inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture 28 – Single Cycle CPU Control I

2004-11-02

Lecturer PSOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Déjà vu all over again! ⇒ **Who Won?**

As of 2am 2004-11-03, it looks like Bush was ahead but hadn't yet clinched it. We may have to wait for a recount and an Ohio tabulation of provisional ballots. A Country Divided!

cnn.com

Review: How to Design a Processor: step-by-step

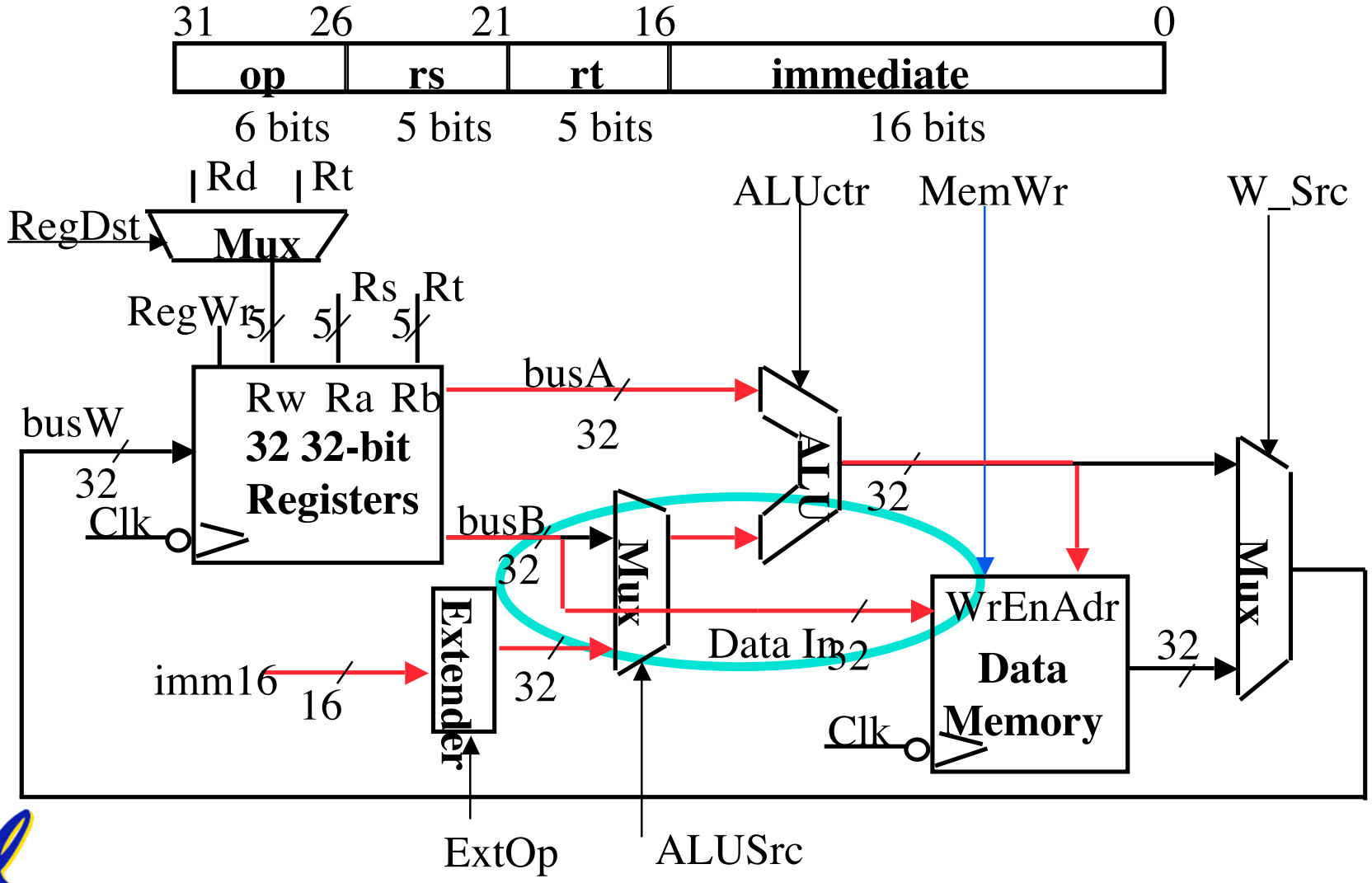
- 1. Analyze instruction set architecture (ISA) => datapath requirements
  - meaning of each instruction is given by the *register transfers*
  - datapath must include storage element for ISA registers
  - datapath must support each register transfer
- 2. Select set of datapath components and establish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

Why do we have two dirty bits?

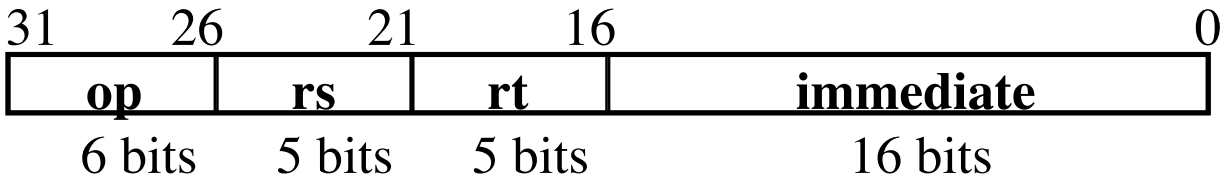
```
if (wEnb)
   if (writeReg!=4'h0)
       begin
          array[writeReg] = writeD;
          dirty1=1'b1;
          dirty2=1'b1;
       end
always @ (readReg1 or dirty1)
   begin
      readD1 = array[readReg1];
      dirty1=0;
   end
always @ (readReg2 or dirty2)
   begin
       readD2 = array[readReg2];
       dirty2=0;
   end
```

Review : 3e: Store Operations

- **Mem[R[rs] + SignExt[imm16]] = R[rt]**
  Ex.: `sw rt, rs, imm16`



3f: The Branch Instruction



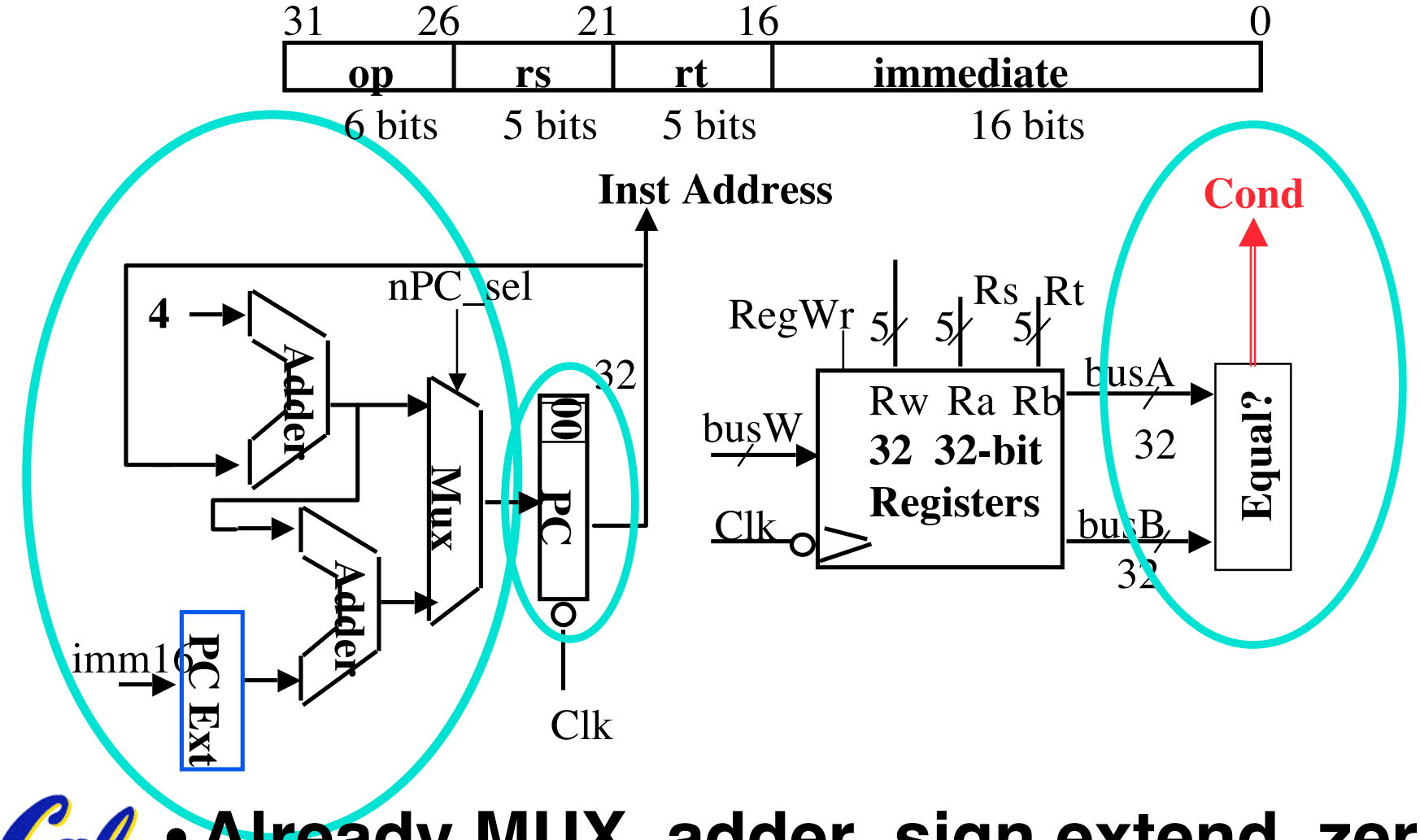
- `beq rs, rt, imm16`
  - mem[PC] Fetch the instruction from memory
  - Equal = R[rs] == R[rt] Calculate branch condition
  - if (Equal) Calculate the next instruction's address
    - PC = PC + 4 + (SignExt(imm16) x 4)

  else
    - PC = PC + 4

Datapath for Branch Operations

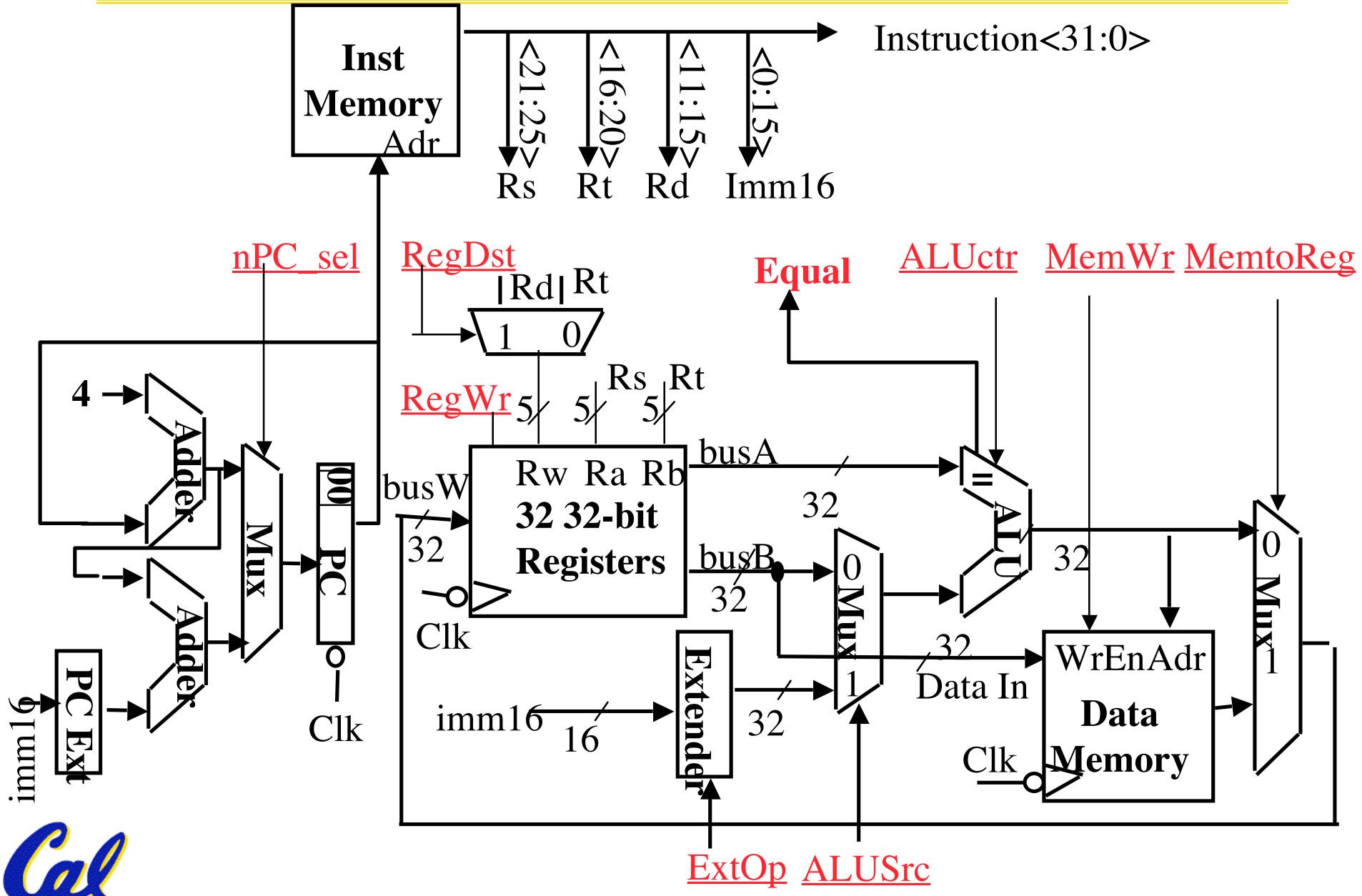
- beq rs, rt, imm16

Datapath generates condition (equal)



- Already MUX, adder, sign extend, zero

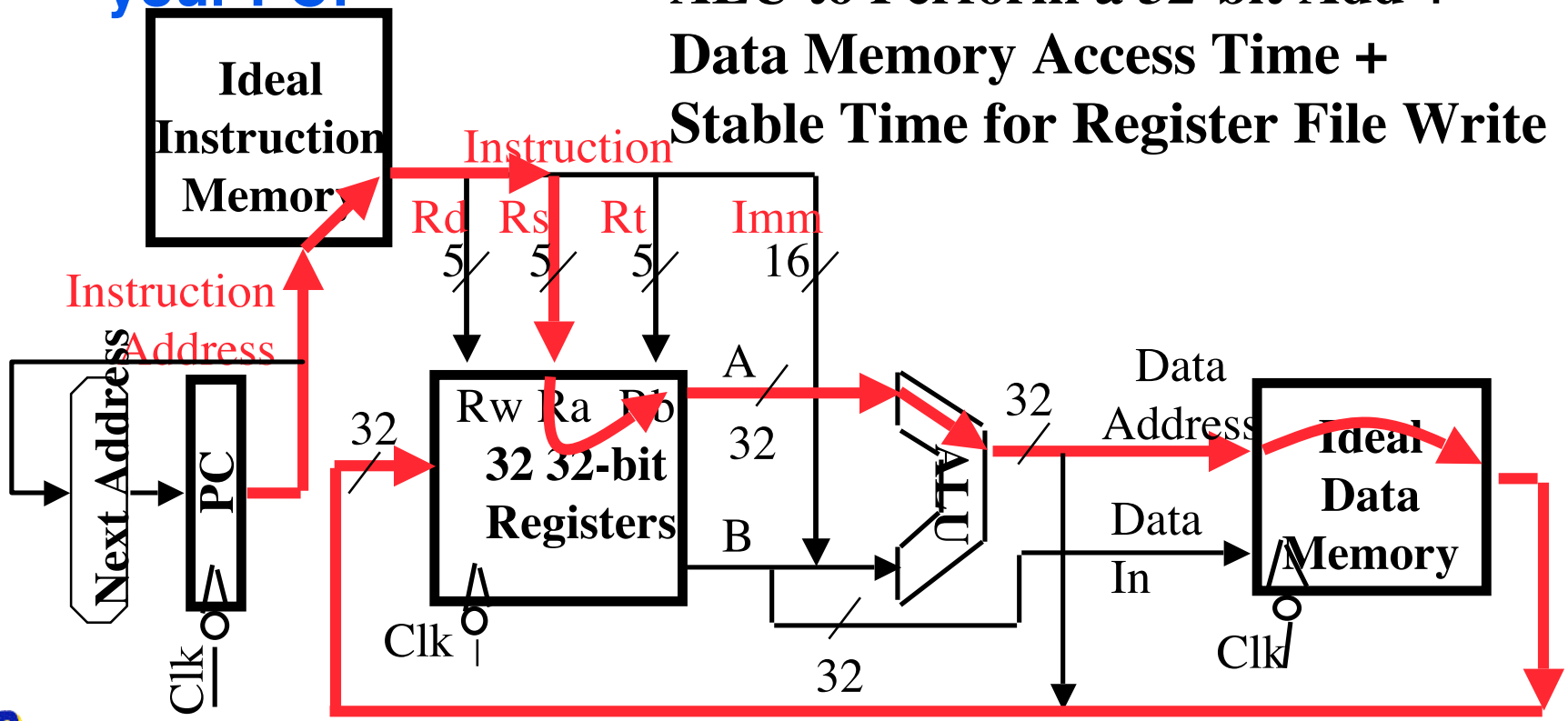
Putting it All Together:A Single Cycle Datapath



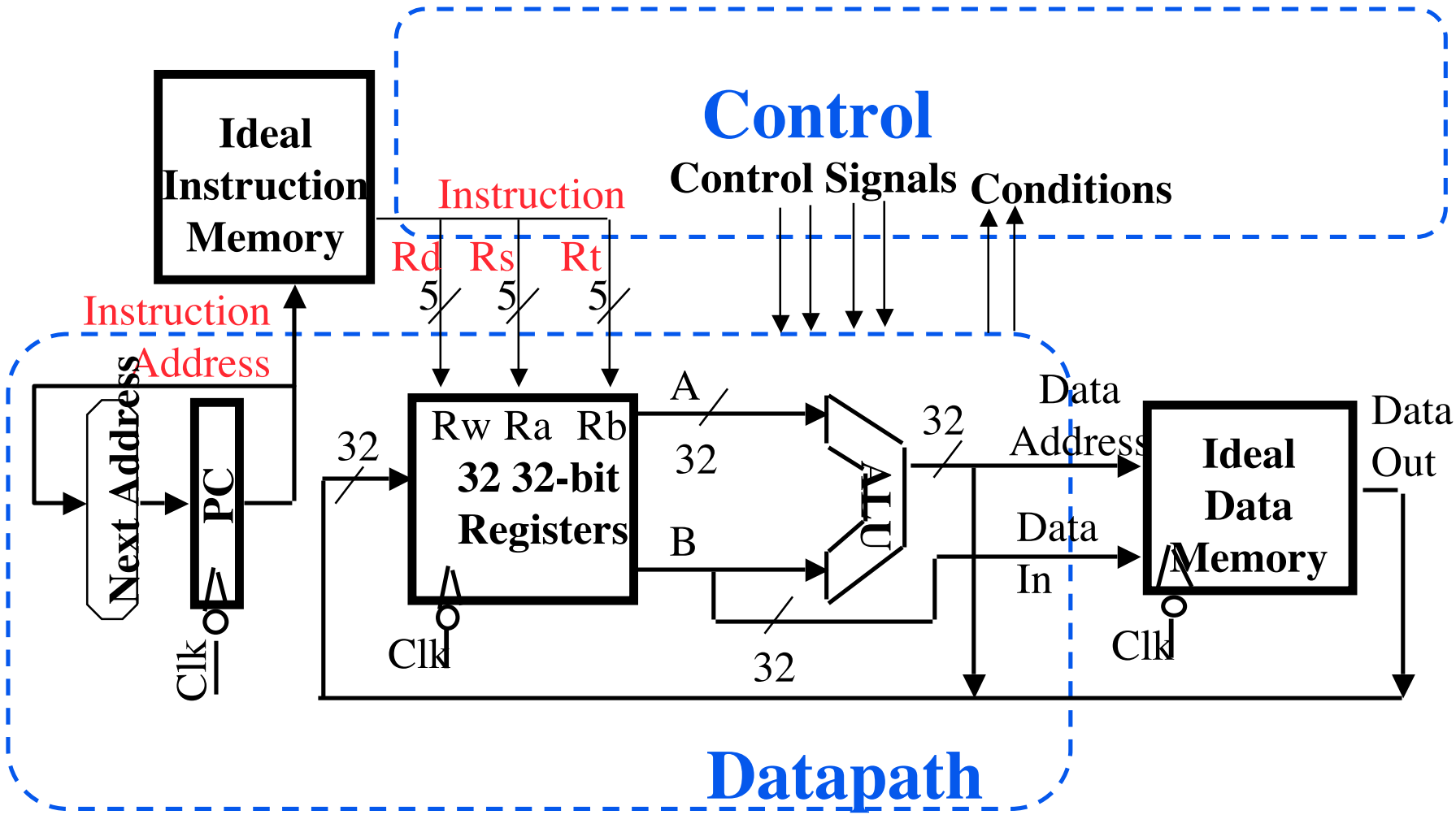
An Abstract View of the Critical Path

- This affects how much you can overclock your PC!

Critical Path (Load Operation) =
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

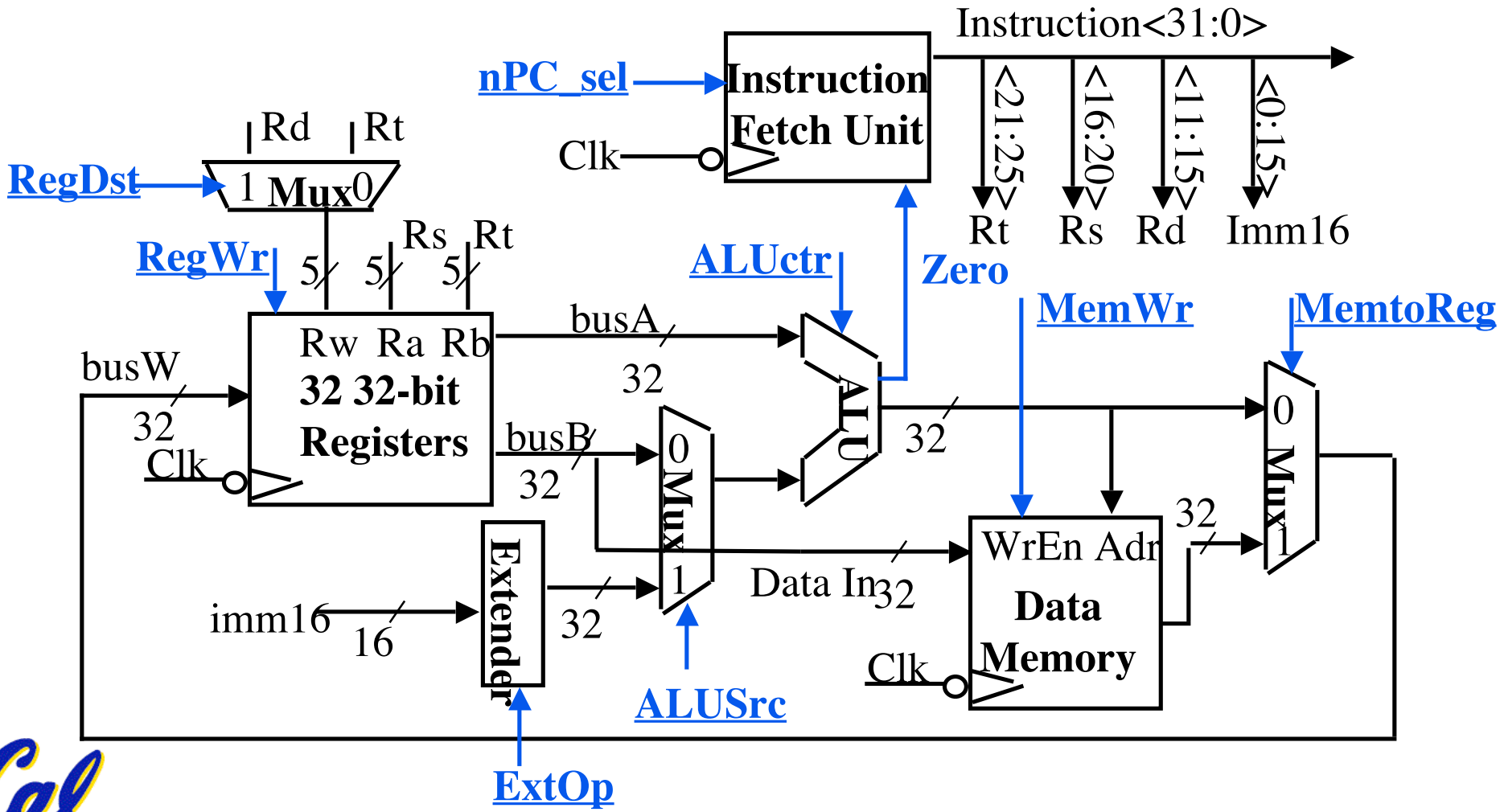


An Abstract View of the Implementation

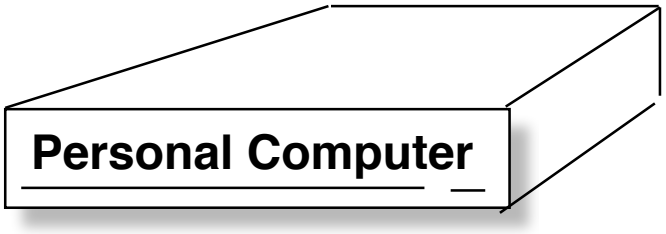


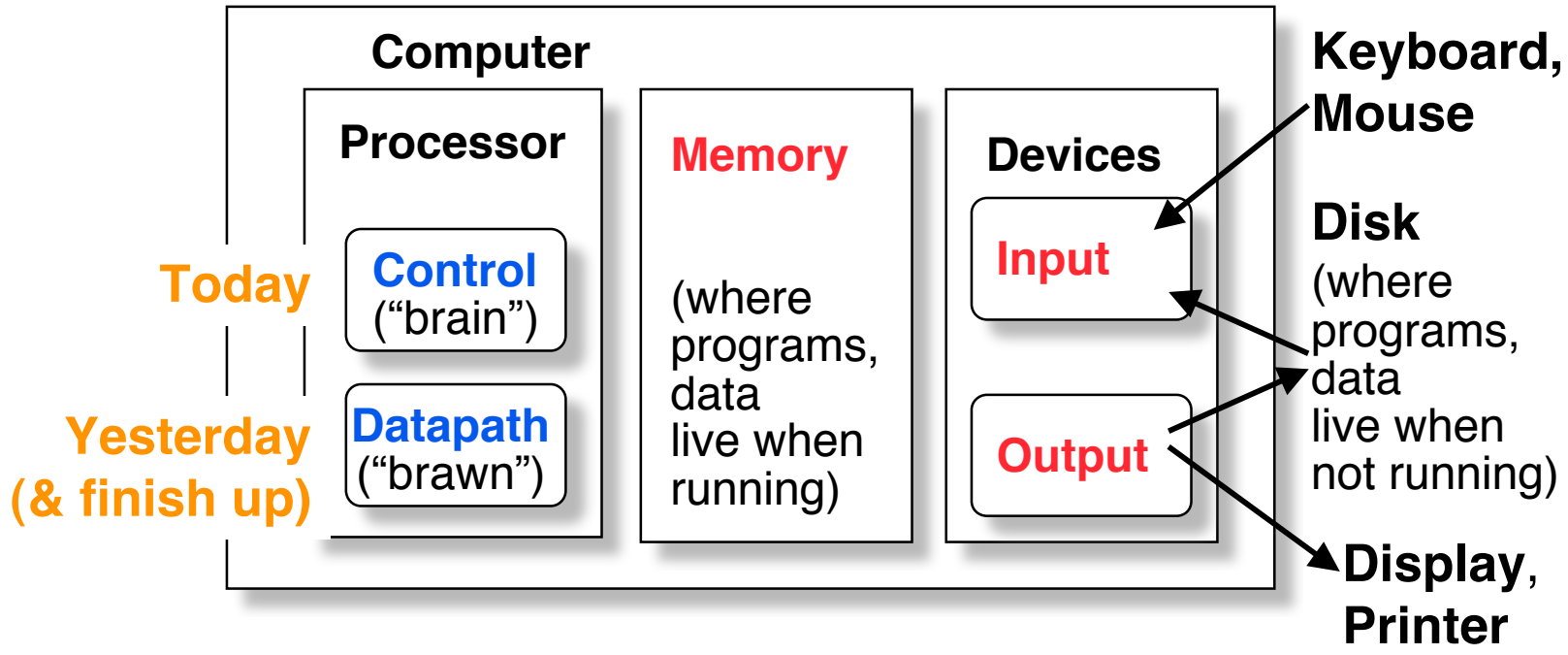
Summary: A Single Cycle Datapath

- Rs, Rt, Rd, Imed16 connected to datapath
- We have everything except control signals



Anatomy Review: 5 components of any Computer



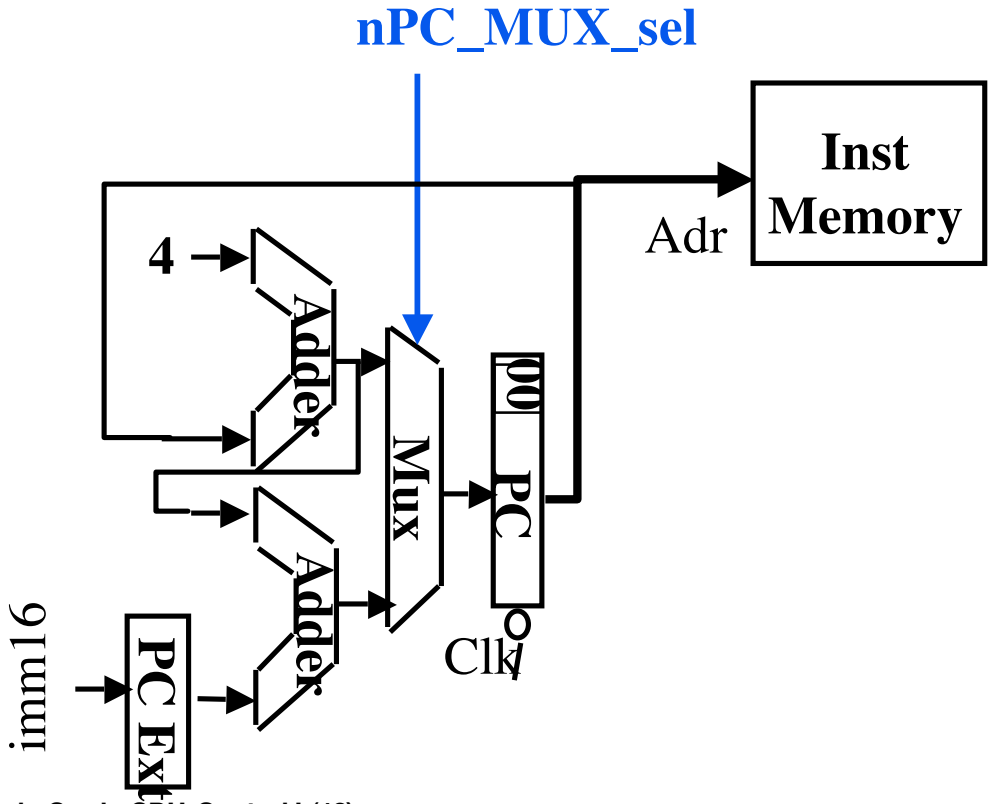


Recap: Meaning of the Control Signals

- **nPC_MUX_sel:** **0 ⇒ PC <– PC + 4**
  1 ⇒ PC <– PC + 4 + {SignExt(Im16) , 00 }

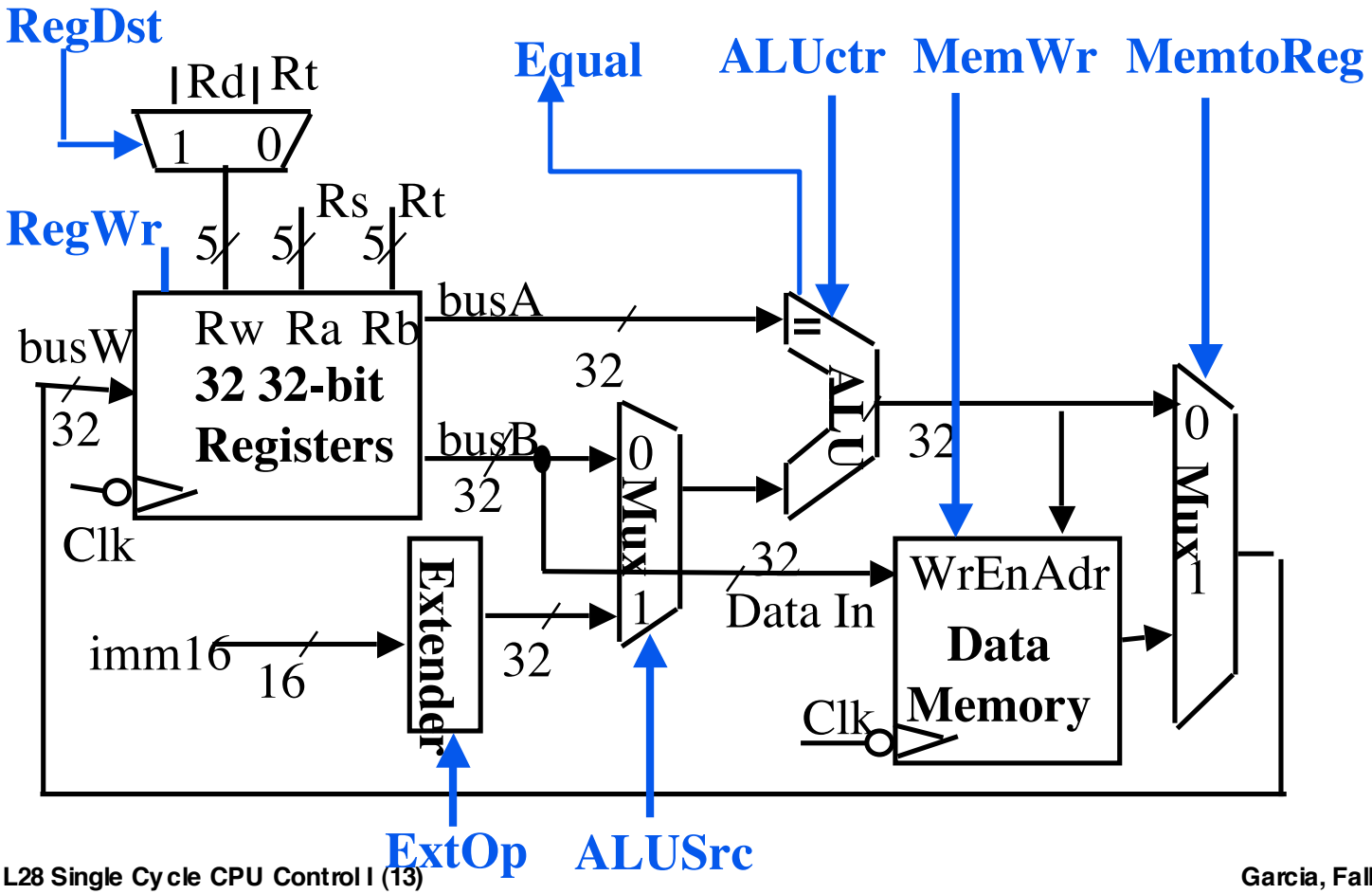
  "n"=next

- **Later in lecture: higher-level connection between mux and branch cond**



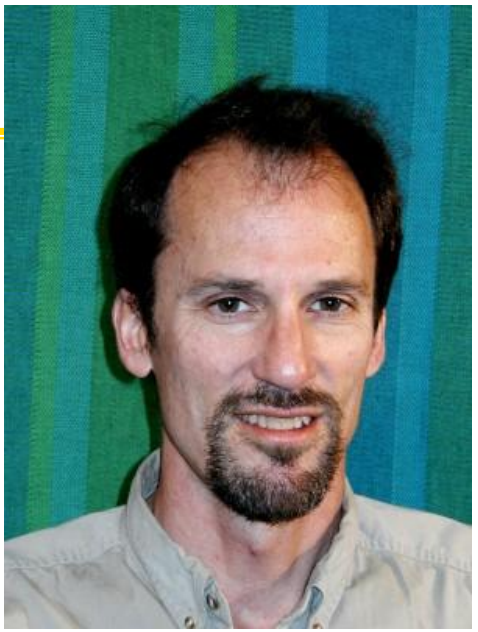
Recap: Meaning of the Control Signals

- ExtOp: "zero", "sign"
- ALUsrc: 0 ⇒ regB; 1 ⇒ immed
- ALUctr: "add", "sub", "or"
- ° MemWr: 1 ⇒ write memory
- ° MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- ° RegDst: 0 ⇒ "rt"; 1 ⇒ "rd"
- ° RegWr: 1 ⇒ write register



Great talk today – Don't miss

306 Soda Hall @ 4pm



- **Dr. David Anderson**
  - **Space Sciences Laboratory, U.C. Berkeley. SETI Director**

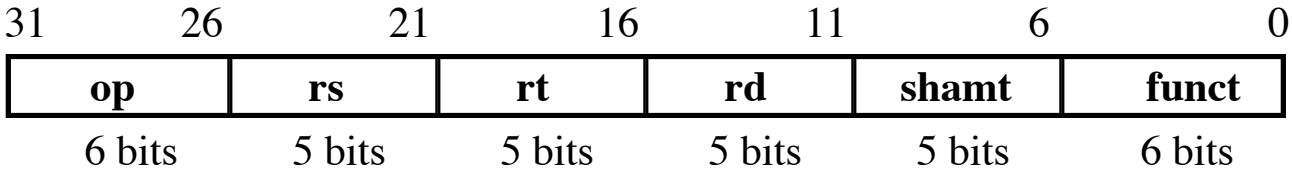
"Public Resource Computing"

The majority of the world's computing power is no longer concentrated in supercomputer centers and machine rooms. Instead it is distributed around the world in hundreds of millions of personal computers and game consoles, many connected to the Internet. A new computing paradigm, "public-resource computing", uses these PCs to do scientific supercomputing. This paradigm enables new research in a number of areas and has social implications as well: it catalyzes global communities centered around common interests and goals, it encourages public awareness of current scientific research, and it may give the public a measure of control over the directions of science progress.

Administrivia

- Dan will be away at a conference on Thursday and Friday, Andrew will cover lecture.
- We regraded all the midterms and your TAs have them to return to you.

RTL: The `Add` Instruction

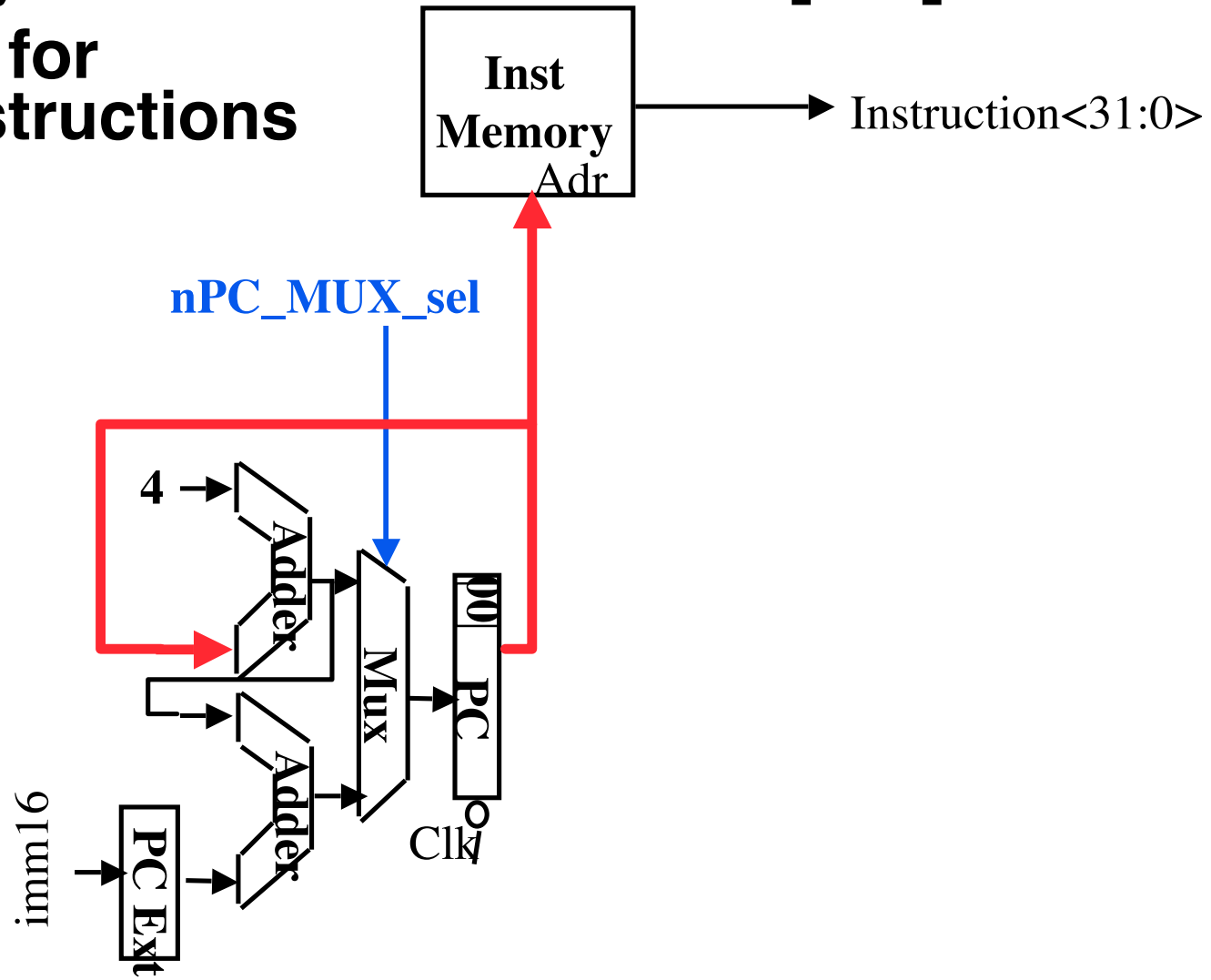



```
add rd, rs, rt
```

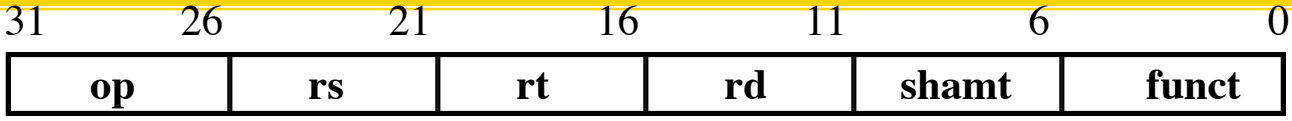
- MEM[PC] Fetch the instruction from memory
- R[rd] = R[rs] + R[rt] The actual operation
- PC = PC + 4 Calculate the next instruction's address

Instruction Fetch Unit at the Beginning of Add

- Fetch the instruction from Instruction memory: Instruction = MEM[PC]
  - same for all instructions



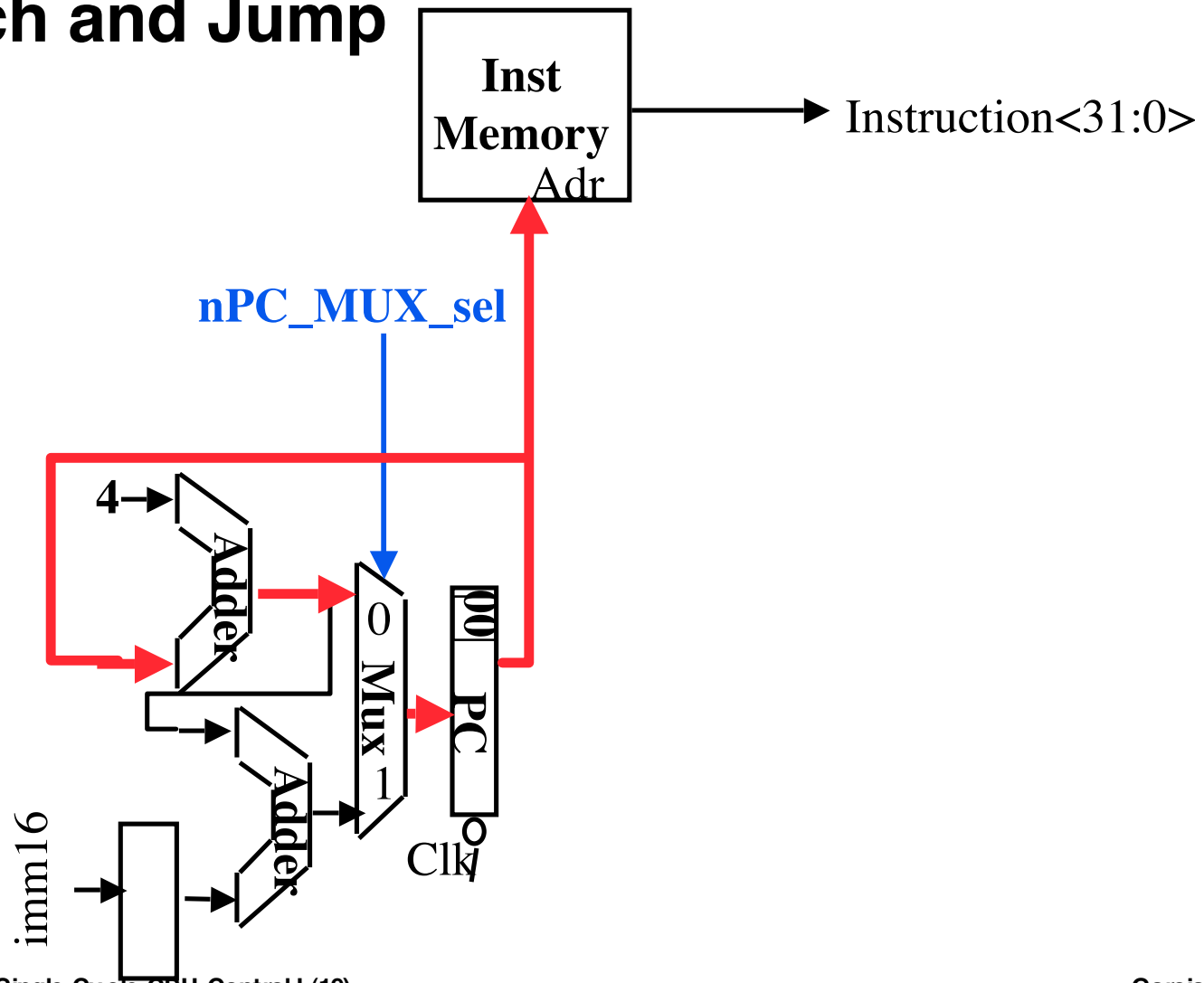
The Single Cycle Datapath during Add



- R[rd] = R[rs] + R[rt]

Instruction Fetch Unit at the End of Add

- PC = PC + 4
  - This is the same for all instructions except: Branch and Jump



Peer Instruction

Suppose we're writing a MIPS interpreter in Verilog. Which sequence below is best organization for the interpreter?

A. `repeat` loop that fetches instructions
B. `while` loop that fetches instructions
C. Decodes instructions using `case` statement
D. Decodes instr. using chained `if` statements
E. Executes each instruction
F. Increments PC by 4

1:	ACEF
2:	ADEF
3:	AECF
4:	AEDF
5:	BCEF
6:	BDEF
7:	BECF
8:	BEDF
9:	EF
0:	FAE

Summary: Single cycle datapath

- 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
- Control is the hard part
- MIPS makes that easier
  - Instructions same size
  - Source registers always in same place
  - Immediates same size, location
  - Operations always on registers/immediates