

## When / Where

Monday, 31 March 2014, 9:30-10:40 CDT

1112 P. Taylor Hall (Here)

## Conditions

Closed Book, Closed Notes

Bring one sheet of notes (both sides), 216 mm × 280 mm.

No use of communication devices.

## Format

Several problems, short-answer questions.

## Resources

Solved tests and homework: <http://www.ece.lsu.edu/ee4720/prev.html>

Statically Scheduled MIPS Study Guide:...

... <http://www.ece.lsu.edu/ee4720/guides/ssched.pdf>

## Study Recommendations

Study **this semester's homework assignments**. Similar problems may appear on the exam.

Solve Old Problems—memorizing solutions **is not the same** as solving.

Following and understanding solutions **is not the same** as solving.

Use the solutions for **brief hints** and **to check** your own solutions.

## MIPS Programming and Instruction Use

Should be able to write MIPS programs.

Should be able to easily understand MIPS programs.

Should be able to use other ISAs' instructions in examples.

Not required to memorize instruction names, except for common MIPS instructions.

## MIPS Implementation

Implementation Diagrams and Pipeline Execution Diagrams

They are a *team*, so study them together.

## ISA Families

MIPS v. SPARC

ISA v. Implementation.

*See Lecture Slides 1*—<http://www.ece.lsu.edu/ee4720/2014/lsl1i01.pdf>

Why separating ISA from implementation is important.

Why it's hard to design an ISA without bias towards first implementation.

## MIPS ISA

*See MIPS Overview*—<http://www.ece.lsu.edu/ee4720/2014/lmips.s.html>

Write and read programs.

Understand how instructions are coded.

## Statically Scheduled MIPS Implementations

See *Lecture Slides 6*—<http://www.ece.lsu.edu/ee4720/2014/lslis06.pdf> and *Statically Scheduled Study Guide*—<http://www.ece.lsu.edu/ee4720/guides/ssched.pdf>

### Unpipelined Implementation

Understand relationship between insn format and connections to register file, etc.

### Pipelined Implementation Versions

Basic (no bypassing).

ALU bypassing, branch in ID.

Many variations covered in class and in assignments.

### Terminology

Dependency Definitions

Hazard Definitions

## MIPS Implementation Skills

For a Given Pipelined Implementation

Show pipeline execution diagrams.

Show register contents at any cycle.

Design control logic.

Determine CPI.

Add bypass paths.

Add functional units.

## ISA Families: RISC, CISC, VLIW

See *ISA Families Overview*—<http://www.ece.lsu.edu/ee4720/2014/lrisc.s.html>

Key distinguishing features and their rationale.

RISC: Goal: Easy pipelining, simple (relatively) implementation.

Fixed-length instructions, balanced work, aligned memory access.

CISC: Goal: Powerful instructions, compact code size.

Flexible operand types, multiple-activity instructions.

VLIW: Goal: Easy superscalar implementation.

Bundled instructions, dependence information.



## RISC to RISC Differences: MIPS v. SPARC

*See ISA Families Overview*—<http://www.ece.lsu.edu/ee4720/2014/lrisc.s.html>

Opcode fields (and their extensions).

Immediate operands, immediate sizes.

Integer branches.

## CISC ISA Features

Goal: Powerful Instructions

Powerful: Many memory addressing modes.

Powerful: Many immediate sizes.

Powerful: Few limits on where addressing modes can be used.

## Compilers and Optimization

*See Optimization Lecture Notes*—<http://www.ece.lsu.edu/ee4720/2014/lopt.s.html>

Steps in building and compiling.

Basic optimization techniques, compiler optimization switches.

Profiling.

Compiler ISA and implementation switches.

How programmer typically uses compiler switches (options).

## Benchmarks

*See Lecture Slides 2—<http://www.ece.lsu.edu/ee4720/2014/lsl102.pdf>,  
SPECcpu description—<http://www.spec.org/benchmarks.html>, and the SPECcpu run and  
reporting rules—<http://www.spec.org/cpu2006/Docs/runrules.html>*

Benchmark types: kernel, synthetic, real progs.

### SPECcpu Benchmark Suite

What suite measures.

Benchmark programs (types, how they were selected).

Why it's useful for computer engineers.

Why results might be trusted:

SPEC membership and their interests.

Rules for running benchmarks and disclosing results.

## Interrupts and Exceptions and Traps

Difference between interrupt, exception, trap.

Causes of exceptions, role of handler.

Privileged Mode.

Pipeline activity leading to execution of handler.

SPARC trap mechanism. (Trap base register, etc.).

Achieving precise exceptions with floating-point operations.

## Floating-Point and other Long-Latency Operations

Types of operations. (Floating point and maybe load.)

Degree of pipelining: Initiation interval, latency.

Detecting functional unit structural hazards.

Detecting WB structural hazards: reservation register.

Detecting and handling RAW hazards: ID-stage v. pre-WB stall.

Control logic.

Handling WAW hazards.

Why precise exceptions more difficult in pipelined impl.

## *n*-Way Superscalar

Duplication of Resources.

For  $n$  instructions / cycle: Fetch, decode, rename, writeback, commit.

For  $< n$  instructions: load/store, floating-point units.

Costs:  $\propto n$  functional units;  $\propto n^2$  bypass, control.

## Performance Limiters

Device Limiters: Lower clock with increasing distances.

Aligned groups impose fetch restrictions that reduce fetch efficiency.

More stalls due to data dependencies.

More squashes due to branches.

## VLIW

Difference with superscalar: instruction bundling.

Dependence information in bundles.



## Deeper (Super) Pipelining

Relationship between stage splitting, clock frequency and performance.

Relationship between stage splitting and cost.

Latch setup time.

More stalls due to data dependencies.

More bypass paths needed.

## Vector Instructions

Vector registers.

How vector instructions operate on vector registers.

Advantages and disadvantages and differences ...

... between vector instructions and superscalar systems.

## Instruction-Level Parallelism

Definition.

Relationship to superscalar implementations.

## Explicit Parallelism

Definition.

Role of programmer and compiler.

Relationship to multi-core chips.