

CS61C – Machine Structures

Lecture 25 - CPU Design: A Gentle Introduction

3/19/2006

John Wawrzynek

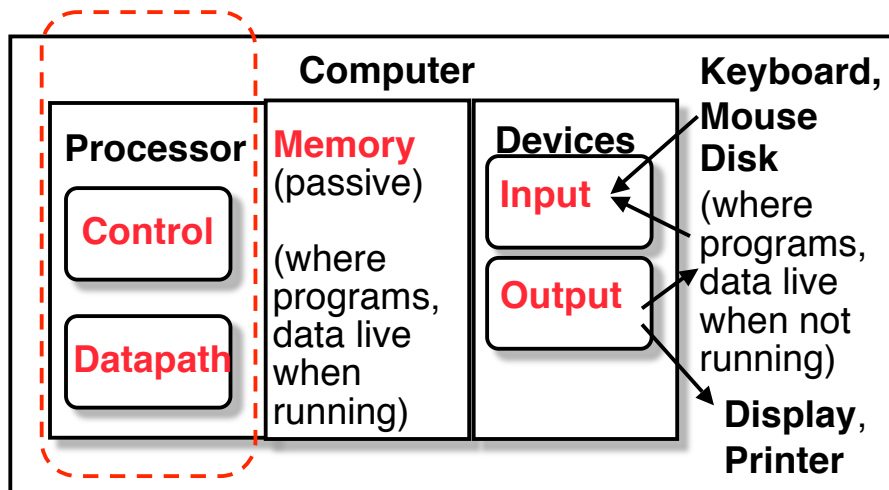
(www.cs.berkeley.edu/~johnw)

www-inst.eecs.berkeley.edu/~cs61c/

CS 61C L23 Intro to CPU Design (1)

Wawrzynek Spring 2006 © UCB

Five Components of a Computer



CS 61C L23 Intro to CPU Design (2)

Wawrzynek Spring 2006 © UCB

The CPU

- **Processor (CPU):** the active part of the computer, which does all the work (data manipulation and decision-making)
- **Datapath:** portion of the processor which contains hardware necessary to perform operations required by the processor (the brawn)
- **Control:** portion of the processor (also in hardware) which tells the datapath what needs to be done (the brain)

Stages of the Datapath (1/6)

- **Problem:** a single, atomic block which “executes an instruction” (performs all necessary operations beginning with fetching the instruction) would be too bulky and inefficient
- **Solution:** break up the process of “executing an instruction” into stages, and then connect the stages to create the whole datapath
 - smaller stages are easier to design
 - easy to optimize (change) one stage without touching the others

Stages of the Datapath (2/6)

- There is a *wide* variety of MIPS instructions: so what general steps do they have in common?
- Stage 1: **Instruction Fetch**
 - no matter what the instruction, the 32-bit instruction word must first be fetched from memory (the cache-memory hierarchy)
 - also, this is where we **Increment PC** (that is, $PC = PC + 4$, to point to the next instruction: byte addressing so + 4)

Stages of the Datapath (3/6)

- Stage 2: **Instruction Decode**
 - upon fetching the instruction, we next gather data from the fields (*decode* all necessary instruction data)
 - first, read the Opcode to determine instruction type and field lengths
 - second, read in data from all necessary registers
 - for `add`, read two registers
 - for `addi`, read one register
 - for `jal`, no reads necessary

Stages of the Datapath (4/6)

◦ Stage 3: **ALU** (Arithmetic-Logic Unit)

- the real work of most instructions is done here: arithmetic (+, -, *, /), shifting, logic (&, |), comparisons (slt)
- what about loads and stores?
 - lw \$t0, 40(\$t1)
 - the address we are accessing in memory = the value in \$t1 PLUS the value 40
 - so we do this addition in this stage

Stages of the Datapath (5/6)

◦ Stage 4: **Memory Access**

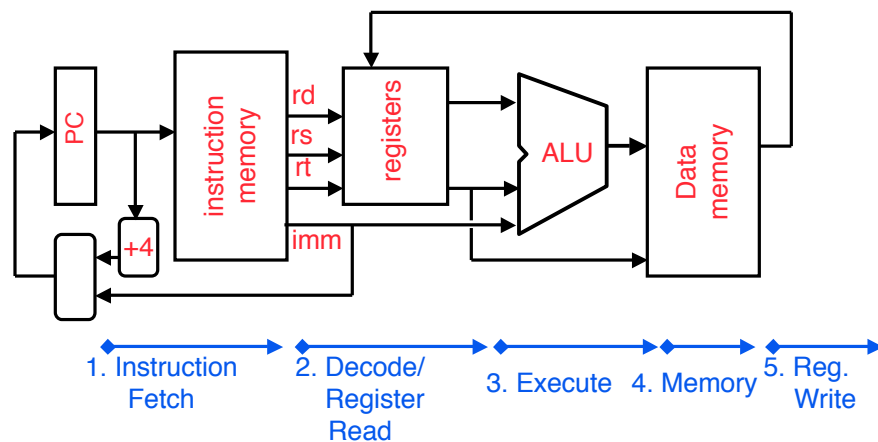
- actually only the load and store instructions do anything during this stage; the others remain idle during this stage or skip it all together
- since these instructions have a unique step, we need this extra stage to account for them
- as a result of the cache system, this stage is expected to be fast

Stages of the Datapath (6/6)

◦ Stage 5: Register Write

- most instructions write the result of some computation into a register
- examples: arithmetic, logical, shifts, loads, `slt`
- what about stores, branches, jumps?
 - don't write anything into a register at the end
 - these remain idle during this fifth stage or skip it all together

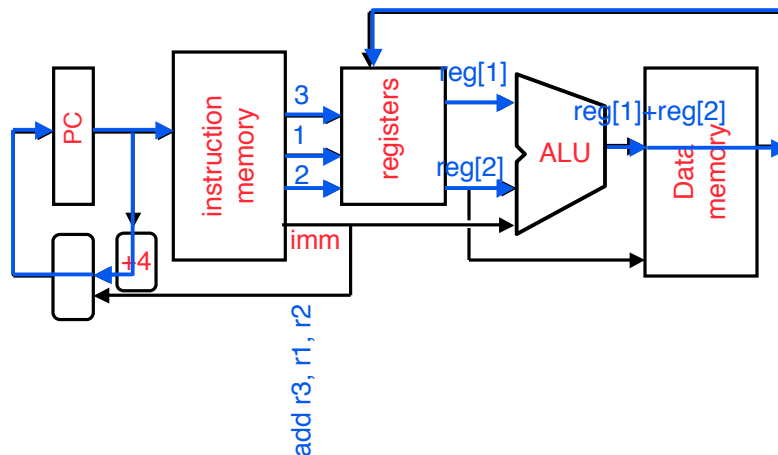
Generic Steps=> Datapath



Datapath Walkthroughs (1/3)

- `add $r3,$r1,$r2 # r3 = r1+r2`
 - Stage 1: fetch this instruction, inc. PC
 - Stage 2: decode to find it's an add, then read registers `$r1` and `$r2`
 - Stage 3: add the two values retrieved in Stage 2
 - Stage 4: idle (nothing to write to memory)
 - Stage 5: write result of Stage 3 into register `$r3`

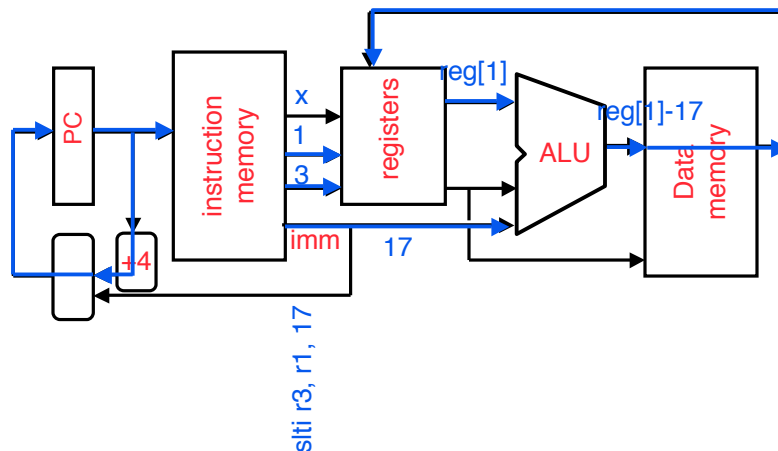
Example: add Instruction



Datapath Walkthroughs (2/3)

- `slti $r3,$r1,17`
 - Stage 1: fetch this instruction, inc. PC
 - Stage 2: decode to find it's an `slt`, then read register `$r1`
 - Stage 3: compare value retrieved in Stage 2 with the integer 17
 - Stage 4: idle
 - Stage 5: write the result of Stage 3 in register `$r3`

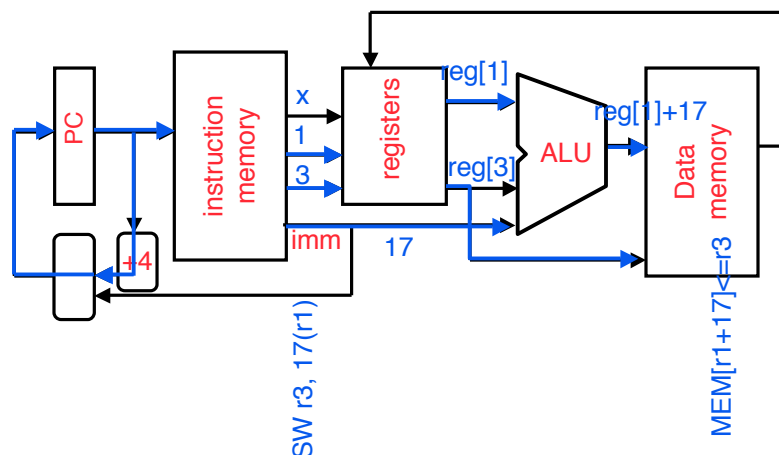
Example: `slt` Instruction



Datapath Walkthroughs (3/3)

- `sw $r3, 17($r1)`
 - Stage 1: fetch this instruction, inc. PC
 - Stage 2: decode to find it's a sw, then read registers \$r1 and \$r3
 - Stage 3: add 17 to value in register \$r1 (retrieved in Stage 2)
 - Stage 4: write value in register \$r3 (retrieved in Stage 2) into memory address computed in Stage 3
 - Stage 5: idle (nothing to write into a register)

Example: sw Instruction



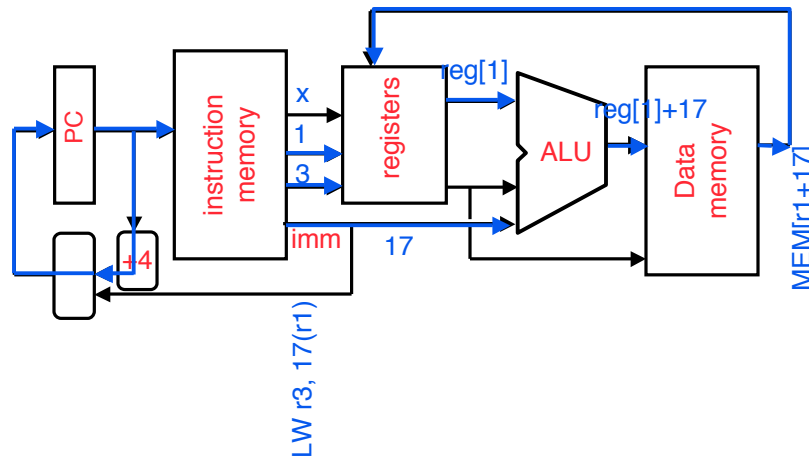
Why Five Stages? (1/2)

- **Could we have a different number of stages?**
 - Yes, and other architectures do
- **So why does MIPS have five if instructions tend to idle for at least one stage?**
 - The five stages are the union of all the operations needed by all the instructions.
 - There is one instruction that uses all five stages: the load

Why Five Stages? (2/2)

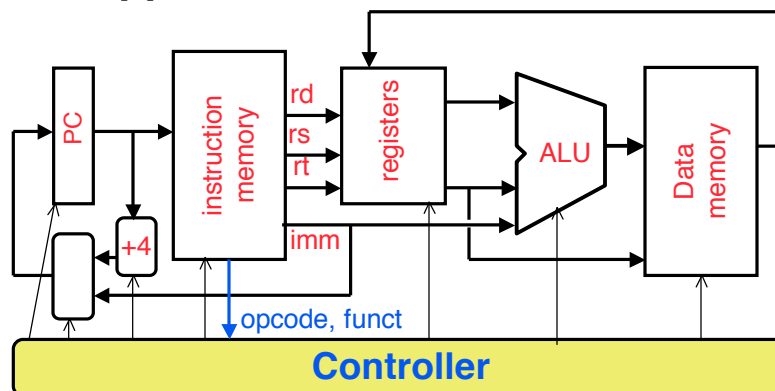
- `lw $r3, 17($r1)`
 - **Stage 1:** fetch this instruction, inc. PC
 - **Stage 2:** decode to find it's a `lw`, then read register `$r1`
 - **Stage 3:** add 17 to value in register `$r1` (retrieved in Stage 2)
 - **Stage 4:** read value from memory address compute in Stage 3
 - **Stage 5:** write value found in Stage 4 into register `$r3`

Example: lw Instruction



Datapath Summary

- The datapath based on data transfers required to perform instructions
- A *controller* causes the right transfers to happen



What Hardware Is Needed? (1/2)

- **PC: a register which keeps track of memory addr of the next instruction**
- **General Purpose Registers**
 - used in Stages 2 (Read) and 5 (Write)
 - MIPS has 32 of these
- **Memory**
 - used in Stages 1 (Fetch) and 4 (R/W)
 - cache system makes these two stages as fast as the others, on average

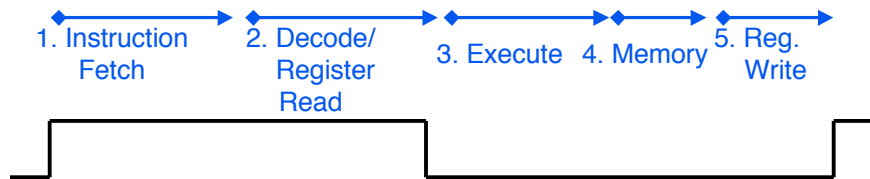
What Hardware Is Needed? (2/2)

- **ALU**
 - used in Stage 3
 - something that performs all necessary functions: arithmetic, logicals, etc.
 - we'll design details later
- **Miscellaneous Registers**
 - In implementations with only one stage per clock cycle, registers are inserted between stages to hold intermediate data and control signals as they travel from stage to stage.
 - Note: Register is a general purpose term meaning something that stores bits. Not all registers are in the "register file".

CPU clocking (1/2)

For each instruction, how do we control the flow of information through the datapath?

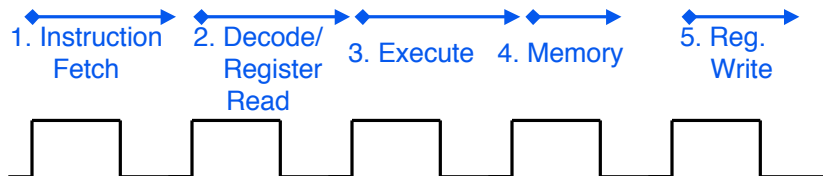
- **Single Cycle CPU:** All stages of an instruction are completed within one *long* clock cycle.
 - The clock cycle is made sufficient long to allow each instruction to complete all stages without interruption and within one cycle.



CPU clocking (1/2)

For each instruction, how do we control the flow of information through the datapath?

- **Multiple-cycle CPU:** Only one stage of instruction per clock cycle.
 - The clock is made as long as the slowest stage.



Several significant advantages over single cycle execution: Unused stages in a particular instruction can be skipped OR instructions can be pipelined (overlapped).