CS152
Computer Architecture and Engineering
Lecture 8: Designing a Single Cycle Datapath

September 24, 1997

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

lecture slides: http://www-inst.eecs.berkeley.edu/~cs152/

©UCB, DAP' 97

Outline of Today's Lecture

- Recap (5 minutes)
- Design a processor: step-by-step
- Requirements of the Instruction Set
- Questions and Administrative Matters (5 minutes)
- Components and Clocking
- Assembling an Adequate Datapath
- Break (5 minutes)
- Controling the datapath

©UCB, DAP' 97

Review: Floating Point

- Define a number system representation that behaves very much like real numbers
  - large dynamic range (-10^{38} to + 10^{38})
  - fine precision (10^{-38})
- Use "scientific notation"
  - normal form for the mantissa ($\pm$1.xxxxxxxx)
  - exponent specifies position of the binary point)
- Arithmetic algorithms just as in high school
  - Addition:
    - shift number with smaller exponent right till have same scale
    - add mantissas
    - renormalize
  - Subtraction? Multiplication? Division?
-

©UCB, DAP' 97

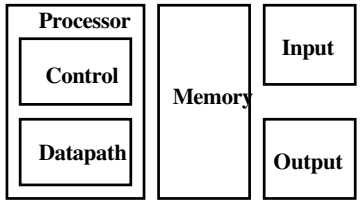
More Review

- IEEE 754: rounding as if computed to full prec. and rounded
  - requires carry, guard, and sticky bit
- IEEE 754: denorms, infinities, and NANs
- Divide can use same hardware as multiply: Hi & Lo registers in MIPS
- Bits have no inherent meaning: operations determine whether they are really ASCII characters, integers, floating point numbers
- Pentium: Difference between bugs that board designers must know about and bugs that potentially affect all users
  - Why not make public complete description of bugs in later category?
  - $200,000 cost in June to repair design
  - $500,000,000 loss in December in profits to replace bad parts
  - How much to repair Intel's reputation?
- What is technologists responsibility in disclosing bugs?

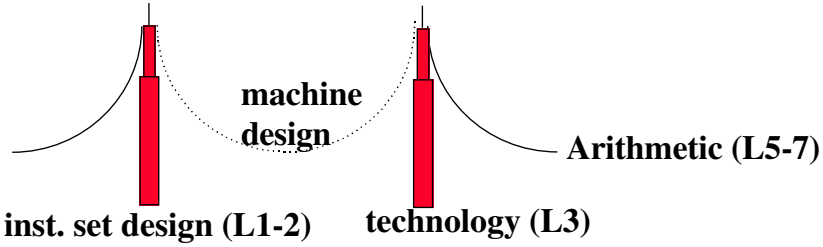
©UCB, DAP' 97

The Big Picture: Where are We Now?

- The Five Classic Components of a Computer



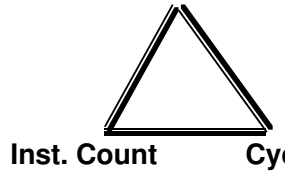
- Today's Topic: Design a Single Cycle Processor



©UCB, DAP' 97

The Big Picture: The Performance Perspective

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



- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction
- Today:
  - Single cycle processor:
    - Advantage: One clock cycle per instruction
    - Disadvantage: long cycle time

©UCB, DAP' 97

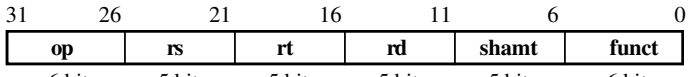
How to Design a Processor: step-by-step

- 1. Analyze instruction set => datapath <u>requirements</u>
  - 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. <u>Assemble</u> 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

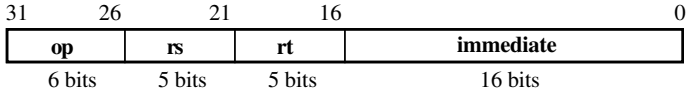
©UCB, DAP' 97

The MIPS Instruction Formats

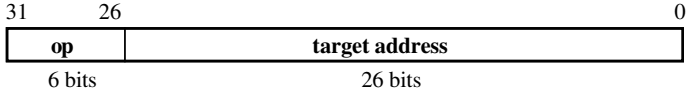
- All MIPS instructions are 32 bits long. The three instruction formats:
  - R-type



  - I-type



  - J-type

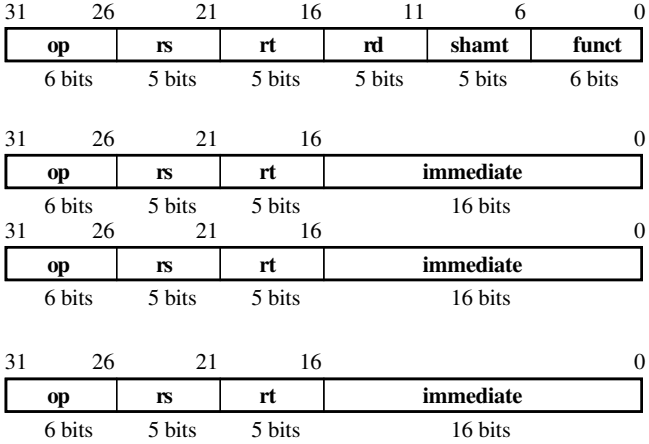


- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: target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

©UCB, DAP' 97

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



©UCB, DAP' 97

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[rs];	PC <– PC + 4
BEQ	if (R[rs] == R[rt]) then PC <– PC + sign_ext(Imm16)] || 00 else PC <– PC + 4	

©UCB, DAP' 97

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 register or extended immediate
- Add 4 or extended immediate to PC

©UCB, DAP' 97

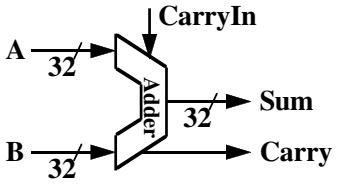
Step 2: Components of the Datapath

- Combinational Elements
- Storage Elements
  - Clocking methodology

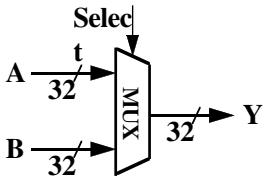
©UCB, DAP' 97

Combinational Logic Elements (Basic Building Blocks)

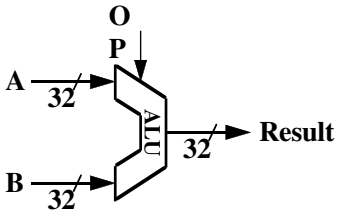
- Adder



- MUX



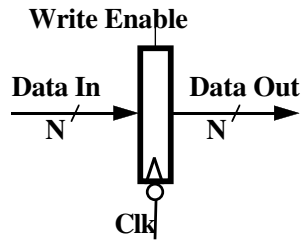
- ALU



©UCB, DAP' 97

Storage Element: Register (Basic Building Block)

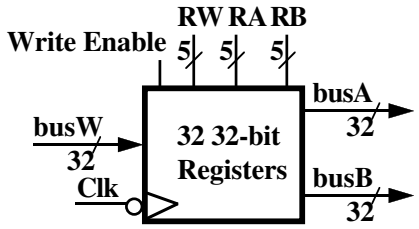
- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
  - Write Enable:
    - negated (0): Data Out will not change
    - asserted (1): Data Out will become Data In



©UCB, DAP' 97

Storage Element: Register File

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

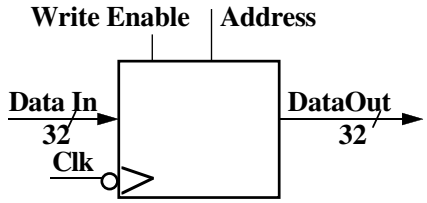


- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid => busA or busB valid after "access time."

©UCB, DAP' 97

Storage Element: Idealized Memory

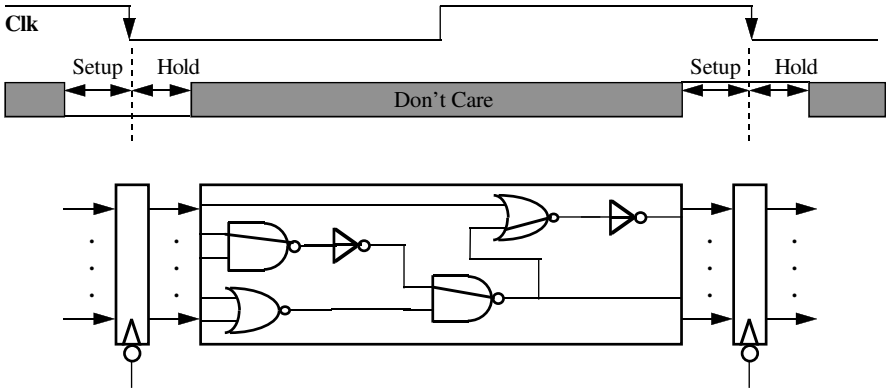
- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out



- 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."

©UCB, DAP' 97

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

©UCB, DAP' 97

Questions and Administrative Matters (5 Minutes)

- Reading Assignment 5.1-5.4
- Project team and computer access:
  - Form four or five people project team.
  - We want you to learn to work in a big team.
  - Each student must pick a permanent discussion section.
- Midterm Wedensday 10/8 in 306 Soda 5:30PM-8:30PM
  - you may bring one double-sided page of notes
  - we'll give you the opcode table from the book
  - review session Sunday September 28 1PM 306 Soda
  - previous midterms and solutions on-line for review

©UCB, DAP' 97

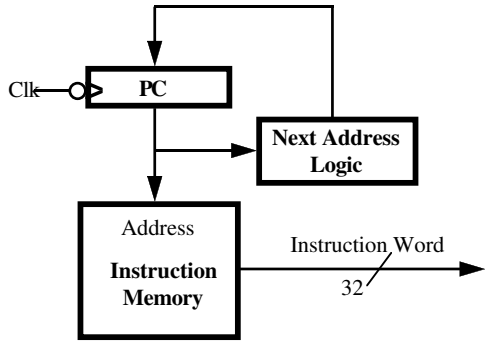
Step 3

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

©UCB, DAP' 97

3a: Overview of the Instruction Fetch Unit

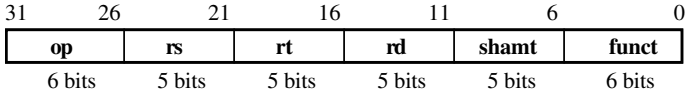
- The common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump: PC <- "something else"

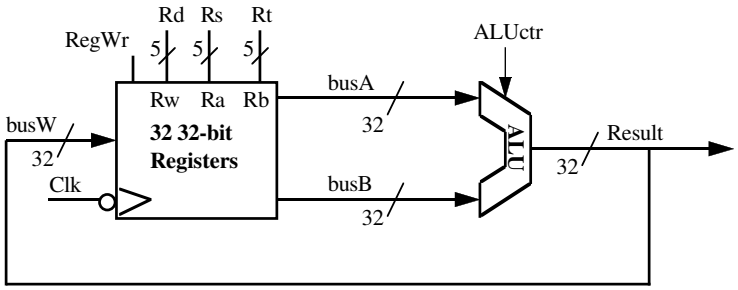


©UCB, DAP' 97

3b: Add & Subtract

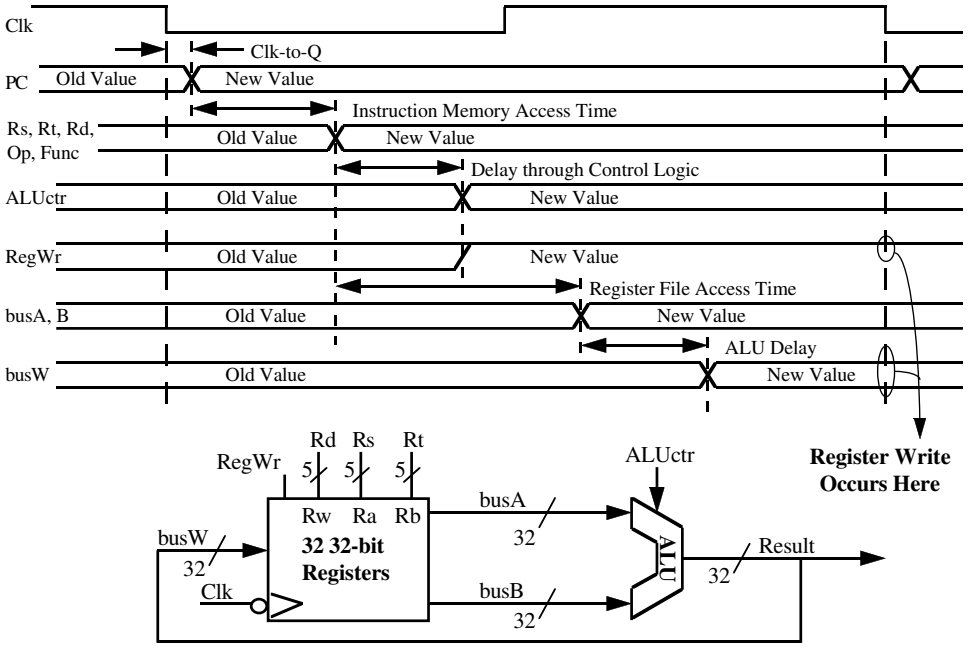
- R[rd] <- R[rs] op R[rt] Example: addU rd, rs, rt
  - Ra, Rb, and Rw come from instruction's rs, rt, and rd fields
  - ALUctr and RegWr: control logic after decoding the instruction





©UCB, DAP' 97

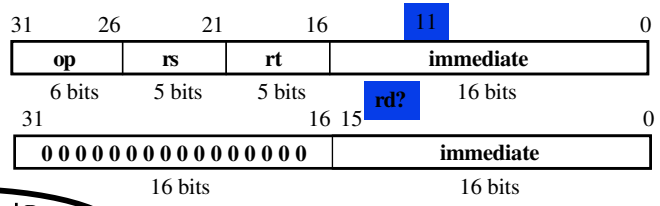
Register-Register Timing

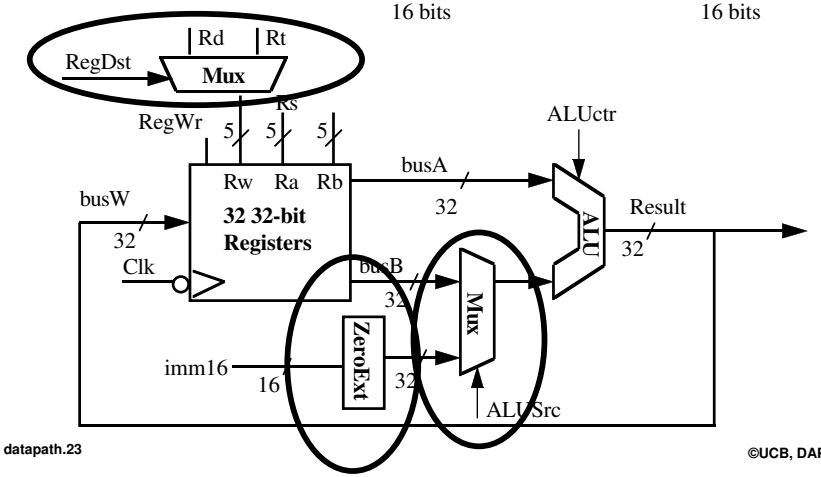


©UCB, DAP' 97

3c: Logical Operations with Immediate

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

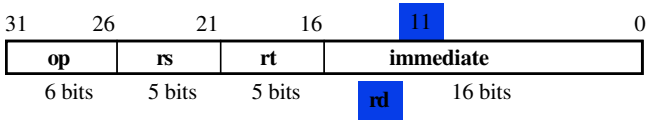


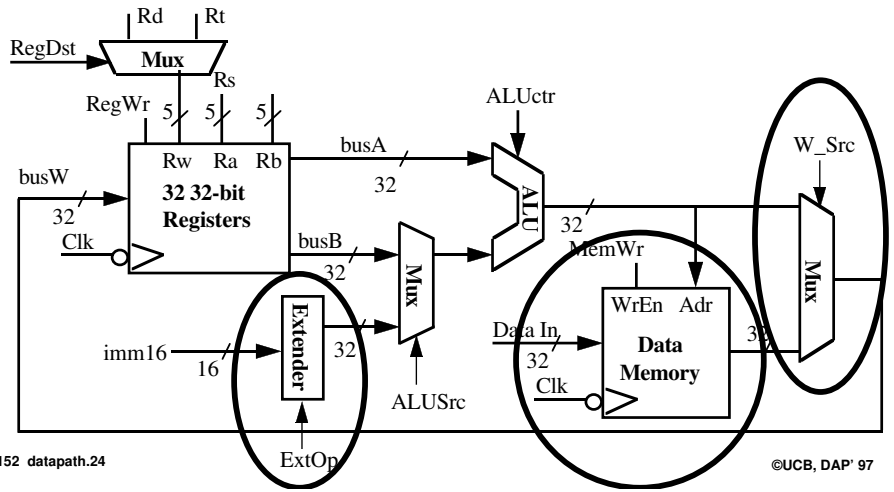


©UCB, DAP' 97

3d: Load Operations

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

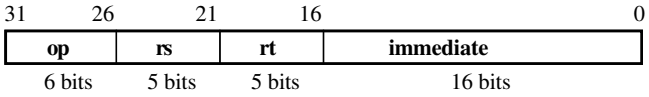


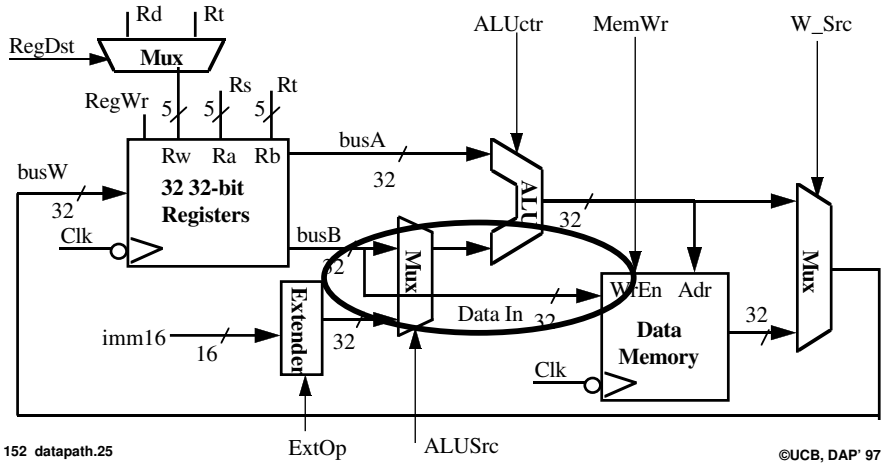


©UCB, DAP' 97

3e: Store Operations

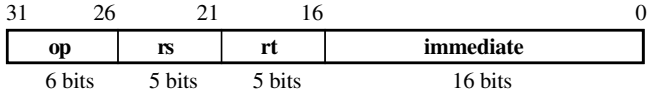
° Mem[R[rs] + SignExt[imm16] <- R[rt]] Example: sw rt, rs, imm16





©UCB, DAP' 97

3f: The Branch Instruction



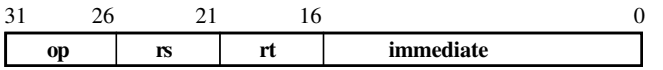
° beq rs, rt, imm16

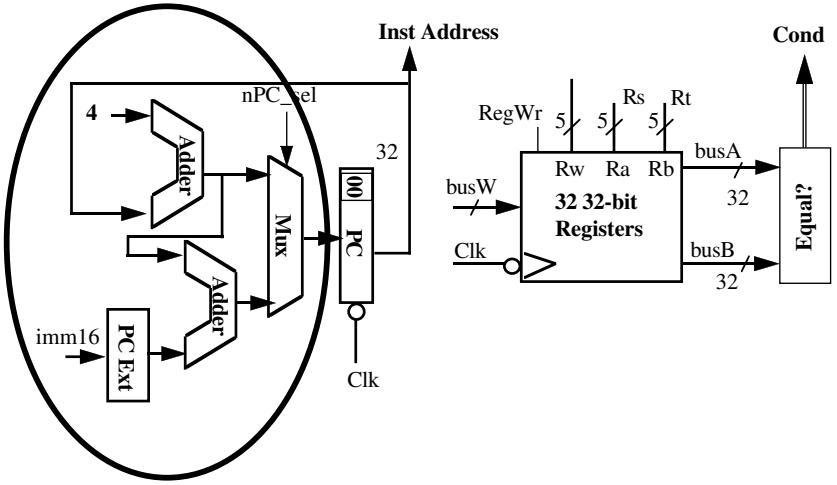
- mem[PC] Fetch the instruction from memory
- Equal <- R[rs] == R[rt] Calculate the branch condition
- if (COND eq 0) Calculate the next instruction's address
  - PC <- PC + 4 + (SignExt(imm16) x 4)
- else
  - PC <- PC + 4

©UCB, DAP' 97

Datapath for Branch Operations

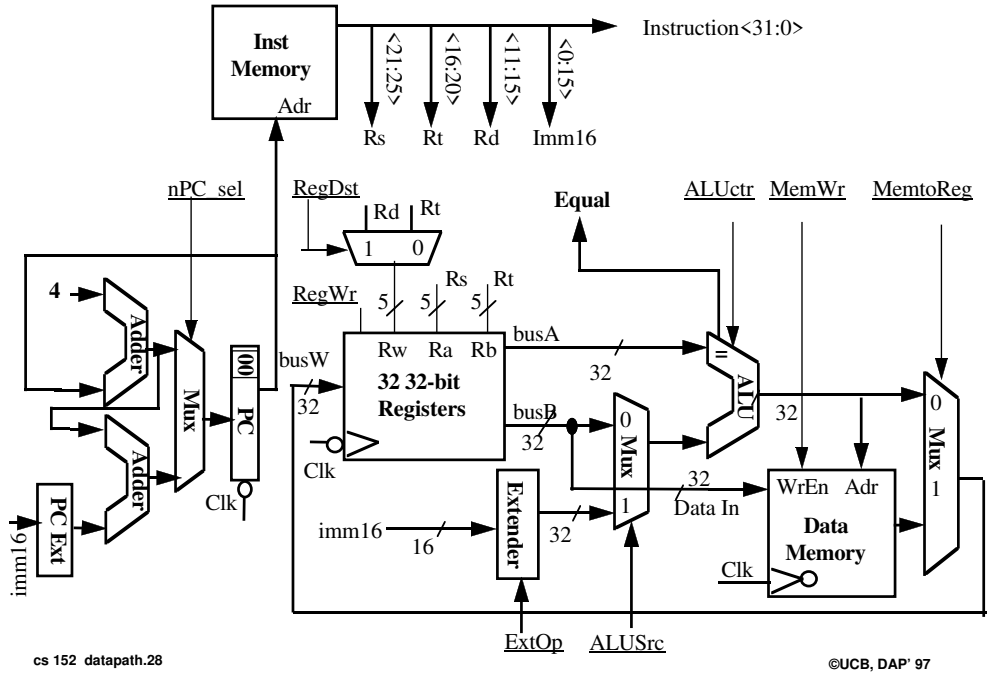
° beq rs, rt, imm16 Datapath generates condition (equal)





©UCB, DAP' 97

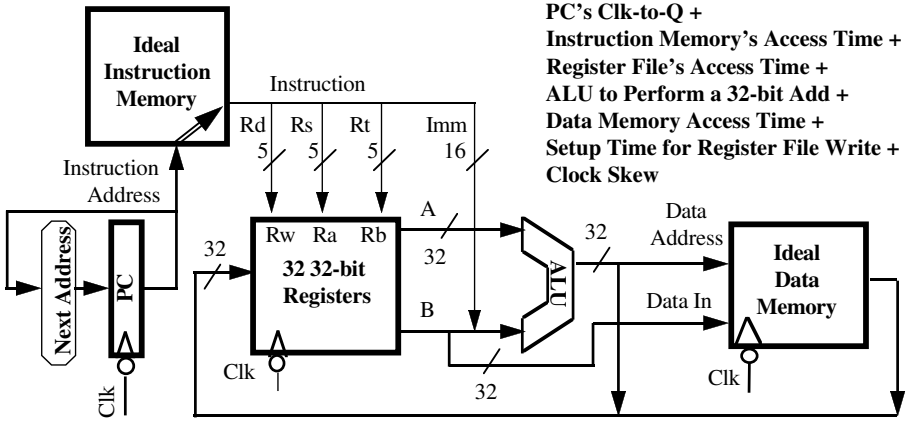
Putting it All Together: A Single Cycle Datapath



©UCB, DAP' 97

An Abstract View of the Critical Path

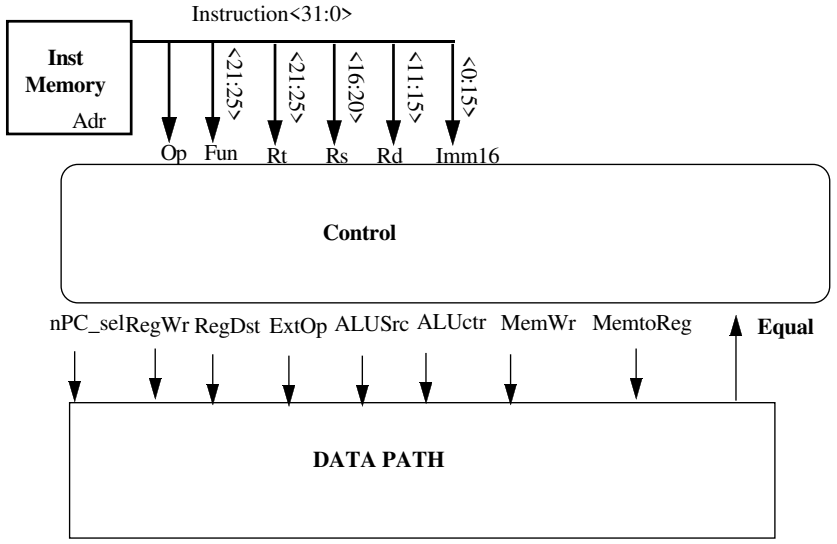
- Register file and ideal memory:
  - The CLK input is a factor ONLY during write operation
  - During read operation, behave as combinational logic:
    - Address valid => Output valid after "access time."



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

©UCB, DAP' 97

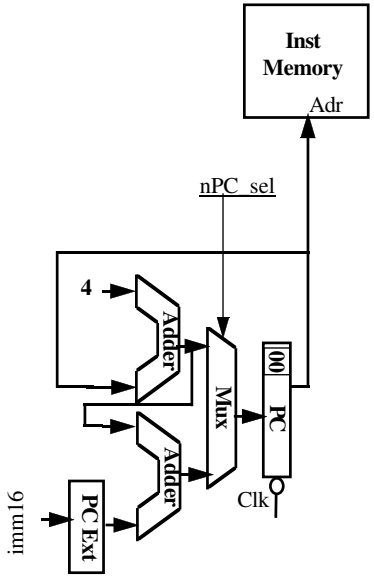
Step 4: Given Datapath: RTL -> Control



©UCB, DAP' 97

Meaning of the Control Signals (see slide 28)

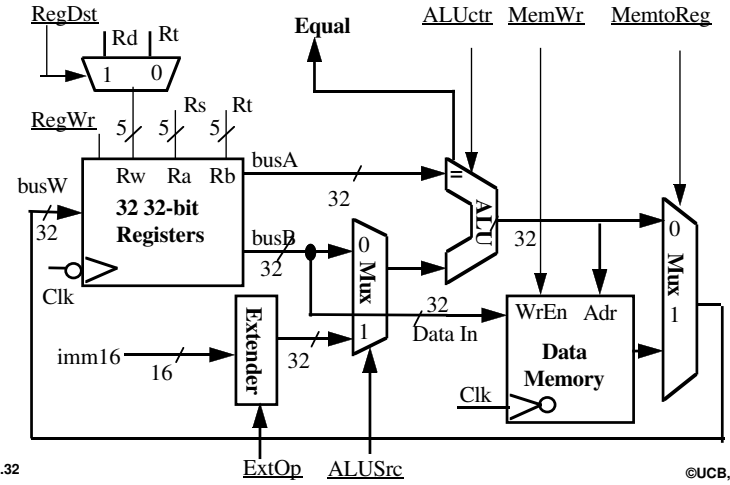
- Rs, Rt, Rd and Imed16 hardwired into datapath
- nPC_sel: 0 => PC <– PC + 4; 1 => PC <– PC + 4 + SignExt(Im16) || 00



©UCB, DAP' 97

Meaning of the Control Signals (see slide 28)

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



©UCB, DAP' 97

Control Signals

inst	Register Transfer	
ADD	R[rd] <– R[rs] + R[rt];	PC <– PC + 4

ALUsrc = RegB, ALUctr = "add", RegDst = rd, RegWr, nPC_sel = "+4"

SUB	R[rd] <– R[rs] – R[rt];	PC <– PC + 4

ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__

ORi	R[rt] <– R[rs] + zero_ext(Imm16);	PC <– PC + 4

ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__

LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4

ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__

STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rs];	PC <– PC + 4

ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__

BEQ if (R[rs] == R[rt]) then PC <– PC + sign_ext(Imm16)] || 00 else PC <– PC + 4

ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__

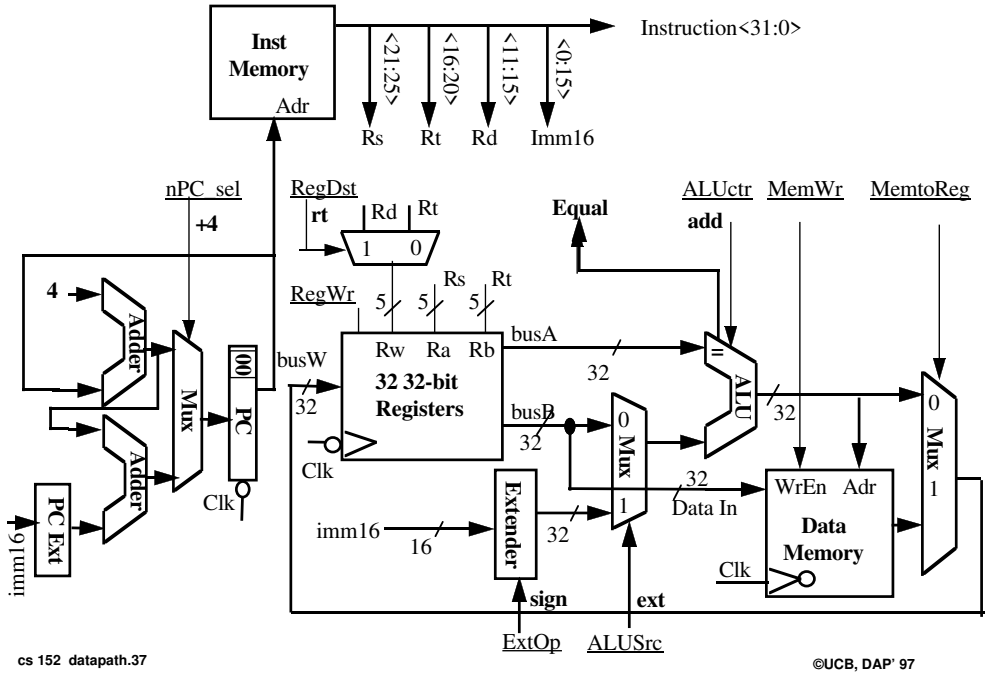
©UCB, DAP' 97

Step 5: Logic for each control signal

- nPC_sel <= if (OP == BEQ) then EQUAL else 0
- ALUsrc <= if (OP == "000000") then "regB" else "immed"
- ALUctr <= if (OP == "000000") then funct
  elseif (OP == ORi) then "OR"
  elseif (OP == BEQ) then "sub"
  else "add"
- ExtOp <= _____
- MemWr <= _____
- MemtoReg <= _____
- RegWr: <=_____
- RegDst: <= _____

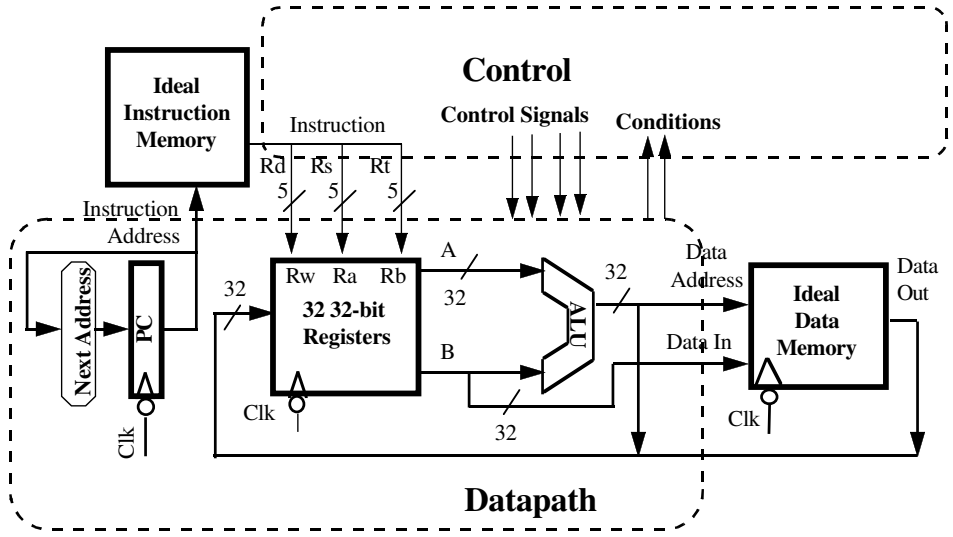
©UCB, DAP' 97

Example: Load Instruction



©UCB, DAP' 97

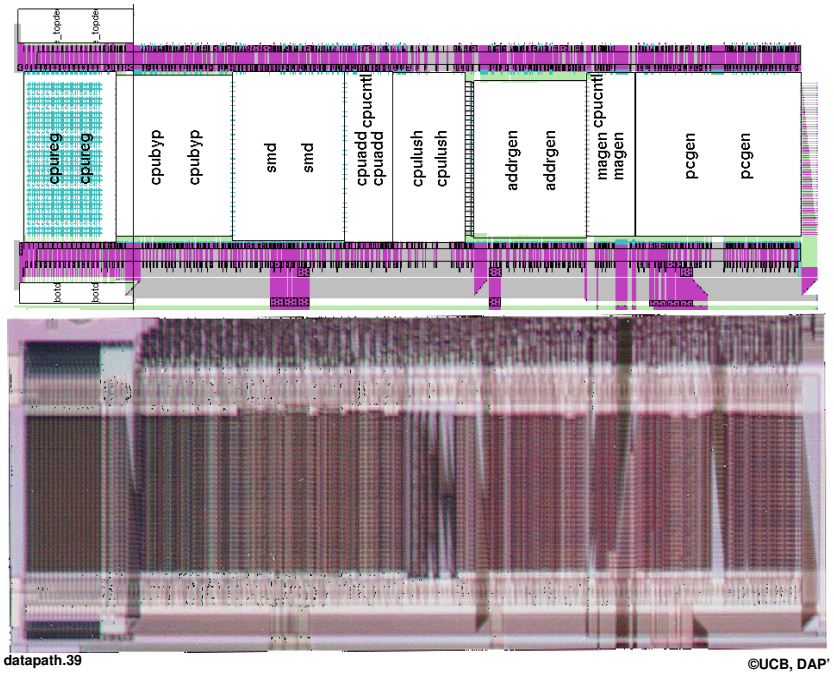
An Abstract View of the Implementation



- Logical vs. Physical Structure

©UCB, DAP' 97

A Real MIPS Datapath (CNS T0)



©UCB, DAP' 97

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

©UCB, DAP' 97