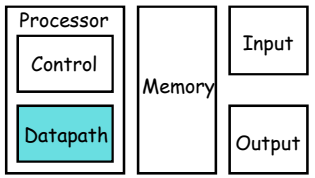
EEM 486: Computer Architecture

Lecture 3

Designing a Single Cycle Datapath

The Big Picture: Where are We Now?

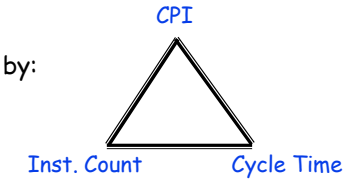
- The Five Classic Components of a Computer



- Today's Topic: Design a Single Cycle Processor

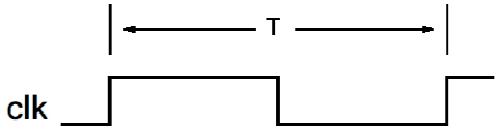
The Big Picture: The Performance Perspective

- Performance of a machine is determined by:
  - Instruction count
  - Clock cycle time
  - Clock cycles per instruction



- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction

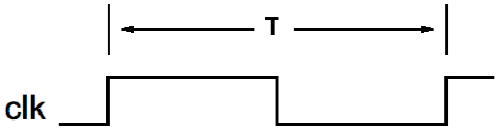
Single-cycle datapath



- All instructions execute in a single cycle of the clock (positive edge to positive edge)
  - Advantage: a great way to learn CPU
  - Unrealistic hardware assumptions, slow clock period

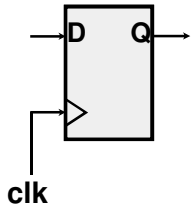
Single cycle data paths: Assumptions

Processor uses synchronous logic design (a "clock")

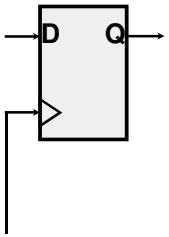


f	T
1 MHz	1 µs
10 MHz	100 ns
100 MHz	10 ns
1 GHz	1 ns

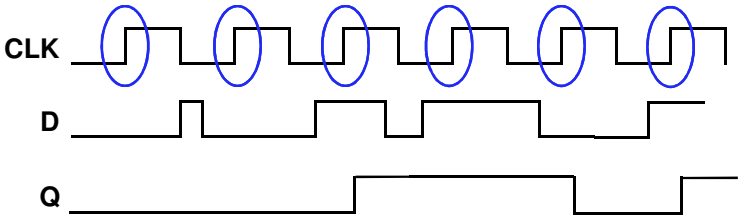
- All state elements act like positive edge-triggered flip flops
- Clocks arrive at all flip flops simultaneously.



Review: Edge-Triggered D Flip Flops



- Value of D is sampled on positive clock edge
- Q outputs sampled value for rest of cycle.

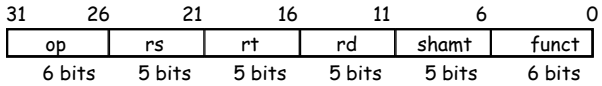


How to Design a Processor: Step-by-Step

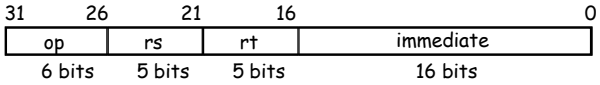
1. Analyze instruction set => datapath requirements
   - Meaning of each instruction is given by the **register transfers**
   - Datapath must include storage element for ISA registers
     - possibly more
   - Datapath must support each register transfer
2. Select set of datapath components and establish 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

The MIPS Instruction Formats

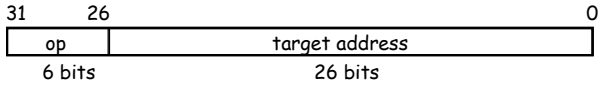
- The three instruction formats:
  - R-type



  - I-type



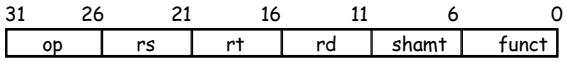
  - J-type



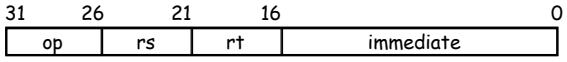
- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: target address of the jump instruction

Step 1a: The MIPS-lite Subset for Today

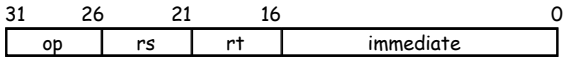
- ADD/SUB
  - addU rd, rs, rt
  - subU rd, rs, rt



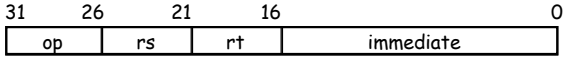
- OR Immediate:
  - ori rt, rs, imm16



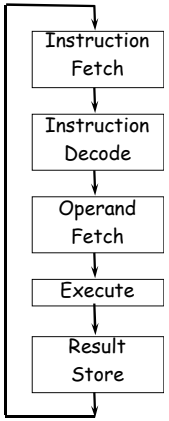
- LOAD/STORE Word
  - lw rt, rs, imm16
  - sw rt, rs, imm16



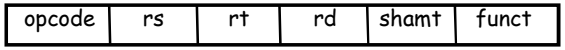
- BRANCH
  - beq rs, rt, imm16



Step 1a: Executing MIPS Instructions



- Fetch next inst from memory
- Get ready for the next instruction



- Decode fields to get a particular instruction
- Retrieve register values (rs, rt)
- Perform the operation (add, sub, or, lw, sw, beq)
- Place the result in a register (rt/rd) / memory, or modify PC

Step 1a: Logical Register Transfers

- RTL gives the meaning of the instructions
- All start by fetching the instruction

op | rs | rt | rd | shamt | funct = MEM[PC]

op | rs | rt | Imm16 = MEM[PC]

inst	Register Transfers	
ADDU	R[rd] <- R[rs] + R[rt];	PC <- PC + 4
SUBU	R[rd] <- R[rs] - R[rt];	PC <- PC + 4
ORi	R[rt] <- R[rs] | zero_ext(Imm16);	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

Step 1: Requirements of the Instruction Set

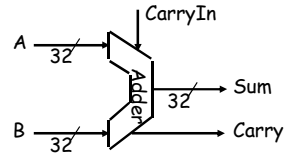
- Memory
  - instruction & data
- Registers (32 x 32)
  - read RS
  - read RT
  - Write RT or RD
- PC
- Extender
- Add and Sub registers or register and extended immediate
- Logical Or of a register and extended immediate
- Add 4 or extended immediate to PC

Step 2: Components of the Datapath

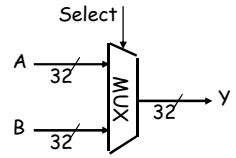
- Combinational Elements
- Storage Elements
  - Clocking methodology

Combinational Logic Elements (Basic Building Blocks)

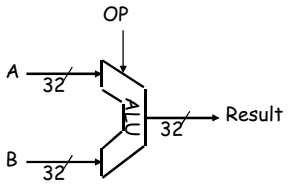
- Adder



- MUX

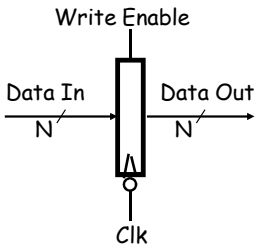


- ALU



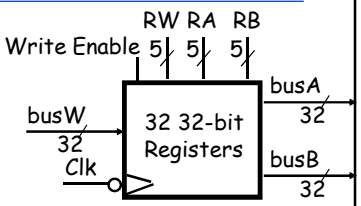
Storage Element: Register (Basic Building Block)

- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
  - Write Enable:
    - Negated (0): Data Out will not change
    - Asserted (1): Data Out becomes Data In



Storage Element: Register File

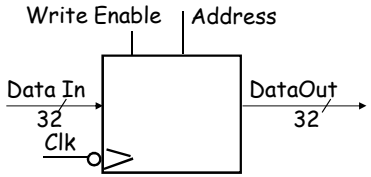
- Register File consists of 32 registers:
  - Two 32-bit output busses:
    busA and busB
  - One 32-bit input bus: busW



- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1

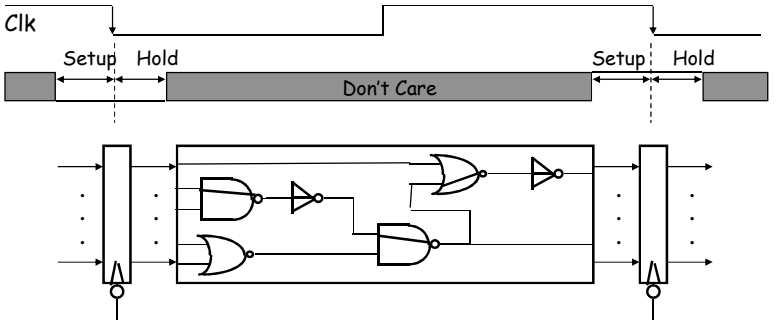
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid => busA or busB valid after "access time."

Storage Element: Idealized Memory



- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is selected by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - Address valid => Data Out valid after "access time."

Clocking Methodology



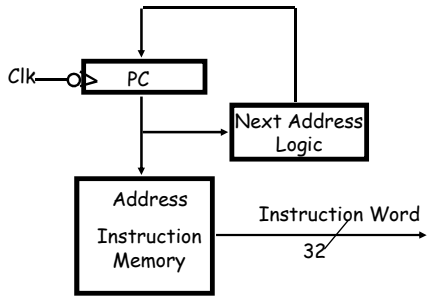
- All storage elements are clocked by the same clock edge
- Being physical devices, flip-flops (FF) and combinational logic have some delays
  - Gates: delay from input change to output change
  - Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF, and we have the usual clock-to-Q delay
- "Critical path" (longest path through logic) determines length of clock period

Step 3: Assemble Datapath Meeting Requirements

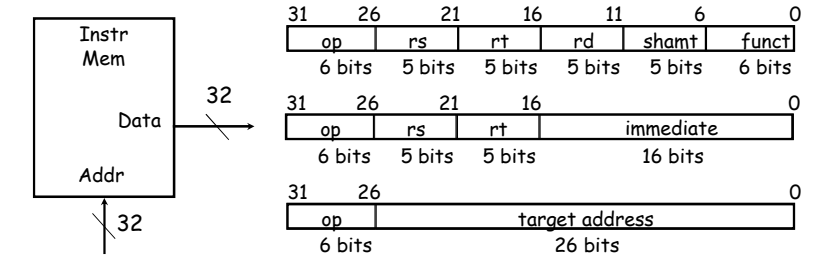
- Register Transfer Requirements
  - ⇒ Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation

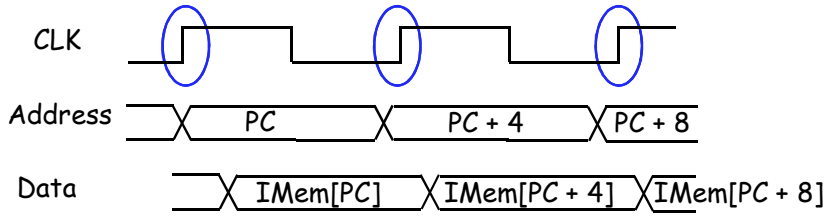
3a: Overview of the Instruction Fetch Unit

- The common RTL operations
  - Fetch the Instruction: mem[PC]
    - PC == Program Counter, points to next instruction
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump: PC <- "something else"



Straight-line Instruction Fetch

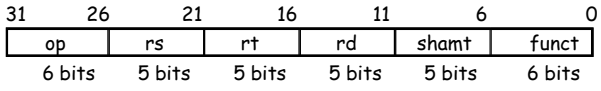


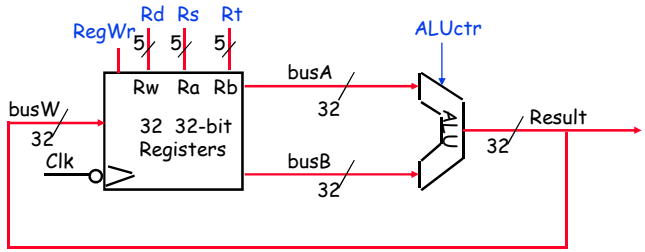


Why +4 and not +1?

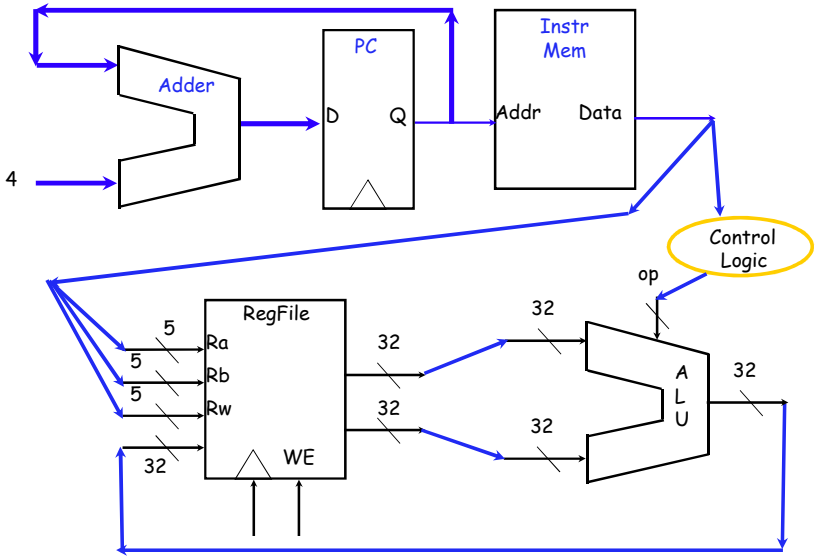
3b: Add & Subtract

- R[rd] <- R[rs] op R[rt] Example: addU rd, rs, rt
  - Ra, Rb, and Rw come from instruction's rs, rt, and rd fields
  - ALUctr and RegWr: control logic produces after decoding the instruction

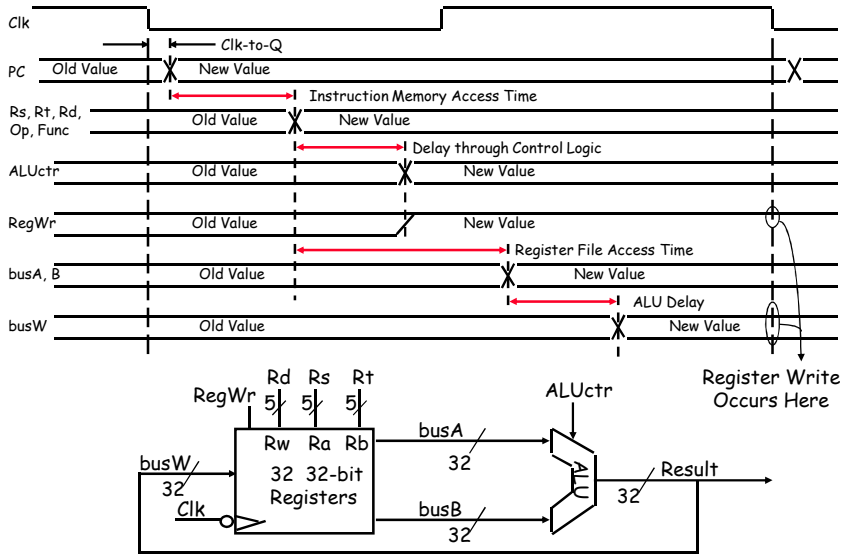




How data flows after posedge

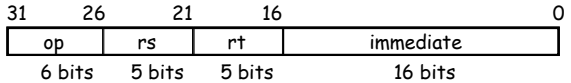


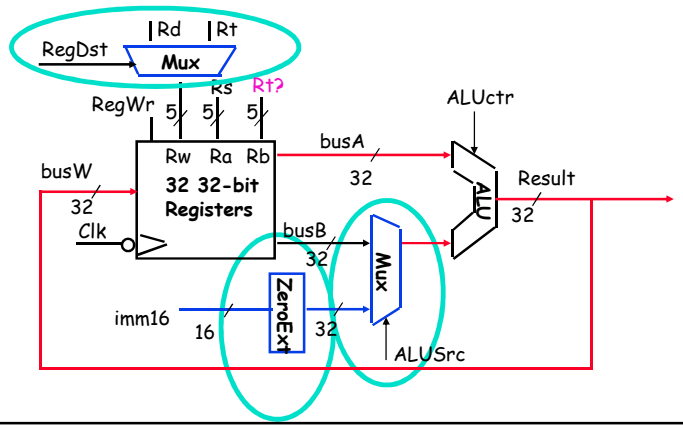
Register-Register Timing: One complete cycle



3c: Logical Operations with Immediate

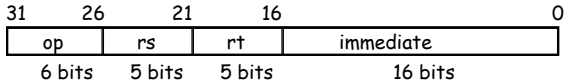
- R[rt] <- R[rs] op ZeroExt[imm16]

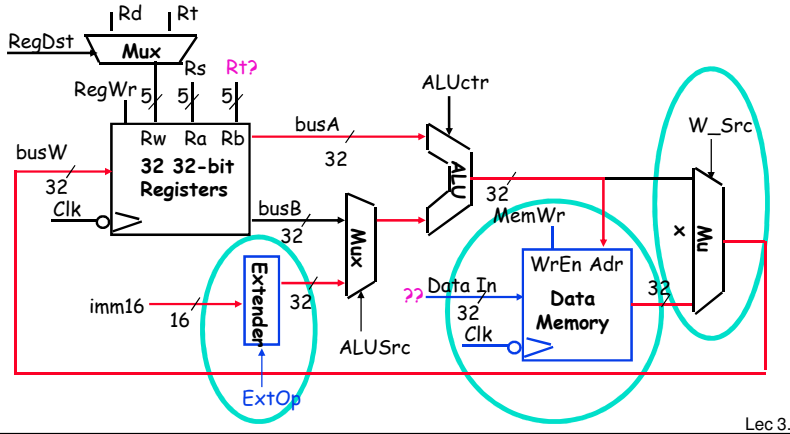




3d: Load Operations

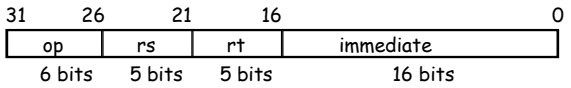
- R[rt] <- Mem[R[rs] + SignExt[imm16]] E.g.: lw rt, rs, imm16

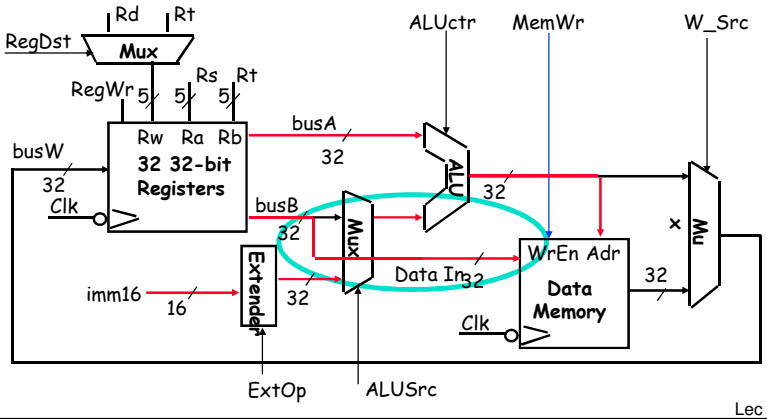




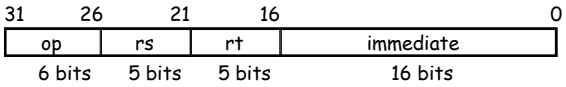
3e: Store Operations

- Mem[R[rs] + SignExt[imm16]] <- R[rt] E.g.: sw rt, rs, imm16





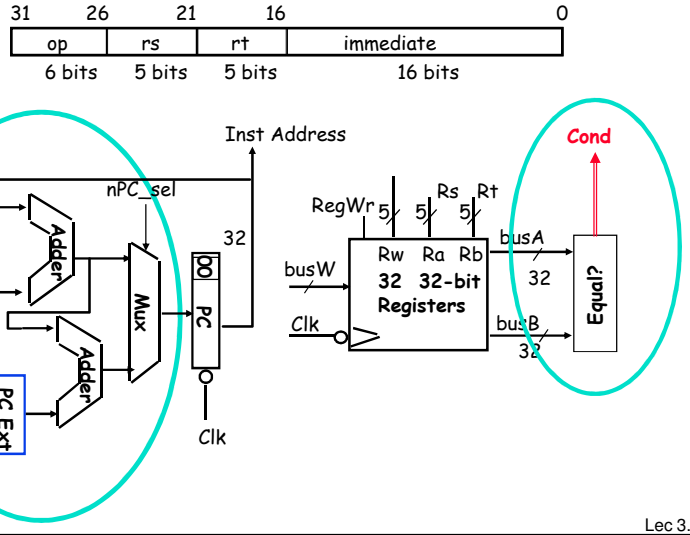
3f: The Branch Instruction



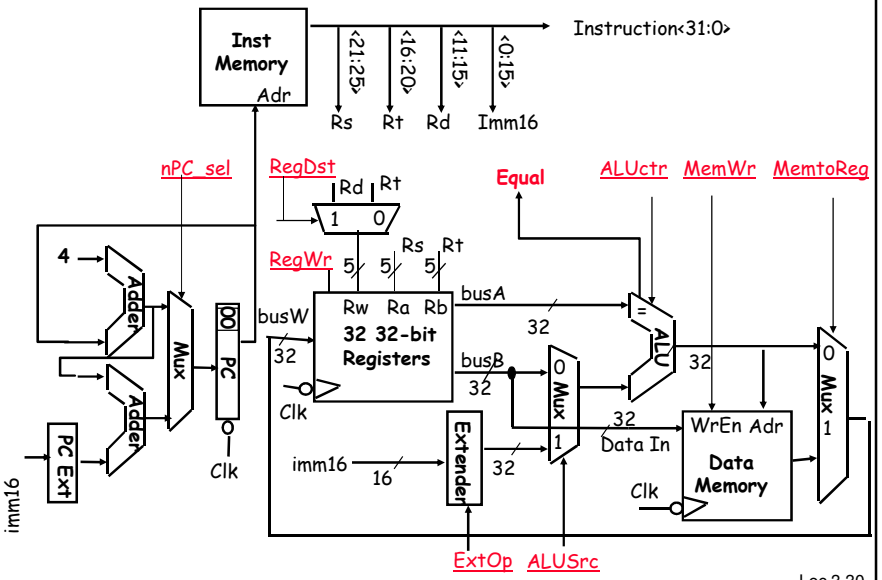
- beq rs, rt, imm16
  - mem[PC] Fetch the instruction from memory
  - Equal <- R[rs] == R[rt] Calculate the branch condition
  - if (Equal) Calculate the next instruction's address
    - PC <- PC + 4 + (SignExt(imm16) x 4)
  - else
    - PC <- PC + 4

Datapath for Branch Operations

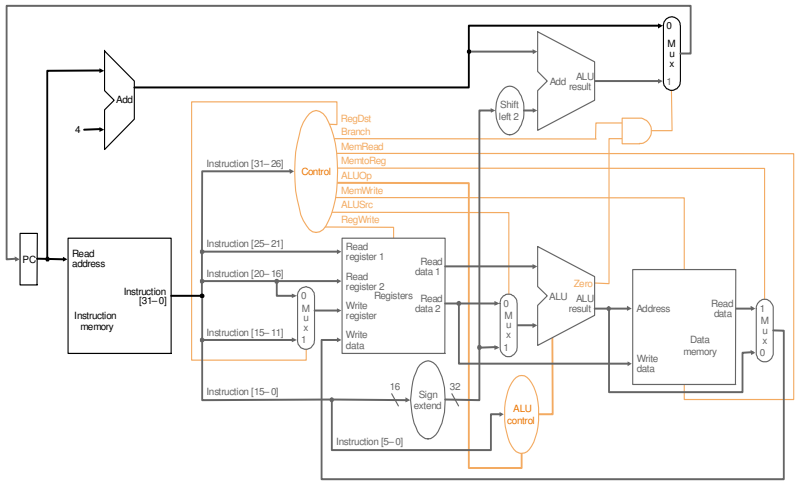
- beq rs, rt, imm16 Datapath generates condition (equal)



Putting it All Together: A Single Cycle Datapath



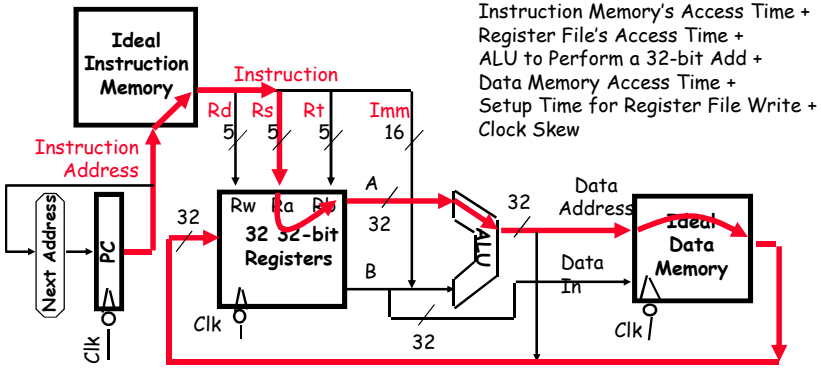
A Single Cycle Datapath



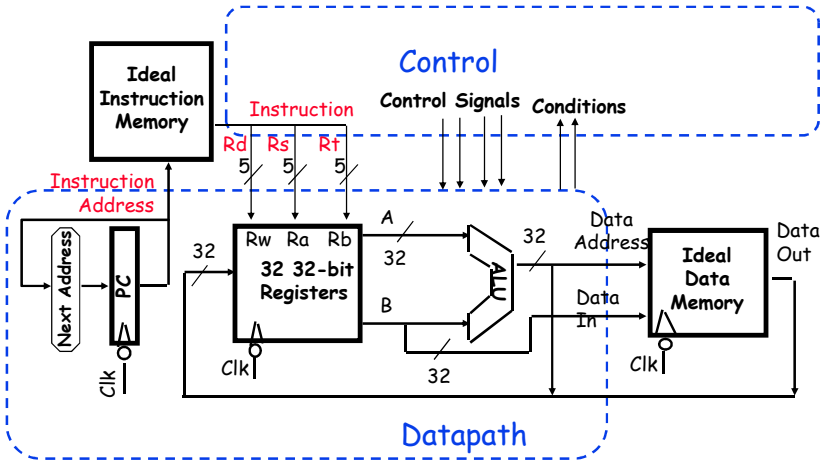
An Abstract View of the Critical Path

- Register file and ideal memory:
  - The CLK input is a factor ONLY during write operation
  - During read operation, behave as combinational logic:
    - Address valid => Output valid after "access time."

Critical Path (Load Operation) =
PC's Clk-to-Q +
Instruction Memory's Access Time +
Register File's Access Time +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Setup Time for Register File Write +
Clock Skew



An Abstract View of the Implementation



Steps 4 & 5: Implement the control

Next time

Summary

- 5 steps to design a processor
  - 1. Analyze instruction set => datapath requirements
  - 2. Select set of datapath components & establish clock 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
- MIPS makes it easier
  - Instructions same size
  - Source registers always in same place
  - Immediates same size, location
  - Operations always on registers/immediates
- Single cycle datapath => CPI=1, CCT => long
- Next time: implementing control