Review 1/1

- Every machine has a convention for how arguments are passed.
- In MIPS, where do the arguments go if you are passing more than 4 words? Stack!
- It is sometimes useful to have a variable number of arguments.
  - The C convention is to use “...”
  - *fmt is used to determine the number of variables and their types.

Outline

- Review Parameter Passing on Stacks
- Pipelining Analogy
- Pipelining Instruction Execution
- Administrivia, “What’s this Stuff Bad for?”
- Hazards to Pipelining
- Solutions to Hazards
- Advanced Pipelining Concepts by Analogy
- Conclusion

Pipelining is Natural! Laundry Example

- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold, and put away
  - Washer takes 30 minutes
  - Dryer takes 30 minutes
  - “Folder” takes 30 minutes
  - “Stasher” takes 30 minutes to put clothes into drawers
Sequential Laundry

○ Sequential laundry takes 8 hours for 4 loads

Pipelined Laundry: Start work ASAP

○ Pipelined laundry takes 3.5 hours for 4 loads!

Pipelining Lessons

○ Pipelining doesn’t help latency of single task, it helps throughput of entire workload

○ Multiple tasks operating simultaneously using different resources

○ Potential speedup = Number pipe stages

○ Time to “fill” pipeline and time to “drain” it reduces speedup: 2.3X v. 4X in this example

○ Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?

○ Pipeline rate limited by slowest pipeline stage

○ Unbalanced lengths of pipe stages also reduces speedup
Review: Steps in Executing MIPS (Lec. 20)

1) Ifetch: Fetch Instruction, Increment PC

2) Decode Instruction, Read Registers

3) Execute:
   Mem-ref: Calculate Address
   Arith-log: Perform Operation
   Branch: Compare if operands ==

4) Memory:
   Load: Read Data from Memory
   Store: Write Data to Memory
   Branch: if operands ==, Change PC

5) Write Back: Write Data to Register

Pipelined Execution Representation

Every instruction takes the same number of steps, also called pipeline “stages”

Graphical Pipeline Representation

(right half highlight means read, left half write)
Example
° Suppose 2 ns for memory access, 2 ns for ALU operation, and 1 ns for register file read or write

° Nonpipelined Execution:
  • \text{lw}: \text{IF} + \text{Read Reg} + \text{ALU} + \text{Memory} + \text{Write Reg} = 2 + 1 + 2 + 2 + 1 = 8 \text{ ns}
  • \text{add}: \text{IF} + \text{Read Reg} + \text{ALU} + \text{Write Reg} = 2 + 1 + 2 + 1 = 6 \text{ ns}

° Pipelined Execution:
  • \text{Max(IF, Read Reg, ALU, Memory, Write Reg)} = 2 \text{ ns}

Administrivia
° Project 6 (last): Due Today
° Next Readings: 7.5
° 11th homework (last): Due Friday 4/30 7PM
  • Exercises 2.6, 2.13, 6.1, 6.3, 6.4

Administrivia: Rest of 61C
F 4/30 Review: Caches/TLB/VM; Section 7.5
M 5/3 Deadline to correct your grade record
W 5/5 Review: Interrupts / Polling; A.7
F 5/7 61C Summary / Your Cal heritage / HKN Course Evaluation
(Due: Final 61C Survey in lab; Return)
Sun 5/9 Final Review starting 2PM (1 Pimintel)
W 5/12 Final (5PM 1 Pimintel)
  • Need Alternative Final? Contact mds@cory

“What’s This Stuff (Potentially) Bad For?”
Linking Entertainment to Violence 100s of studies in recent decades have revealed a direct correlation between exposure to media violence--including video games--and increased aggression.
  • "We are reaching the stage of desensitization at which the inflicting of pain and suffering has become a source of entertainment; vicarious pleasure rather than revulsion. We are learning to kill, and we are learning to like it." Like the tobacco industry, “the evidence is there.”
  • The 14-year-old boy who opened fire on a prayer group in a Ky. school foyer in 1997 was a video-game expert. He had never fired a pistol before, but in the ensuing melee, he fired 8 shots, hit 8 people, and killed 3. The average law enforcement officer in the United States, at a distance of 7 yards, hits fewer than 1 in 5 shots.
  • Because of freedom of speech is a value that we don’t want to compromise, “it really comes down to the people creating these games. That's where the responsibility lies.”
N.Y. Times, 4/26/99
Pipeline Hazard: Matching socks in later load

A depends on D; stall since folder tied up

Problems for Computers

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support this combination of instructions (single person to fold and put clothes away)
  - Control hazards: Pipelining of branches & other instructions stall the pipeline until the hazard “bubbles” in the pipeline
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline (missing sock)

Single Memory is a Structural Hazard

Read same memory twice in same clock cycle

Structural Hazards limit performance

- Example: if 1.3 memory accesses per instruction (30% of instructions executed loads and stores) and only one memory access per cycle then
  - Average CPI \( \geq 1.3 \)
  - Otherwise resource is more than 100% utilized
Control Hazard Solutions

- Stall: wait until decision is clear
  - Move up decision to 2nd stage by adding hardware to check registers as being read
  - Impact: 2 clock cycles per branch instruction ⇒ slow

- Predict: guess one direction, then back up if wrong
  - For example, Predict not taken
  - Impact: 1 clock per branch instruction if right, 2 if wrong (right ∼ 50% of time)
  - More dynamic scheme: history of 1 branch (∼ 90%)

Example Nondelayed vs. Delayed Branch

- Redefine branch behavior (takes place after next instruction) “delayed branch”
  - Impact: 1 clock cycles per branch instruction if can find instruction to put in “slot” (∼ 50% of time)
Data Hazard on Register $1

add $1, $2, $3
sub $4, $1, $3
and $6, $1, $7
or $8, $1, $9
xor $10, $1, $11

Data Hazard on $1:

Dependencies backwards in time are hazards

add $1, $2, $3
sub $4, $1, $3
and $6, $1, $7
or $8, $1, $9
xor $10, r1, $11

Data Hazard Solution:

• “Forward” result from one stage to another
• “or” OK if define read/write properly

Forwarding (or Bypassing): What about Loads

• Dependencies backwards in time are hazards
• Can’t solve with forwarding
• Must stall instruction dependent on loads
Data Hazard Even with Forwarding

- Must insert stall or bubble in pipeline

Time (clock cycles)

```
lw $1, 0($2)
sub $4,$1,$6
and $6,$1,$7
or $8,$1,$9
```

Software Scheduling to Avoid Load Hazards

Try producing fast code for

```
a = b + c;
d = e - f;
a, b, c, d ,e, and f in memory
```

Slow code:
```
lw $2,b
lw $3,c
add $1,$2,$3
sw $1,a
lw $5,e
lw $6,f
sw $4,$5,$6
```

Fast code:
```
lw $2,b
lw $3,c
lw $5,e
sw $1,a
lw $6,f
sw $4,$5,$6
```

Advanced Pipelining Concepts (if time)

- Out-of-order Execution
- Superscalar execution
- State-of-the-Art Microprocessor

Pipeline Hazard: Stall

```
6 PM 7 8 9 10 11 12 1 2 AM
```

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

State of the Art: Alpha 21264

° 15 Million transistors
° 2 64KB caches on chip; 16MB L2 cache off chip
° Clock cycle time <1.7 nsec, or Clock Rate >600 MHz (Fastest Cray Supercomputer: T90 2.2 nsec)
  • 90 watts per chip!
° Superscalar: fetch up to 6 instructions/clock cycle, retires up to 4 instruction/clock cycle
° Execution out-of-order
Summary 1/2: Pipelining Overview

- Pipelining is a fundamental concept
  - Multiple steps using distinct resources
  - Exploiting parallelism in instructions

What makes it easy? (MIPS vs. 80x86)
- All instructions are the same length ➞ simple instruction fetch
- Just a few instruction formats ➞ read registers before decode instruction
- Memory operands only in loads and stores ➞ fewer pipeline stages
- Data aligned ➞ 1 memory access / load, store

Summary 2/2: Pipelining Overview

What makes it hard?
- Structural hazards: suppose we had only one cache?
  ➞ Need more HW resources
- Control hazards: need to worry about branch instructions?
  ➞ Branch prediction, delayed branch
- Data hazards: an instruction depends on a previous instruction?
  ➞ need forwarding, compiler scheduling