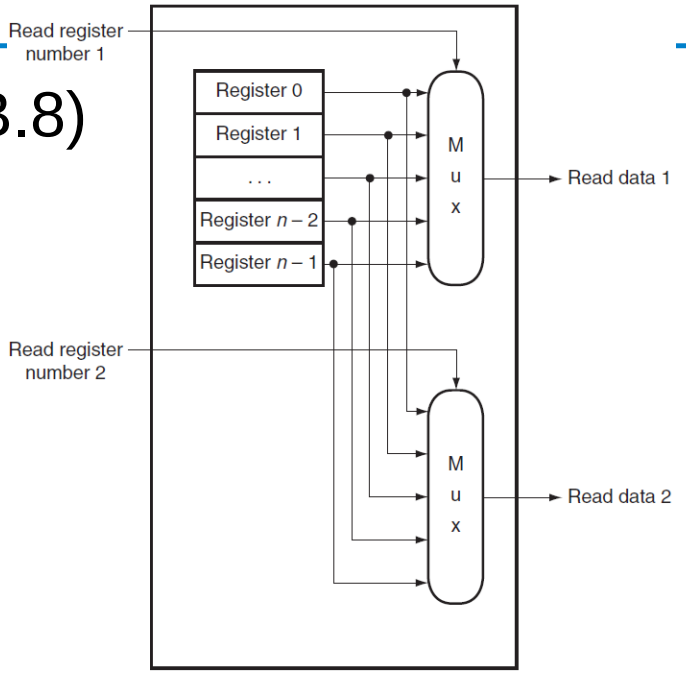
State Element: Instruction Memory and PC

- Instruction Memory
  - Input: InstructionAddress (32-bit)
  - Output: Instructions (32-bit)
- Program Counter
  - Input: InstructionAddress (32-bit)
  - Output: InstructionAddress (32-bit)
- No control signals

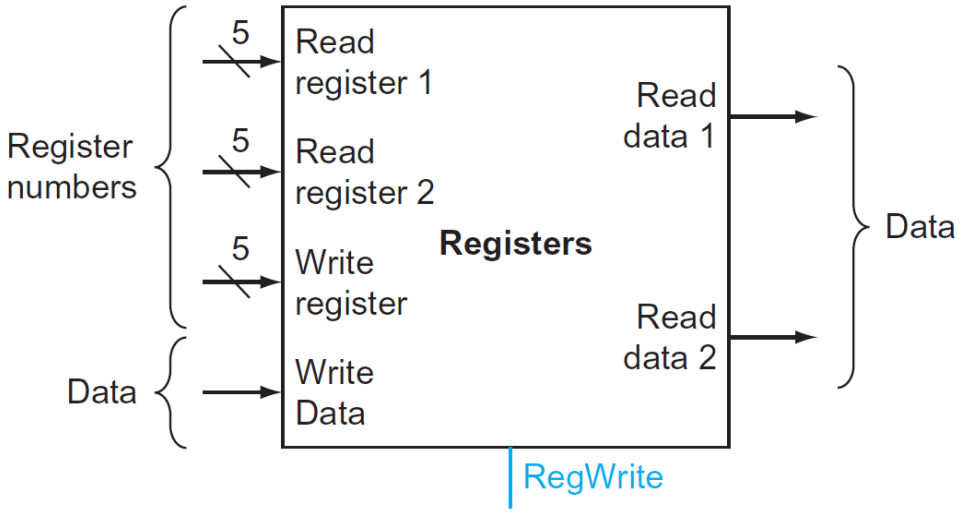


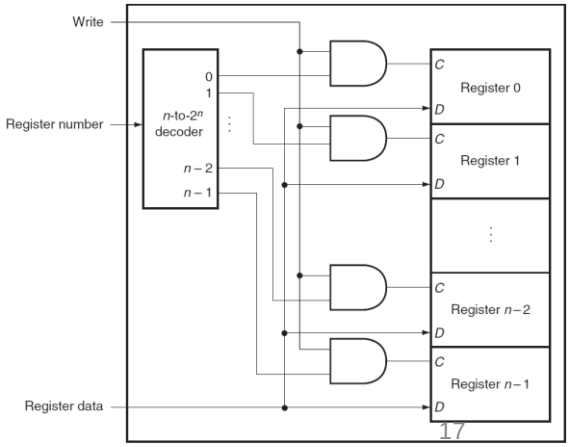
State Element: Register Files

- Consists of 32 registers: (Appendix B.8)
  - Input:
    - three register numbers (5-bit *3)
    - write-in data(32-bit)
    - RegWrite (1-bit)
  - Output:
    - readdata1 (32-bit),
    - readdata2 (32-bit)



Register Read

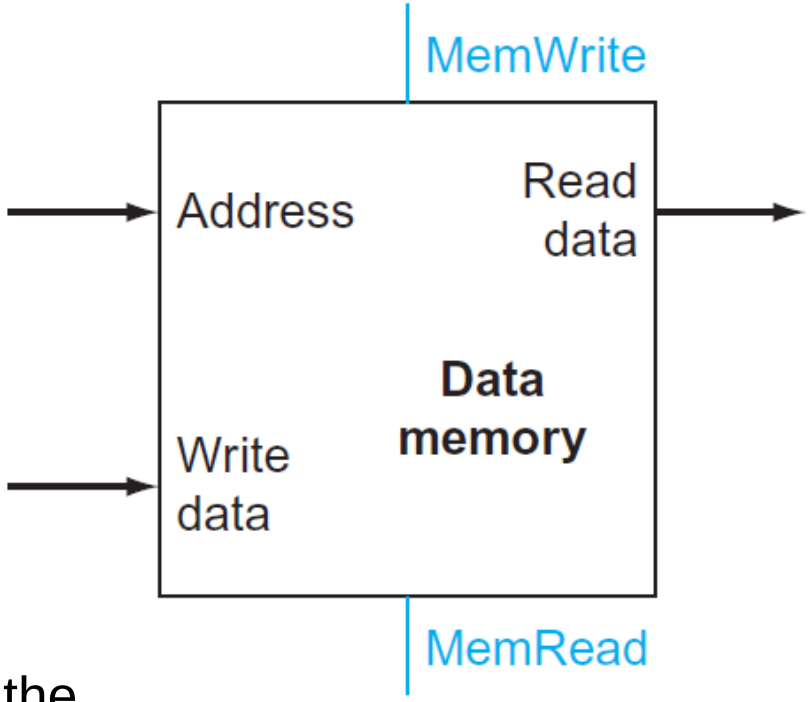




Register Write

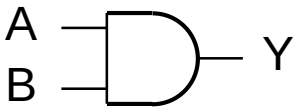
State Element: Data Memory

- Data Memory (Appendix B.9)
  - Input:
    - Address(32-bit)
    - write-in data(32-bit)
    - MemWrite (1-bit)
    - MemRead (1-bit)
    - clock
  - Output:
    - readdata1 (32-bit)
- Word is selected by:
  - Read Enable = 1: Address selects the word to put on Read data bus
  - Write Enable = 1: address selects the memory word to be written via the Write data bus

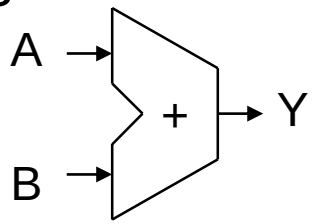


Combinational Elements

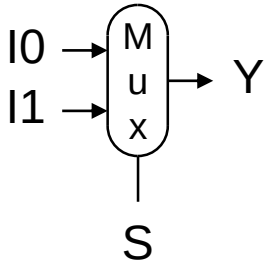
- Basic building blocks of combinational logic elements :
  - AND-gate
    - Y = A & B



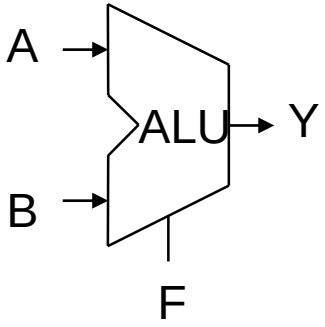
  - Adder
    - Y = A + B



  - Multiplexer
    - Y = S ? I1 : I0

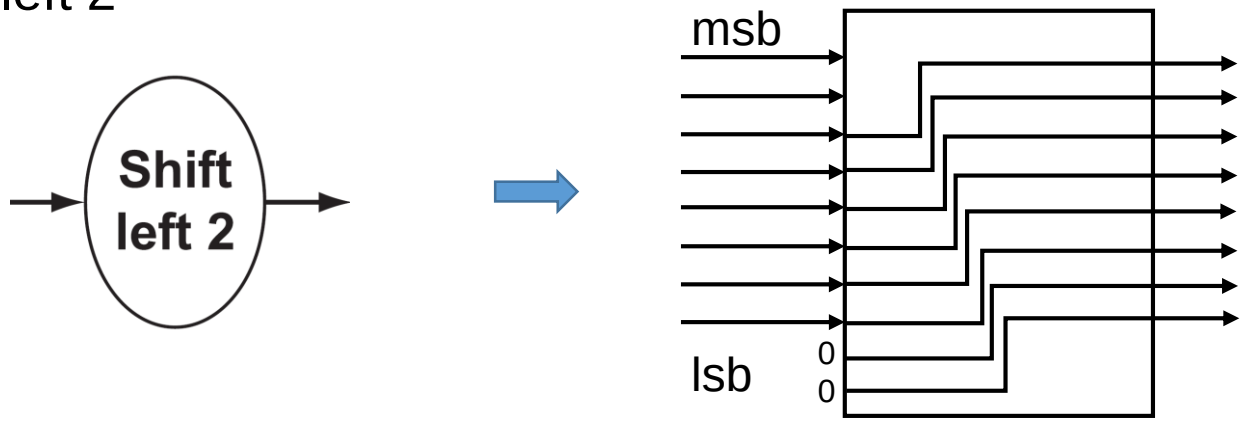


  - Arithmetic/Logic Unit
    - Y = F(A, B)

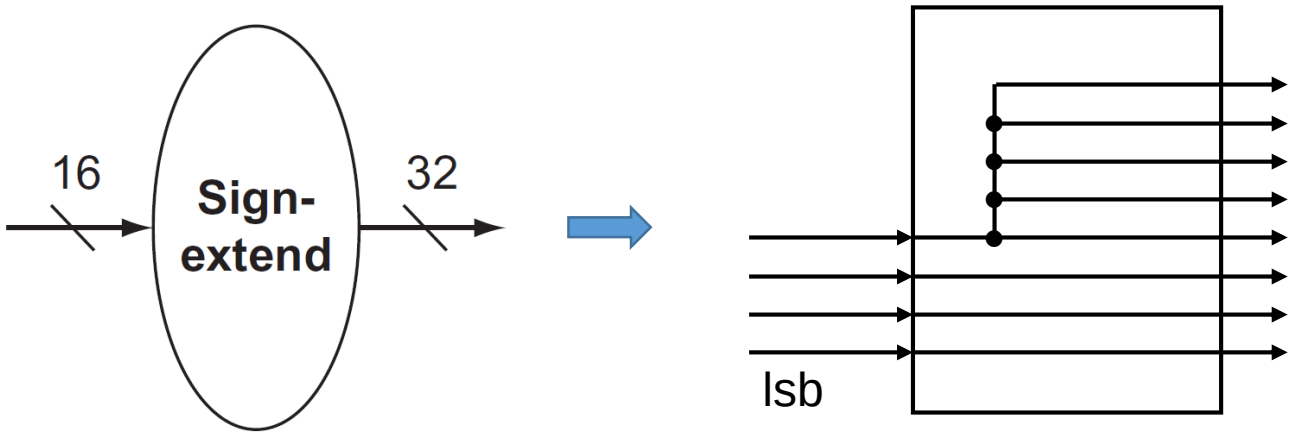


Combinational Elements

- Other useful Basic building blocks:
  - Shift left 2

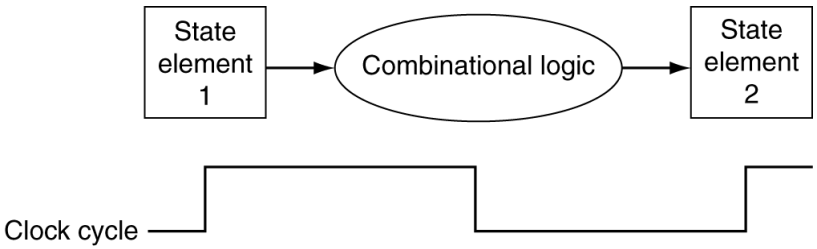


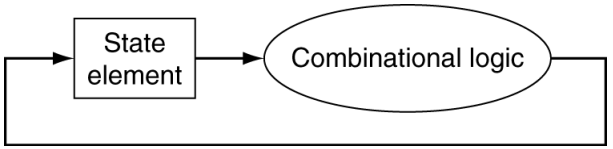
  - Sign-extend



Clocking Methodology

- Defines when signals can be read and when they can be written
  - Edge-triggered clocking: all state changes occur on a clockedge.
- Clock time > the time needed for signals to propagate from SE1through combinatorial element to SE2
- A state element can be read and written in the same clock cyclewithout creating a race, but the clock cycle should be longenough





How to Design a Processor?

1. Analyze instruction set (datapath requirements)
   - The meaning of each instruction is given by the register transfers
   - Datapath must include storage element
   - Datapath must support each register transfer
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting the requirements
4. Assemble the control logic
   - by analyzing implementation of each instruction to determine setting of control points effecting register transfer

Step 3: Datapath Assembly

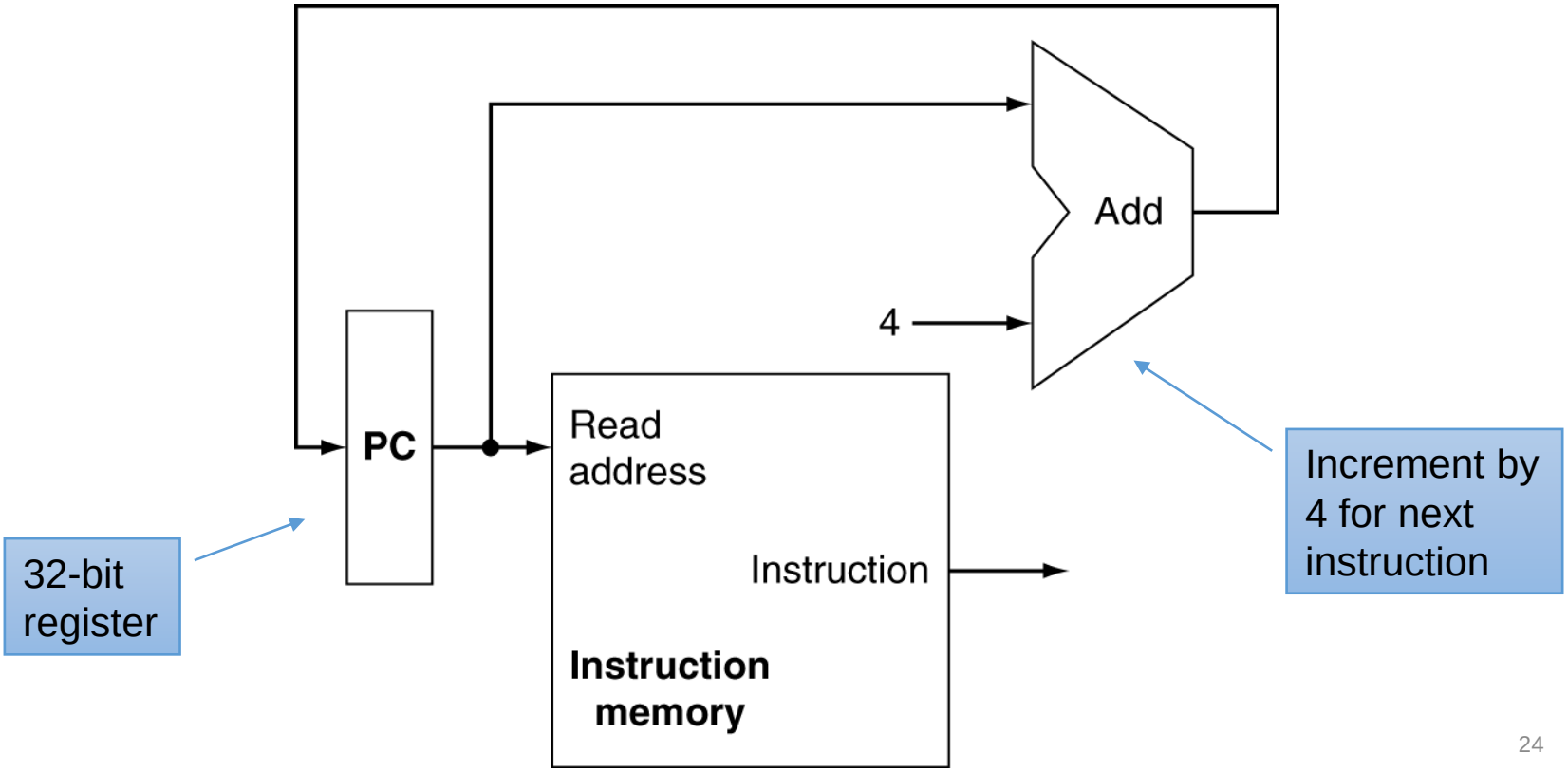
- All start by fetching the instruction, read registers, then use ALU => simplicity and regularity help

```
MEM[PC] = op | rs | rt | rd | shamt | funct
   or   = op | rs | rt |   Imm16
   or   = op | Imm26 (added at the end)
```

Inst	Register transfers
ADD	R[rd] ← R[rs] + R[rt]; PC ← PC + 4
SUB	R[rd] ← R[rs] - R[rt]; PC ← PC + 4
LOAD	R[rt] ← MEM[R[rs] + sign_ext(Imm16)]; PC ← PC + 4
STORE	MEM[R[rs] + sign_ext(Imm16)] ←R[rt]; PC ← PC + 4
BEQ	if(R[rs] == R[rt]) then PC ← PC + 4 + (sign_ext(Imm16)] : 00) else PC ← PC + 4

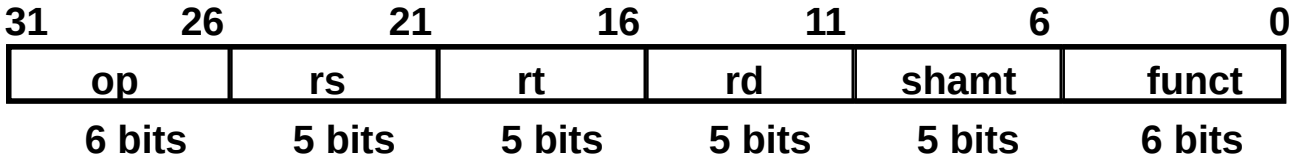
Instruction Fetch

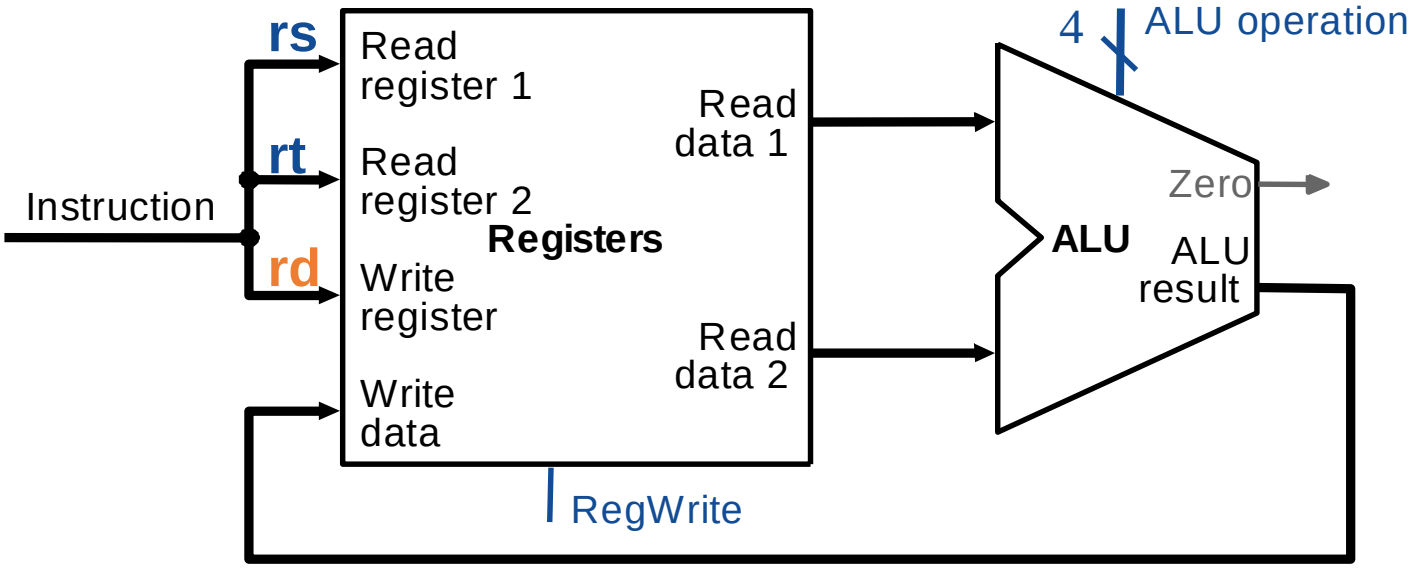
- Instruction fetch unit: common operations
  - Fetch the instruction: mem[PC]
    - Sequential code: PC <- PC + 4
    - Branch and Jump: PC <- “Something else”



R-Format Operations

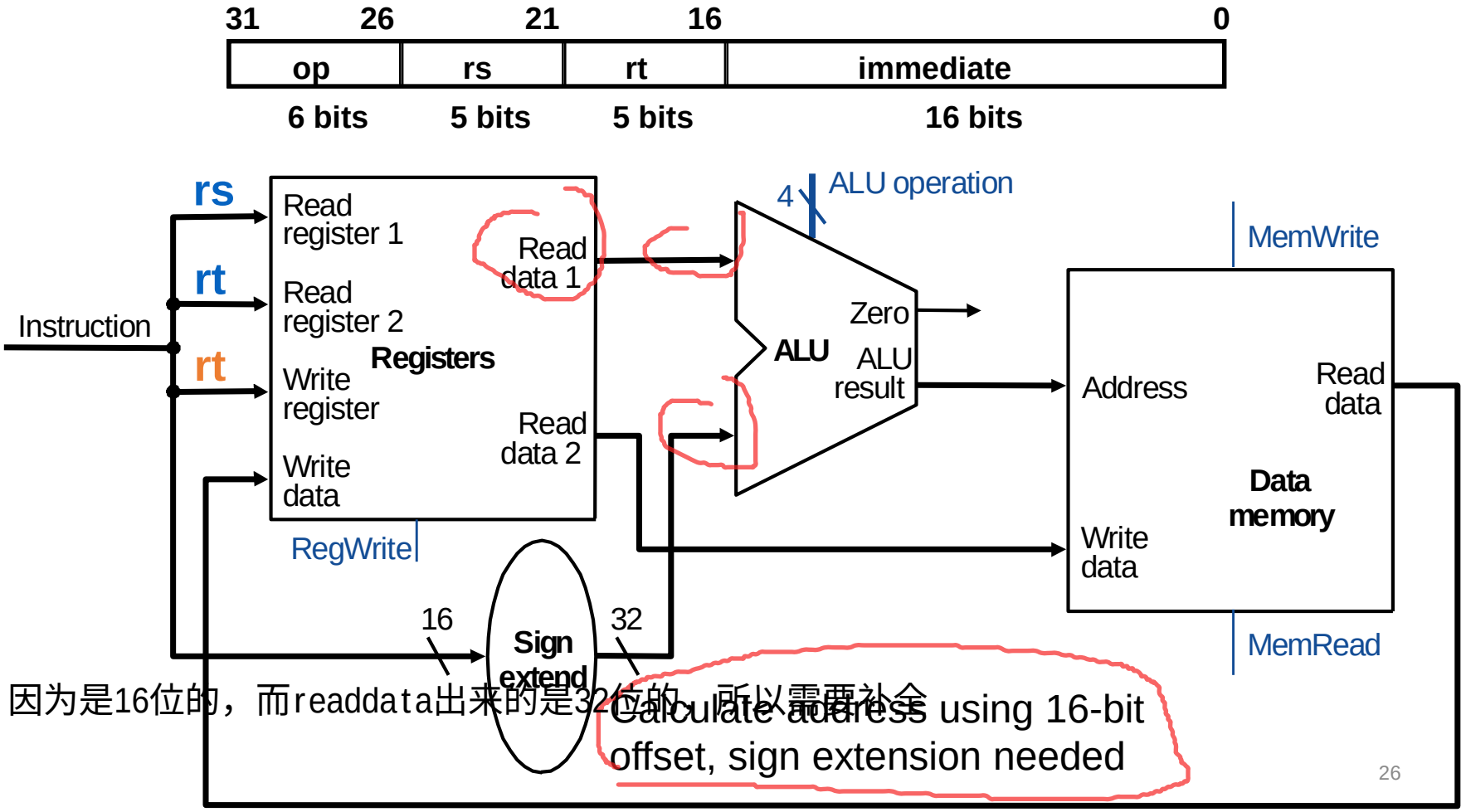
- R[rd] <- R[rs] op R[rt] Ex: add rd, rs, rt
  - rs, rt, rd come from instruction's rs, rt, and rd fields
  - ALU and RegWrite: control logic after decode





Load/Store Operations

- R[rt]←Mem[R[rs]+SignExt[imm16]] Ex: lw rt,rs,imm16
- R[rt]→Mem[R[rs]+SignExt[imm16]] Ex: sw rt,rs,imm16



Branch Operations

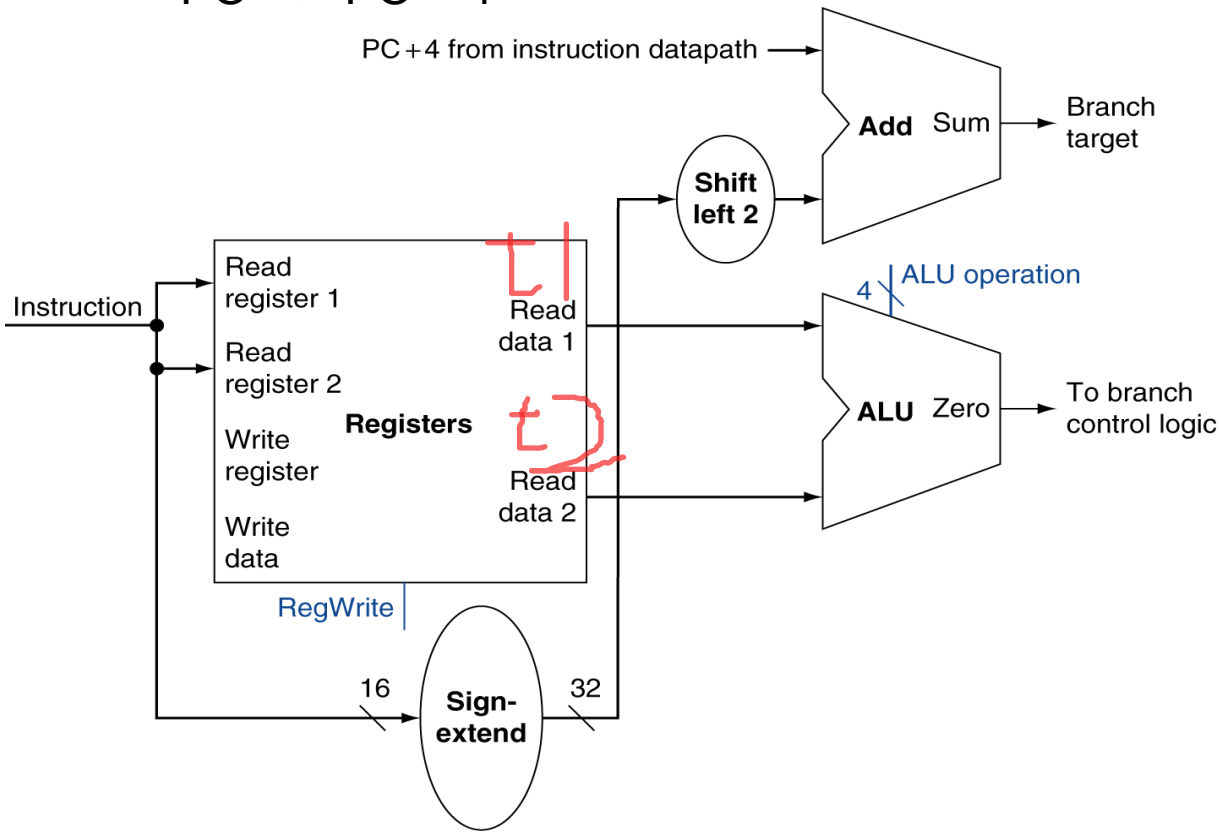
- beq rs, rt, imm16

  mem[PC] Fetch inst. from memory

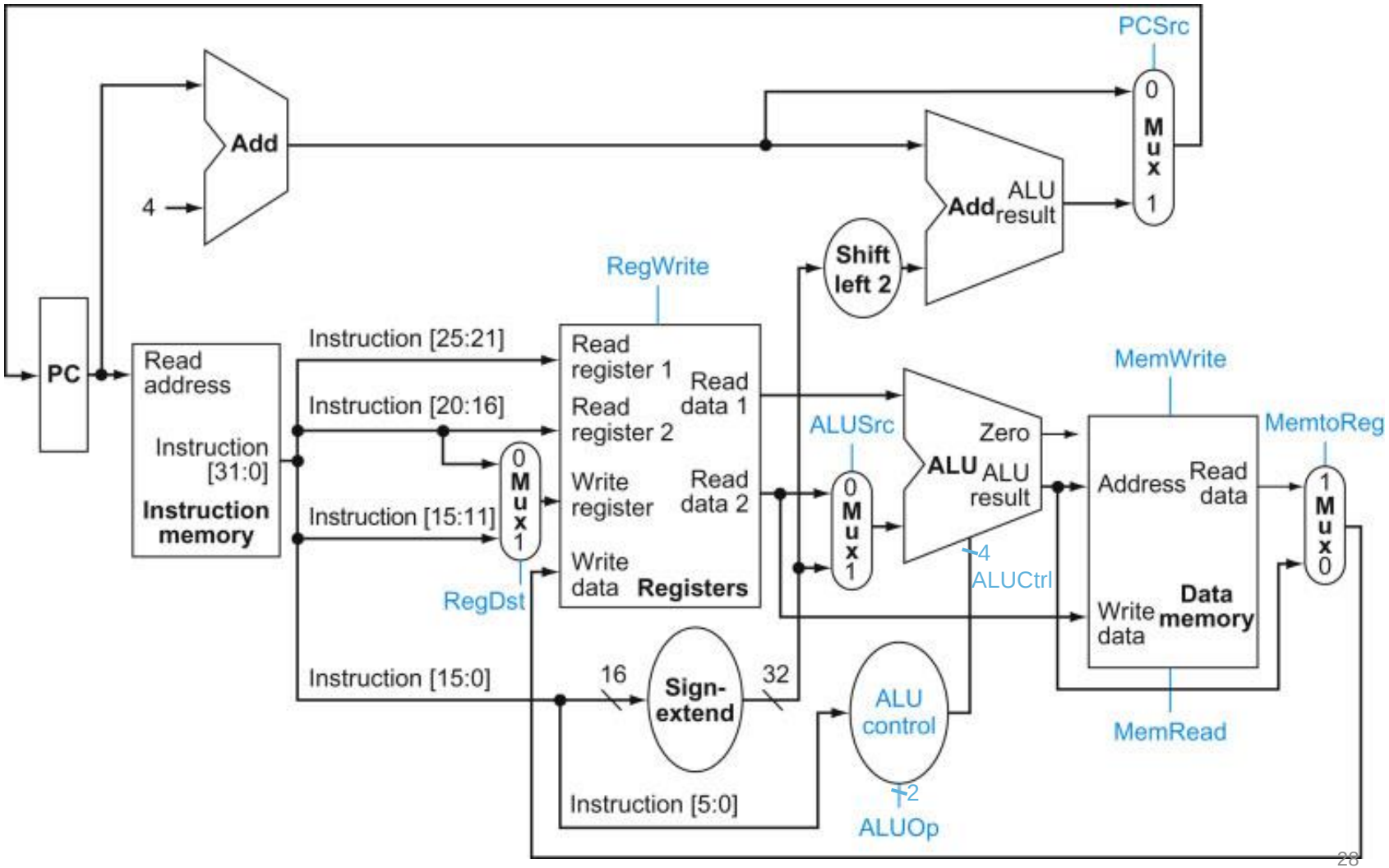
  Zero ← R[rs] == R[rt] subtract and check Zero output

  if (Zero == 1) PC <- PC + 4 + (SignExt(imm16) << 2)

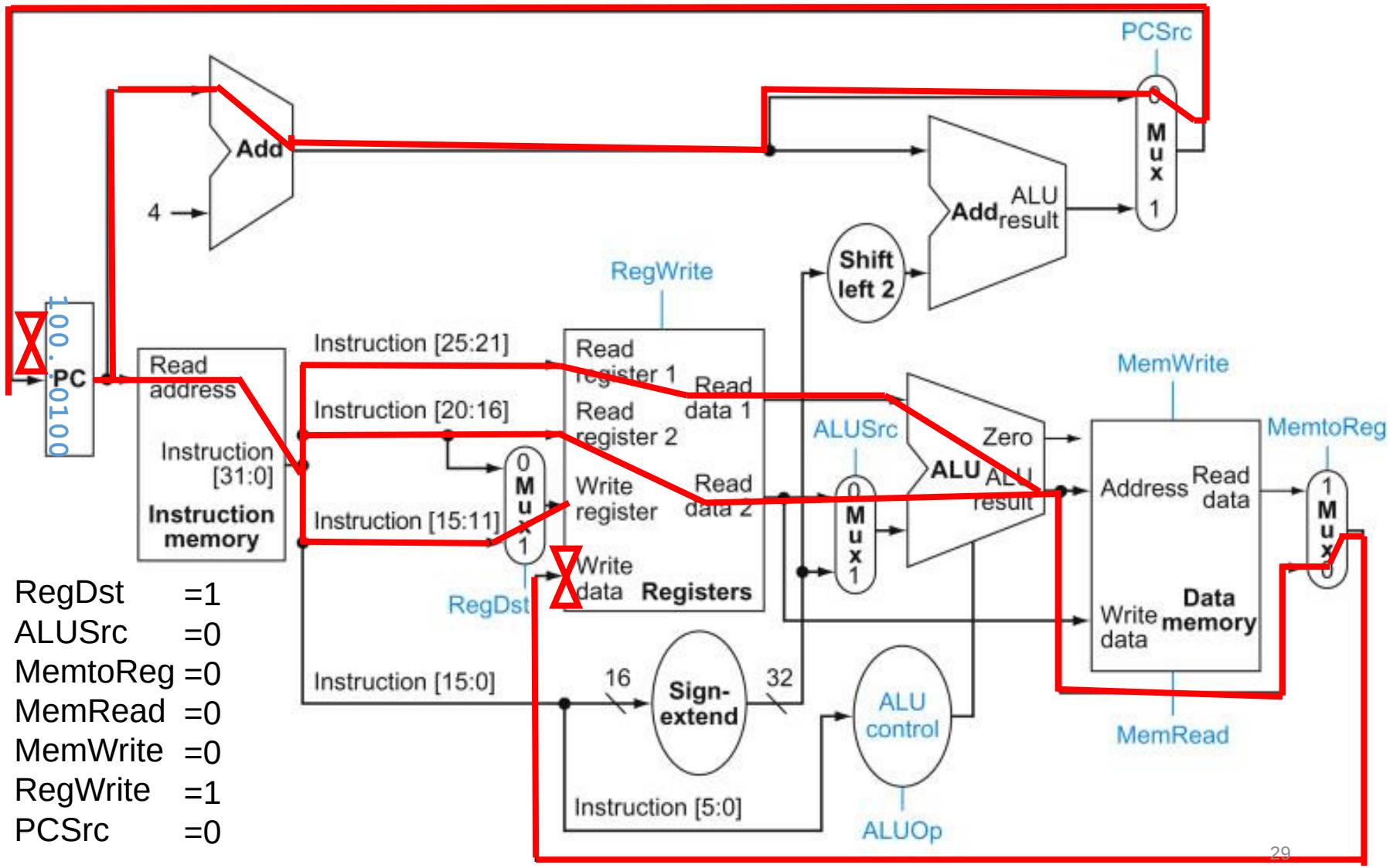
  else PC <- PC + 4



A Single Cycle Datapath



Example: Data Flow during Add

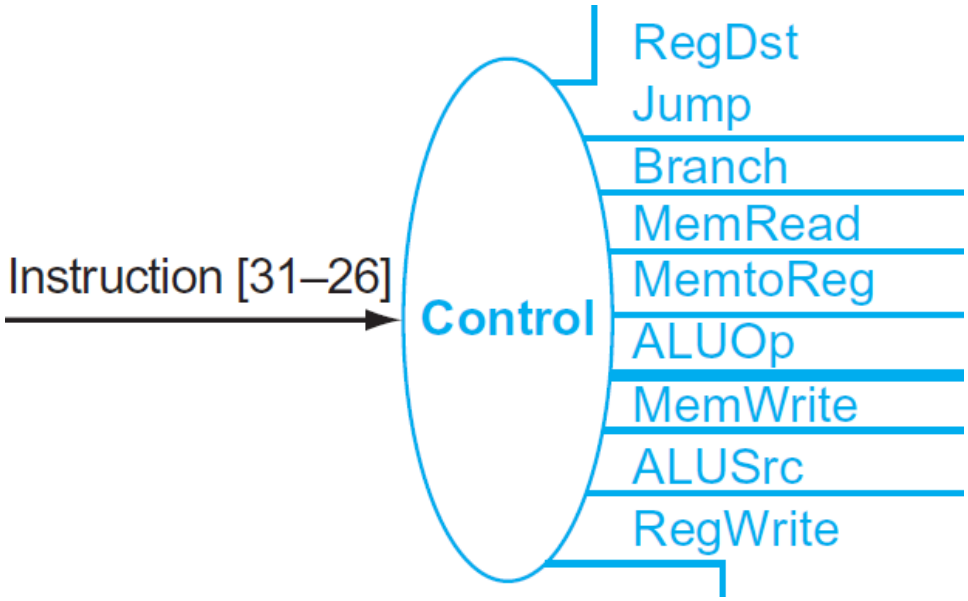


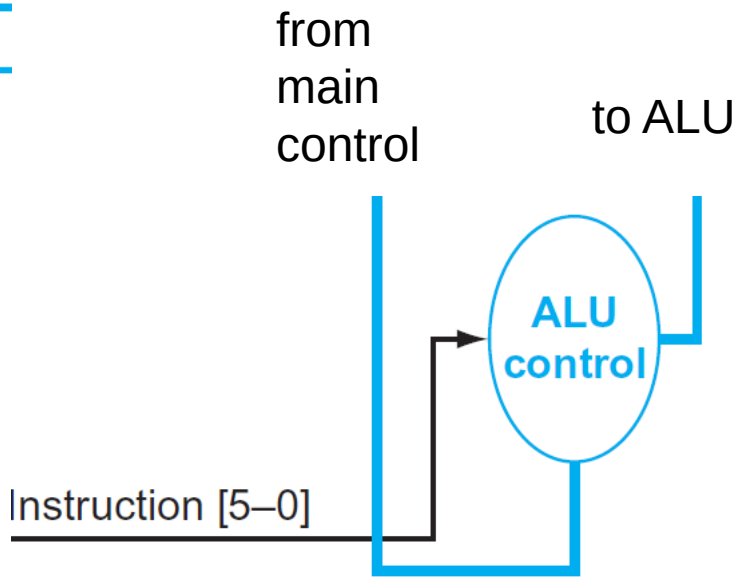
RegDst =1
ALUSrc =0
MemtoReg =0
MemRead =0
MemWrite =0
RegWrite =1
PCSrc =0

How to Design a Processor?

1. Analyze instruction set (datapath requirements)
   - The meaning of each instruction is given by the register transfers
   - Datapath must include storage element
   - Datapath must support each register transfer
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting the requirements
4. Assemble the control logic
   - by analyzing implementation of each instruction to determine setting of control points effecting register transfer

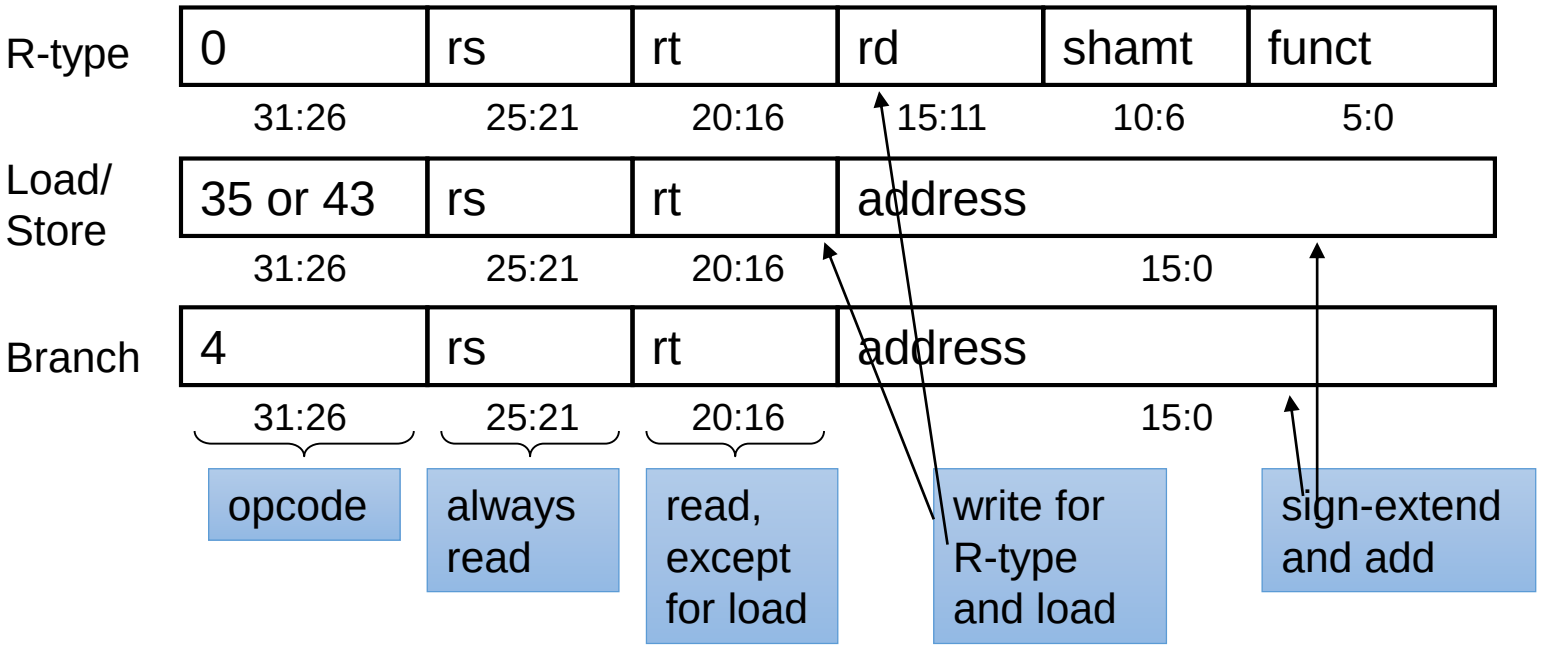
Step 4: Control Points and Signals

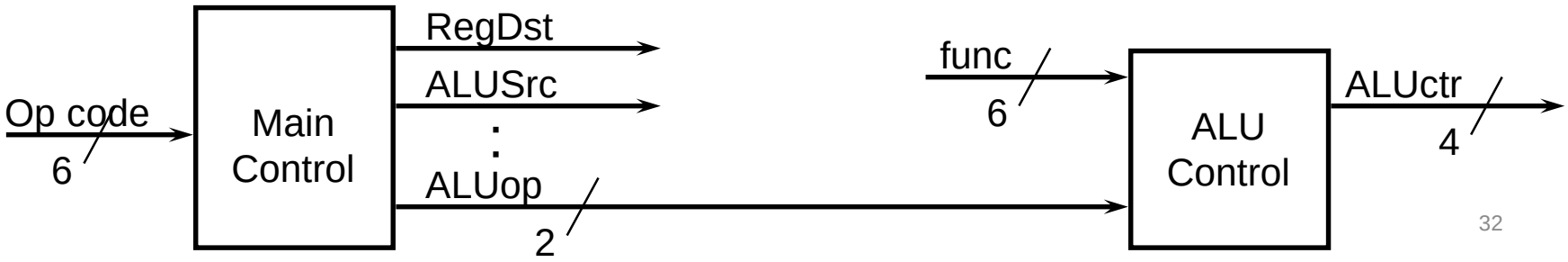




The Main Control Unit

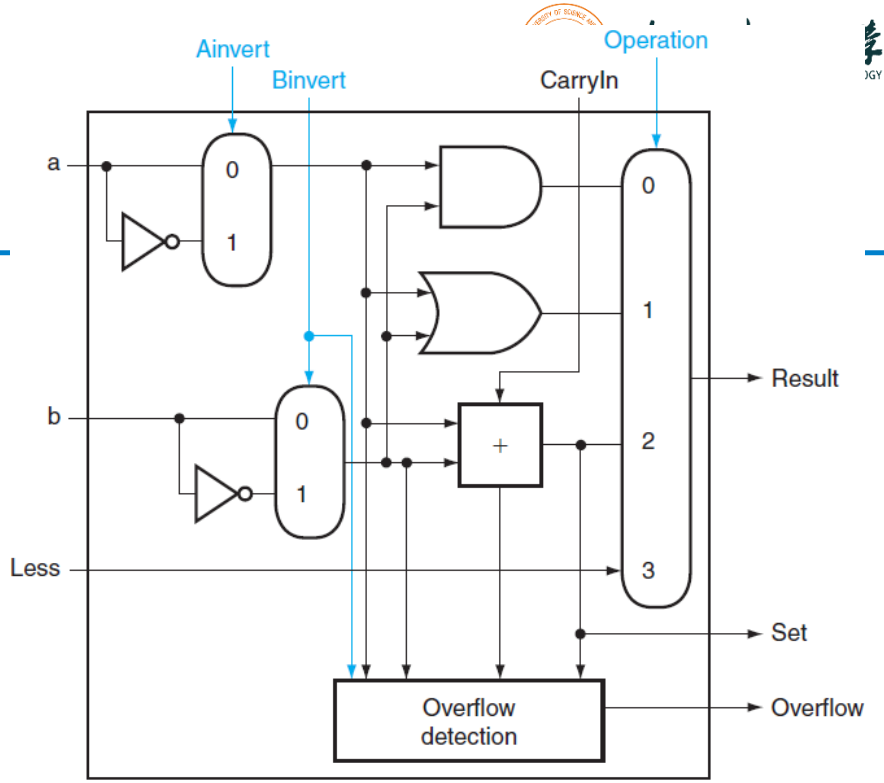
- Control signals derived from instruction





ALU Control

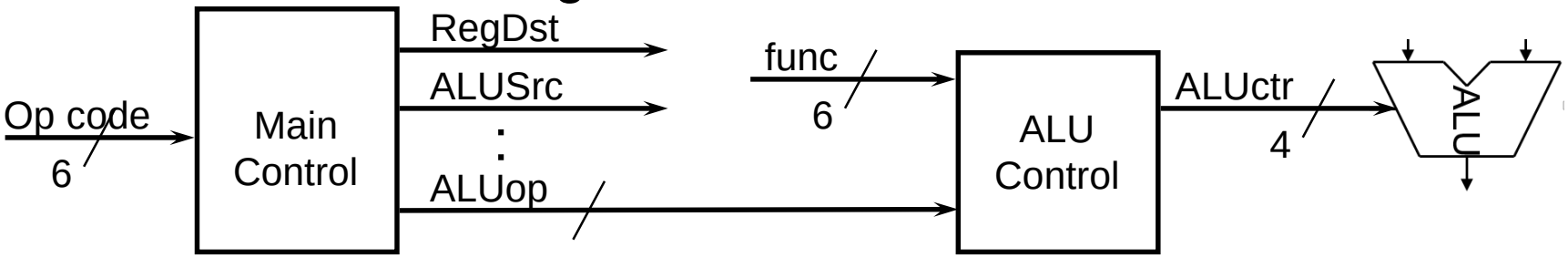
- ALU design Appendix B.5
- ALU used for
  - Load/Store: F = add
  - Branch: F = subtract
  - R-type: F depends on funct field



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

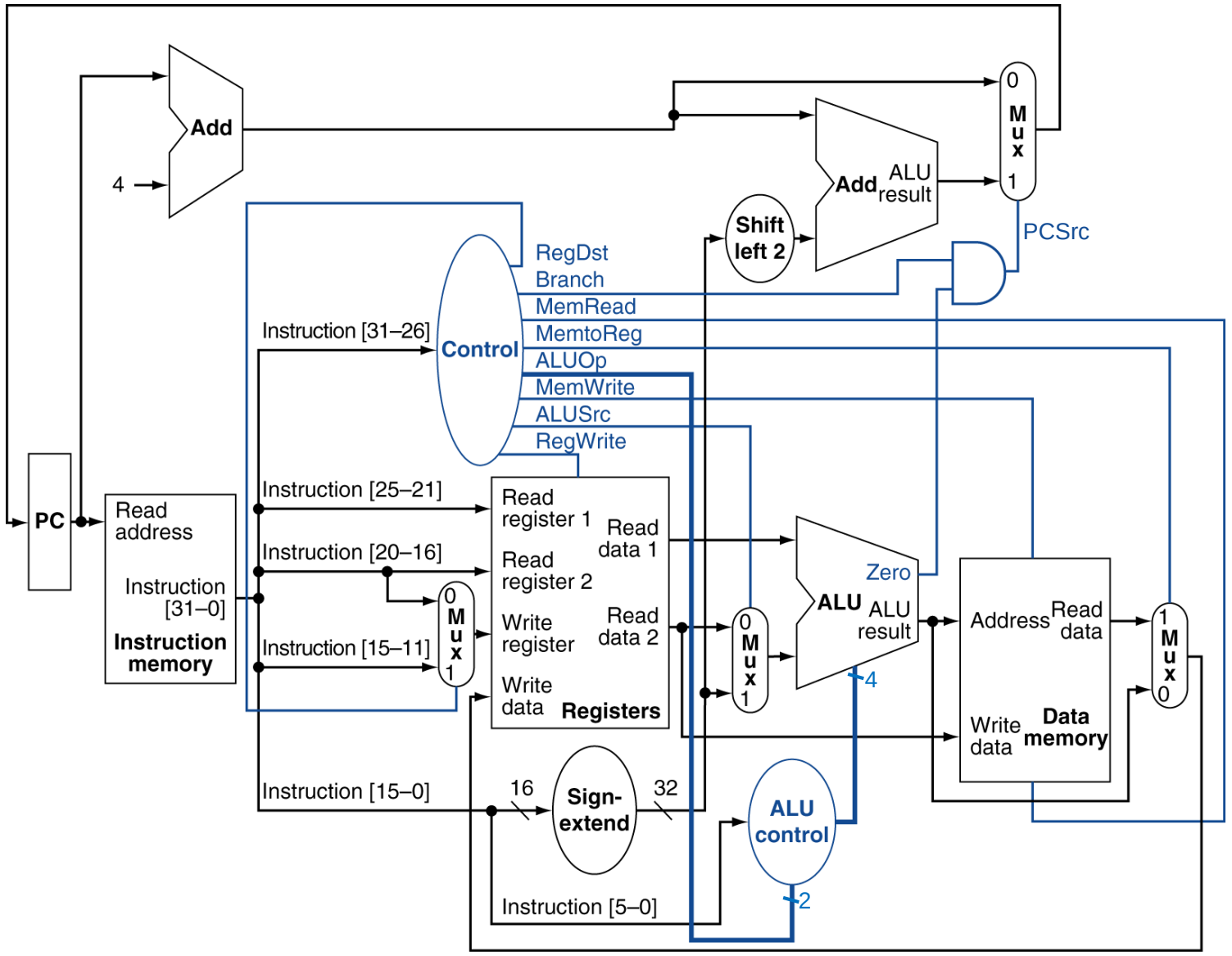
ALU Control

- Assume 2-bit ALUOp derived from opcode
  - Combinational logic derives ALU control

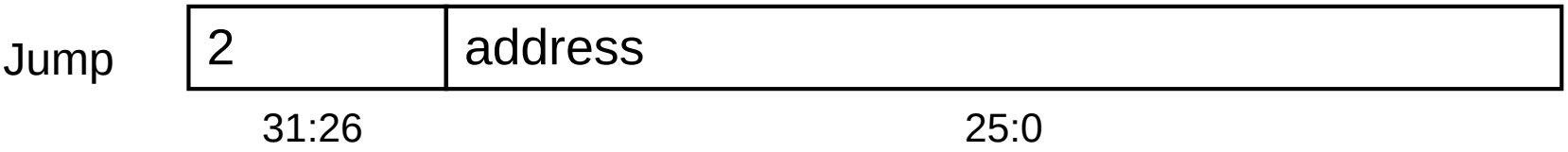


opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		slt	101010	slt	0111

Full Datapath With Control

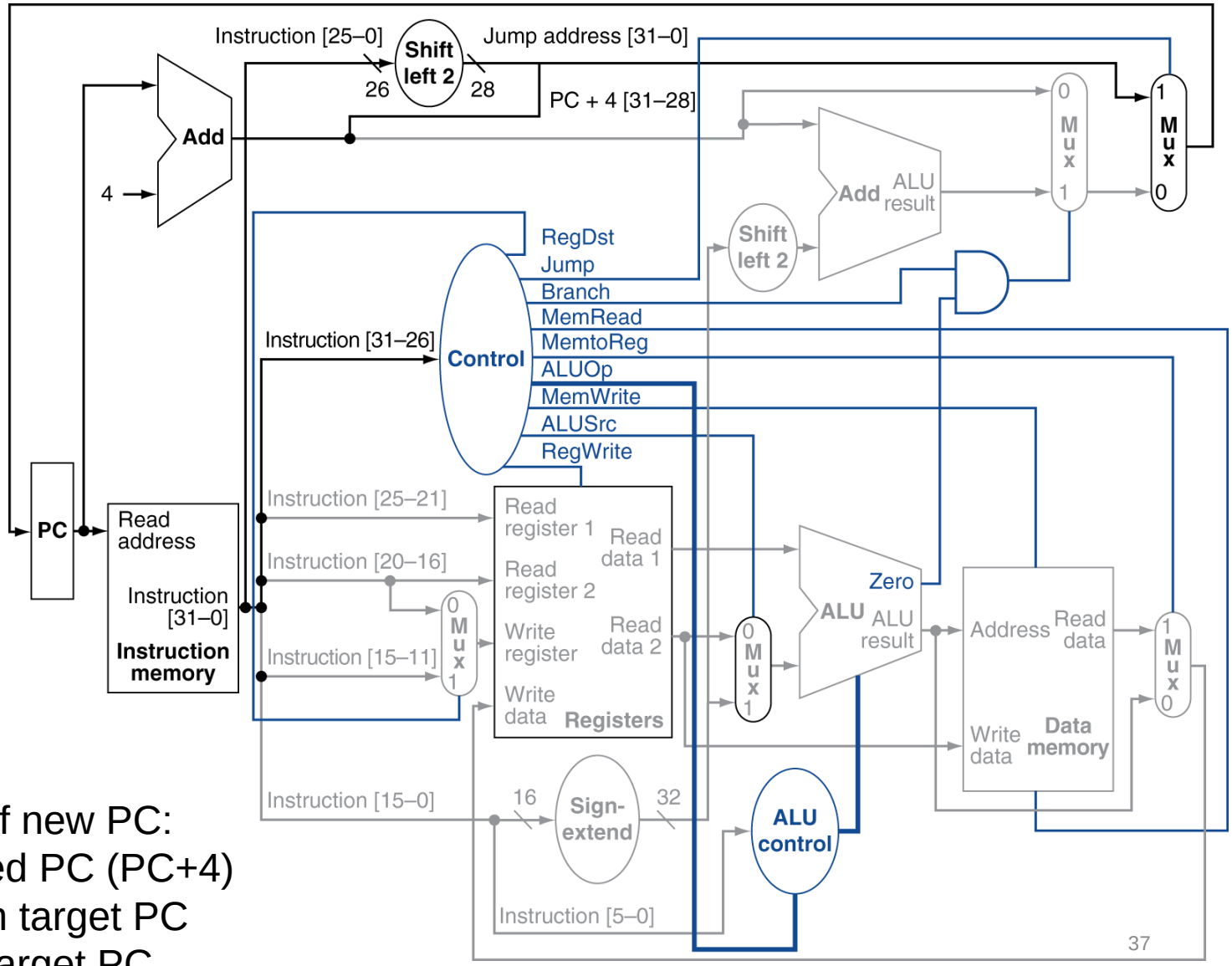


Implementing Jumps



- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
  - 00
- Need an extra control signal decoded from opcode

Full Datapath With Jumps Added



The sources of new PC:

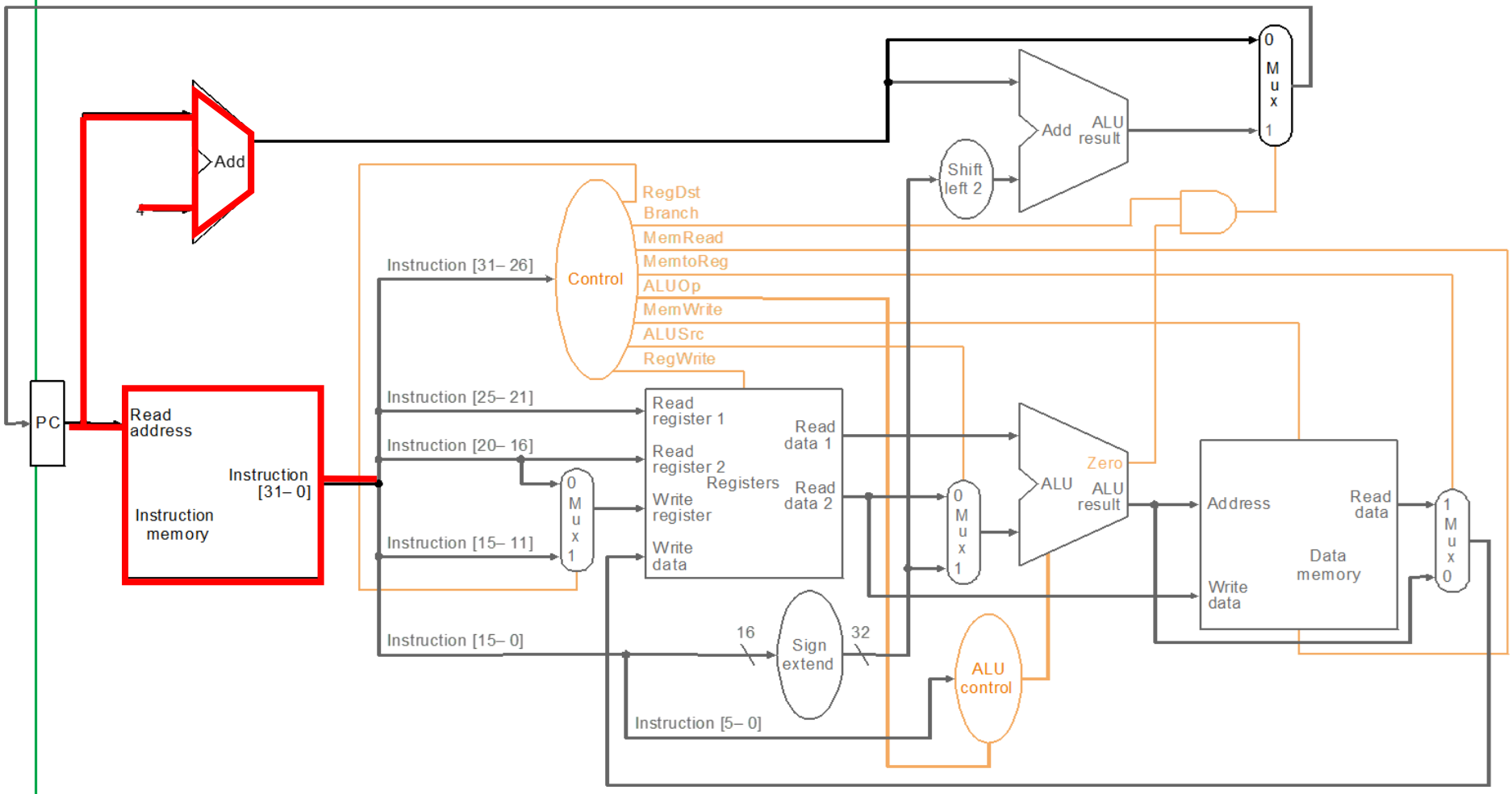
1. Incremented PC (PC+4)
2. The branch target PC
3. The jump target PC

Concluding Remarks

- Single cycle datapath => CPI=1, Clock cycle time long
- MIPS makes control easier
  - Instructions same size
  - Source registers always in same place
  - Immediates same size, location
  - Operations always on registers/immediates
- Building blocks needed for datapath?
- Which units need a clock(meaning sequential)?

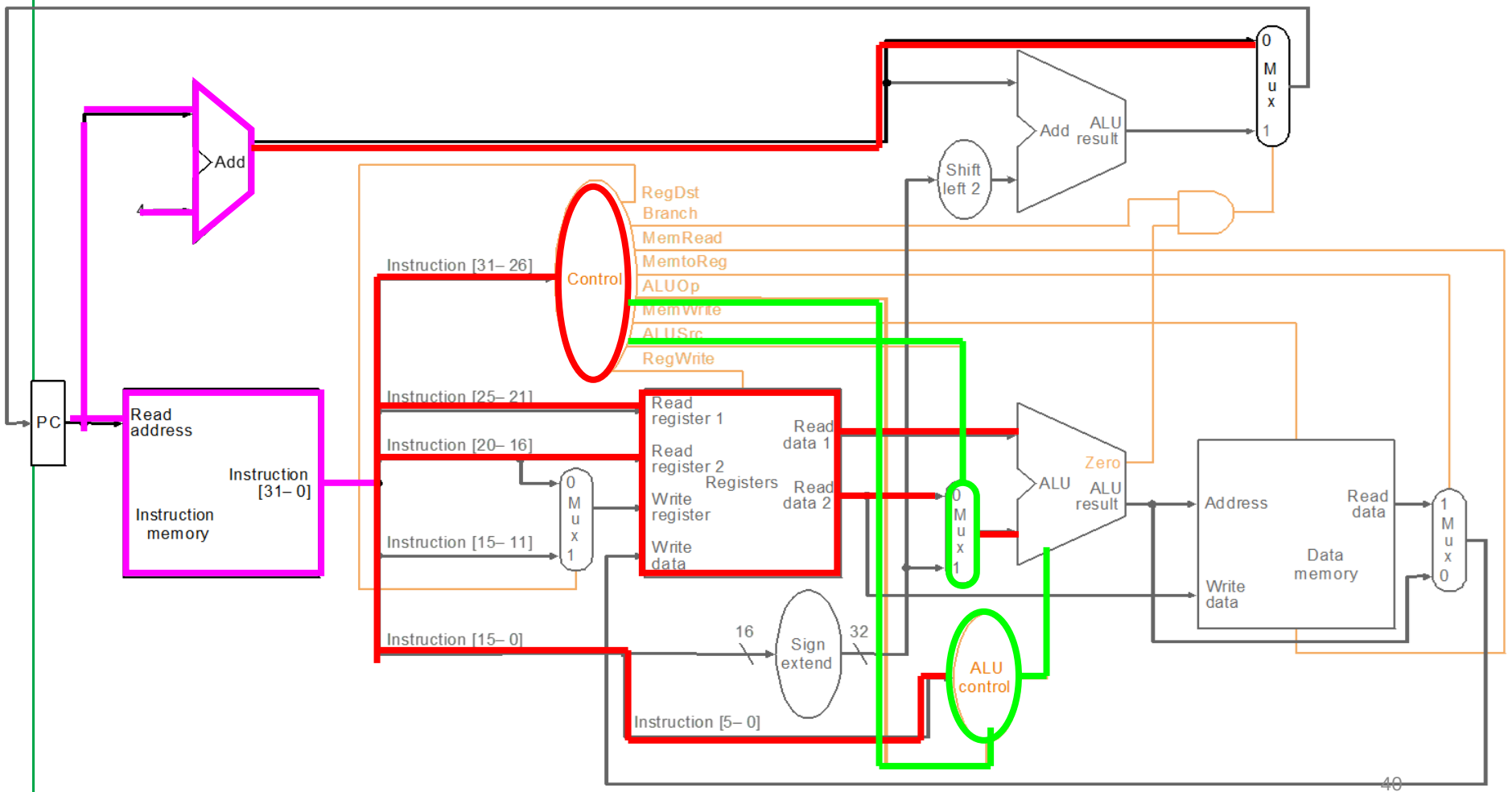
Instruction Fetch at Start of add

- instruction <- mem[PC]; PC + 4



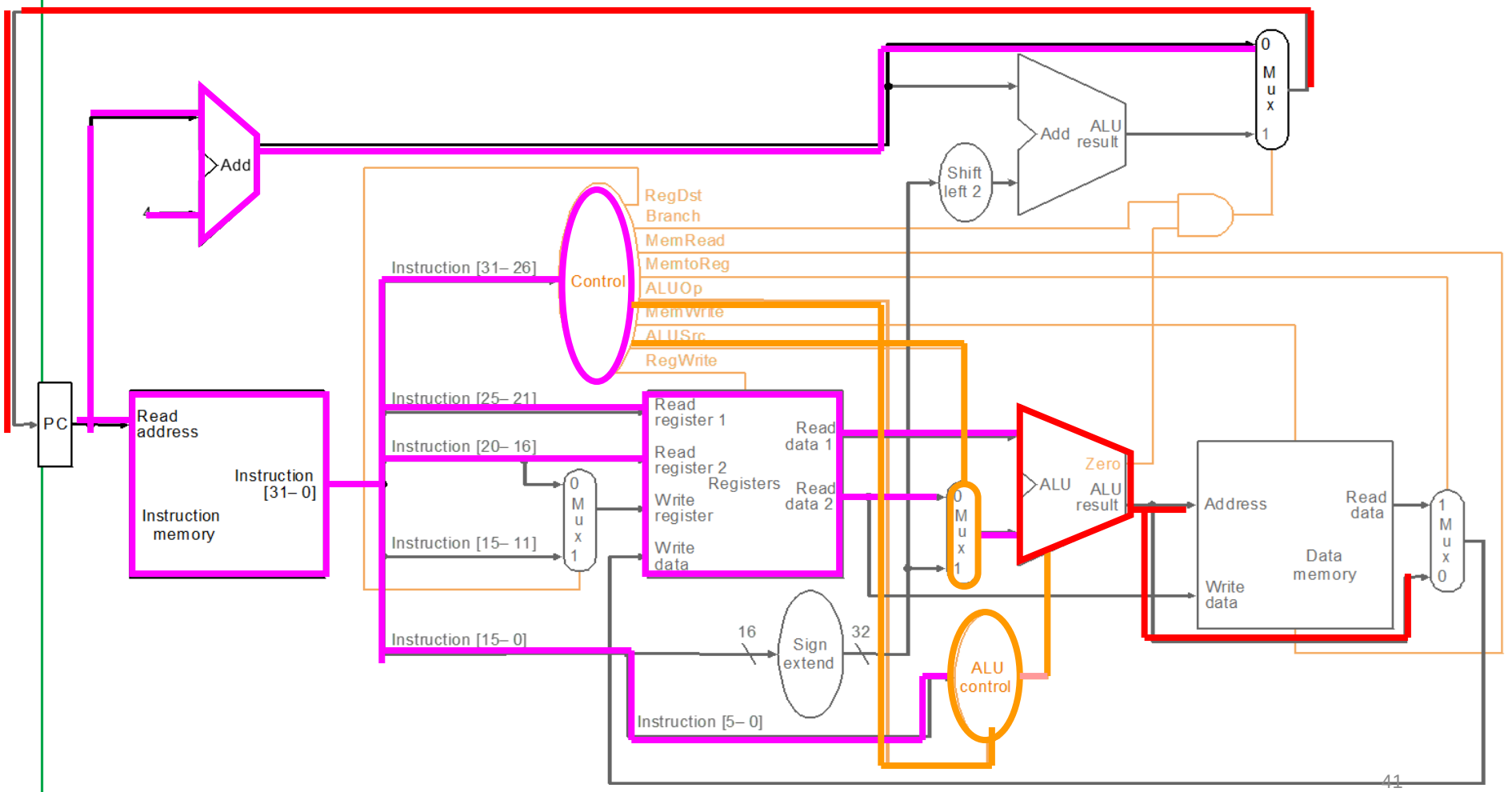
Instruction Decode of add

- Fetch the two operands and decode instruction:



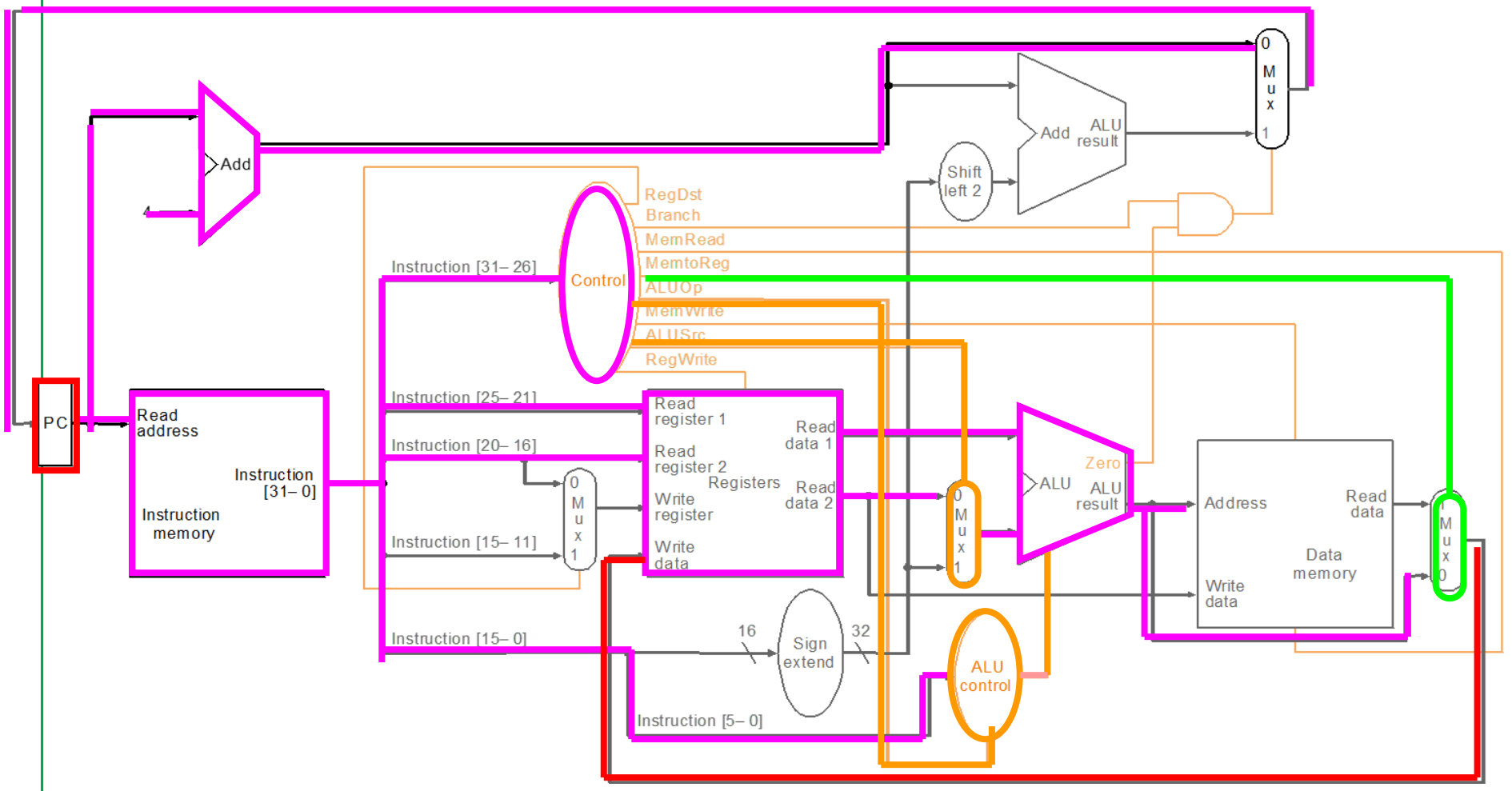
ALU Operation during add

- R[rs] + R[rt]



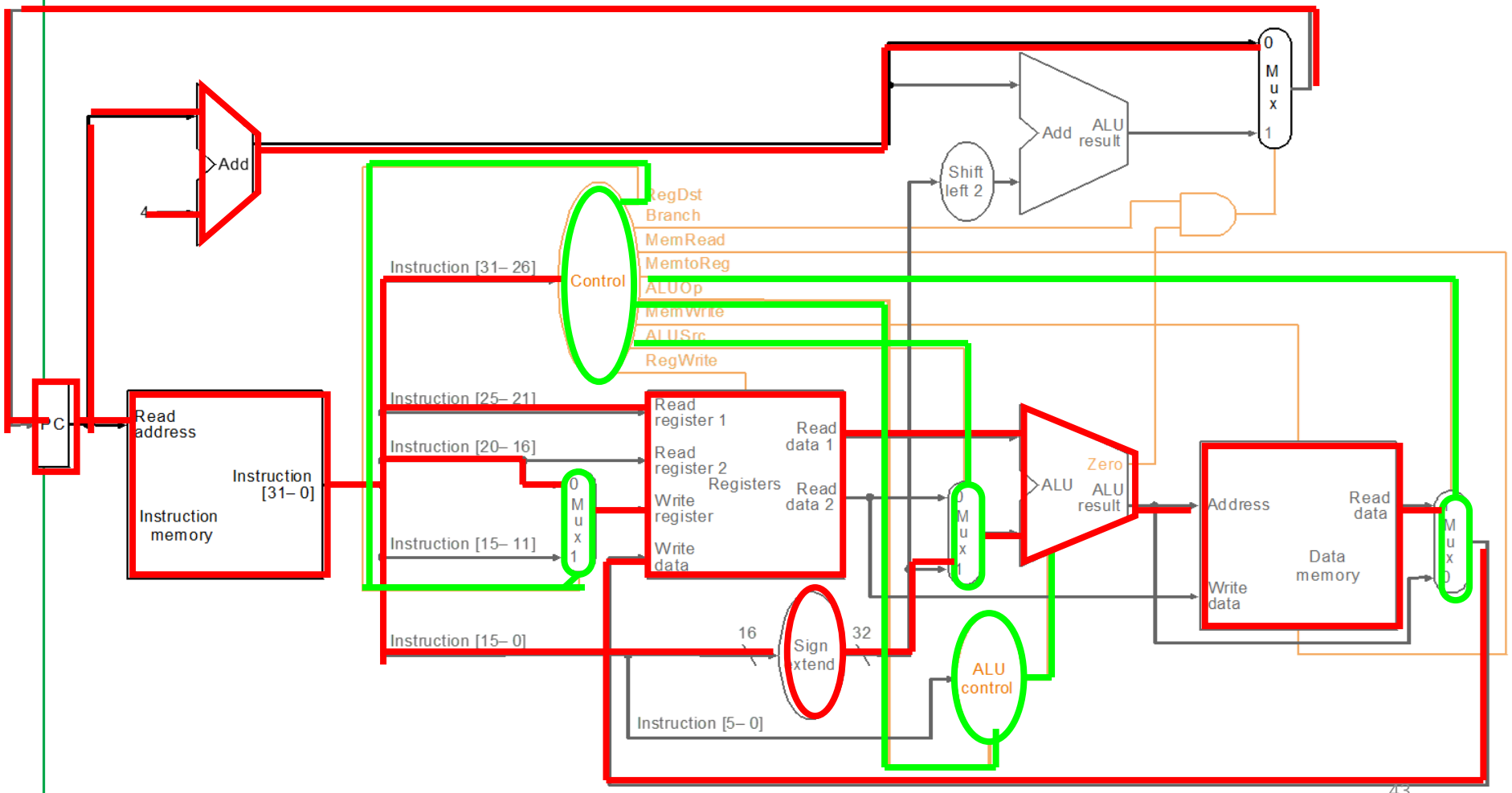
Write Back at the End of add

- R[rd] <- ALU; PC <- PC + 4



Datapath Operation for lw

- R[rt] <- Memory {R[rs] + SignExt[imm16]}



Datapath Operation for beq

if (R[rs]-R[rt]==0) then Zero<-1 else Zero<-0

if (Zero==1) then PC=PC+4+signExt[imm16]*4; else PC = PC + 4