Computer Organization
CS 231-01

Processors

Dr. William H. Robinson
August 30, 2004

http://eecs.vanderbilt.edu/courses/cs231/

Topics

“It is quite a three-pipe problem.”
– Sir Arthur Conan Doyle
The Adventures of Sherlock Holmes

• General announcements
• Example computers from Tanenbaum
• Five parts of a computer
• Improving processor performance
• Assignment
  – Continue reading Chapter 2 in Tanenbaum

Supplemental Textbook

• Computer Organization and Design Second Edition:
The Hardware/Software Interface
  – David Patterson
  – John Hennessy

Computer Examples

• Pentium II
  – From Intel
  – Complex Instruction Set Computer (CISC)
• UltraSPARC II
  – From Sun Microsystems
  – Reduced Instruction Set Computer (RISC)
• picoJava II
  – Based on Java Virtual Machine (JVM)
  – Stack machine
Introduction to Pentium II

- All Intel chips are backward compatible
- Pentium comes from Greek word for “five”

### CISC Architectures

- S/W interpreter increased the computing power of simple H/W design
- Provided 200 – 300 instructions
  - But how many of those were really used?
- Must still provide support for legacy code
  - Intel processors are CISC/RISC hybrid

### Introduction to UltraSPARC II

- Entrepreneurs from Stanford
  - Stanford University Network (SUN-1)
- Scalable Processor ARCHitecture
- Difference between SPARC architecture and SPARC chips
- Textbook refers to 64-bit V9 chip

### RISC Design Principles

- All instructions directly executed by H/W
  - Eliminate overhead of interpretation
- Maximize rate instructions are issued
  - Utilize parallelism within program
- Instructions should be easy to decode
  - Quickly determine resources required
- Only loads and stores should reference memory
  - Memory access is longer and unpredictable
- Provide plenty of registers
  - Avoid memory access penalty for flushing data set
Introduction to picoJava II

- Java programming language
  - C++ without security problems

- Essentially a H/W Java Virtual Machine (JVM)
  - Directly execute JVM binaries

- Value comes from embedded systems market
  - Performance requirements prevent interpretation
  - Lack of memory prevents Just In Time (JIT) compilation

Instruction Set Architecture

- Interface between software and hardware
  - Both compilers and hardware must understand ISA
    - Compilers translate high-level language into object code
    - Hardware must directly execute or interpret ISA

Which is easier to change/design???

Comparing Number of Instructions

<table>
<thead>
<tr>
<th>Code sequence for ((C = A + B)) for four classes of instruction sets:</th>
<th>Stack</th>
<th>Accumulator</th>
<th>Register (register-memory)</th>
<th>Register (load-store)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push A</td>
<td>Load A</td>
<td>Load R1,A</td>
<td>Load R1,A</td>
<td></td>
</tr>
<tr>
<td>Push B</td>
<td>Add B</td>
<td>Add R1,B</td>
<td>Add R1,B</td>
<td>Load R2,B</td>
</tr>
<tr>
<td>Add</td>
<td>Store C</td>
<td>Store C, R1</td>
<td>Store C, R1</td>
<td></td>
</tr>
<tr>
<td>Pop C</td>
<td></td>
<td>Add R3,R1,R2</td>
<td>Store C,R3</td>
<td></td>
</tr>
</tbody>
</table>

The Big Picture

- Since 1946 (ENIAC I) all computers have had 5 components
Computer Organization

Five Basic Parts
- Control
- Datapath
  - ALU
  - Registers
- Memory
- Input
- Output

Data Path of von Neumann Machine

- Critical path of processor design
- Does the computational work of computer

Instruction Execution
1. Fetch next instruction from memory
2. Change PC to next instruction
3. Decode instruction
4. Determine data memory location, if necessary
5. Get data word, if necessary
6. Execute instruction
7. Go to step 1

Architecture Implementation

- **H/W and S/W are logically equivalent**
  - Basic fetch-decode-execute cycle can be built with either method

- **Determining Factors**
  - Cost? H/W is more expensive
  - Speed? H/W is faster
  - Reliability? Depends on the design
  - Frequency of expected changes? S/W can change easier
Improve Performance with Parallelism

• **Data-Level Parallelism (DLP)**
  – Perform same instruction on different data
  – Extending the ISA

• **Instruction-Level Parallelism (ILP)**
  – Perform independent instructions concurrently
  – H/W and S/W techniques

• **Thread-Level Parallelism (TLP)**
  – Perform independent tasks concurrently
  – Multiprocessors

Pipelining is Natural!

**Laundry Example**

• Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
  - 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
- If they learned pipelining, how long would laundry take?

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
- Use different resources to execute multiple tasks simultaneously
- Potential speedup = Number pipe stages
- Time to “fill” pipeline and time to “drain” it reduces speedup
- Stall for Dependences

Adapted from David Patterson’s CS 252 lecture notes. Copyright 2001 UCB.

Classic Five-Stage Pipeline

- Ifetch: Instruction Fetch
  – Fetch the instruction from the Instruction Memory
- Reg/Dec: Registers Fetch and Instruction Decode
- Exec: Perform calculation for instruction
- Mem: Access the data from the Data Memory
- Wr: Write the data back to the register file

Adapted from David Patterson’s CS 252 lecture notes. Copyright 2001 UCB.

Pipelining

- Improve performance by increasing instruction throughput
- Ideal speedup is number of stages in the pipeline
- Do we achieve this?

Conventional Pipelined Execution Representation

- Assume each stage is one cycle
- # of cycles = # of instructions + (# of stages – 1)
  – Assuming full utilization of pipeline

Adapted from David Patterson’s CS 252 lecture notes. Copyright 2001 UCB.
Five-Stage Pipeline

- One instruction completes every cycle after pipeline is full

\[ \text{MIPS} = \frac{1000}{\text{cycle} \times \text{time(ns)}} \]

Superscalar Architectures

- Duplicates hardware; becomes costly beyond 2 pipelines

Array Processor

- Exploit data parallelism in applications
- Specialized architecture for higher performance efficiency
- Single-Instruction, Multiple-Data (SIMD)
Multiprocessors

Fig. 2-8. (a) A single-bus multiprocessor. (b) A multicomputer with local memories.

Summary

• Datapath performs the computational work of the computer
• RISC design principles seek to improve processor performance
• Pipelining improves performance by increasing instruction throughput