

inst.eecs.berkeley.edu/~cs61c

UC Berkeley CS61C : Machine Structures

Lecture 26

CPU Design: Designing a Single-cycle CPU, pt 2



2010-03-31

Hello to Vinay Kumar
listening from Columbia Univ!

Lecturer SOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Apple to end AT&T monopoly? ⇒

The Wall Street Journal reports that

Apple is developing a CDMA cell phone that could be used by Verizon and Sprint. Most carriers (incl AT&T) use GSM. Lots of people declining to comment, but if there's any truth to this, it's great!



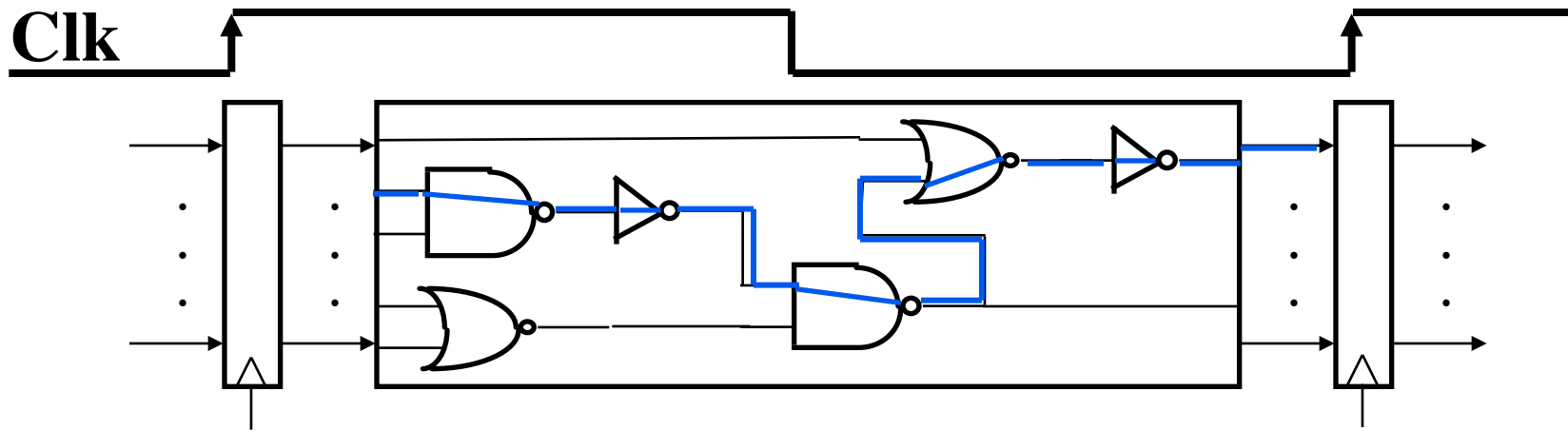
How to Design a Processor: step-by-step

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



5. Assemble the control logic

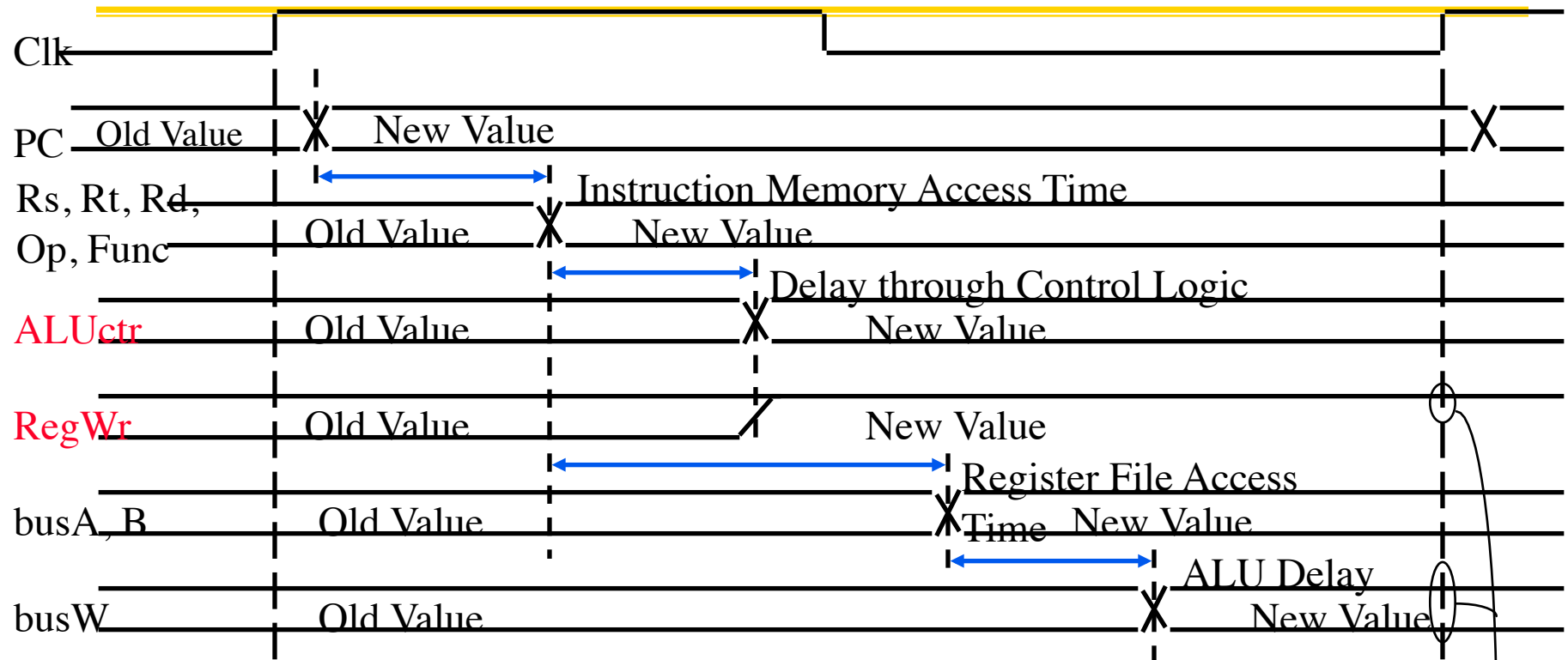
Clocking Methodology



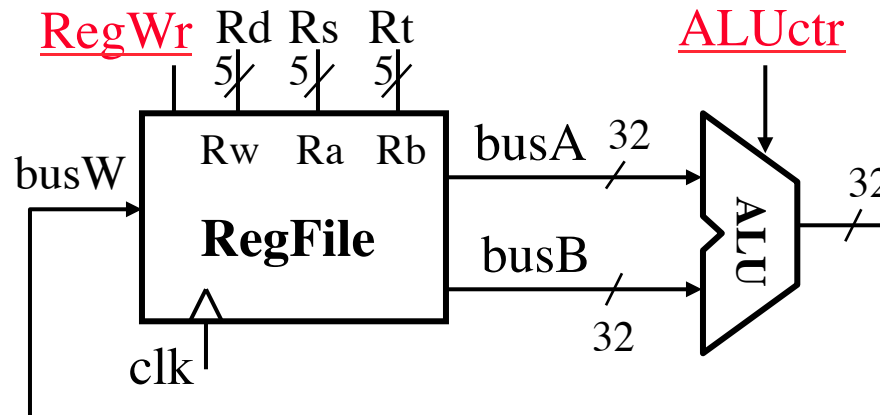
- **Storage elements clocked by same edge**
- **Being physical devices, flip-flops (FF) and combinational logic have some delays**
 - **Gates: delay from input change to output change**
 - **Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF (set-up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



Register-Register Timing: One complete cycle

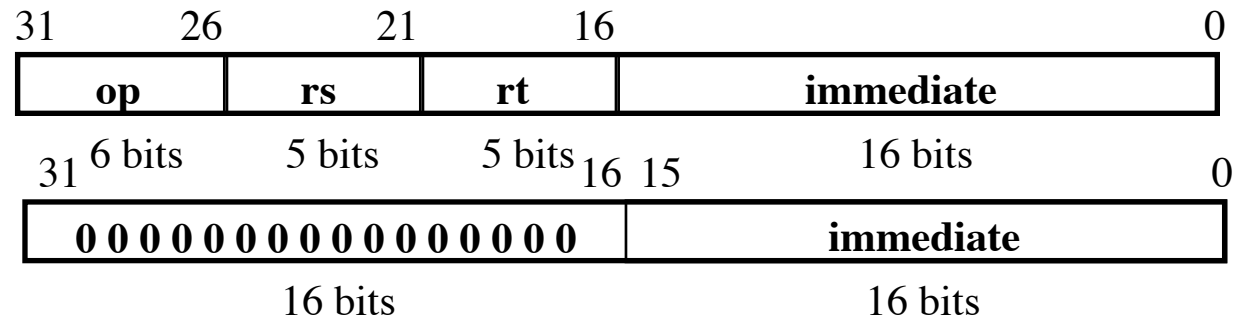


Register Write Occurs Here

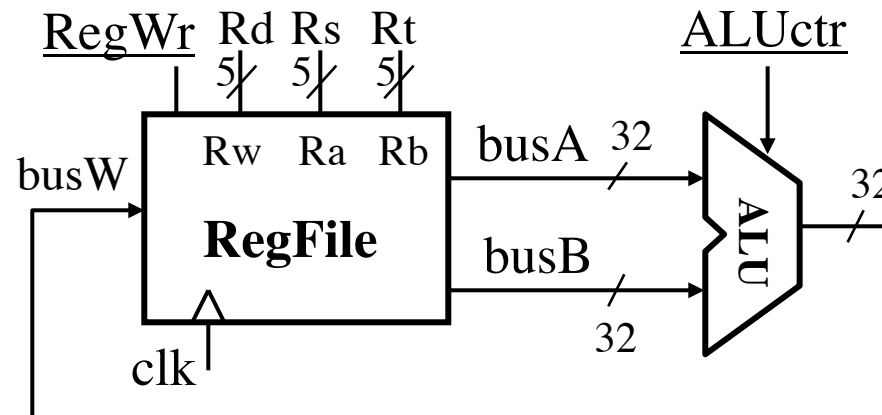


3c: Logical Operations with Immediate

- $R[rt] = R[rs] \text{ op ZeroExt}[imm16]$

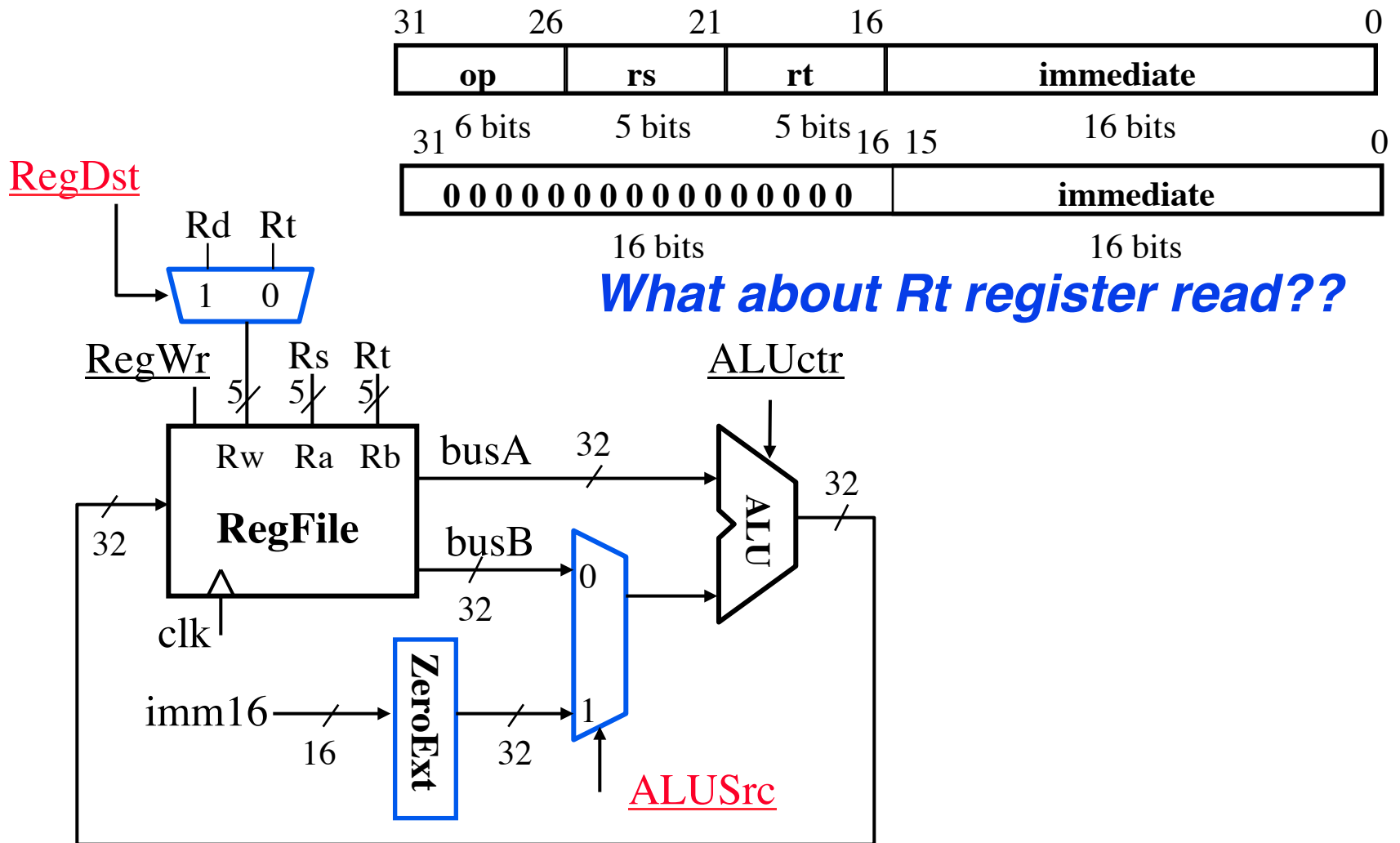


But we're writing to Rt register??



3c: Logical Operations with Immediate

- $R[rt] = R[rs] \text{ op ZeroExt}[imm16]$

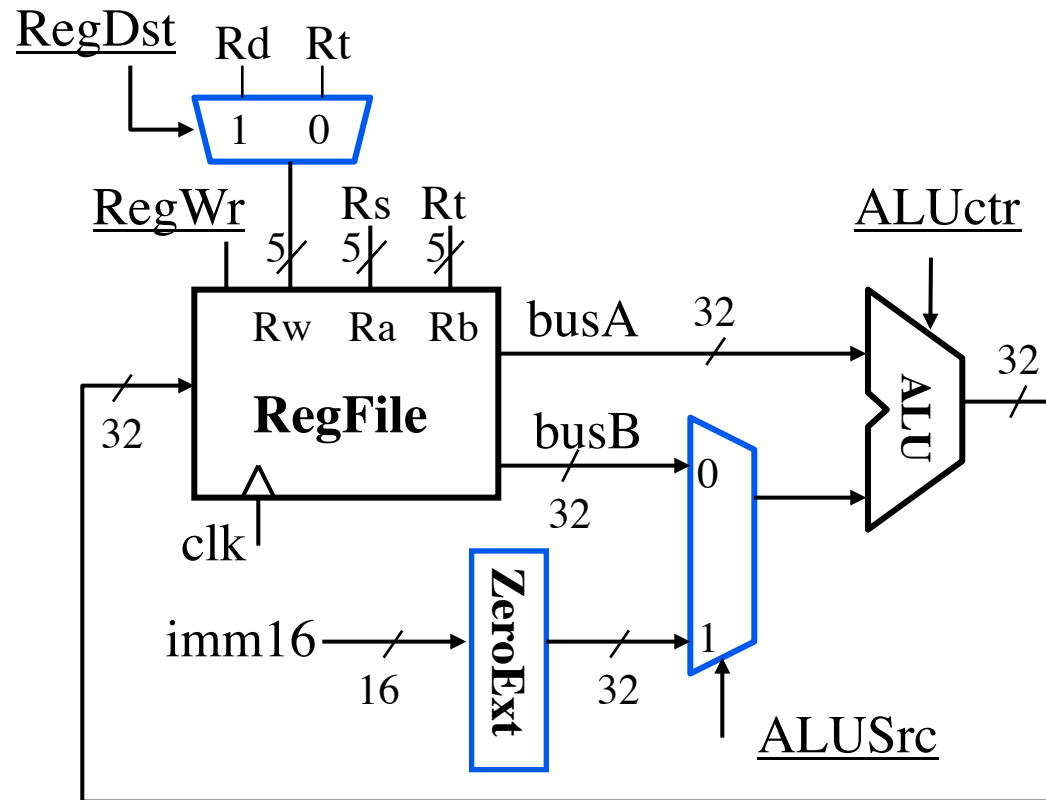
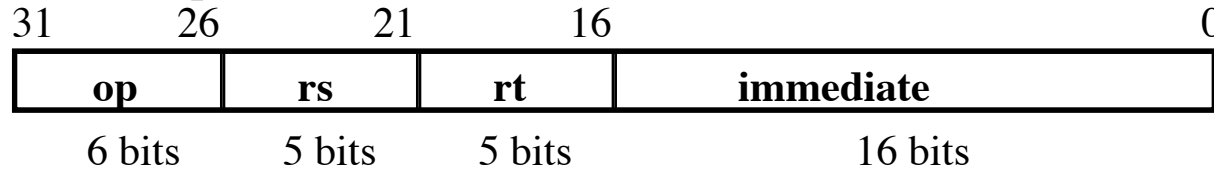


- Already defined 32-bit MUX; Zero Ext?



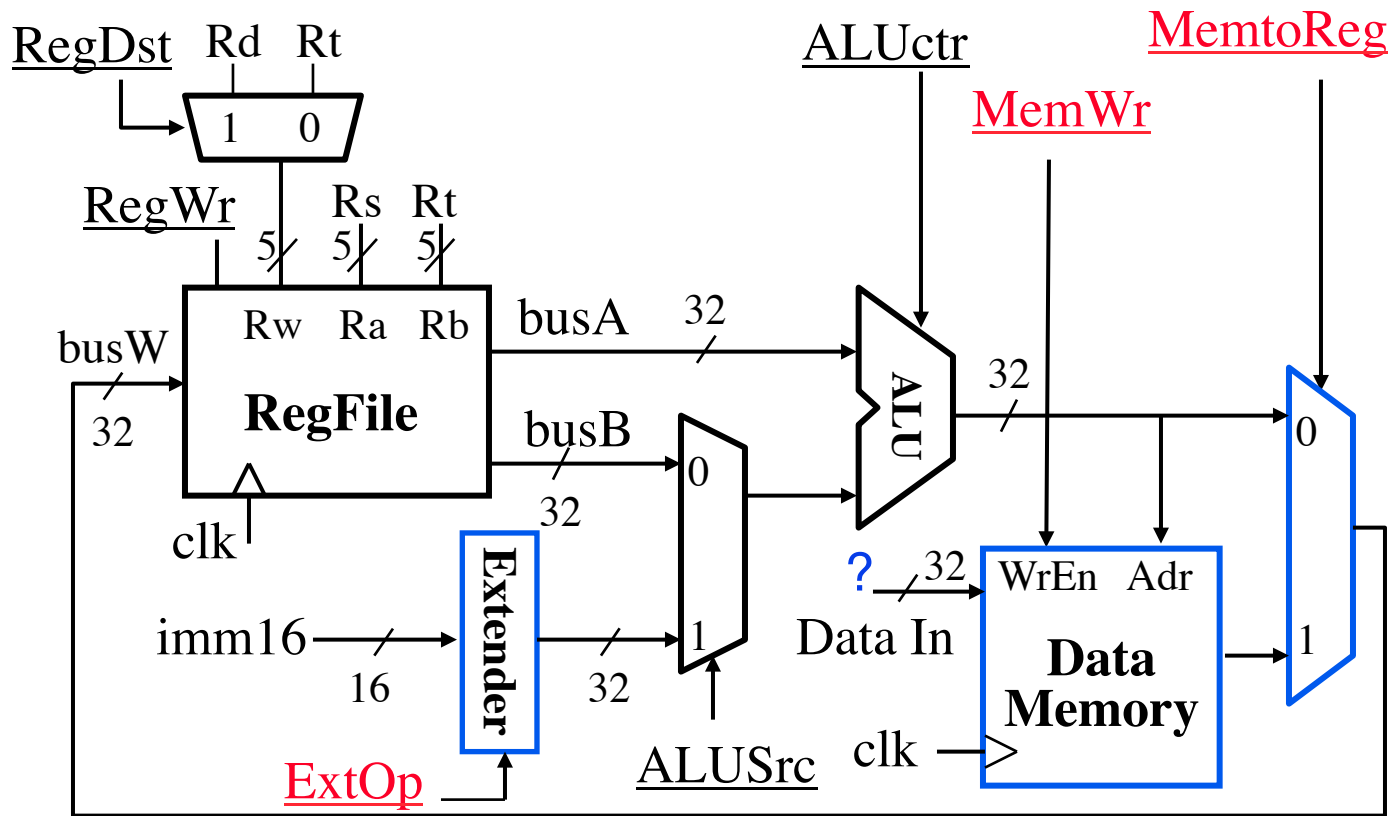
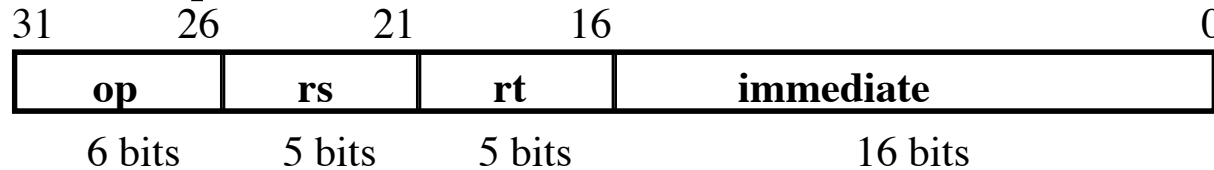
3d: Load Operations

- $R[rt] = Mem[R[rs] + SignExt[imm16]]$
Example: `lw rt, rs, imm16`



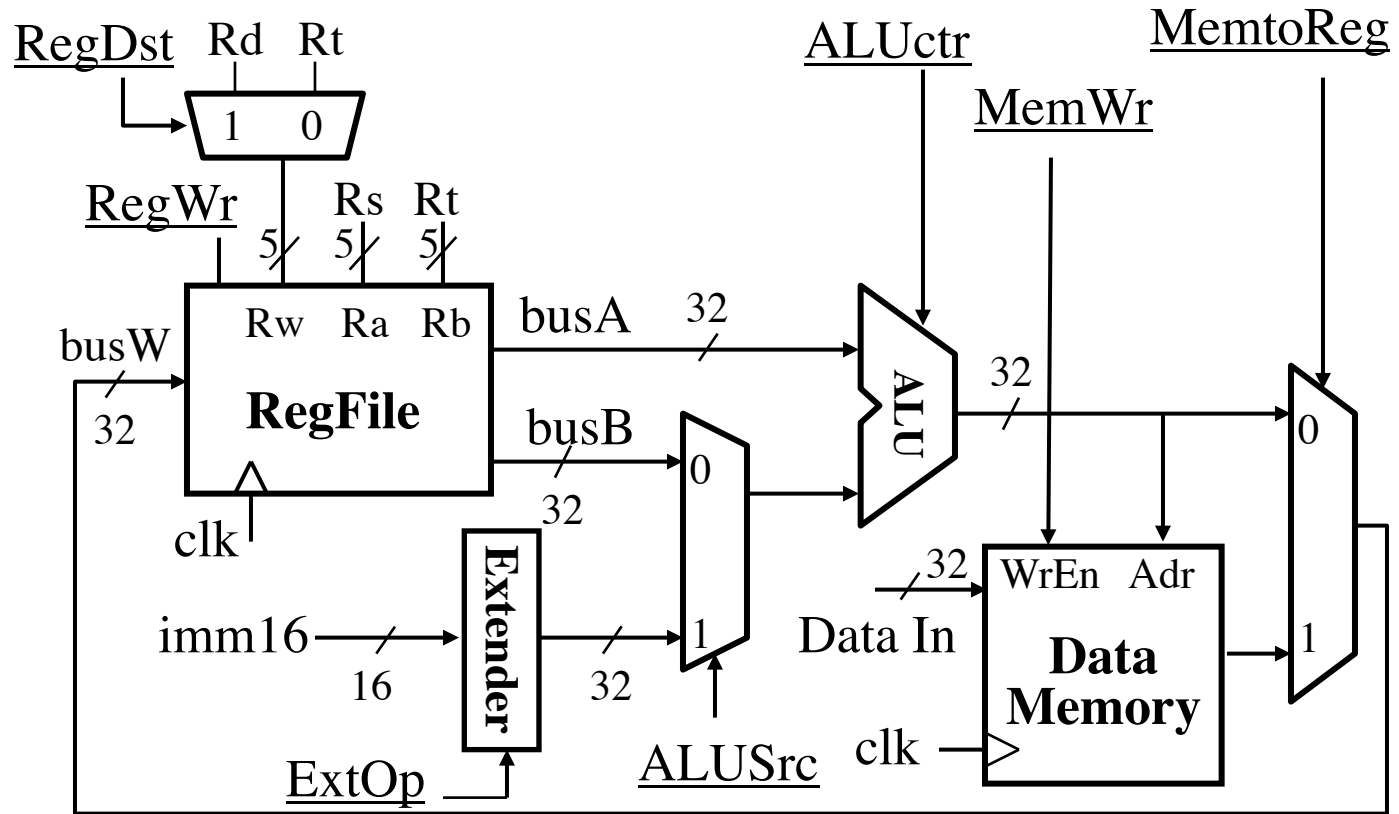
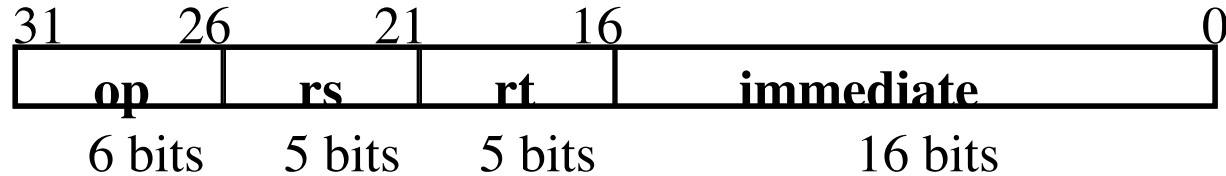
3d: Load Operations

- $R[rt] = Mem[R[rs] + SignExt[imm16]]$
Example: lw rt, rs, imm16



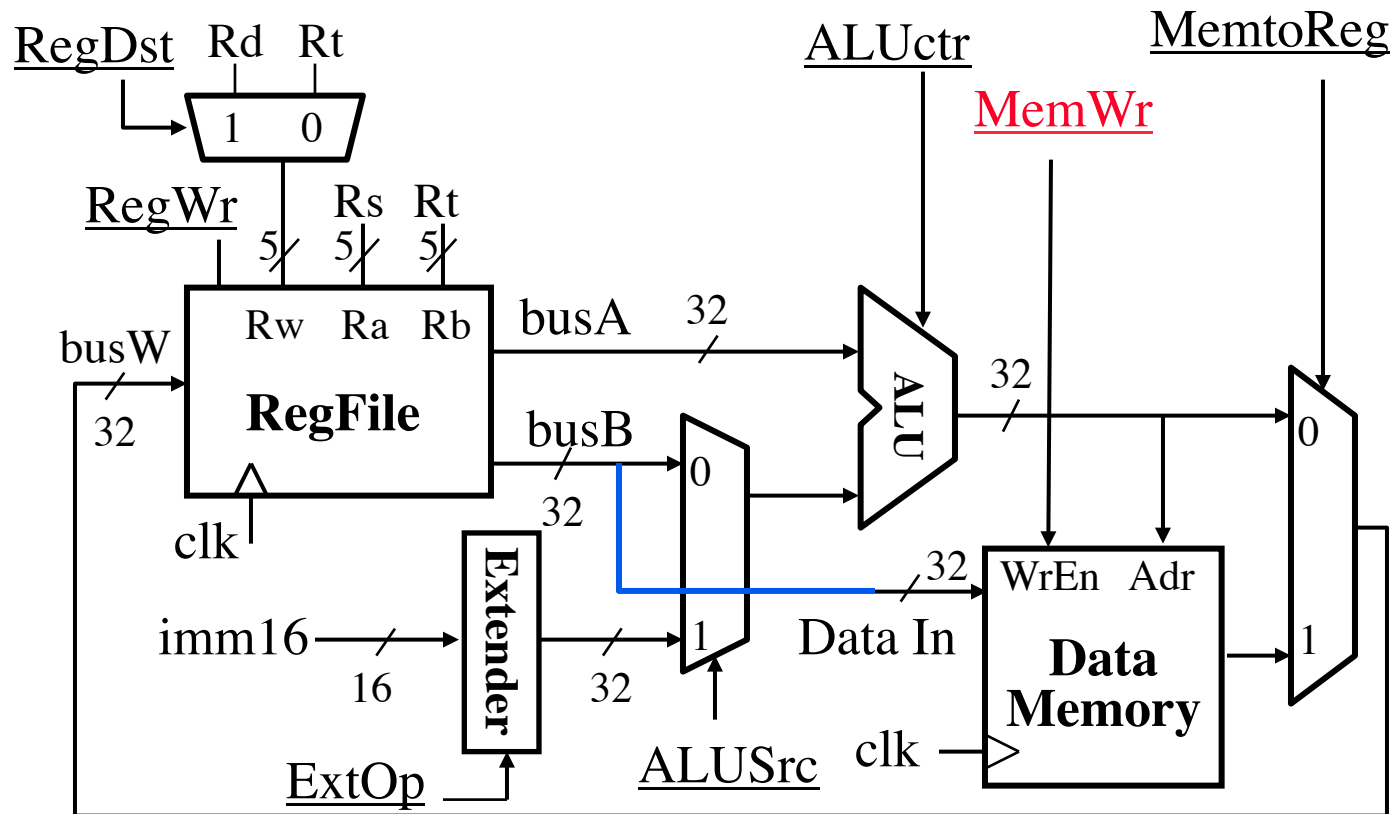
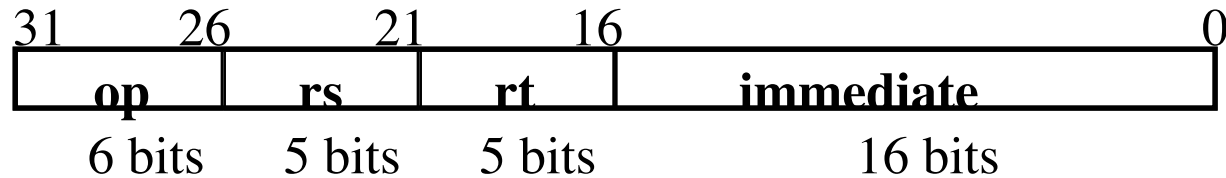
3e: Store Operations

- $\text{Mem}[R[\text{rs}] + \text{SignExt}[\text{imm16}]] = R[\text{rt}]$
 Ex.: `sw rt, rs, imm16`

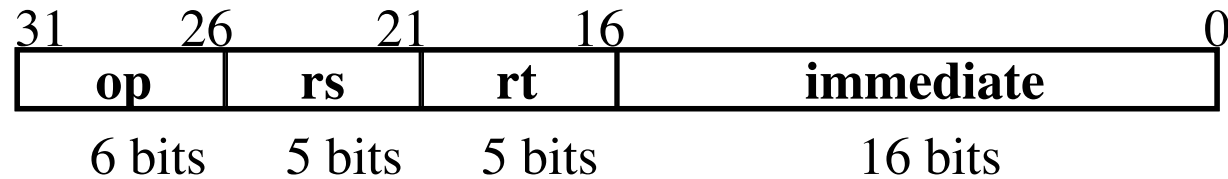


3e: Store Operations

- $\text{Mem}[\text{R}[\text{rs}] + \text{SignExt}[\text{imm16}]] = \text{R}[\text{rt}]$
 Ex.: `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 branch condition**
- **if (Equal) Calculate the next instruction's address**

- **$PC = PC + 4 + (\text{SignExt}(\text{imm16}) \times 4)$**

else

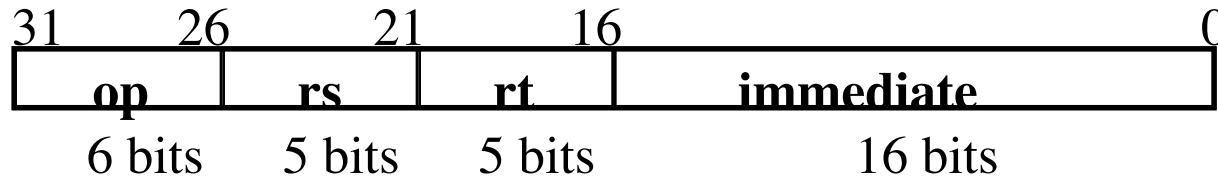
- **$PC = PC + 4$**



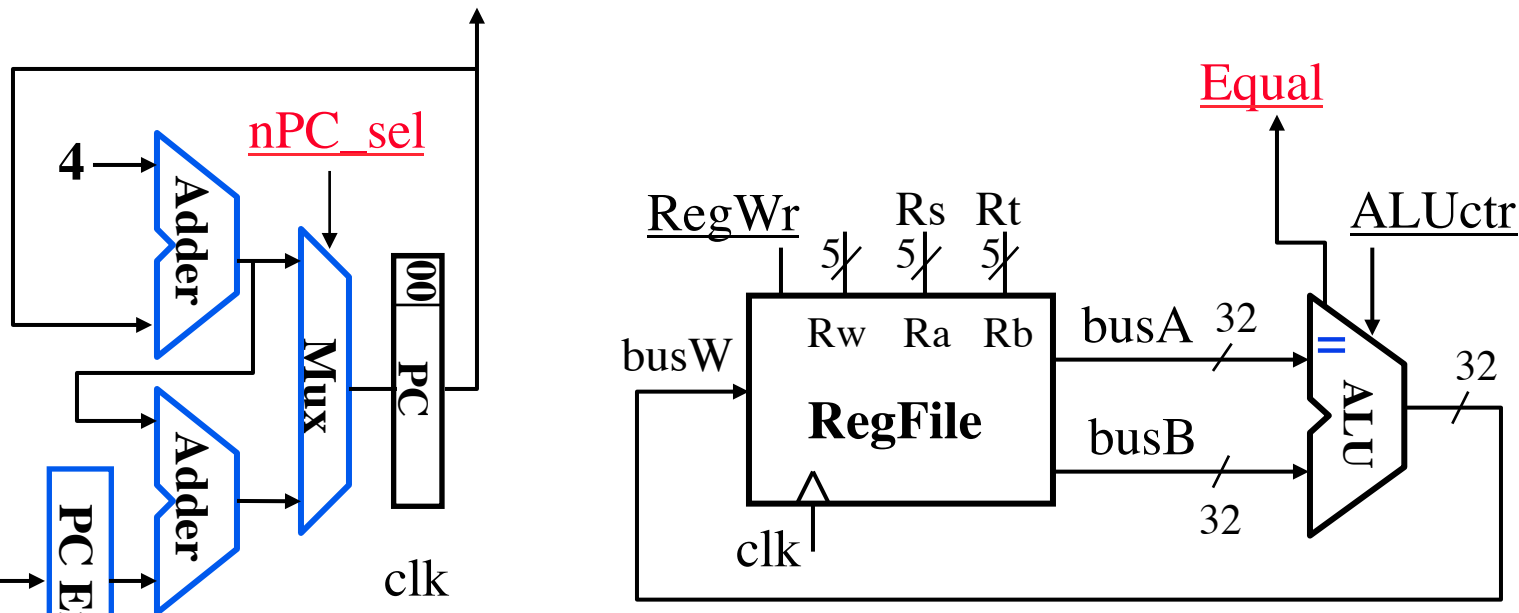
Datapath for Branch Operations

- **beq rs, rt, imm16**

Datapath generates condition (equal)



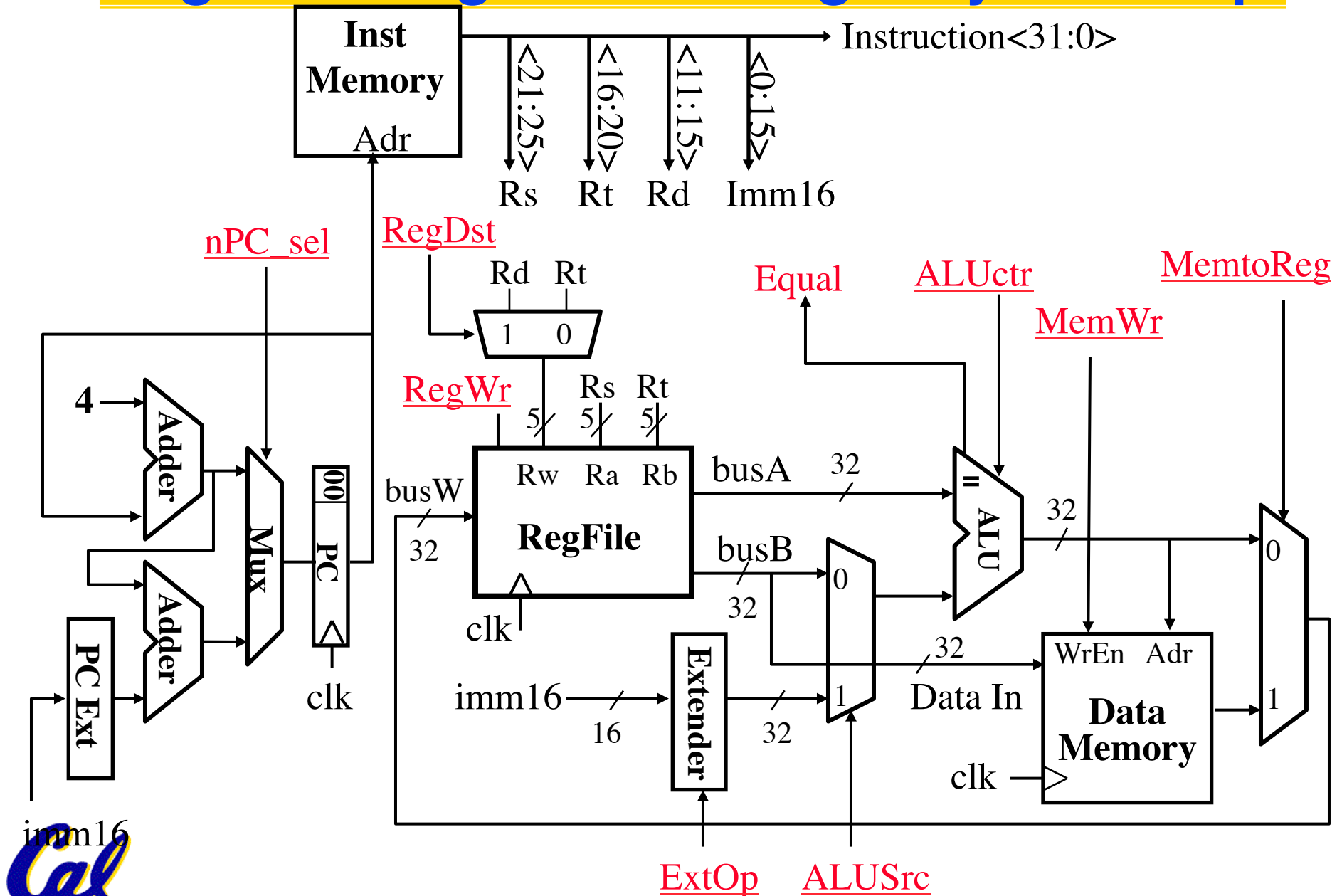
Inst Address



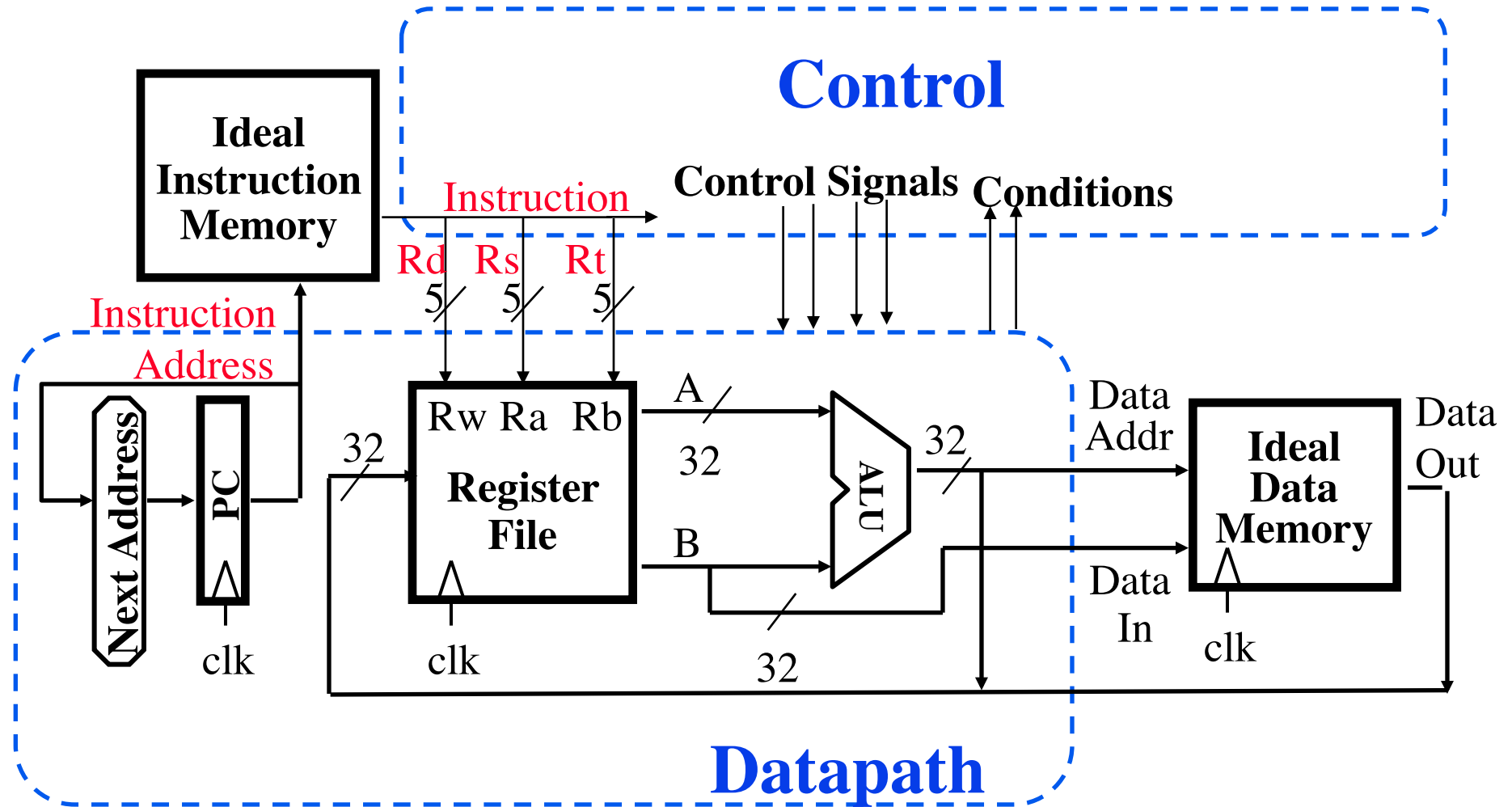
Already have mux, adder, need special sign extender for PC, need equal compare (sub?)



Putting it All Together: A Single Cycle Datapath



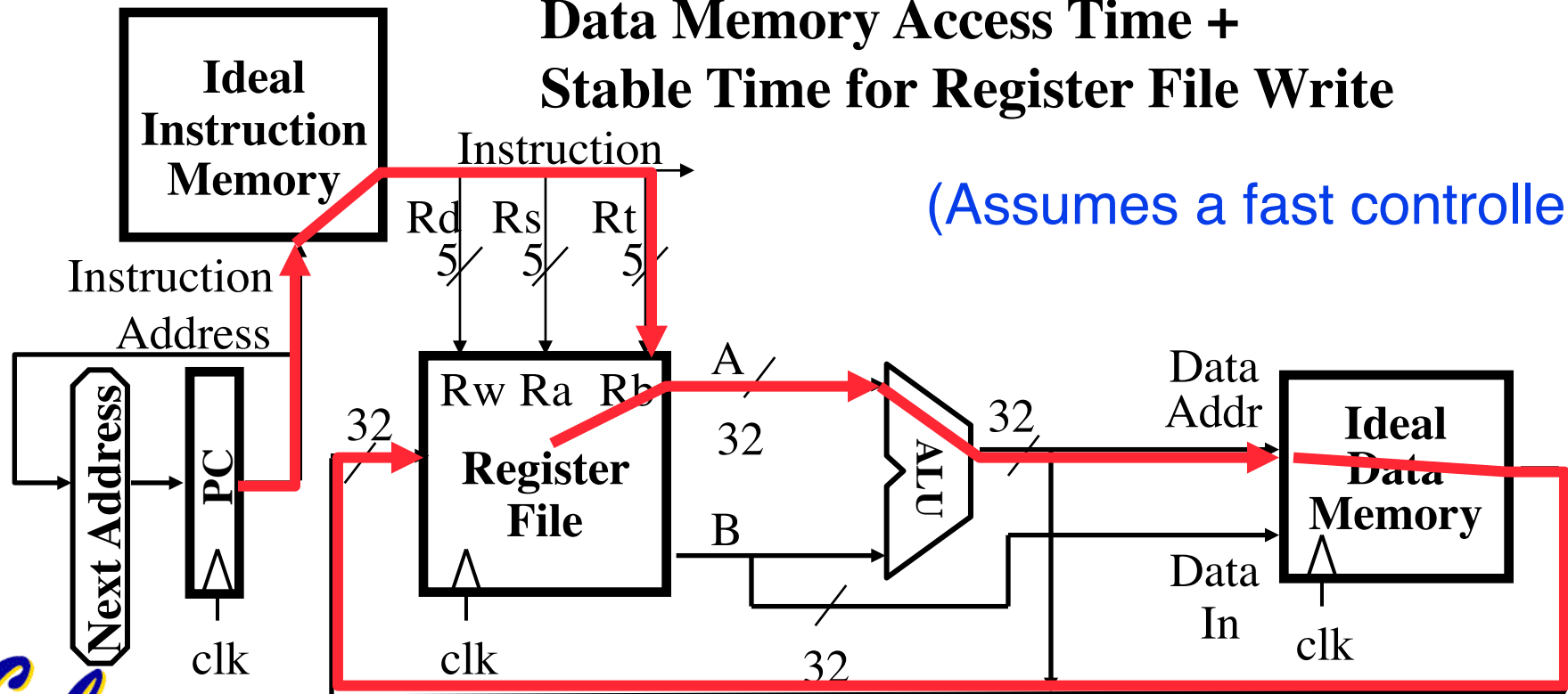
An Abstract View of the Implementation



An Abstract View of the Critical Path

**Critical Path (Load Instruction) =
Delay clock through PC (FFs) +
Instruction Memory's Access Time +
Register File's Access Time, +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Stable Time for Register File Write**

(Assumes a fast controller)



Administrivia

- **Any administrivia to announce?**



Peer Instruction

- 1) In the worst case, the biggest delay is the memory access time
- 2) With only changes to control, our datapath could write to memory and registers in one cycle.

	12
a)	FF
b)	FT
c)	TF
d)	TT



Summary: A Single Cycle Datapath

- We have everything except control signals

