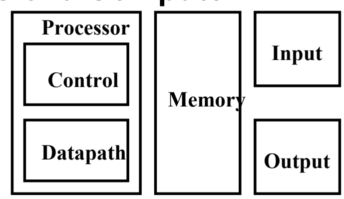
361
Computer Architecture
Lecture 8: Designing a Single Cycle Datapath

Outline of Today's Lecture

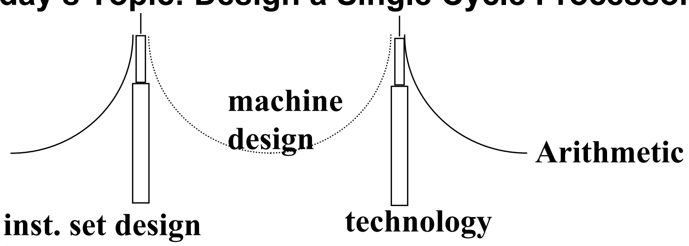
- Introduction
- Where are we with respect to the BIG picture?
- Questions and Administrative Matters
- The Steps of Designing a Processor
- Datapath and timing for Reg-Reg Operations
- Datapath for Logical Operations with Immediate
- Datapath for Load and Store Operations
- Datapath for Branch and Jump Operations

The Big Picture: Where are We Now?

° The Five Classic Components of a Computer



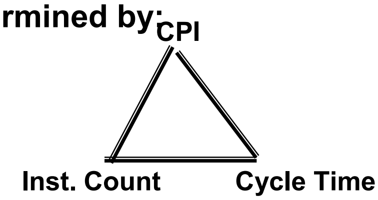
° Today's Topic: Design a Single Cycle Processor



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

° Today:

- Single cycle processor:
  - Advantage: One clock cycle per instruction
  - Disadvantage: long cycle time

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

The MIPS Instruction Formats

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

- R-type

31 26	21	16	11	6	0
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- I-type

31 26	21	16	0
op	**rs**	**rt**	**immediate**
6 bits	5 bits	5 bits	16 bits

- J-type

31 26	0
op	**target address**
6 bits	26 bits

° 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**

31 op 26	rs 21	rt 16	rd 11	shamt 6	funct 0
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

° **OR Immediate:**

- **ori rt, rs, imm16**

31 op 26	rs 21	rt 16	immediate 0
6 bits	5 bits	5 bits	16 bits

° **LOAD and STORE Word**

- **lw rt, rs, imm16**
- **sw rt, rs, imm16**

31 op 26	rs 21	rt 16	immediate 0
6 bits	5 bits	5 bits	16 bits

° **BRANCH:**

- **beq rs, rt, imm16**

31 op 26	rs 21	rt 16	immediate 0
6 bits	5 bits	5 bits	16 bits

Logical Register Transfers

° **RTL gives the <u>meaning</u> 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 + sign_ext(Imm16)] || 00 else PC <– PC + 4	

Step 1: Requirements of the Instruction Set

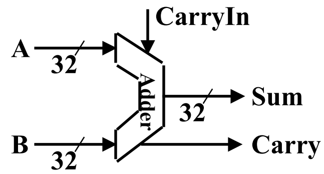
- Memory
  - instruction & data
- Registers (32 x 32)
  - read RS
  - read RT
  - Write RT or RD
- PC
- Extender
- Add and Sub register or extended immediate
- Add 4 or extended immediate to PC

Step 2: Components of the Datapath

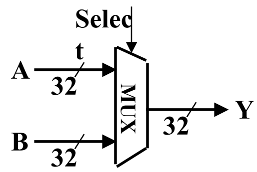
- Combinational Elements
- Storage Elements
  - Clocking methodology

Combinational Logic Elements (Basic Building Blocks)

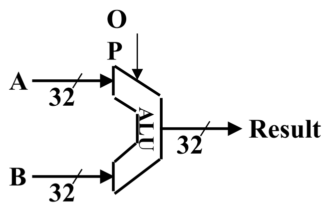
° **Adder**



° **MUX**



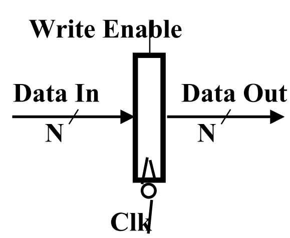
° **ALU**



Storage Element: Register (Basic Building Block)

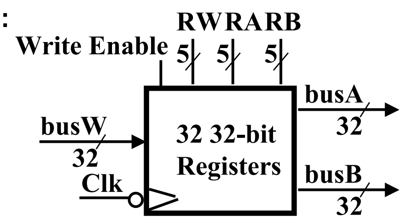
° **Register**

- **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 will become Data In**



Storage Element: Register File

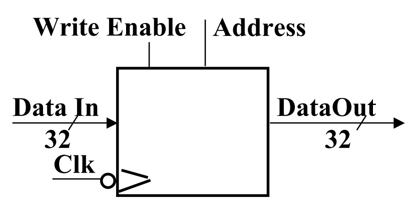
° Register File consists of 32 registers:
- Two 32-bit output busses: busA and busB
- One 32-bit input bus: busW



° 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

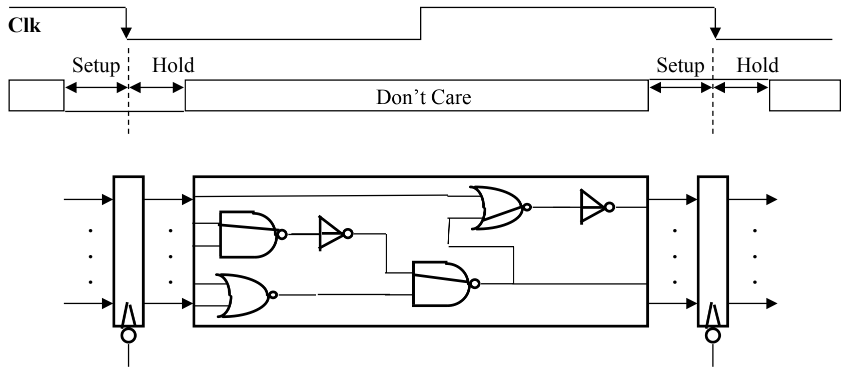


° Memory (idealized)
- One input bus: Data In
- One output bus: Data Out

° 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.”

Clocking Methodology



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

Questions and Administrative Matters

Step 3

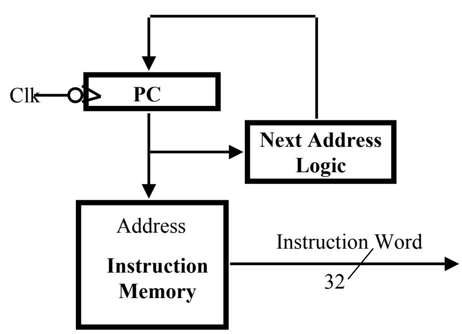
° Register Transfer <u>Requirements</u>
–> Datapath <u>Assembly</u>

° Instruction Fetch

° Read Operands and Execute Operation

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"



RTL: The ADD Instruction

31 26	21	16	11	6	0
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

° **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**

RTL: The Subtract Instruction

31 26	21	16	11	6	0
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

° **sub 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**

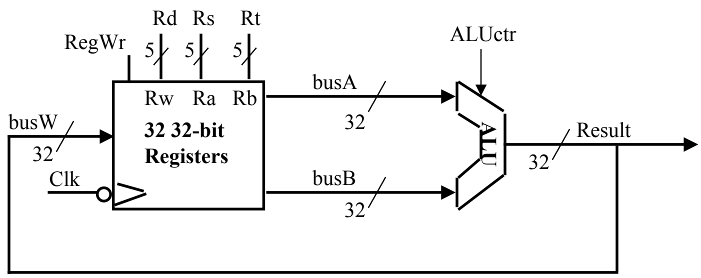
3b: Add & Subtract

° **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**

31 26 21 16 11 6 0

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



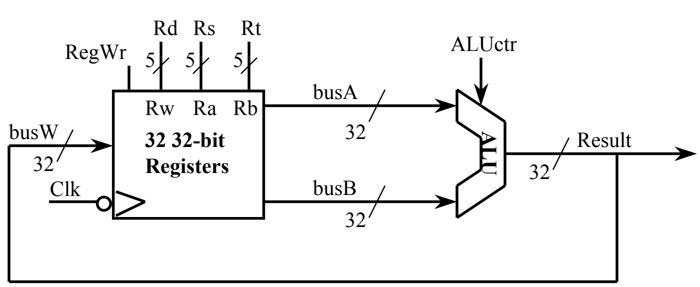
Datapath for Register-Register Operations (in general)

° **R[rd] <- R[rs] op R[rt]** **Example: add rd, rs, rt**

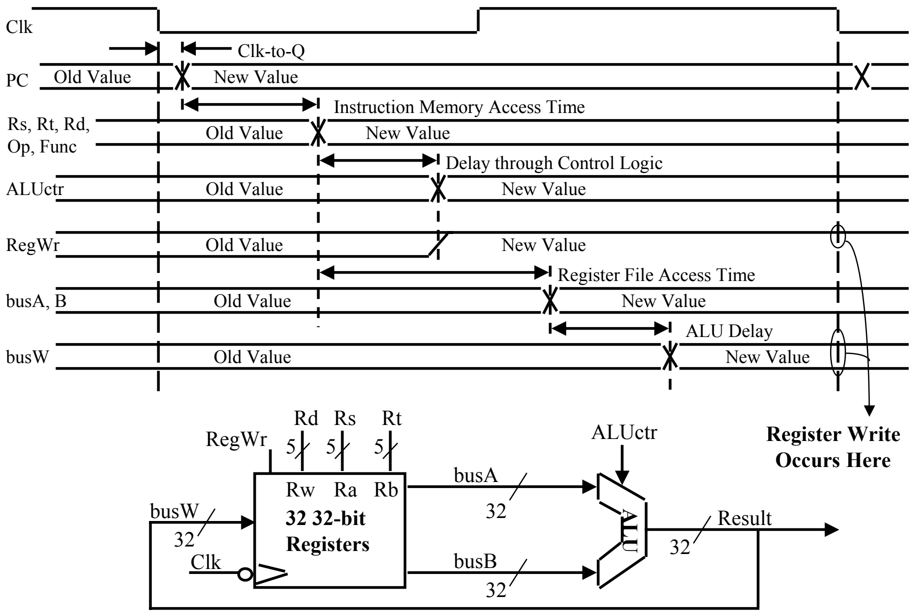
- **Ra, Rb, and Rw comes from instruction's rs, rt, and rd fields**
- **ALUctr and RegWr: control logic after decoding the instruction**

31 26 21 16 11 6 0

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



Register-Register Timing



361 datapath.23

RTL: The OR Immediate Instruction

31	26	21	16	0
op	**rs**	**rt**	**immediate**	
6 bits	5 bits	5 bits	16 bits	

° **ori rt, rs, imm16**

- **mem[PC]** **Fetch the instruction from memory**
- **R[rt] <- R[rs] or ZeroExt(imm16)**
  The OR operation
- **PC <- PC + 4** **Calculate the next instruction's address**

31	16 15	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	**immediate**	
16 bits	16 bits	

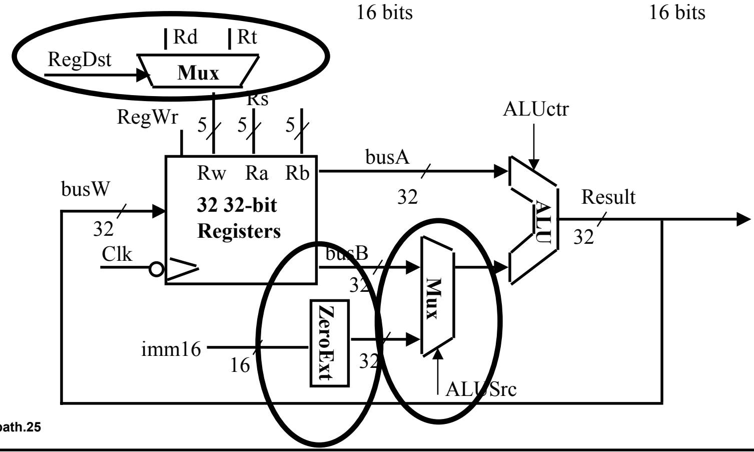
361 datapath.24

3c: Logical Operations with Immediate

° R[rt] <- R[rs] op ZeroExt[imm16]]

31	26	21	16	11	0
op	rs	rt	immediate		
6 bits	5 bits	5 bits	rd?	16 bits	

31	16	15	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		immediate	
16 bits		16 bits	



RTL: The Load Instruction

° lw rt, rs, imm16

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	

- mem[PC] Fetch the instruction from memory
- Addr <- R[rs] + SignExt(imm16) Calculate the memory address
  R[rt] <- Mem[Addr] Load the data into the register
- PC <- PC + 4 Calculate the next instruction's address

31	16	15	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0 immediate	
16 bits		16 bits	

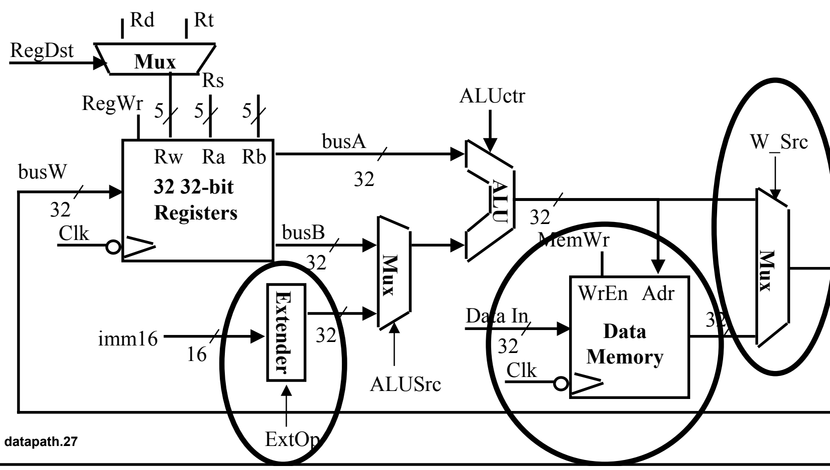
31	16	15	0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		1 immediate	
16 bits		16 bits	

3d: Load Operations

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

31 26 21 16 11 0

op	rs	rt	immediate
6 bits	5 bits	5 bits	rd 16 bits



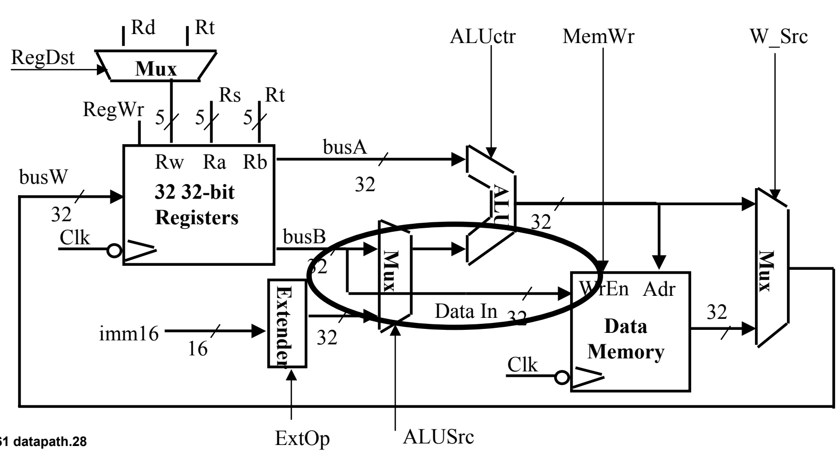
361 datapath.27

3e: Store Operations

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

31 26 21 16 0

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits



361 datapath.28

3f: The Branch Instruction

31 — 26	21	16	— 0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

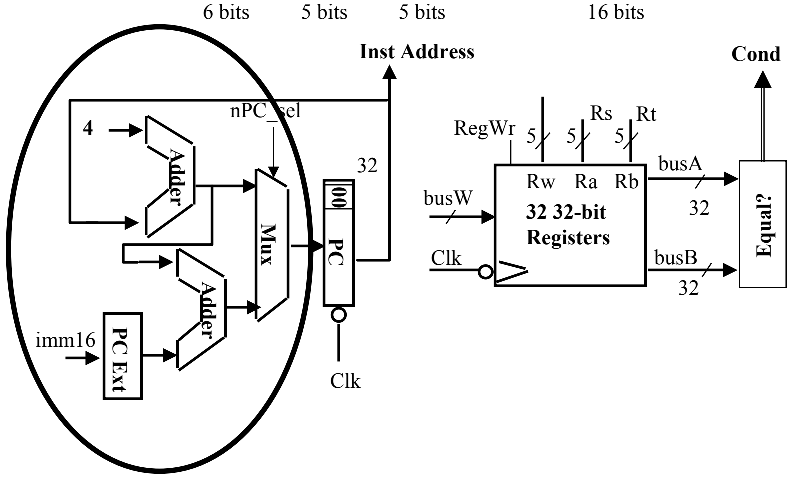
° **beq rs, rt, imm16**

- **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**

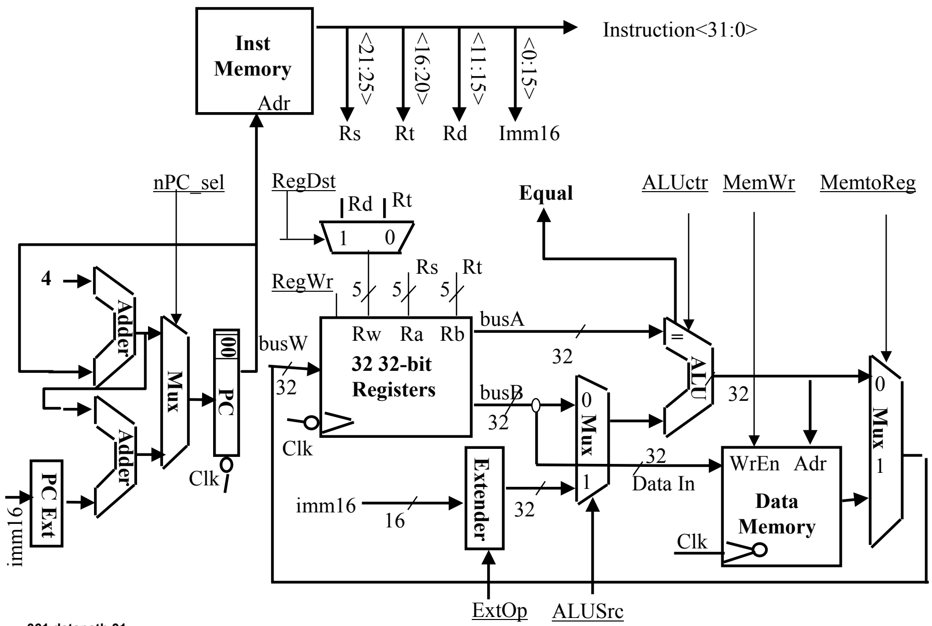
Datapath for Branch Operations

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

31 — 26	21	16	— 0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits



Putting it All Together: A Single Cycle Datapath

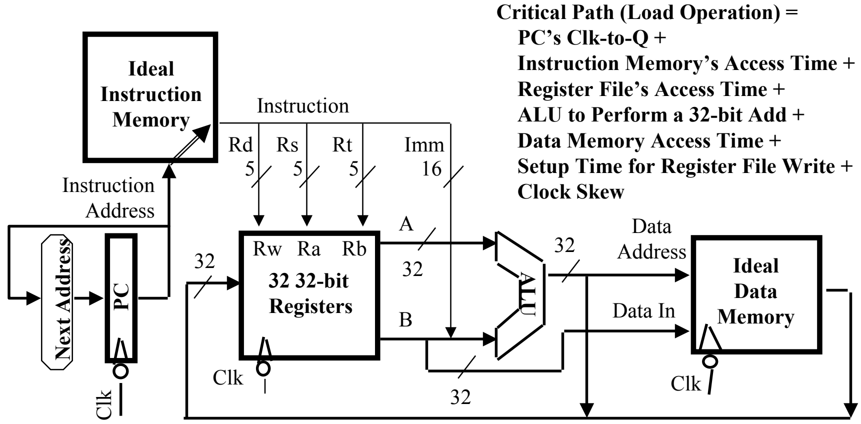


361 datapath.31

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



361 datapath.32

Binary Arithmetics for the Next Address

° In theory, the PC is a 32-bit byte address into the instruction memory:
  - Sequential operation: PC<31:0> = PC<31:0> + 4
  - Branch operation: PC<31:0> = PC<31:0> + 4 + SignExt[Imm16] * 4

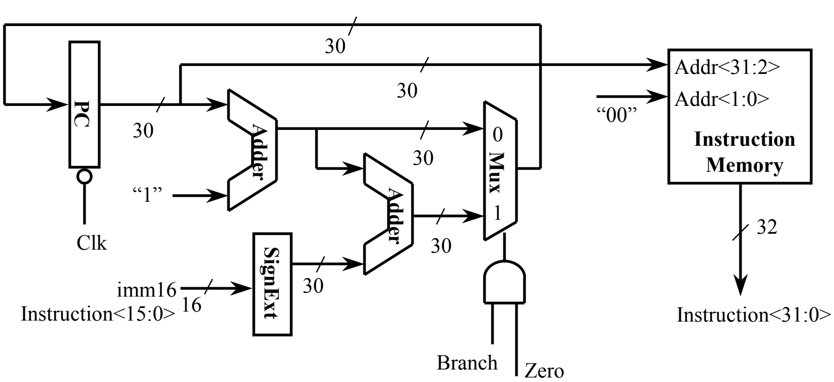
° The magic number "4" always comes up because:
  - The 32-bit PC is a byte address
  - And all our instructions are 4 bytes (32 bits) long

° In other words:
  - The 2 LSBs of the 32-bit PC are always zeros
  - There is no reason to have hardware to keep the 2 LSBs

° In practice, we can simply the hardware by using a 30-bit PC<31:2>:
  - Sequential operation: PC<31:2> = PC<31:2> + 1
  - Branch operation: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
  - In either case: Instruction Memory Address = PC<31:2> concat "00"

Next Address Logic: Expensive and Fast Solution

° Using a 30-bit PC:
  - Sequential operation: PC<31:2> = PC<31:2> + 1
  - Branch operation: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
  - In either case: Instruction Memory Address = PC<31:2> concat "00"



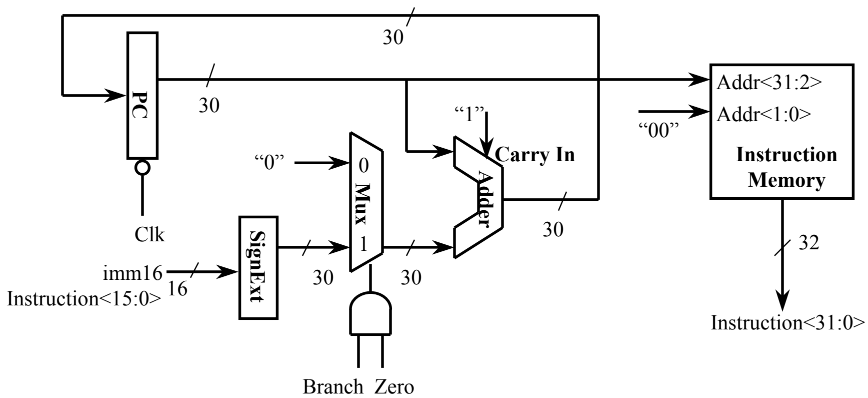
Next Address Logic: Cheap and Slow Solution

° **Why is this slow?**

- **Cannot start the address add until Zero (output of ALU) is valid**

° **Does it matter that this is slow in the overall scheme of things?**

- **Probably not here. Critical path is the load operation.**



361 datapath.35

RTL: The Jump Instruction

31 op (6 bits) 26	target address (26 bits) 0
op	target address
6 bits	26 bits

° **j   target**

- **mem[PC]**   **Fetch the instruction from memory**

- **PC<31:2>   <- PC<31:28> concat target<25:0>**

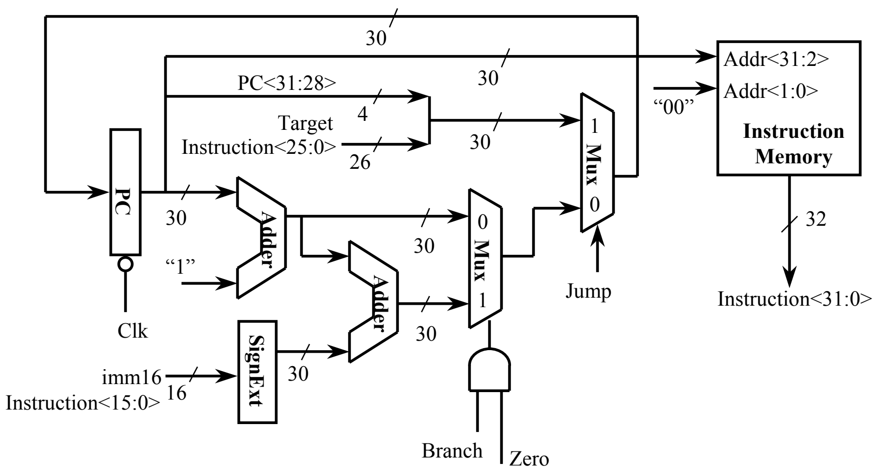
  Calculate the next instruction's address

361 datapath.36

Instruction Fetch Unit

° j target

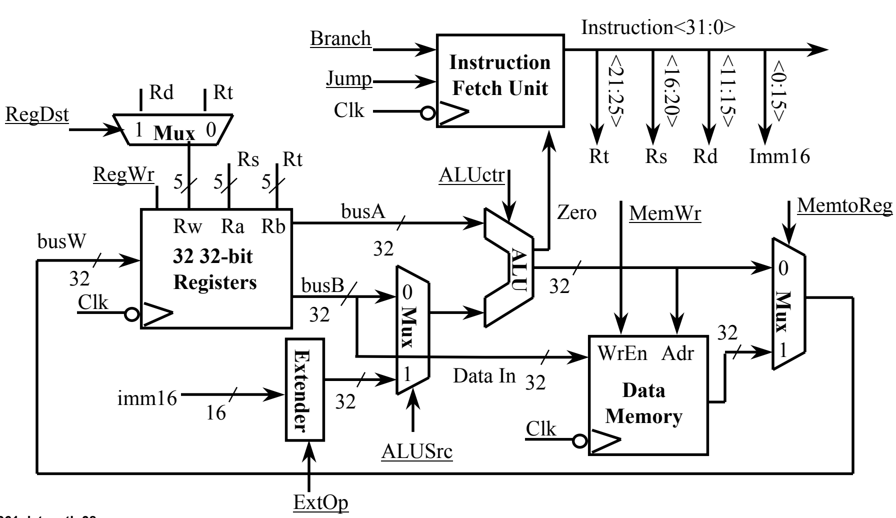
• PC<31:2> <- PC<31:28> concat target<25:0>



361 datapath.37

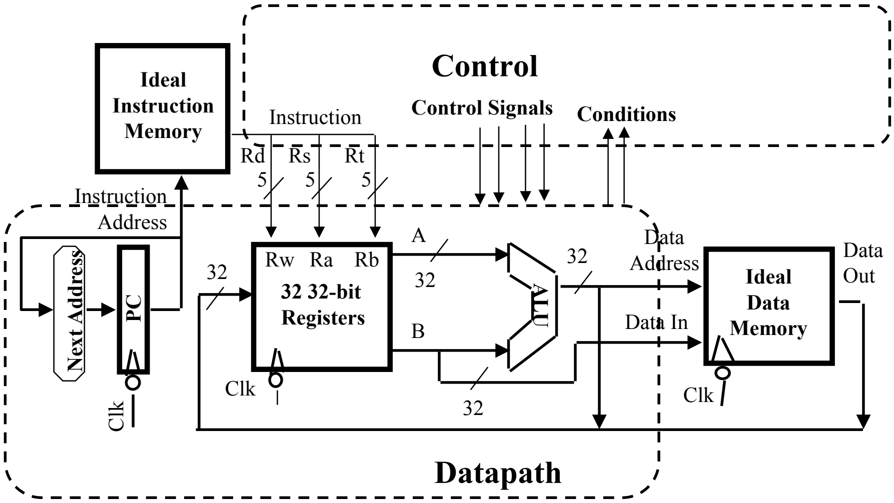
Putting it All Together: A Single Cycle Datapath

° We have everything except control signals (underline)



361 datapath.38

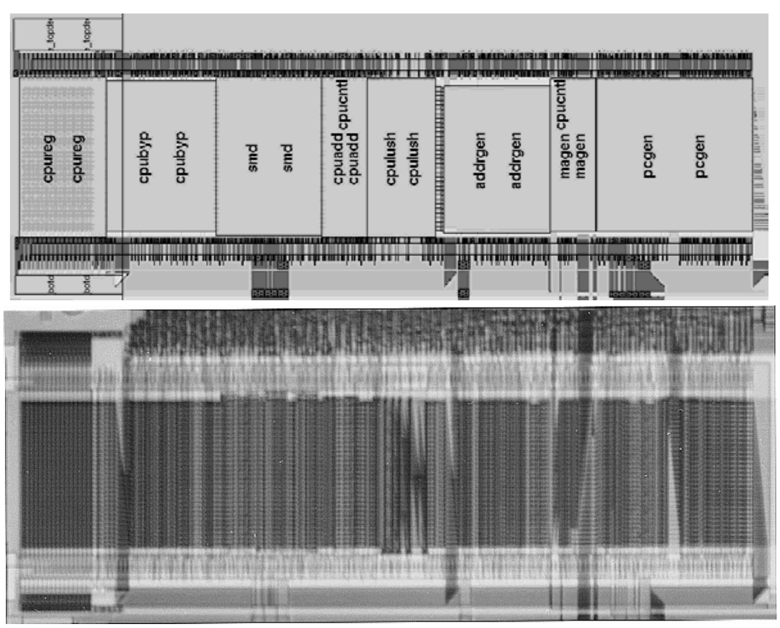
An Abstract View of the Implementation



° **Logical vs. Physical Structure**

361 datapath.39

A Real MIPS Datapath



361 datapath.40

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 (Steps 4 and 5)**