inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture #20
Introduction to Pipelined Execution, pt II

CPS today!

2005-11-09

There is one handout today at the front and back of the room!

Lecturer PSOE, new dad Dan Garcia

www.cs.berkeley.edu/~ddgarcia

History's worst SW bugs! ⇒ How does your Proj1 Peg Solitaire bug compares to the top 10 worst bugs of all time? How many can you name? Wired chose the list.

wired.com/news/technology/bugs/0,2924,69355,00.html

Review: Pipeline (1/2)

- Optimal Pipeline
  - Each stage is executing part of an instruction each clock cycle.
  - One inst. finishes during each clock cycle.
  - On average, execute far more quickly.
- What makes this work?
  - Similarities between instructions allow us to use same stages for all instructions (generally).
  - Each stage takes about the same amount of time as all others: little wasted time.

Review: Pipeline (2/2)

- Pipelining is a BIG IDEA
  - widely used concept
- What makes it less than perfect?
  - Structural hazards: suppose we had only one cache? ⇒ Need more HW resources
  - Control hazards: need to worry about branch instructions? ⇒ Delayed branch
  - Data hazards: an instruction depends on a previous instruction?

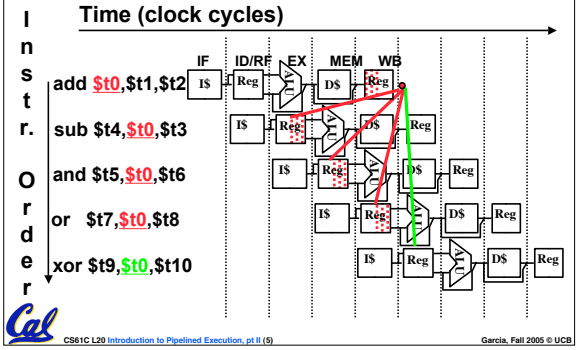
Data Hazards (1/2)

- Consider the following sequence of instructions

```
add $t0, $t1, $t2
sub $t4, $t0 ,$t3
and $t5, $t0 ,$t6
or  $t7, $t0 ,$t8
xor $t9, $t0 ,$t10
```

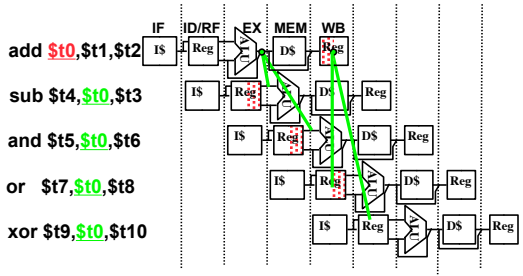

Data Hazards (2/2)

Dependencies backwards in time are hazards



Data Hazard Solution: Forwarding

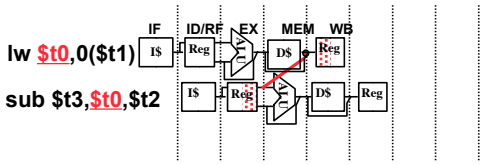
- Forward result from one stage to another



"or" hazard solved by register hardware

Data Hazard: Loads (1/4)

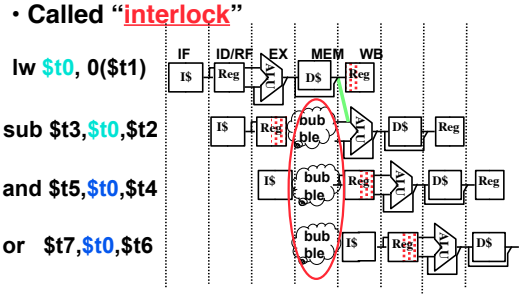
- Dependencies backwards in time are hazards



- Can't solve with forwarding
- Must stall instruction dependent on load, then forward (more hardware)

Data Hazard: Loads (2/4)

- Hardware must stall pipeline
- Called "interlock"

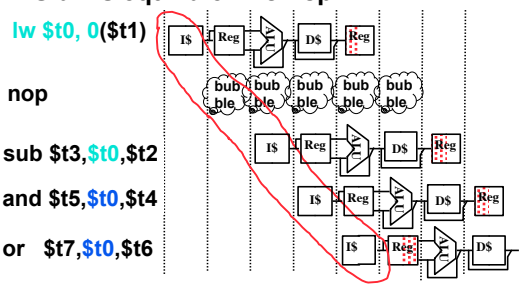


Data Hazard: Loads (3/4)

- Instruction slot after a load is called "load delay slot"
- If that instruction uses the result of the load, then the hardware interlock will stall it for one cycle.
- If the compiler puts an unrelated instruction in that slot, then no stall
- Letting the hardware stall the instruction in the delay slot is equivalent to putting a nop in the slot (except the latter uses more code space)

Data Hazard: Loads (4/4)

- Stall is equivalent to nop



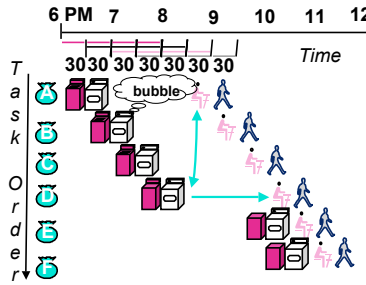
Historical Trivia

- First MIPS design did not interlock and stall on load-use data hazard
- Real reason for name behind MIPS:
  Microprocessor without
  Interlocked
  Pipeline
  Stages
  - Word Play on acronym for Millions of Instructions Per Second, *also* called MIPS

Administrivia

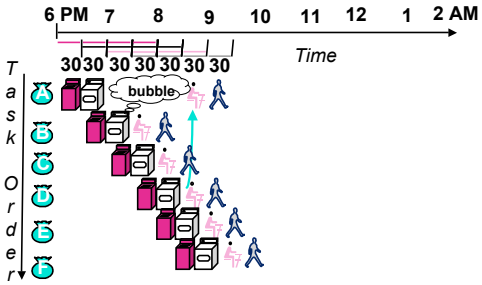
- Any administrivia?
- Advanced Pipelining!
  - "Out-of-order" Execution
  - "Superscalar" Execution

Review Pipeline Hazard: Stall is dependency



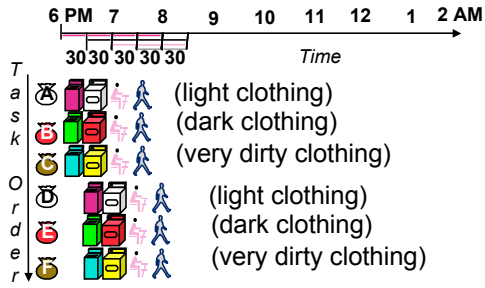
A depends on D; stall since folder tied up

Out-of-Order Laundry: Don't Wait



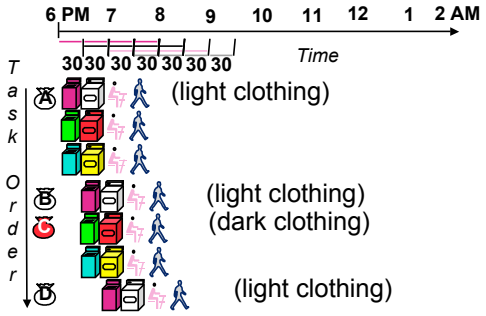
A depends on D; rest continue; need more resources to allow out-of-order

Superscalar Laundry: Parallel per stage



More resources, HW to match mix of parallel tasks?

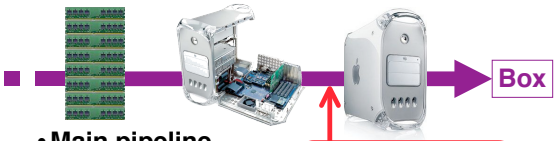
Superscalar Laundry: Mismatch Mix



Task mix underutilizes extra resources

Real-world pipelining problem

- You're the manager of a HUGE assembly plant to build computers.

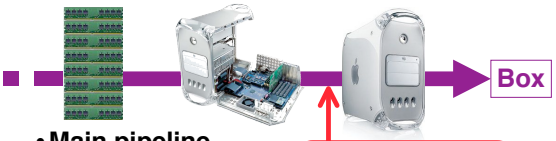


- Main pipeline
  - 10 minutes/ pipeline stage
  - 60 stages
  - Latency: 10hr

Problem: need to run 2 hr test before done..help!

Real-world pipelining problem solution 1

- You remember: "a pipeline frequency is limited by its slowest stage", so…

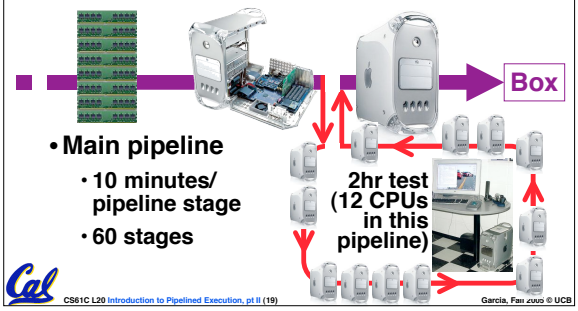


- Main pipeline
  - 2hours/ pipeline stage
  - 60 stages
  - Latency: 120hr

Problem: need to run 2 hr test before done..help!

Real-world pipelining problem solution 2

- Create a sub-pipeline!



- Main pipeline
  - 10 minutes/ pipeline stage
  - 60 stages

Wired Magazine: History's Worst SW Bugs!

- Engineers learn from failure!
- Failure for most other engineers usually means people die as a result
  - Be thankful you weren't a civil engineer designing steel truss bridges in the early 1900s, since the failure rate was > 40%!

- First "Bug": in 1947 engineers found a dead moth in Panel F, Relay #70 of the Harvard Mark 1 computer system.
  - The following are sorted by date.

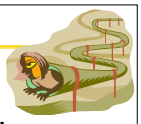
`wired.com/news/technology/bugs/0,2924,69355,00.html`

July 28, 1962 – Mariner I space probe

- "A bug in the flight software for the Mariner 1 causes the rocket to divert from its intended path on launch. Mission control destroys the rocket over the Atlantic Ocean.
- The investigation into the accident discovers that a formula written on paper in pencil was improperly transcribed into computer code, causing the computer to miscalculate the rocket's trajectory."

`wired.com/news/technology/bugs/0,2924,69355,00.html`

1982 – Soviet gas pipeline



- "Operatives working for the CIA allegedly plant a bug in a Canadian computer system purchased to control the trans-Siberian gas pipeline.
- Soviets had obtained the system as part of a wide-ranging effort to covertly purchase or steal sensitive U.S. technology.
- The CIA reportedly found out about the program and decided to make it backfire with equipment that would pass Soviet inspection and then fail once in operation.
- The resulting event is reportedly the largest non-nuclear explosion in the planet's history…"

`wired.com/news/technology/bugs/0,2924,69355,00.html`

1985-1987 – Therac-25 medical accelerator

- "A radiation therapy device malfunctions and delivers lethal radiation doses at several medical facilities. Based upon a previous design, the Therac-25 was an 'improved' therapy system that could deliver 2 different kinds of radiation: a low-power electron beam (beta particles) or X-rays.
- The Therac-25's X-rays were generated by smashing high-power electrons into a metal target positioned between the electron gun and the patient. A second 'improvement' was the replacement of the older Therac-20's electromechanical safety interlocks with software control, a decision made because software was perceived to be more reliable.

  What engineers didn't know was that both the 20 and the 25 were built upon an operating system that had been kludged together by a programmer with no formal training. Because of a subtle bug called a 'race condition,' a quick-fingered typist could accidentally configure the Therac-25 so the electron beam would fire in high-power mode but with the metal X-ray target out of position. At least five patients die; others are seriously injured."

`wired.com/news/technology/bugs/0,2924,69355,00.html`

1988 – Buffer overflow in Unix finger daemon

- "The first internet worm (the so-called Morris Worm) infects between 2,000 and 6,000 computers in less than a day by taking advantage of a buffer overflow. The specific code is a function in the standard input/output library routine called `gets()` designed to get a line of text over the network. Unfortunately, `gets()` has no provision to limit its input, and an overly large input allows the worm to take over any machine to which it can connect.
- Programmers respond by attempting to stamp out the `gets()` function in working code, but they refuse to remove it from the C programming language's standard I/O library, where it remains to this day."

`wired.com/news/technology/bugs/0,2924,69355,00.html`

1988-1996 – Kerberos Random # Generator

- "The authors of the Kerberos security system neglect to properly 'seed' the program's random number generator with a truly random seed.
- As a result, for eight years it is possible to trivially break into any computer that relies on Kerberos for authentication. It is unknown if this bug was ever actually exploited."

wired.com/news/technology/bugs/0,2924,69355,00.html

January 15, 1990 – AT&T Network Outage

- "A bug in a new release of the software that controlled AT&T's #4ESS long distance switches caused these mammoth computers to crash when they received a specific message from one of their neighboring machines – a message that the neighbors send out when they recover from a crash.
- One day a switch in New York crashed and rebooted, causing its neighboring switches to crash, then their neighbors' neighbors, and so on. Soon, 114 switches were crashing and rebooting every six seconds, leaving an estimated 60,000 people without long distance service for nine hours. The fix: engineers load the previous software release.."

wired.com/news/technology/bugs/0,2924,69355,00.html

1993 – Intel Pentium floating point divide

- "A silicon error causes Intel's highly promoted Pentium chip to make mistakes when dividing floating-pt #s that occur within a specific range.
- For example, dividing `4195835.0/3145727.0` yields `1.33374` instead of `1.33382`, an error of `0.006` percent. Although the bug affects few users, it becomes a public relations nightmare.
- With an estimated 3 million to 5 million defective chips in circulation, at first Intel only offers to replace Pentium chips for consumers who can prove that they need high accuracy; eventually the company relents and agrees to replace the chips for anyone who complains. The bug ultimately costs Intel $475 million."

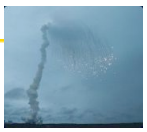
wired.com/news/technology/bugs/0,2924,69355,00.html

1995/1996 – The Ping of Death

- "A lack of sanity checks and error handling in the IP fragmentation reassembly code makes it possible to crash a wide variety of operating systems by sending a malformed 'ping' packet from anywhere on the internet.
- Most obviously affected are computers running Windows, which lock up and display the so-called 'blue screen of death' when they receive these packets. But the attack also affects Macintoshes and Unixes too."

wired.com/news/technology/bugs/0,2924,69355,00.html

June 4, 1996 – Ariane 5 Flight 501



- Working code for the Ariane 4 rocket is reused in the Ariane 5, but the Ariane 5's faster engines trigger a bug in an arithmetic routine inside the rocket's flight computer.
- The error is in the code that converts a 64-bit floating-point number to a 16-bit signed integer. The faster engines cause the 64-bit numbers to be larger in the Ariane 5 than in the Ariane 4, triggering an overflow condition that results in the flight computer crashing.
- First Flight 501's backup computer crashes, followed 0.05 seconds later by a crash of the primary computer. As a result, the rocket's primary processor overpowers the rocket's engines and causes the rocket to disintegrate 40 seconds after launch.

wired.com/news/technology/bugs/0,2924,69355,00.html

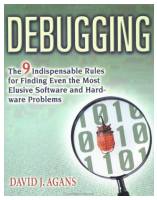
November 2000 – NCI, Panama City

- In a series of accidents, therapy planning SW created by Multidata Systems International, a U.S. firm, miscalculates the proper dosage of radiation for patients undergoing radiation therapy.
- Multidata's SW allows a radiation therapist to draw on a computer screen the placement of metal shields called "blocks" designed to protect healthy tissue from the radiation. But the software will only allow technicians to use four shielding blocks, and the Panamanian doctors wish to use five.
- The doctors discover that they can trick the software by drawing all five blocks as a single large block with a hole in the middle. What the doctors don't realize is that the Multidata software gives different answers in this configuration depending on how the hole is drawn: draw it in one direction and the correct dose is calculated, draw in another direction and the software recommends twice the necessary exposure.
- At least 8 patients die, while another 20 receive overdoses likely to cause significant health problems. The physicians, who were legally required to double-check the computer's calculations by hand, are indicted for murder.

wired.com/news/technology/bugs/0,2924,69355,00.html

What can you do about it?

- Debug & test rigorously, as if lives depended on it
  - They just might!
- Learn from this book:



Peer Instruction (1/2)

Assume 1 instr/clock, delayed branch, 5 stage pipeline, forwarding, interlock on unresolved load hazards (after 10^3 loops, so pipeline full)

```
Loop:   lw    $t0, 0($s1)
        addu  $t0, $t0, $s2
        sw    $t0, 0($s1)
        addiu $s1, $s1, -4
        bne   $s1, $zero, Loop
        nop
```

- How many pipeline stages (clock cycles) per loop iteration to execute this code?

1 2 3 4 5 6 7 8 9 10

Peer Instruction (2/2)

Assume 1 instr/clock, delayed branch, 5 stage pipeline, forwarding, interlock on unresolved load hazards (after 10^3 loops, so pipeline full). Rewrite this code to reduce pipeline stages (clock cycles) per loop to as few as possible.

```
Loop:   lw    $t0, 0($s1)
        addu  $t0, $t0, $s2
        sw    $t0, 0($s1)
        addiu $s1, $s1, -4
        bne   $s1, $zero, Loop
        nop
```

- How many pipeline stages (clock cycles) per loop iteration to execute this code?

1 2 3 4 5 6 7 8 9 10

"And in Conclusion.."

- Pipeline challenge is hazards
  - Forwarding helps w/many data hazards
  - Delayed branch helps with control hazard in 5 stage pipeline
- More aggressive performance:
  - Superscalar
  - Out-of-order execution
- You can be creative with your pipelines
- Learn from our top 10 worst SW bugs…
  - Test, test, test. Expect the unexpected.
  - Design w/failure as possibility! Redundancy!