

## CS 61C: Great Ideas in Computer Architecture (Machine Structures)

### Instruction Level Parallelism

Instructors:  
Randy H. Katz  
David A. Patterson

<http://inst.eecs.Berkeley.edu/~cs61c/fa10>

11/4/10 Fall 2010 -- Lecture #29 1

## Agenda

- Review
- Instruction Set Design and Pipelined Execution
- Control Hazards
- Administrivia
- Branch Prediction
- Higher Level ILP
- Summary

11/4/10 Fall 2010 -- Lecture #29 2

## Review

### The BIG Picture

- Pipelining improves performance by increasing instruction throughput: exploits ILP
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Subject to hazards
  - Structure, data, control
- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure

E&E/0110 -- Lecture #29 3

## Pipelining and ISA Design

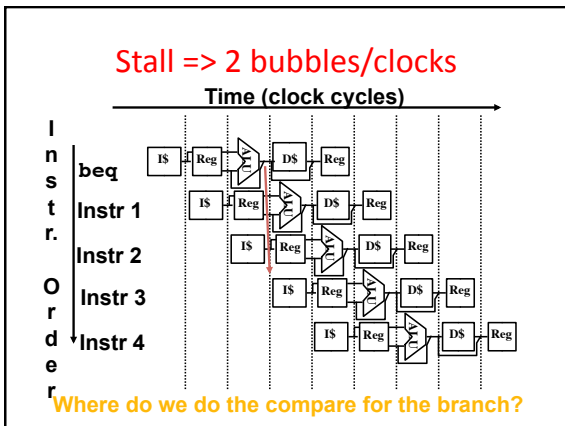
- MIPS Instruction Set designed for pipelining
- All instructions are 32-bits
  - Easier to fetch and decode in one cycle
  - x86: 1- to 17-byte instructions (x86 HW actually translates to internal RISC instructions!)
- Few and regular instruction formats, 2 source register fields always in same place
  - Can decode and read registers in one step
- Memory operands only in Loads and Stores
  - Can calculate address 3<sup>rd</sup> stage, access memory 4<sup>th</sup> stage
- Alignment of memory operands
  - Memory access takes only one cycle

E&E/0110 -- Lecture #29 4

## Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - Still working on ID stage of branch
- BEQ, BNE in MIPS pipeline
- Simple solution Option 1: *Stall* on every branch until have new PC value
  - Would add 2 bubbles/clock cycles for every Branch! (~ 20% of instructions executed)

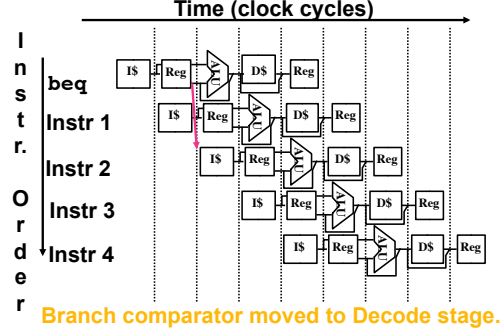
E&E/0110 -- Lecture #29 5



### Control Hazard: Branching

- Optimization #1:
  - insert **special branch comparator** in Stage 2
  - as soon as instruction is decoded (Opcode identifies it as a branch), immediately make a decision and set the new value of the PC
  - Benefit: since branch is complete in Stage 2, only one unnecessary instruction is fetched, so only one no-op is needed
  - Side Note: This means that branches are idle in Stages 3, 4 and 5.

### One Clock Cycle Stall



### Control Hazards

- Option 2: **Predict** outcome of a branch, fixup if guess wrong
  - Must cancel all instructions in pipeline that depended on guess that was wrong
- Simplest hardware if predict all branches NOT taken

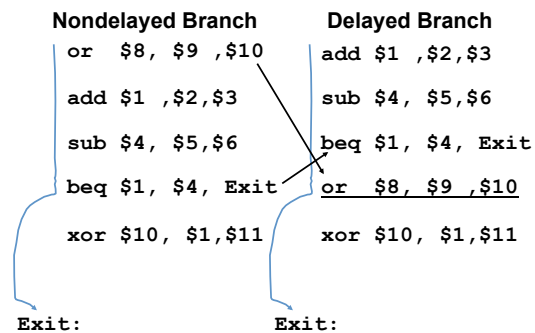
### Control Hazard: Branching

- Option #3: Redefine branches
  - Old definition: if we take the branch, none of the instructions after the branch get executed by accident
  - New definition: whether or not we take the branch, the single instruction immediately following the branch gets executed (called the **branch-delay slot**)
- The term “**Delayed Branch**” means we always execute inst after branch
- This optimization is used with MIPS

### Control Hazard: Branching

- Notes on **Branch-Delay Slot**
- Worst-Case Scenario: put a no-op in the branch-delay slot
- Better Case: find instruction preceding branch placed in the branch-delay slot without affecting flow of program
  - Re-ordering instructions is common way to speed up programs
  - Compiler usually finds such an instruction 50% of time
  - Jumps also have a delay slot...

### Example: Nondelayed vs. Delayed Branch



### Delayed Branch/Jump and MIPS ISA?

- Why does JAL put PC+8 in register 31?
- JAL executes following instruction (PC+4) so should return to PC+8

11/4/10 Fall 2010 – Lecture #29 13

### Agenda

- Review
- Instruction Set Design and Pipelined Execution
- Control Hazards
- Administrivia
- Branch Prediction
- Higher Level ILP
- Summary

11/4/10 Fall 2010 – Lecture #29 14

### Administrivia

- Project 3: Thread Level Parallelism + Data Level Parallelism + Cache Optimization
  - Due Part 2 due Saturday 11/13
- Project 4: Single Cycle Processor in Logicsim
  - Due Part 2 due Saturday 11/27
  - Face-to-Face grading: Signup for time slot in last week
- Extra Credit: Fastest Version of Project 3
  - Due Monday 11/29 Midnight
- Final Review: TBD (Vote via Survey!)
  - “Please narrow what we need to study on review”
- Final: Mon Dec 13 8AM-11AM (TBD)

11/4/10 Fall 2010 – Lecture #27 15

### Greater Instruction-Level Parallelism (ILP)

- Deeper pipeline (5 => 10 => 15 stages)
  - Less work per stage => shorter clock cycle
- Multiple issue “superscalar”
  - Replicate pipeline stages => multiple pipelines
  - Start multiple instructions per clock cycle
  - CPI < 1, so use Instructions Per Cycle (IPC)
  - E.g., 4GHz 4-way multiple-issue
    - 16 BIPS, peak CPI = 0.25, peak IPC = 4
  - But dependencies reduce this in practice

11/4/10 Fall 2010 – Lecture #29 16

§4.10 Parallelism and Advanced Instruction Level Parallelism

### Multiple Issue

- Static multiple issue
  - Compiler groups instructions to be issued together
  - Packages them into “issue slots”
  - Compiler detects and avoids hazards
- Dynamic multiple issue
  - CPU examines instruction stream and chooses instructions to issue each cycle
  - Compiler can help by reordering instructions
  - CPU resolves hazards using advanced techniques at runtime

11/4/10 Fall 2010 – Lecture #29 17

### Superscalar Laundry: Parallel per stage

11/4/10 Fall 2010 – Lecture #29 18

### Static Multiple Issue

- Compiler groups instructions into “issue packets”
  - Group of instructions that can be issued on a single cycle
  - Determined by pipeline resources required
- Think of an issue packet as a very long instruction
  - Specifies multiple concurrent operations

EEB/40100 – Lecture #29

19

### Scheduling Static Multiple Issue

- Compiler must remove some/all hazards
  - Reorder instructions into issue packets
  - No dependencies with a packet
  - Possibly some dependencies between packets
    - Varies between ISAs; compiler must know!
  - Pad with nop if necessary

EEB/40100 – Lecture #29

20

### MIPS with Static Dual Issue

- Two-issue packets
  - One ALU/branch instruction
  - One load/store instruction
  - 64-bit aligned
    - ALU/branch, then load/store
    - Pad an unused instruction with nop

Address	Instruction type	Pipeline Stages						
		IF	ID	EX	MEM	WB		
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store		IF	ID	EX	MEM	WB	
n + 8	ALU/branch			IF	ID	EX	MEM	WB
n + 12	Load/store				IF	ID	EX	MEM
n + 16	ALU/branch					IF	ID	EX
n + 20	Load/store						IF	ID

EEB/40100 – Lecture #29

21

### Hazards in the Dual-Issue MIPS

- More instructions executing in parallel
- EX data hazard
  - Forwarding avoided stalls with single-issue
  - Now can't use ALU result in load/store in same packet
    - add \$t0, \$s0, \$s1
    - load \$s2, 0(\$t0)
    - Split into two packets, effectively a stall
- Load-use hazard
  - Still one cycle use latency, but now two instructions
- More aggressive scheduling required

EEB/40100 – Lecture #29

22

### Scheduling Example

- Schedule this for dual-issue MIPS

```

Loop: lw  $t0, 0($s1)    # $t0=array element
      addu $t0, $t0, $s2 # add scalar in $s2
      sw  $t0, 0($s1)    # store result
      addi $s1, $s1, -4  # decrement pointer
      bne $s1, $zero, Loop # branch $s1!=0
    
```

	ALU/branch	Load/store	cycle
Loop:	nop	lw \$t0, 0(\$s1)	1
	addi \$s1, \$s1, -4	nop	2
	addu \$t0, \$t0, \$s2	nop	3
	bne \$s1, \$zero, Loop	sw \$t0, 4(\$s1)	4

■  $IPC = 5/4 = 1.25$  (c.f. peak  $IPC = 2$ )

EEB/40100 – Lecture #29

23

### Loop Unrolling

- Replicate loop body to expose more parallelism
  - Reduces loop-control overhead
- Use different registers per replication
  - Called “register renaming”
  - Avoid loop-carried “anti-dependencies”
    - Store followed by a load of the same register
    - Aka “name dependence”
      - Reuse of a register name

EEB/40100 – Lecture #29

24

### Loop Unrolling Example

	ALU/branch	Load/store	cycle
Loop:	addi \$s1, \$s1, -16	lw \$t0, 0(\$s1)	1
	nop	lw \$t1, 12(\$s1)	2
	addu \$t0, \$t0, \$s2	lw \$t2, 8(\$s1)	3
	addu \$t1, \$t1, \$s2	lw \$t3, 4(\$s1)	4
	addu \$t2, \$t2, \$s2	sw \$t0, 16(\$s1)	5
	addu \$t3, \$t4, \$s2	sw \$t1, 12(\$s1)	6
	nop	sw \$t2, 8(\$s1)	7
	bne \$s1, \$zero, Loop	sw \$t3, 4(\$s1)	8

- IPC = 14/8 = 1.75
  - Closer to 2, but at cost of registers and code size

EE/QT300 – Lecture #29

25

### Dynamic Multiple Issue

- “Superscalar” processors
- CPU decides whether to issue 0, 1, 2, ... each cycle
  - Avoiding structural and data hazards
- Avoids the need for compiler scheduling
  - Though it may still help
  - Code semantics ensured by the CPU

EE/QT300 – Lecture #29

26

### Dynamic Pipeline Scheduling

- Allow the CPU to execute instructions *out of order* to avoid stalls
  - But commit result to registers in order
- Example
 

```
lw    $t0, 20($s2)
addu  $t1, $t0, $t2
subu  $s4, $s4, $t3
slli  $t5, $s4, 20
```

  - Can start subu while addu is waiting for lw

EE/QT300 – Lecture #29

27

### Why Do Dynamic Scheduling?

- Why not just let the compiler schedule code?
- Not all stalls are predictable
  - e.g., cache misses
- Can’t always schedule around branches
  - Branch outcome is dynamically determined
- Different implementations of an ISA have different latencies and hazards

EE/QT300 – Lecture #29

28

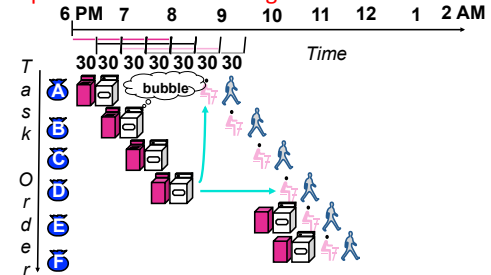
### Speculation

- “Guess” what to do with an instruction
  - Start operation as soon as possible
  - Check whether guess was right
    - If so, complete the operation
    - If not, roll-back and do the right thing
- Common to static and dynamic multiple issue
- Examples
  - Speculate on branch outcome (Branch Prediction)
    - Roll back if path taken is different
  - Speculate on load
    - Roll back if location is updated

EE/QT300 – Lecture #29

29

### Pipeline Hazard: Matching socks in later load



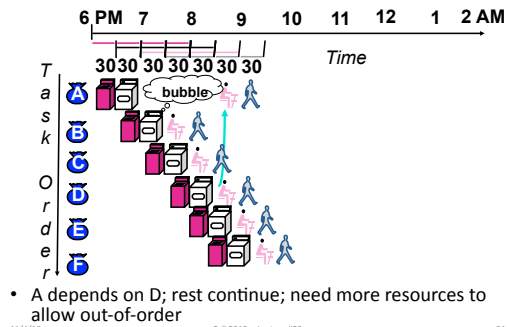
- A depends on D; stall since folder tied up;

11/4/10

Fall 2010 – Lecture #29

30

## Out-of-Order Laundry: Don't Wait



## Does Multiple Issue Work?

### The BIG Picture

- Yes, but not as much as we'd like
- Programs have real dependencies that limit ILP
- Some dependencies are hard to eliminate
  - e.g., pointer aliasing
- Some parallelism is hard to expose
  - Limited window size during instruction issue
- Memory delays and limited bandwidth
  - Hard to keep pipelines full
- Speculation can help if done well

EB/0200 - Lecture #29

32

## “And in Conclusion..”

- Pipeline challenge is hazards
  - Forwarding helps w/many data hazards
  - Delayed branch helps with control hazard in 5 stage pipeline
  - Load delay slot / interlock necessary
- More aggressive performance:
  - Longer pipelines
  - Superscalar
  - Out-of-order execution
  - Speculation

11/4/10

Fall 2010 - Lecture #29

33