CS152 – Computer Architecture and Engineering

Lecture 6 – Single Cycle and Design Notebook

2003-09-11

Dave Patterson
(www.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs152/

Performance Review

- Latency v. Throughput
- Performance doesn't depend on any single factor: need to know Instruction Count, Clocks Per Instruction and Clock Rate to get valid estimations
- 2 Definitions of times:
  - User Time: time user needs to wait for program to execute (multitasking affects)
  - CPU Time: time spent executing a single program: (no multitasking)
- Amdahl's Law: law of diminishing returns

Design Process Review

- Divide and Conquer (e.g., ALU)
  - Formulate a solution in terms of simpler components.
  - Design each of the components (subproblems)
- Generate and Test (e.g., ALU)
  - Given a collection of building blocks, look for ways of putting them together that meets requirement
- Successive Refinement
  - Solve "most" of the problem (i.e., ignore some constraints or special cases), examine and correct shortcomings.
- Formulate High-Level Alternatives
  - Articulate many strategies to "keep in mind" while pursuing any one approach.
- Work on the Things you Know How to Do
  - The unknown will become “obvious” as you make progress.

Outline

- Single Cycle Datapath
  - 5 steps
  - Performance?
    (Instruction Count x CPI x Clock Cycle Time)
- Online Notebook
  - Capturing design and implementation process, decisions so that can understand evolution of design, fix bugs

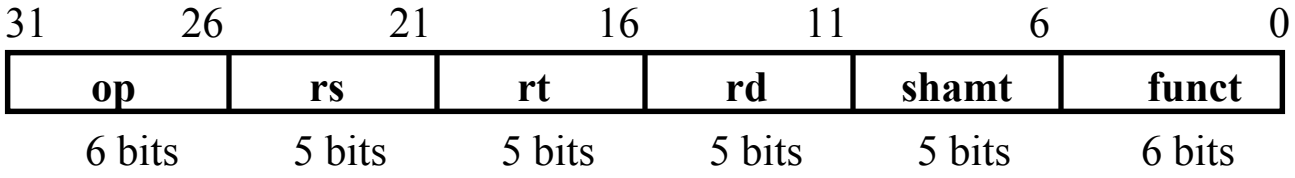
How to Design a Processor: step-by-step

- 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 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

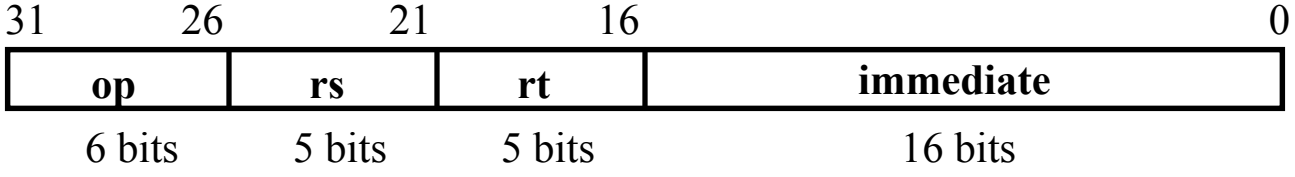
The MIPS Instruction Formats

° All MIPS instructions are 32 bits long. The three instruction formats:

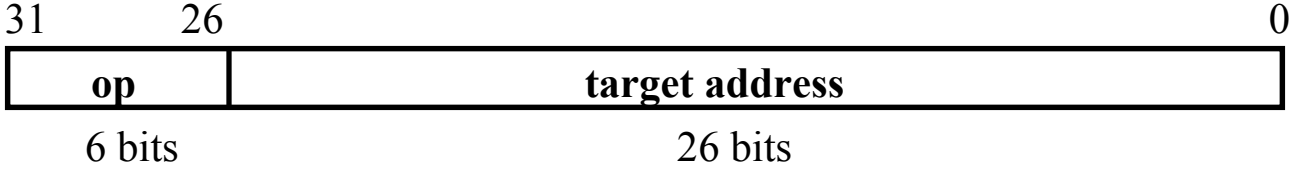
- R-type



- I-type



- J-type

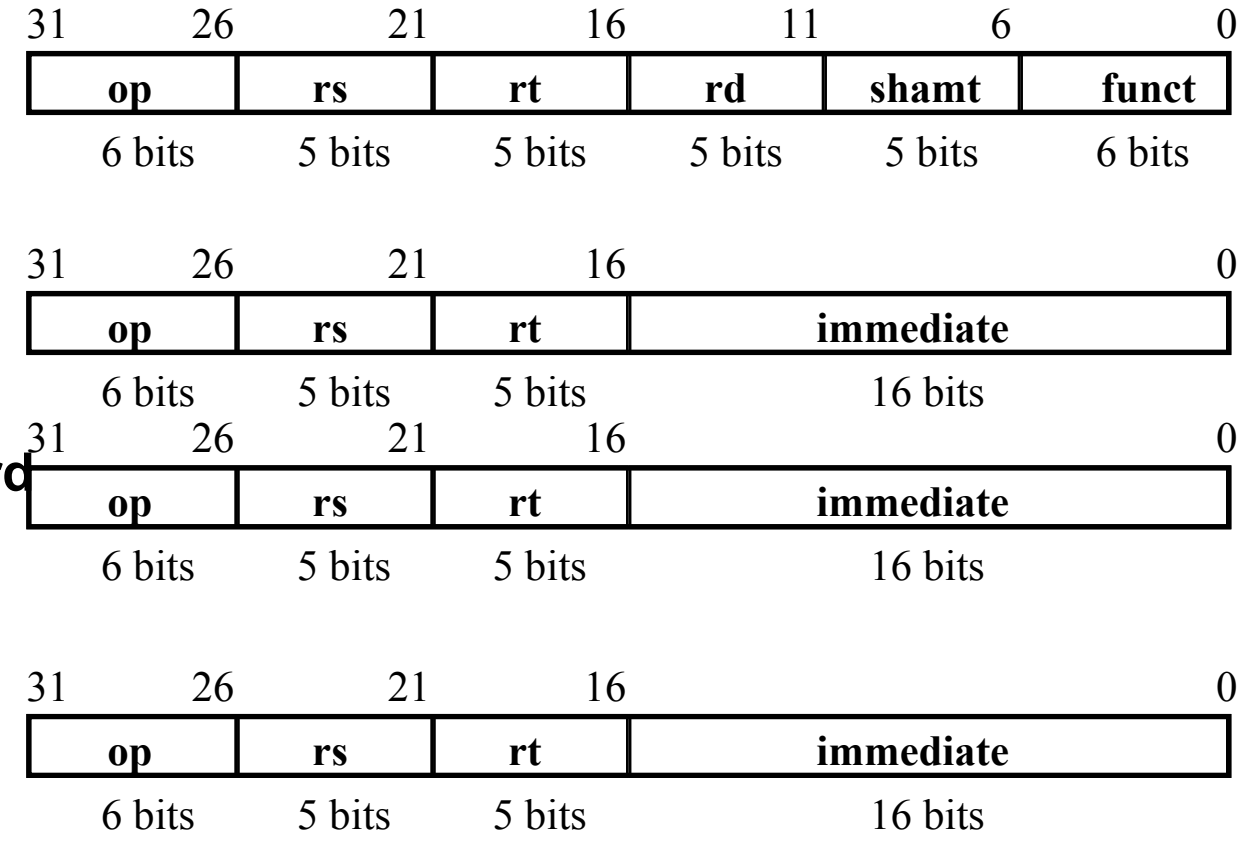


° The different fields are:

- 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 and SUB
  - addU rd, rs, rt
  - subU rd, rs, rt
- OR Immediate:
  - ori rt, rs, imm16
- LOAD and STORE Word
  - lw rt, rs, imm16
  - sw rt, rs, imm16
- BRANCH:
  - beq rs, rt, imm16



Using Hardware Description Lang.

° All start by fetching the instruction

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

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

inst	HDL description	
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]) PC <= PC + 4 + {sign_ext(Imm16)], 2b00 } else PC <= PC + 4	

Step 1: Requirements of the Instruction Set

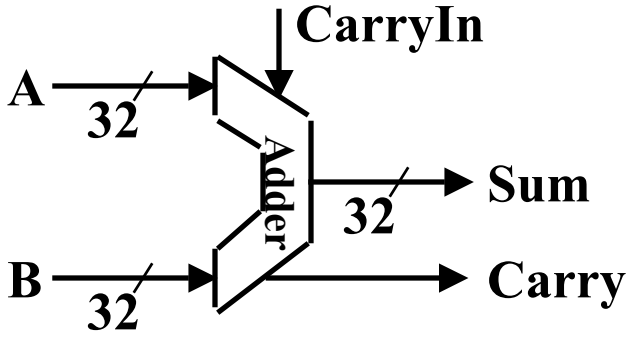
- Memory
  - One for instructions, one for data
- Registers (32 x 32bit)
  - read RS
  - read RT
  - Write RT or RD
- PC
- Sign Extender (for immediate field)
- Add and Sub register or extended immediate
- Add 4 or extended immediate to PC

Step 2: Components of the Datapath

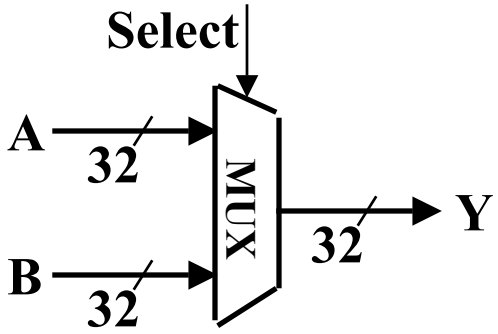
- Combinational Logic Elements
- Storage Elements
  - Clocking methodology

Combinational Logic Elements (Basic Building Blocks)

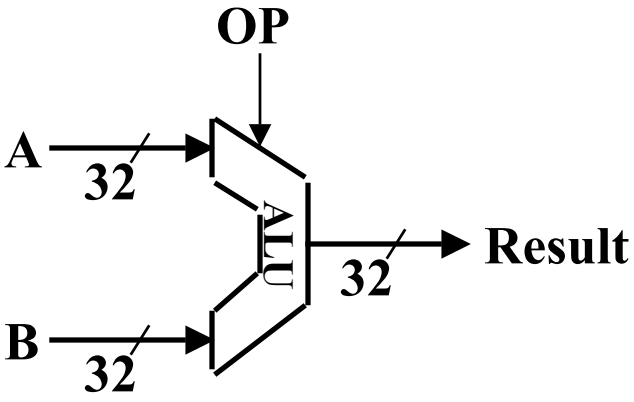
- Adder



- MUX

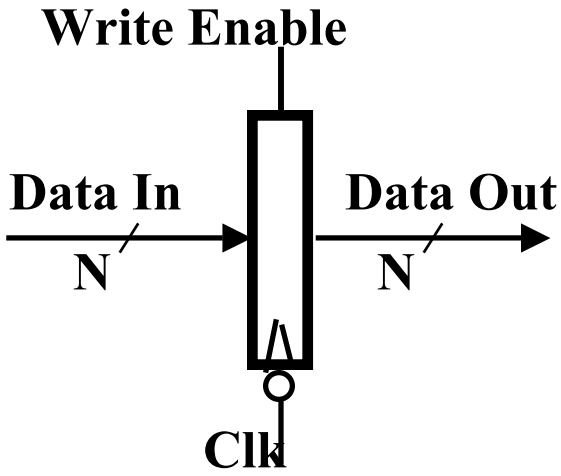


- ALU



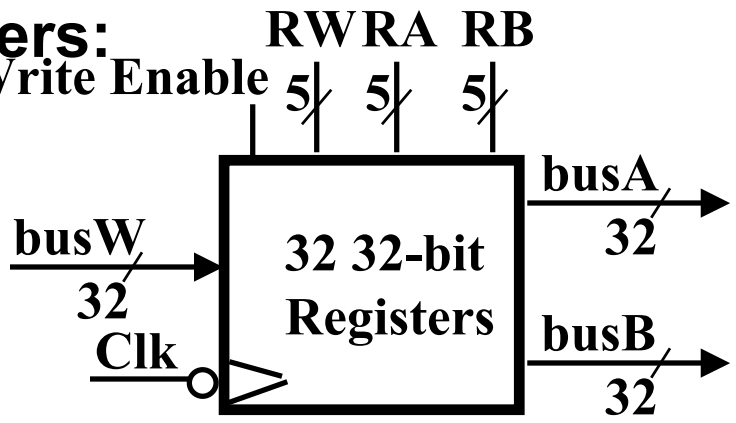
Storage Element: Register (Basic Building Block)

- 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 will become Data In



Storage Element: Register File

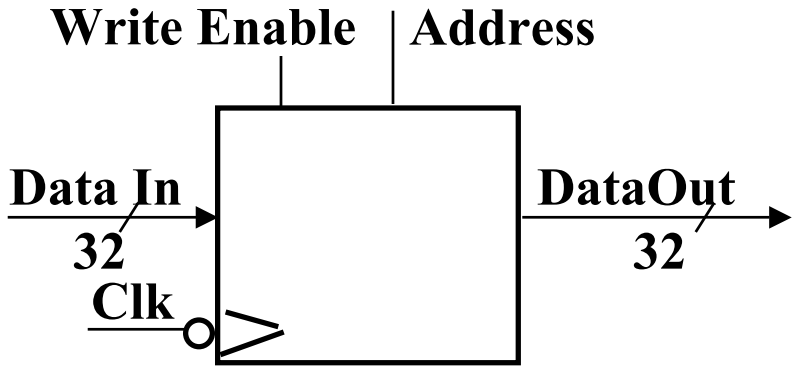
° Register File consists of 32 registers:
  - Two 32-bit output busses:
    busA and busB
  - One 32-bit input bus: busW



° 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

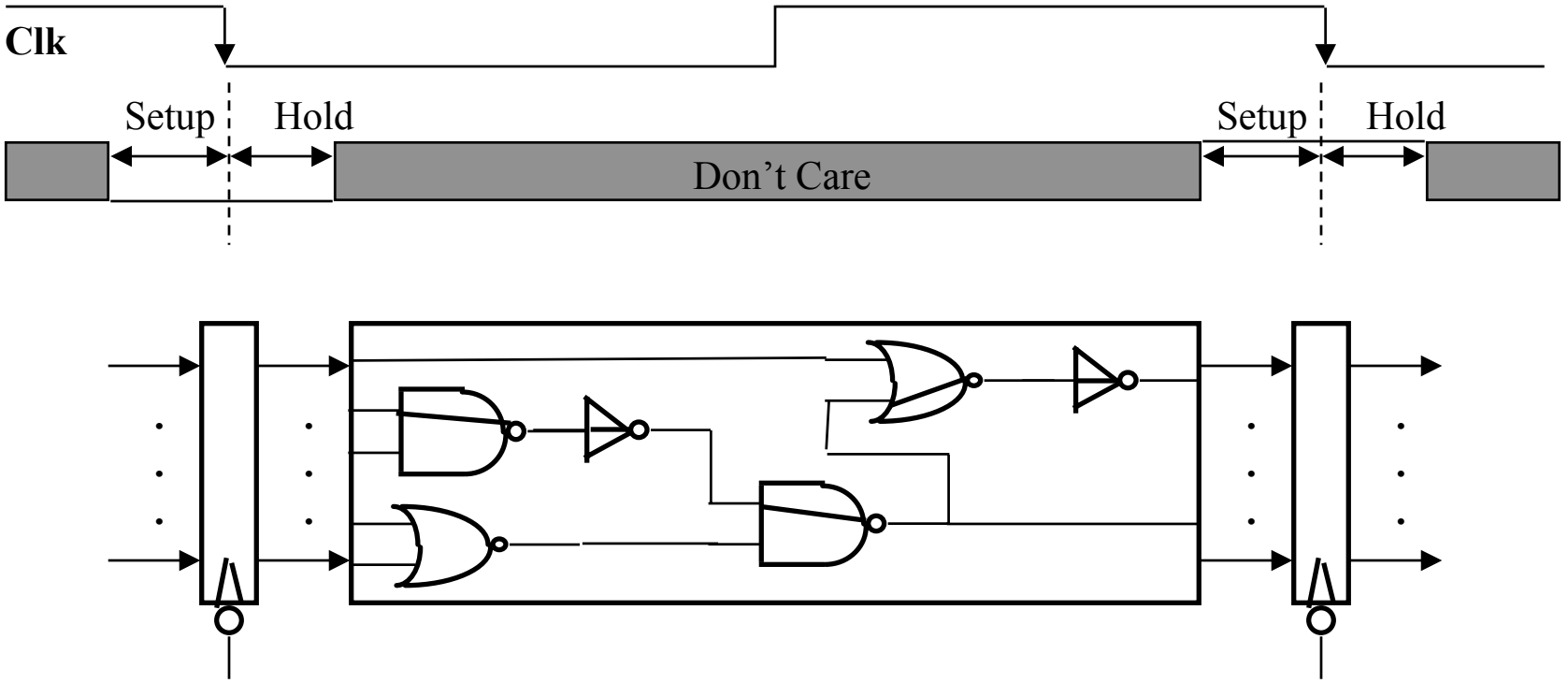


- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is selected by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - The CLK input is a factor ONLY during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

Administrivia

- Lab #2 Due Monday 9/15 before midnight
  - Demo during Friday discussion in 119 Cory
  - If not done Friday, schedule demo with TA after deadline
- Form 4 or 5 person teams by Friday 9/12
  - Who have full teams? Needs teammates?
- Office hours in Lab
  - Mon 4 – 5:30 Jack, Mon 3 – 4:30 John
- Dave's office hours Tue 3:30 – 5
- Reading: Sections 5.1 to 5.4 in Beta ed.

Clocking Methodology



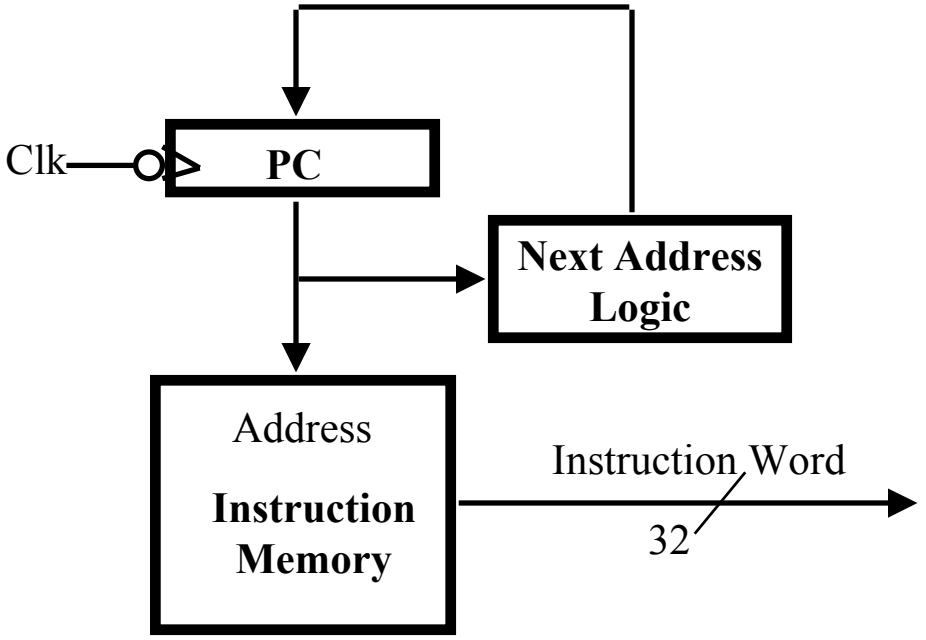
- All storage elements are clocked by the same clock edge
- Cycle Time = CLK-to-Q + Longest Delay Path + Setup + Clock Skew
- (CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time

Step 3: Assemble Datapath meeting our requirements

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

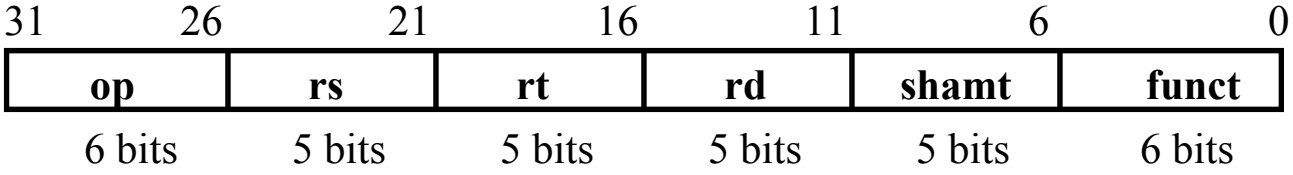
3a: Overview of the Instruction Fetch Unit

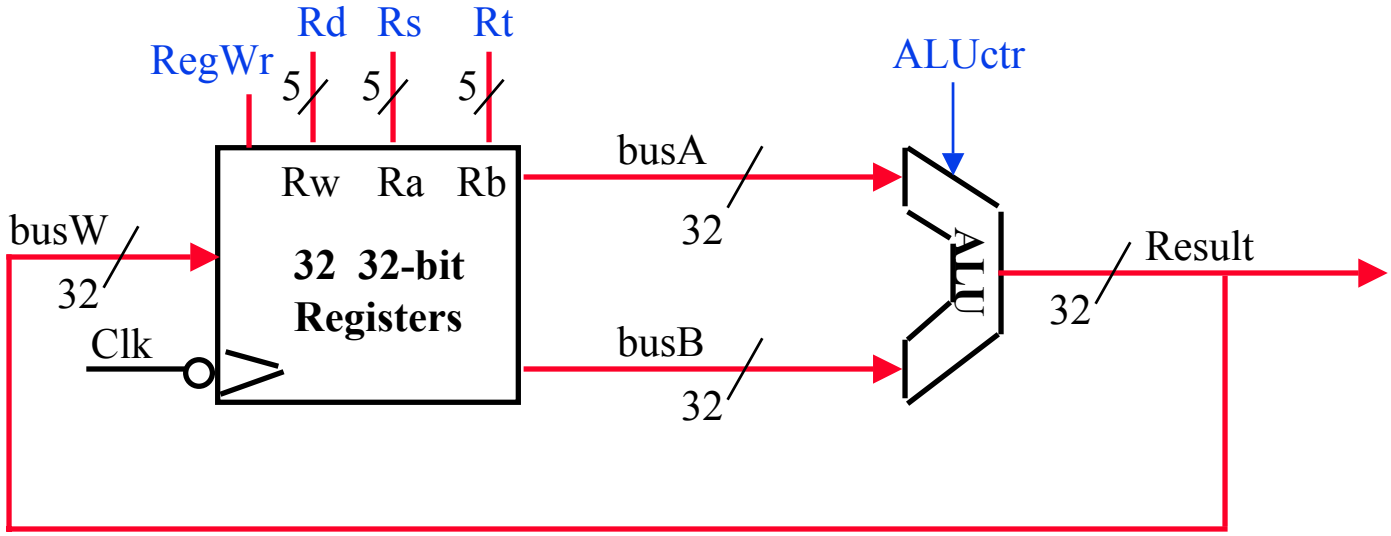
° The common operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC <= PC + 4
    - Branch and Jump: PC <= "something else"



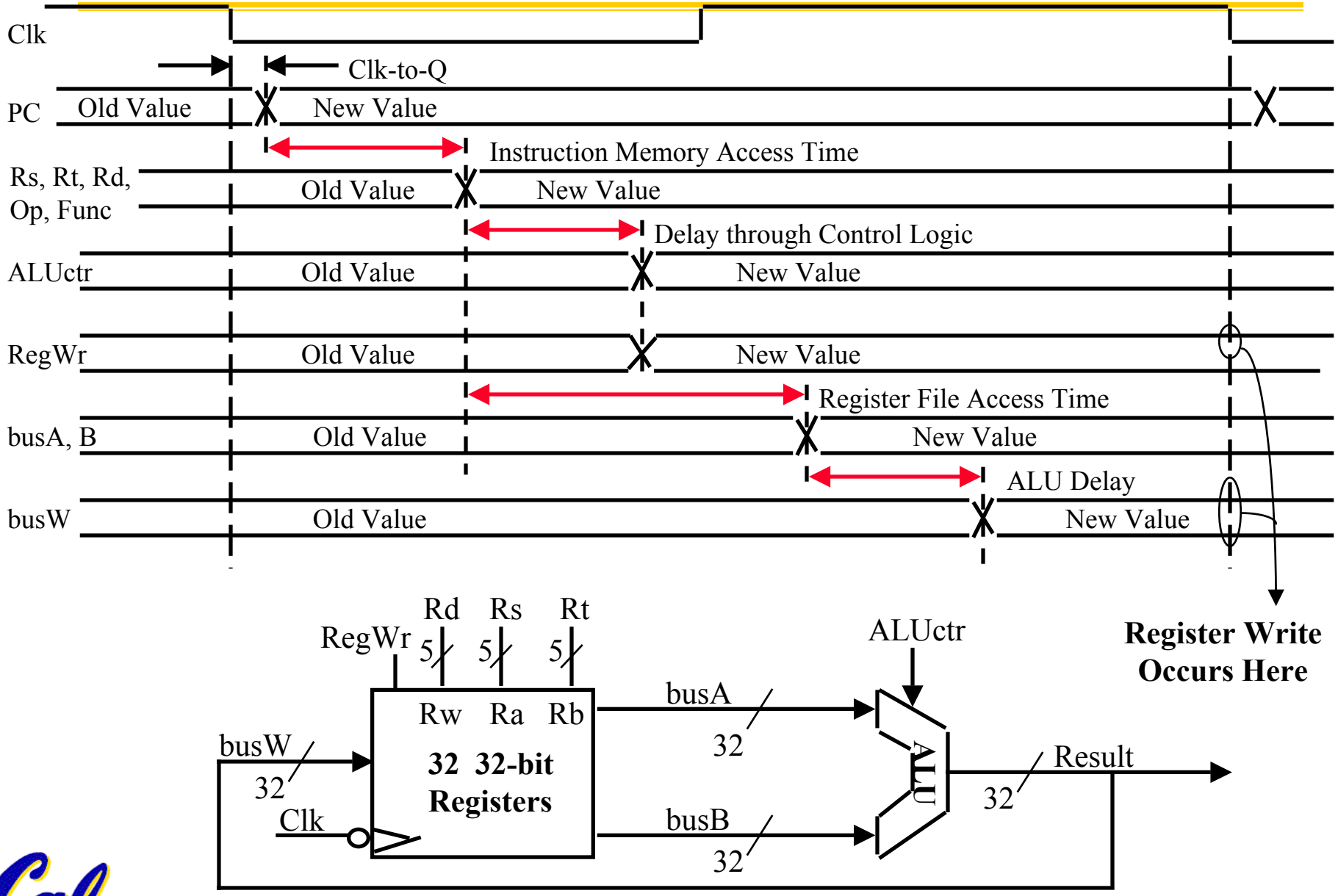
3b: Add & Subtract

- 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 after decoding the instruction



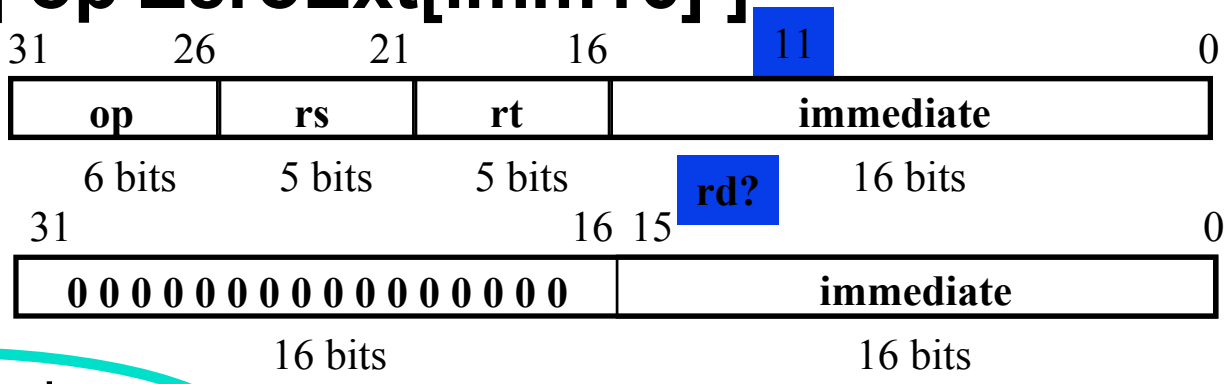


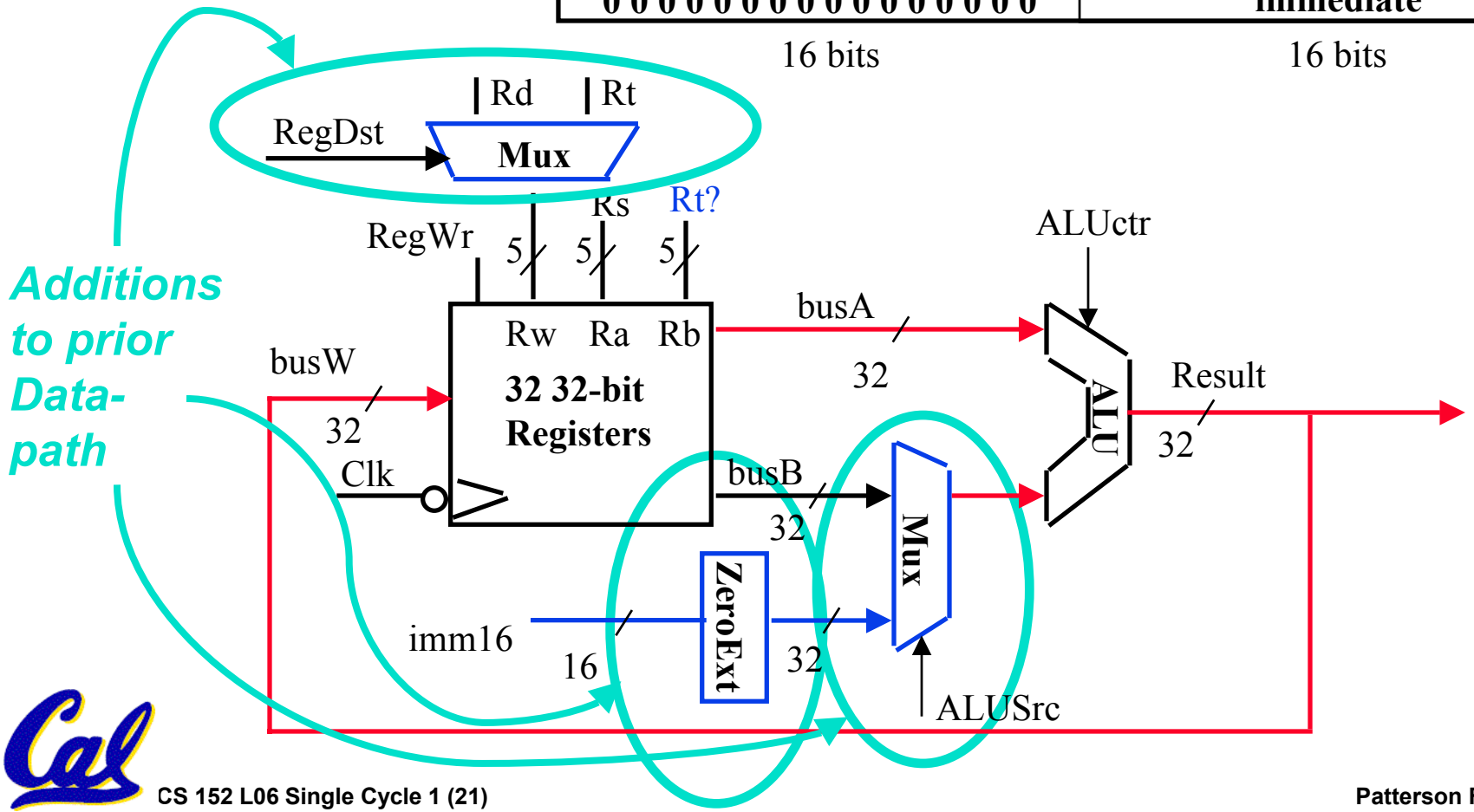
Register-Register Timing: One complete cycle



3c: Logical Operations with Immediate

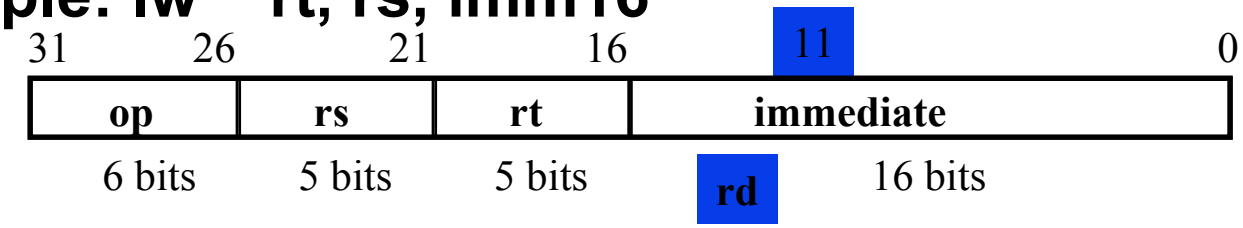
° R[rt] <= R[rs] op ZeroExt[imm16]]

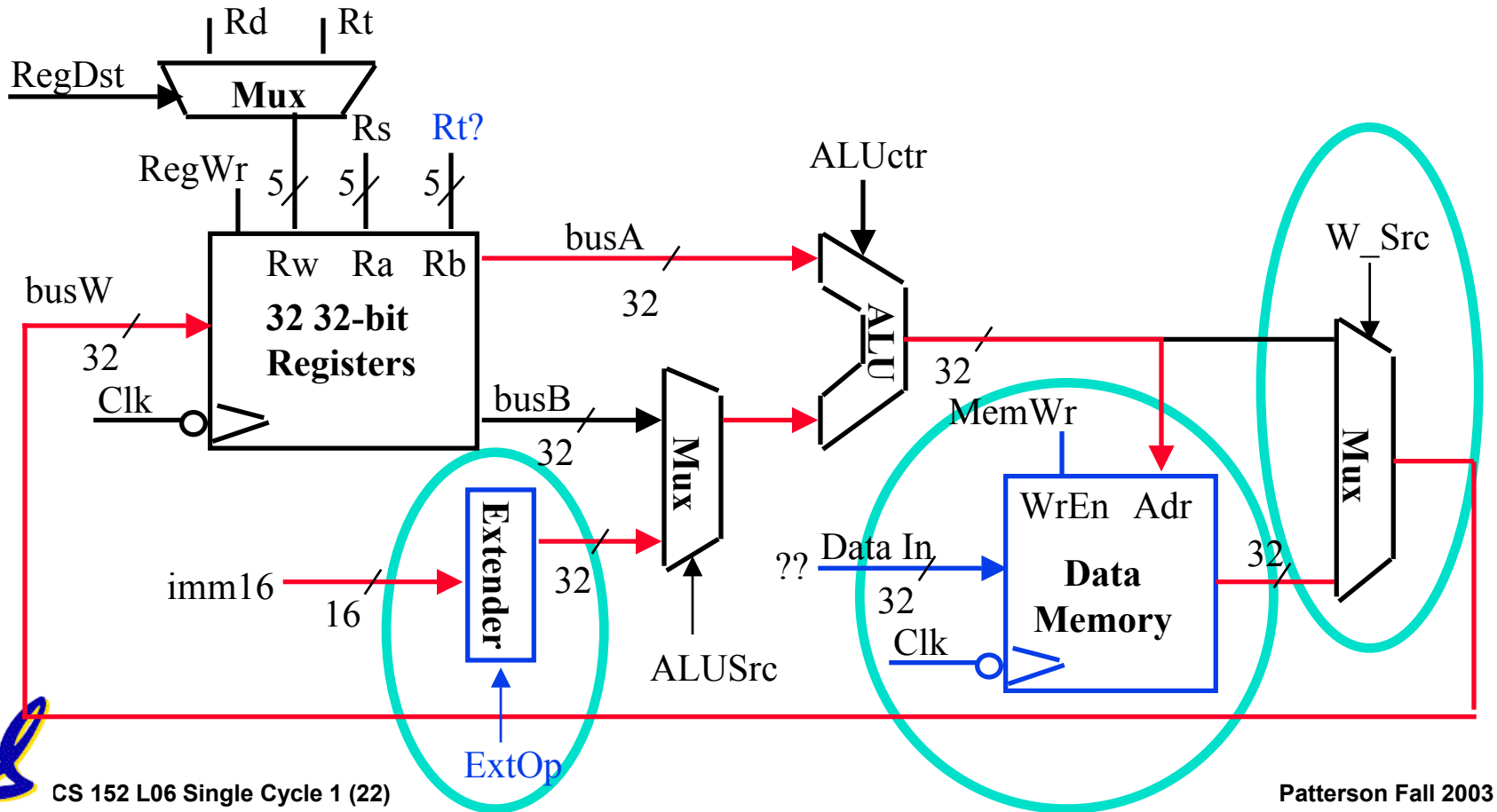




3d: Load Operations

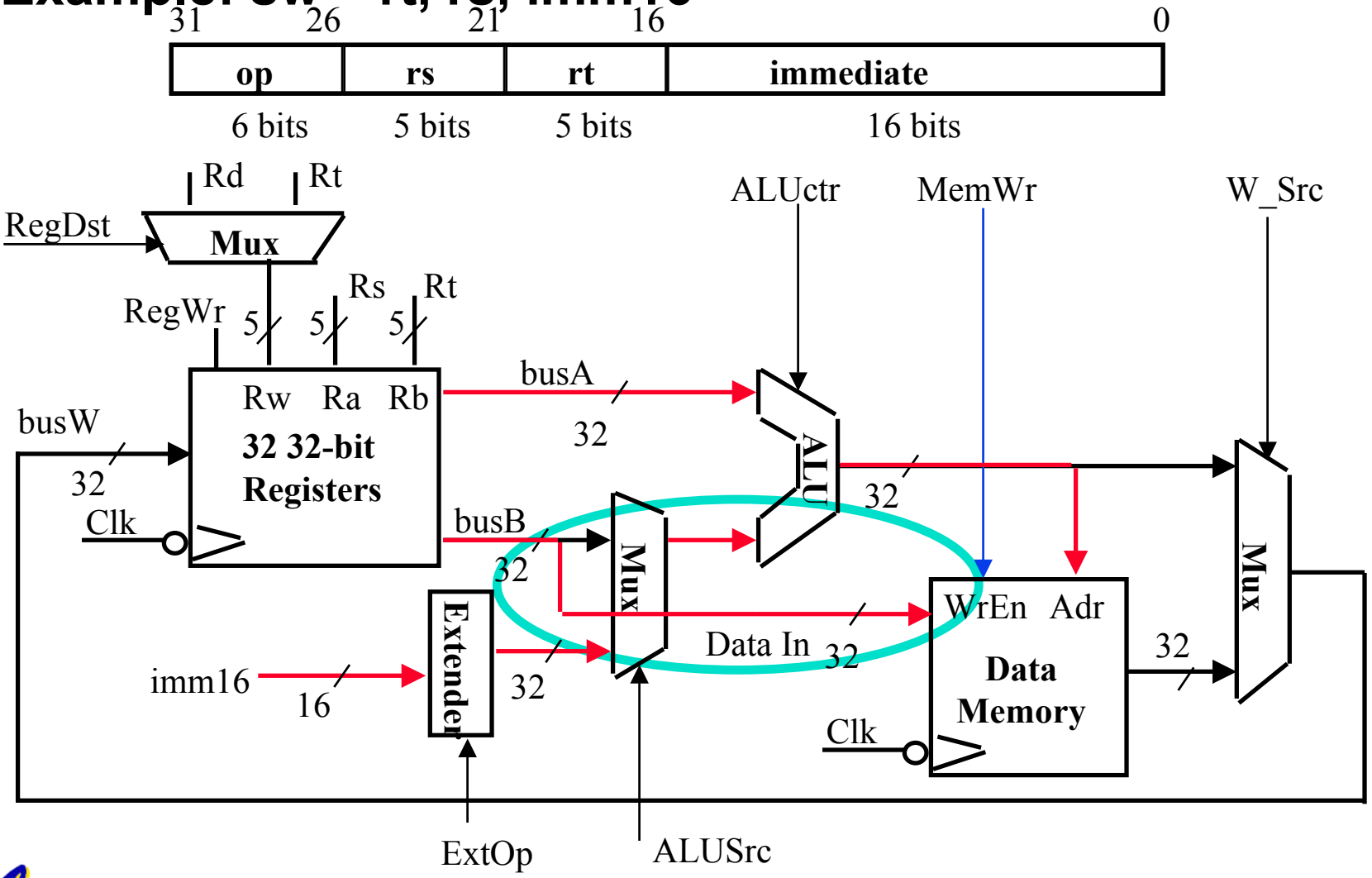
° R[rt] <= Mem[R[rs] + SignExt[imm16]]
Example: lw rt, rs, imm16



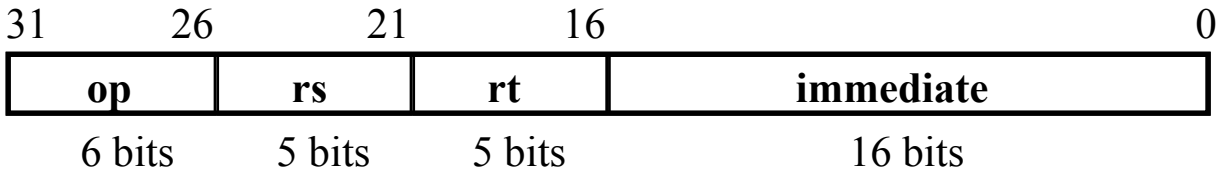


3e: Store Operations

° **Mem[R[rs] + SignExt[imm16] <= R[rt]]**
Example: sw rt, rs, imm16



3f: The Branch Instruction



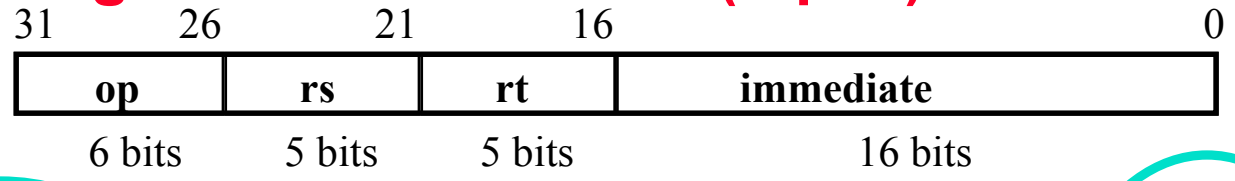
° beq rs, rt, imm16

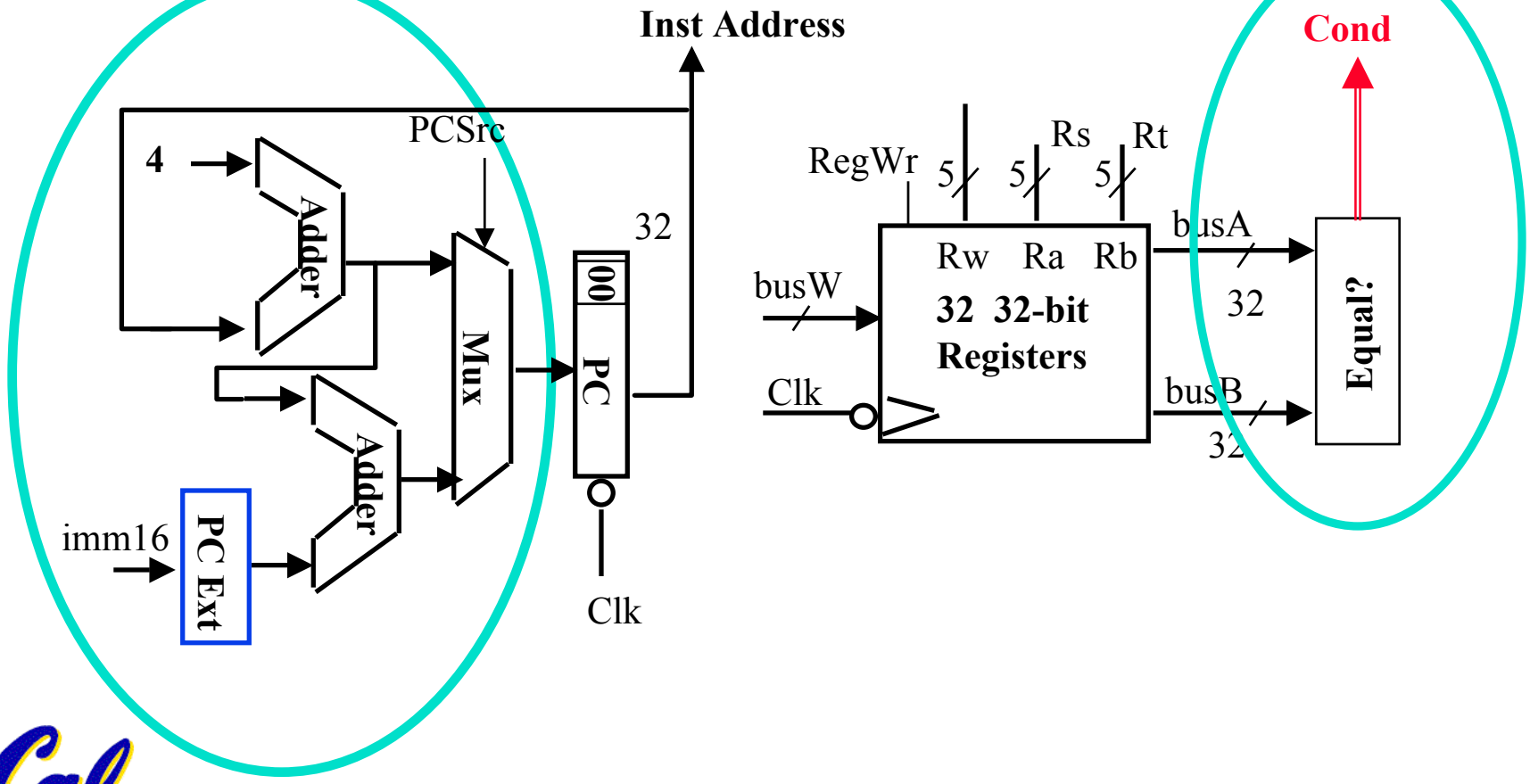
- mem[PC] Fetch the instruction from memory
- Equal <= (R[rs] == R[rt]) Calculate the branch condition
- if (Equal) Calculate the next instruction's address
  - PC <= PC + 4 + { SignExt(imm16) , 2b00 }
- else
  - PC <= PC + 4

Datapath for Branch Operations

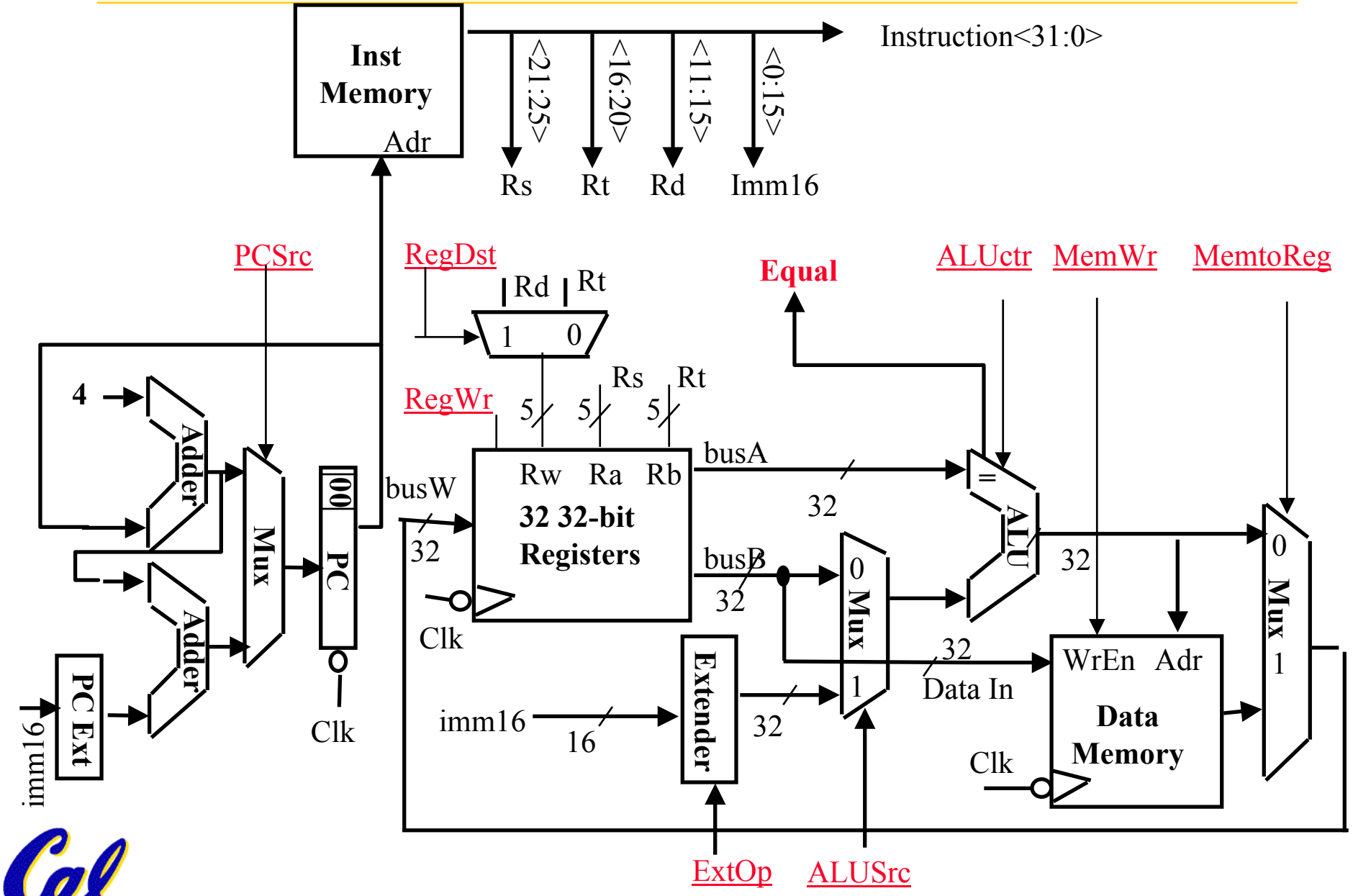
° beq rs, rt, imm16

Datapath generates condition (equal)





Putting it All Together: A Single Cycle Datapath



Lectures vs. Chapter 5 Beta 3/e

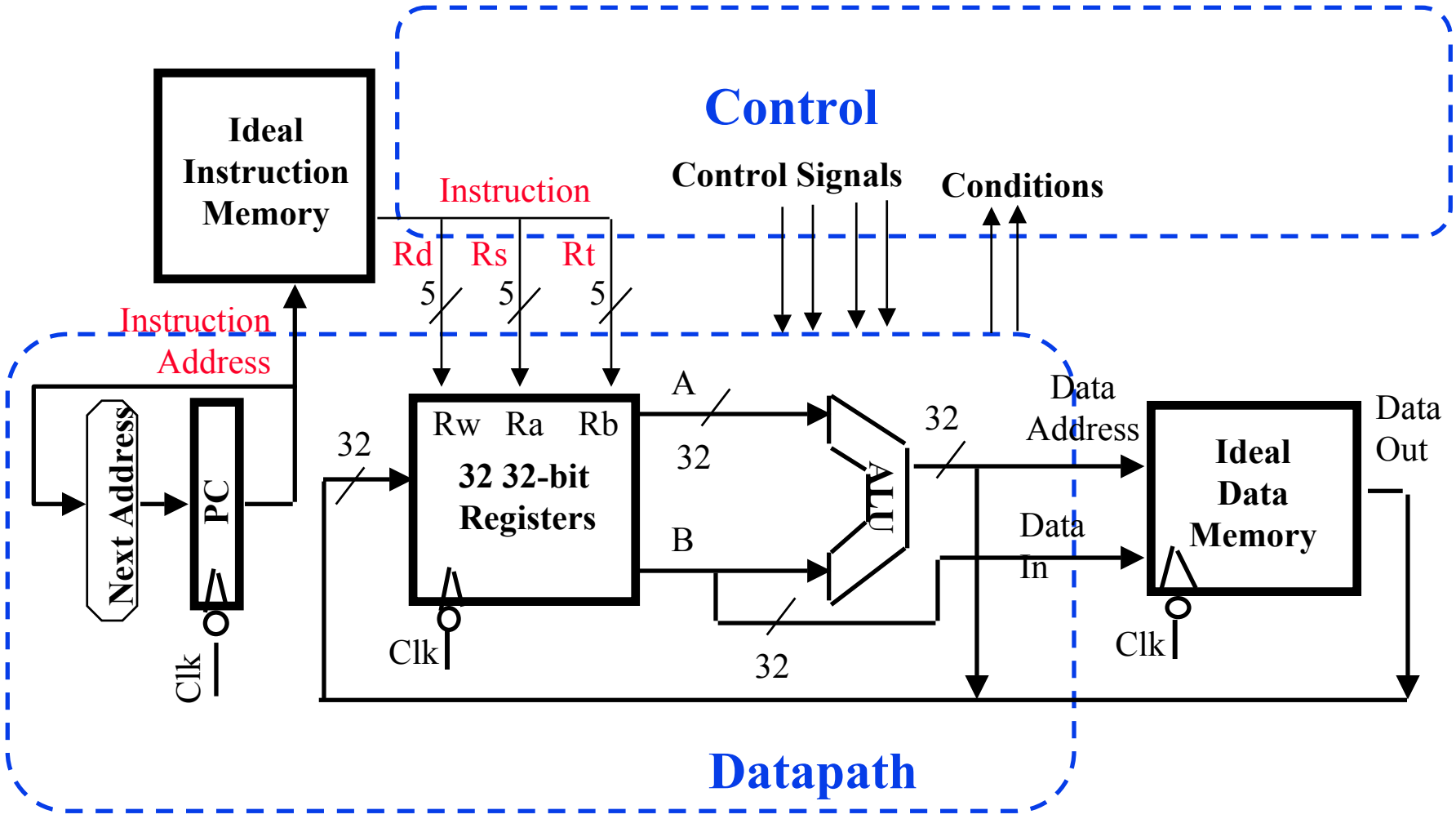
Lectures

- MIPS-lite subset:
  - AddU, SubU, LW, SW
  - BEQ, ORI
- Control lines names
  - MemtoReg, PCSrc, ALUSrc, RegDst
  - MemWr, RegWr
  - ExtOp (zero extend or sign extend)
  - ALUctr 3 bits (no NOR)
  - MemWr=0 => MemRead

Book

- MIPS-lite subset:
  - AddU, SubU, LW, SW
  - BEQ, OR
  - AND, SLT, J
- Control lines names
  - MemtoReg, PCSrc, ALUSrc, RegDst
  - MemWrite, RegWrite
  - No ExtOp since subset immediates sign extend
  - ALUoperation 4 bits
  - MemRead & MemWrite

An Abstract View of the Implementation



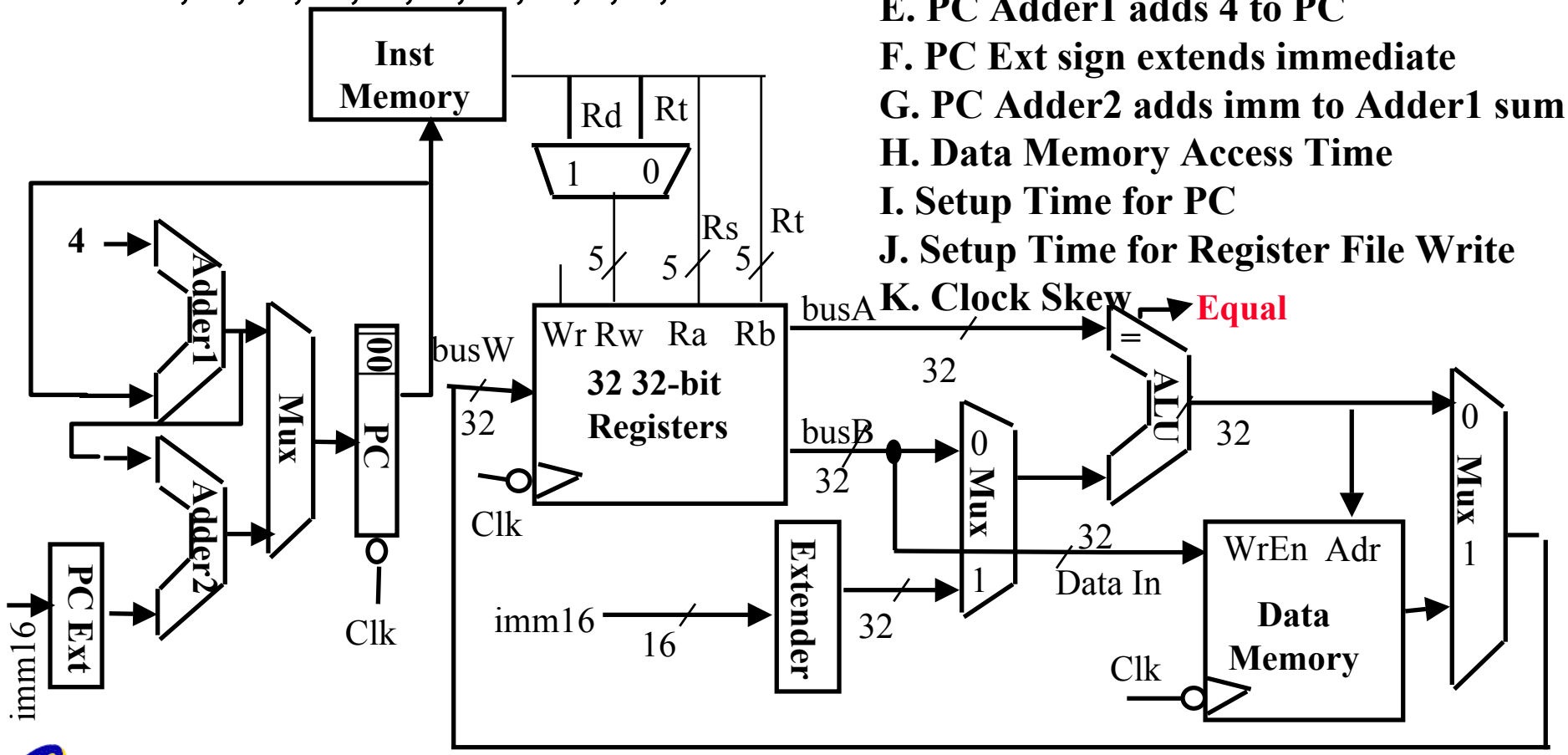
Steps 4 & 5: Implement the control

Next Time!

Peer Instruction: What is critical path for BEQ?

1. A, B, C, D, I, K
2. A, B, C, D, G, I, K
3. A, B, C, D, E, F, G, I, J, K
4. A, B, C, D, E, F, G, H, I, J, K

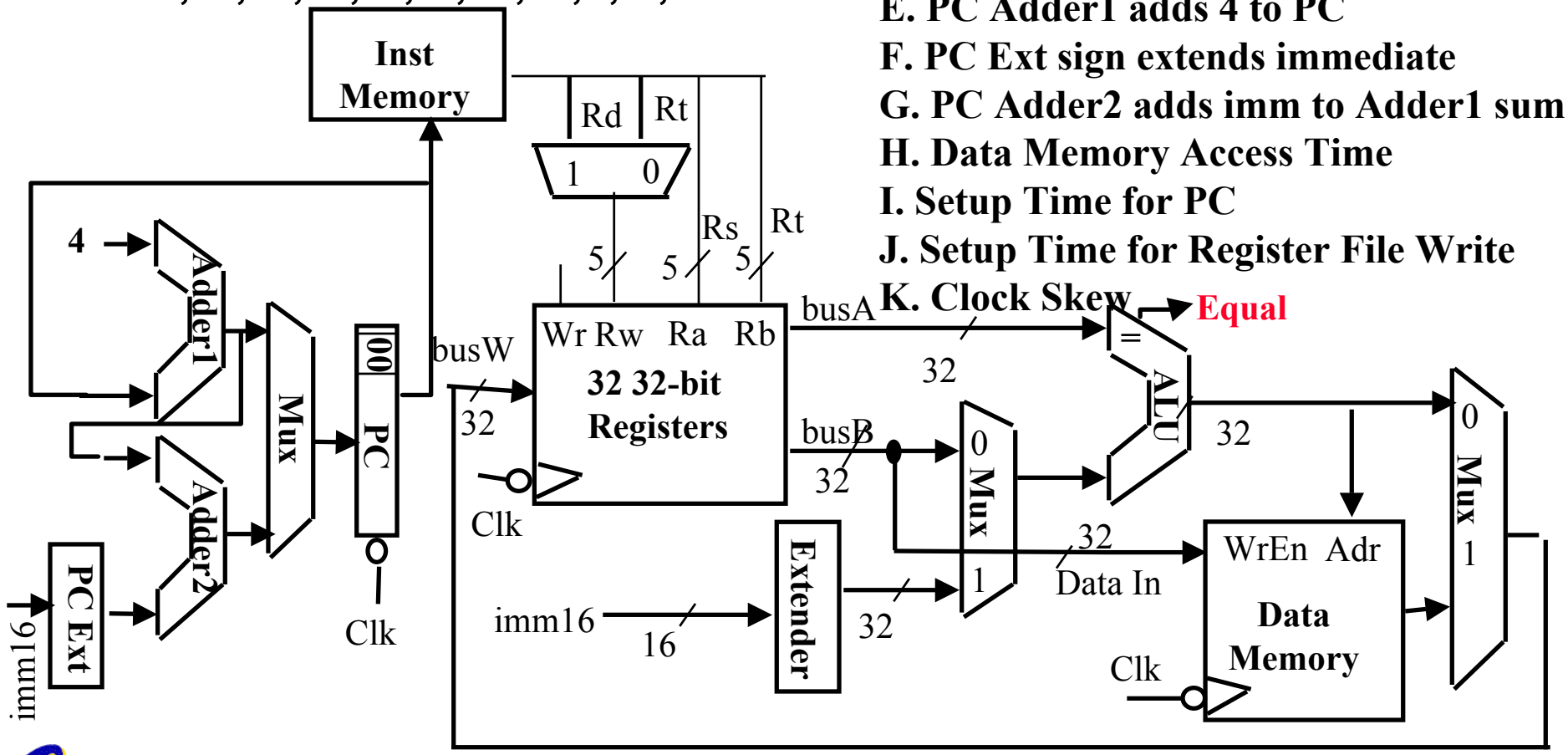
A. PC's Clk-to-Q
B. Instruction Memory's Access Time
C. Register File's Access Time
D. ALU to Perform a 32-bit Operation
E. PC Adder1 adds 4 to PC
F. PC Ext sign extends immediate
G. PC Adder2 adds imm to Adder1 sum
H. Data Memory Access Time
I. Setup Time for PC
J. Setup Time for Register File Write
K. Clock Skew



Peer Instruction: What is critical path for LW?

1. A, B, C, D, H, J, K
2. A, B, C, D, H, I, J, K
3. A, B, C, D, E, H, I, J, K
4. A, B, C, D, E, F, G, H, I, J, K

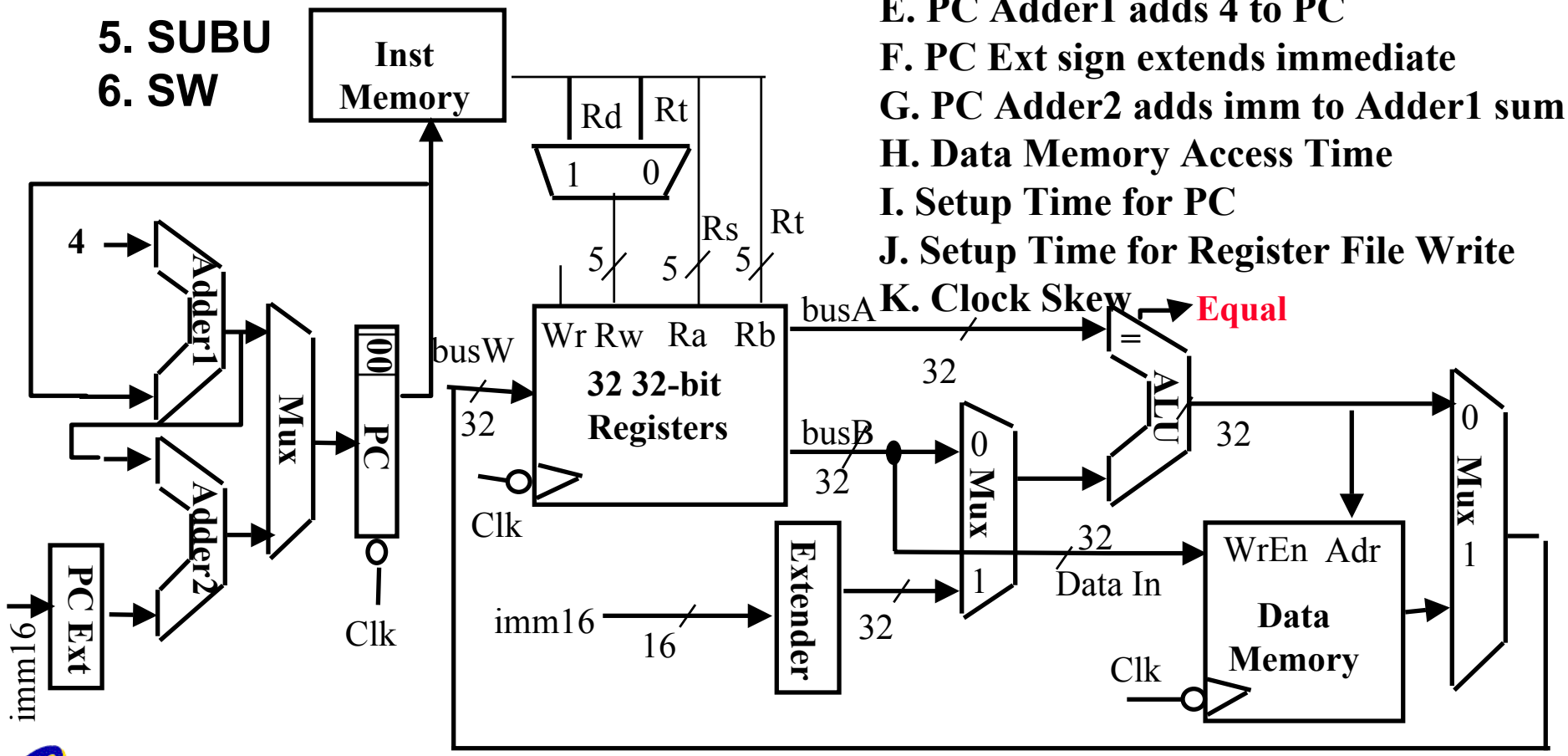
A. PC's Clk-to-Q
B. Instruction Memory's Access Time
C. Register File's Access Time
D. ALU to Perform a 32-bit Operation
E. PC Adder1 adds 4 to PC
F. PC Ext sign extends immediate
G. PC Adder2 adds imm to Adder1 sum
H. Data Memory Access Time
I. Setup Time for PC
J. Setup Time for Register File Write
K. Clock Skew



Peer Instruction: Which has longest critical path?

1. ADDU
2. BEQ
3. LW
4. ORI
5. SUBU
6. SW

A. PC's Clk-to-Q
B. Instruction Memory's Access Time
C. Register File's Access Time
D. ALU to Perform a 32-bit Operation
E. PC Adder1 adds 4 to PC
F. PC Ext sign extends immediate
G. PC Adder2 adds imm to Adder1 sum
H. Data Memory Access Time
I. Setup Time for PC
J. Setup Time for Register File Write
K. Clock Skew



Why should you keep a design notebook?

- Keep track of the design decisions and the reasons behind them
  - Otherwise, it will be hard to debug and/or refine the design
  - Write it down so that can remember in long project: 2 weeks ->2 yrs
  - Others can review notebook to see what happened
- Record insights you have on certain aspect of the design as they come up
- Record of the different design & debug experiments
  - Memory can fail when very tired
- Industry practice: learn from others mistakes

Why do we keep it on-line?

- You need to force yourself to take notes!
  - Open a window and leave an editor running while you work
    1) Acts as reminder to take notes
    2) Makes it easy to take notes
  - 1) + 2) => will actually do it
- Take advantage of the window system's "cut and paste" features
- It is much easier to read your typing than your writing
- Also, paper log books have problems
  - Limited capacity => end up with many books
  - May not have right book with you at time vs. networked screens
  - Can use computer to search files/index files to find what looking for

How should you do it?

- Keep it simple
  - DON'T make it so elaborate that you won't use (fonts, layout, ...)
- Separate the entries by dates
  - type "`date`" command in another window and cut&paste
- Start day with problems going to work on today
- Record output of simulation into log with cut&paste; add date
  - May help sort out which version of simulation did what
- Record key email with cut&paste
- Record of what works & doesn't helps team decide what went wrong after you left
- Index: write a one-line summary of what you did at end of each day

On-line Notebook Example

- Refer to the handout "Example of On-Line Log Book" on CS 152 home page:

`~cs152/handouts/online_notebook_example.html`

1st page of On-line notebook (Index + Wed. 9/6/95)

```
* Index ========================================================

Wed Sep  6 00:47:28 PDT 1995 - Created the 32-bit comparator component
Thu Sep  7 14:02:21 PDT 1995 - Tested the comparator
Mon Sep 11 12:01:45 PDT 1995 - Investigated bug found by Bart in
                               comp32 and fixed it
+ ========================================================
Wed Sep  6 00:47:28 PDT 1995

  Goal: Layout the schematic for a 32-bit comparator

  I've layed out the schemtatics and made a symbol for the comparator.
  I named it comp32.  The files are
      ~/wv/proj1/sch/comp32.sch
      ~/wv/proj1/sch/comp32.sym

Wed Sep  6 02:29:22 PDT 1995
- ========================================================
```

- Add 1 line index at front of log file at end of each session: date+summary
- Start with date, time of day + goal
- Make comments during day, summary of work
- End with date, time of day (and add 1 line summary at front of file)

2nd page of On-line notebook (Thursday 9/7/95)

```
+ ============================================================
Thu Sep  7 14:02:21 PDT 1995

  Goal: Test the comparator component

  I've written a command file to test comp32.  I've placed it
  in ~/wv/proj1/diagnostics/comp32.cmd.

  I ran the command file in viewsim and it looks like the comparator
  is working fine.  I saved the output into a log file called
  ~/wv/proj1/diagnostics/comp32.log

  Notified the rest of the group that the comparator
  is done.

Thu Sep  7 16:15:32 PDT 1995
- ============================================================
```

3rd page of On-line notebook (Monday 9/11/95)

```
+ ============================================================
Mon Sep 11 12:01:45 PDT 1995

  Goal: Investigate bug discovered in comp32 and hopefully fix it

  Bart found a bug in my comparator component. He left the following
  e-mail.

-------------------
From bart@simpsons.residence Sun Sep 10 01:47:02 1995
Received: by wayne.manor (NX5.67e/NX3.0S)
        id AA00334; Sun, 10 Sep 95 01:47:01 -0800
Date: Wed, 10 Sep 95 01:47:01 -0800
From: Bart Simpson <bart@simpsons.residence>
To: bruce@wanye.manor, old_man@gokuraku, hojo@sanctuary
Subject: [cs152] bug in comp32
Status: R

Hey Bruce,
I think there's a bug in your comparator.
The comparator seems to think that ffffffff and fffffff7 are equal.

Can you take a look at this?
Bart
---------------
```

4th page of On-line notebook (9/11/95 contd)

```
I verified the bug. here's a viewsim of the bug as it appeared..
  (equal should be 0 instead of 1)
------------------
SIM>stepsize 10ns
SIM>v a_in A[31:0]
SIM>v b_in B[31:0]
SIM>w a_in b_in equal
SIM>a a_in ffffffff\h
SIM>a b_in fffffff7\h
SIM>sim
time =    10.0ns  A_IN=FFFFFFFF\H B_IN=FFFFFFF7\H EQUAL=1
Simulation stopped at 10.0ns.
-------------------
```

Ah. I've discovered the bug. I mislabeled the 4th net in the comp32 schematic.

I corrected the mistake and re-checked all the other labels, just in case.

I re-ran the old diagnostic test file and tested it against the bug Bart found. It seems to be working fine. hopefully there aren't any more bugs:)

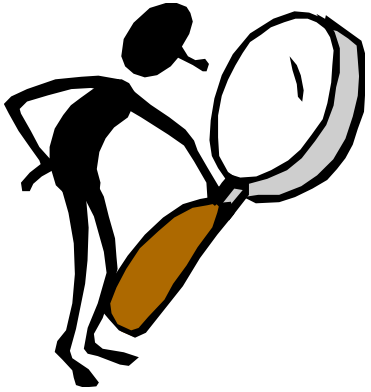
5th page of On-line notebook (9/11/95 contd)

```
  On second inspectation of the whole layout, I think I can
  remove one level of gates in the design and make it go faster.
  But who cares! the comparator is not in the critical path
  right now. the delay through the ALU is dominating the critical
  path.  so unless the ALU gets a lot faster, we can live with
  a less than optimal comparator.

  I e-mailed the group that the bug has been fixed

Mon Sep 11 14:03:41 PDT 1995
- ============================================================
```

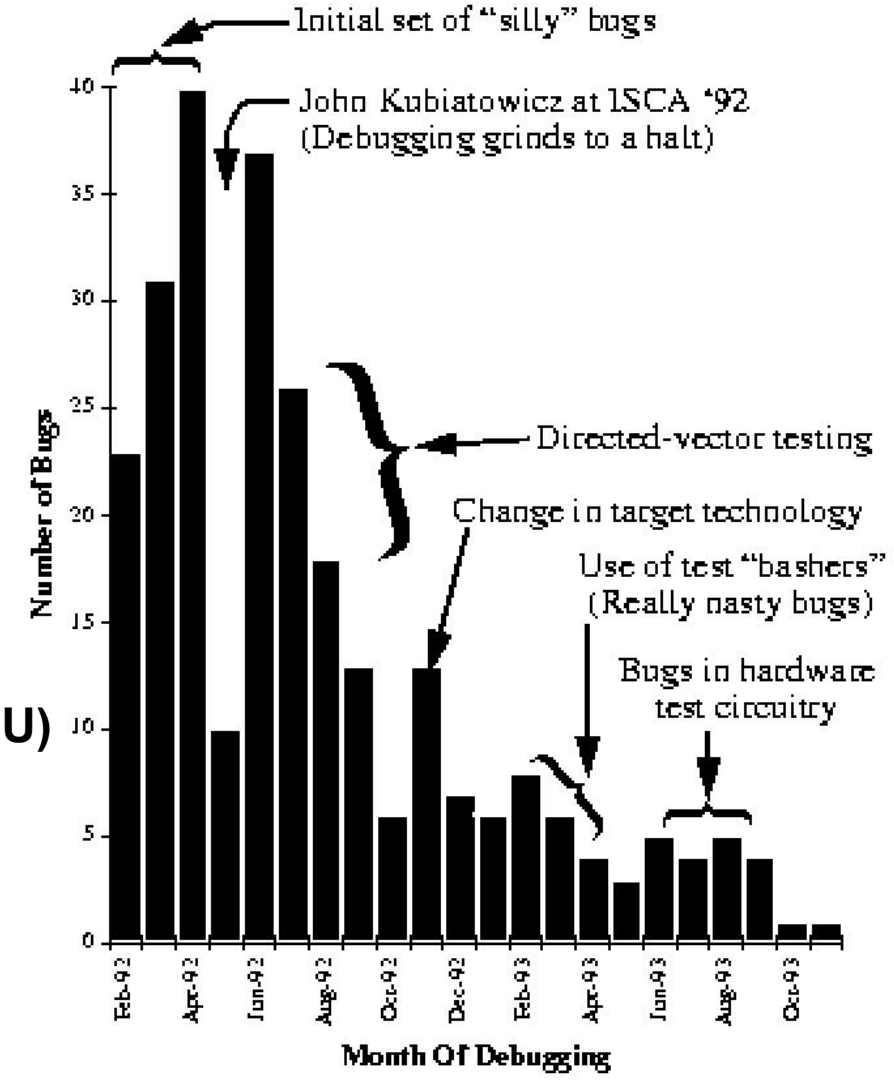
- Perhaps later critical path changes;
- What was idea to make comparator faster?
- Check on-line notebook!

Added benefit: cool post-design statistics

Sample graph from the Alewife project:



- **For the Communications and Memory Management Unit (CMMU)**
- **These statistics came from on-line record of bugs**



Lecture 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 (Next Lecture)
- MIPS makes it easier
  - Instructions same size; Source registers, immediates always in same place
  - Operations always on registers/immediates
- Single cycle datapath => CPI=1, CCT => long
- On-line Design Notebook
  - Open a window and keep an editor running while you work;cut&paste
  - Former CS 152 students (and TAs) say they use on-line notebook for programming as well as hardware design; one of most valuable skills