Discussion 12 Richard Guo
CPU and Pipelining 07/30/08

1. CPU Says: My Clock's on Speed!

The MIPS processor is just a wrapper over an advanced calculator. You've already seen the basics of how it operates in the ALU homework. The CPU has a few main parts:

Datapath: The hardware elements that can process all possible instructions.
Control: Determines which path through the datapath instructions will flow.

The heuristics for determining a CPU's performance are clock speed (how fast each cycle is completed) and IPC (instructions per clock, or how much work is done in each cycle). In MIPS, we don't worry too much about IPC because we have pipelined the stages to contain around equal amounts of work.

Clock speed is an issue if we don't pipeline, because we have to wait for an instruction to sift through the entire *datapath* before the next instruction can be *in flight*. Instead, if we add registers between certain discrete CL units, the longest path between two nodes will be much shorter and the clock speed can be increased. [Baseball analogy]

2. Pumping Gas Through the Pipeline

Pipelining improves throughput, not latency. A single instruction will actually take a bit more time to processed because of the extra delay introduced by registers. However, more instructions can be in flight as the same time. [*Historical Aside: Designers can get with crazy pipelining too many things. Intel's Pentium 4 was legendary for having 20-30 stages in its pipeline. Clock speeds were high, but power consumption varies proportionally with clock speed, and making the incorrect predictions requires flushing the entire pipeline, an expensive proposition.*] Thus, throughput is improved.

Fetch: Loads PC. Retrieve the instruction from memory or from cache. Increments the PC counter by 4.
Decode: Reads data from the registers as specified by opcode
ALU or Execute: Executes the operation (+, -, *, /, &, |, >>, <<, <) specified on the inputs.
Memory Access: Loads or stores values from or to the memory. (only used by loads and stores)
Write Back or Register Write: Stores values back to a register. Not used by stores, branches, and jumps

No instruction needs all of the datapath. The most intensive one is lw, which uses every stage.
The control examines the opcode and funct and sets certain bits that signal which parts of path to take.

3. [Dukes of] Hazard

Hazards can occur when the different stages of the pipeline conflict (yet another reason to avoid long pipelines). The focus for this course is on data and control hazards. Structural ones are abstracted away.

1. Structural (memory): e.g. Fetch and decode in same cycle. Solved by caching and then delaying if required data for both instructions is not in cache and needs to be loaded from memory.
2. Structural (registers): e.g. Read and write to the same register in a cycle. Solved by splitting the time in half. Actually what's done (p 408) is internal forwarding where the input to the write, which happens first, is piped out to the read, which normally happens second. This is because the clk-to-q time may not be short enough to ensure the read gets the updated value.

3. Data: Instruction relies on the updated value of a register, but that update hasn't hit the write back stage yet. Forwarding pipes out the output of the ALU or Memory and feeds it into the later ALU or Memory when it is required. There is a special case when load word causes the forwarding to be backwards in time. The output of memory cannot be piped to the ALU of the next stage because they happen at the same cycle. Solution is to delay the 2nd instruction or to reorder the code to move an unrelated instruction up.
4. Control: Since the pipeline is filled sequentially, branches and jumps that move out of order may cause the wrong instructions to be loaded into the pipeline. Solved by bypassing the ALU and changing the PC right after decoding the instruction. This still means that the instruction sequentially after the branch will be loaded before PC is updated. Improved by forcing the compiler to be smart and to reorder instruction so that something from before the branch/jump is placed after into the *branch delay slot*. Optimization called *out-of-order*.

4. Unscramble Me

1. One instruction type that can cause wrong instructions further down on the pipeline: **Branch**
2. Another instruction type that can wrong instructions further down on the pipeline: **Jump**
4. The stage of the MIPS processor that performs slr: **ALU**
5. Another name for #4: **Execute**
6. A technique to segment work into discrete tasks. **Pipelining**
7. How many inputs does #4 have (a number)? **3**
8. This stage of the MIPS processor can conflict with the memory stage: **Fetch**
9. The last stage of the MIPS processor (2 words): **Write Back**
10. How do we eliminate non-load data hazards? **Forwarding**
11. Which metric does #6 maximize? **Throughput**
12. Another metric does #6 maximize? **Clock speed**
13. How do we solve control hazards (3 words each beginning with the same letter)? **Out of Order**
14. What do we add to implement #6? **Registers**

Now take the first letters/number of the answer to each and form words (each vowel can be used more than once; 1 consonant is not used): _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (Don't yell this out.)

Answer: **Beware of Project 3**

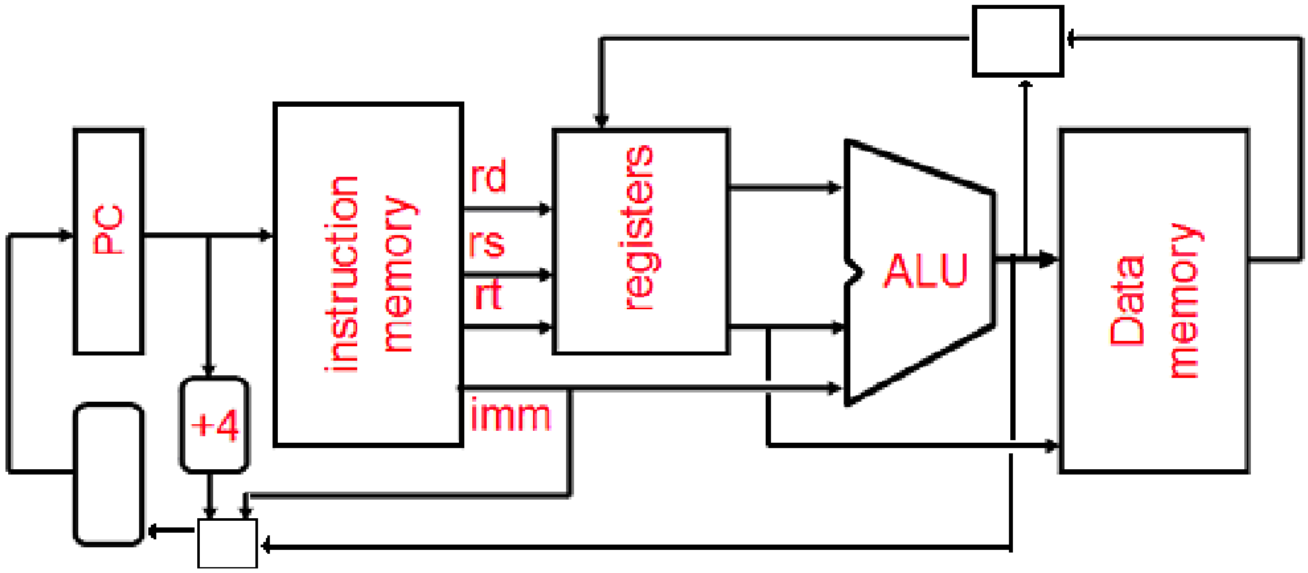
5. Problems

MIPS creates 5 stages in its datapath. How many total registers in the datapath does this require? {**4 for after each stage. PC for a total of 5**}

Why is a 50 stage CPU possibly worse than a 5 stage CPU? {**higher power consumption, register delay is more significant part of total delay = higher latency, data hazards are more often**}

6. A More "Challenging" Problem

Trace out the instructions jal and jr. Then modify the following datapath to support the instruction jalr.


PC
instruction memory
rd
rs
rt
imm
registers
ALU
Data memory
+4