What is the Course About?

- Instruction level parallelism
- Static and dynamic scheduling
- Pipelined architecture
- Chapters 3 & 4, Appendix A

Supplemented with case studies

- Cache design
- Memory system optimizations
- Chapter 5

- Storage technologies
- I/O systems
- Chapter 7

- Parallel architectures
- Shared and Distributed Memory Systems
- Interconnection Networks
- Clusters
- Chapters 6 & 8
The Processor Datapath

- **Performance Goal**: Exploit instruction-level parallelism to execute multiple instructions per cycle

  - CPI > 1
  - CPI < 1

  - Pipelining
  - Instruction level parallelism

  \[ \text{CPI} = 1.0 \]

- Multi-cycle datapaths

- Architectural and compiler techniques to systematically increase the number of instructions executed per clock cycle
  - Note CPI is not the whole story!

- Both hardware and compiler techniques are dependent on the instruction set architecture (ISA)
  - Set of instructions and resources they manipulate
  - The ISA is interface between the hardware and the compiler
Basic Instruction Set Classes

- Specificity vs. generality in the use of registers
  - What would ease the compiler’s burden?
- What does technology favor?
Role of the Compiler

- What does the compiler see?
  - Resources such as registers and caches
  - Datapath such as the pipeline structure
- What can the compiler optimize?
Basics of RISC Instruction Sets

- Load/store architecture
  - All operations are on data stored in registers
  - Memory operations only move data to/from registers
- Instruction formats: Simple, fixed field, small number
  - Lead to simplifications in the handling of pipelined processors
  - Uniformity of instruction decoding and control
- Compiler plays an integral role

- Register to register
- Conditional branches
- Immediate operations
- Memory operations
- Unconditional branches
The MIPS Instruction Set Architecture

- 32, 64 bit general purpose registers
  - R0 hardwired to 0
- 32 single precision floating point registers
  - Pairs used for double precision
- 8, 16, 32, and 64 bit data types
  - Instructions for manipulating different types
- Only immediate and displacement addressed modes
Sample Instructions

- **LD R1, 64(R2)**
  - displacement addressing mode
  - Load/store instructions for multiple operand sizes
- **ADD R4, R1, R2**
  - Register-to-register operations
- **SW R6, 22(R2)**
  - Indexed addressed mode
- **BNE R1, R2, LOOP, SLT R1, R2, R3**
  - Use general purpose registers to store results of relational operations
  - Comparison can be used to synthesize all manner of conditions
- **Figure 2.31, pg. 137**
The Road to Performance

- Performance can be defined in many distinct ways
  - Cost, execution time, power
  - We will focus on execution time
- Overriding concern is program correctness
- We will use sequential consistency as our specification of correctness
  - Informal Definition: The effect of program execution must be as if the program instructions were executed in the order that they appeared in the program
Pipelining Basics

- Multiple instructions in various stages of execution concurrently
- Ideally we have CPI = 1
- What are the constraints on concurrent execution?
Hurdles to Pipelining

- **Structural hazards**
  - Results from conflict for shared resources

- **Data Hazards**
  - Can occur when data dependencies exist between two concurrently executing instructions

```
LD R4, 16(R2)  
ADD R6, R4, R2
```

- **Control hazards**
  - Caused by delays in evaluating the result of a branch instruction that changes the flow of control

- **Occurrence of hazards make it necessary to stall the pipeline**
Handling Structural Hazards

- **Examples of sources of stalls**
  - Register file ports
  - Data and instruction memories

- **Options**
  - Stall
    - Add hardware → increased cost

- **Instruction set design can simplify resource usage**
Data Hazards

- Updates to register file happen after operand read by data dependent instruction
- What are the different types of data hazards?
Data Hazards

• **Read After Write (RAW)**
  - \(\text{instr}_J\) tries to read operand before \(\text{instr}_I\) writes it

• **Example:**
  
  I: ADD R1, R2, R3  
  
  J: SUB R4, R1, R3
  
  - This results from a data dependence in the program

• **Write After Read (WAR)**
  - \(\text{instr}_J\) tries to write operand before \(\text{instr}_I\) reads it

• **Example:**
  
  I: ADD R1, R2, R3  
  
  J: SUB R2, R1, R3
  
  - Generally does not occur in most pipelines
  
  - This results from an anti-dependence in the program
• **Write After Write (WAW)**
  - instr$_j$ tries to write an operand before instr$_i$ writes it
    
    I: ADD R4, R2, R3
    
    J: SUB R4, R1, R3
  
  - All writes serialized in the MIPS pipeline in WB and hence cannot occur
  
  - This results from an output dependence in the source program

• **Anti- and output dependencies are also called name dependencies**
Forwarding to Avoid Data Hazards

- Note that instructions are actually executed in order!
  - The results are in the pipeline and not in the register file
  - Go get them!
- Forwarding can take place from anywhere in the pipeline
Implementation of Forwarding

- Provide data paths from all possible sources of data
- Provide pipeline control to select the correct operands
- Figure A.22, pg. A-36
Load Hazard

- The load instruction hazard
- Structure of the pipeline does not permit forwarding
  - Add stall cycle
- How about compiler scheduling?
The Pipelined Datapath

- Behavior of conditional branches?
- Branch penalty?
- Solution #1 – stall the pipeline
- Solution #2 – predict not taken
- Solution #3 - predict taken
Load Delay Slots

- Solution #4 - compiler scheduling to fill branch delay slots
Dealing with Exceptions

- Exceptions can occur in any stage
- Precise exceptions/interrupts and re-startable pipelines
- Challenges in handling precise exceptions
  - Out of order instructions
  - Branch instructions
  - Out order occurrence of exceptions
Review

- Operation of pipelined, in-order issue, in-order execution datapath
- Techniques to avoid structural, data, and control hazards
  - Hardware techniques vs. software techniques
  - Performance implications
- Handling of exceptions
- Impact of the ISA on the complexity of pipelined operation and control
- Now how do we get beyond the CPI < 1 barrier?