Advanced Topics
Announcements

• **HW8 deadline extended to Friday 5/14**

• **Instructor office hours next week**
  – Thursday 5/20, 4:30-6:00pm
  – Sunday 5/23, 10:00-11:30am

• **Additional TA office hours during study period will be posted on Piazza soon**

• **Fill out 2300 course evaluation (by Mon 5/17)**
  – Comments not required but very welcome
Final Exam

• **When:** Sunday May 23, 7:30PM (110mins)

• **Where:** Exam zoom room
  – Same protocol (open book, open notes, camera on)

• **What:** Covers the entire course
  – With a particular emphasis on computer organization (Lectures 15~24)
    • Programmable microprocessor, pipelining, caches, performance measurement, virtual memory, exceptions, I/O
  – A sample exam is posted on CMS
    • Solution to be released over the weekend
Performance Tradeoffs

CPU execution time = I x CPI x CT

• A decision regarding the ISA or processor organization often improves one aspect of CPU execution time at the expense of another
  – I versus CPI
  – I versus CT
  – CPI versus CT
Discussion: Performance Tradeoff
Example 1

ADD 8(R1),0(R1),4(R1)  vs.  LW   R2,0(R1)
LW     R3,4(R1)
ADD    R2,R2,R3
SW     R2,8(R1)

Complex Instruction vs. Simple Instruction
Discussion: Performance Tradeoff

Example 2

Shallow vs. Deep Pipelining
Parallelism: Making our Processor *Fast*

- Processor architects improve performance through hardware that exploits the different types of *parallelism* within computer programs.

- **Instruction-Level Parallelism (ILP)**
  - (fine-grain) parallelism within a sequential program

- **Thread-level parallelism (TLP)**
  - (coarse-grain) parallelism among different threads in a program

- **Data-level parallelism (DLP)**
  - parallelism among the data within a program
Instruction-Level Parallelism (ILP)

• Refers to the parallelism found within a sequential program

• Consider the ILP in this program segment

  ADD  R1,R2,R3
  OR   R4,R4,R3
  SUB  R5,R2,R3
  AND  R6,R6,R2
  ADDI R7,R7,3
  LW   R2,R3,0

• Superscalar pipelines exploit ILP by duplicating the pipeline hardware
Two-Way Superscalar Pipeline

IF    ID    EX    MEM    WB
IM    Reg    ALU    DM    Reg
IF/ID  ID/EX  EX/MEM  MEM/WB
Instruction Sequence on 2W SS

IM
ADD R1, R2, R3
OR R4, R4, R3
SUB R5, R2, R3
AND R6, R6, R2
ADDI R7, R7, 3
LW R2, R3, 0

Reg

ALU

ALU

DM

Reg
Instruction Sequence on 2W SS

ADD R1, R2, R3
OR R4, R4, R3
SUB R5, R2, R3
AND R6, R6, R2
ADDI R7, R7, 3
LW R2, R3, 0

ADD R1, R2, R3
OR R4, R4, R3
Instruction Sequence on 2W SS

- **IM**
  - ADD R1, R2, R3
  - OR R4, R4, R3
  - SUB R5, R2, R3
  - AND R6, R6, R2
  - ADDI R7, R7, 3
  - LW R2, R3, 0

- **Reg**
- **ALU**
- **DM**
- **Reg**

- SUB R5, R2, R3
- ADD R1, R2, R3
- AND R6, R6, R2
- OR R4, R4, R3

Lecture 25: 12
Instruction Sequence on 2W SS

ADD R1, R2, R3
OR R4, R4, R3
SUB R5, R2, R3
AND R6, R6, R2
ADDI R7, R7, 3
LW R2, R3, 0
ARM Cortex-A8 Microprocessor

Apple iPhone 4, iPod Touch (3rd & 4th gen), iPad; Motorola Droid, Droid X, Droid 2; Palm Pre, Pre 2; Samsung Omnia HD, Wave 8500, i9000 Galaxy S, P1000 Galaxy Tab; HTC Desire; Google Nexus S; Nokia N900; Sony Ericsson Satio, Xperia X10, etc, etc
Cortex-A8 Processor Pipeline

- 2-way superscalar

- Average CPI of 1.1

- 13 stages for integer instructions, 3 major sections
  - Instruction Fetch, Instruction Decode, Execute

- Up to 1GHz clock frequency

- ~0.5W @ 1GHz (processor core only)
Caches and TLBs

• **Identical L1 instruction and data caches**
  – 16KB or 32KB
  – 4-way set associative
  – 64-byte block size
  – Random replacement policy

• **32 entry, fully associative ITLB and DTLB**

• **L2 cache**
  – Up to 1MB, 8-way set associative, 64 byte block size, random replacement
Data Dependences Limit SS Execution

• Consider this program sequence

```
ADD  R1, R2, R3
OR   R4, R1, R3
SUB  R5, R2, R3
AND  R6, R6, R5
```

• The ADD and the OR, and the SUB and AND, cannot execute at the same time
  – Cortex-A8 limits dual issue in this case

• Addressed by \textit{out-of-order execution}
Out-of-Order Execution

- Processor can execute instructions out of the original program order

\[
\begin{align*}
\text{ADD} & \quad \text{R1, R2, R3} \\
\text{OR} & \quad \text{R4, R1, R3} \\
\text{SUB} & \quad \text{R5, R2, R3} \\
\text{AND} & \quad \text{R6, R6, R5}
\end{align*}
\]

- One key component is an *issue queue* that tracks the availability of source operands

![Diagram](diagram.png)
ARM Cortex-A9

- **Successor to the Cortex-A8**
- **Superscalar pipeline with out-of-order execution**
  - Issues up to 4 instruction each clock cycle
- **ITLB and DTLB + L2 TLB**

Apple iPhone 4S, iPad2; Motorola Droid Bionic, Altrix 4G, Xoom; Blackberry Playbook; Samsung Galaxy S II, Galaxy S III; HTC Sensation, EVO 3D; LG Optimus 2X, Optimus 3D; Lenovo IdeaPad K2, ASUS Eee Pad Transformer; Acer ICONIA TAB A-series, etc, etc
A More Troublesome Piece of Code

• Now consider this program sequence

\[
\text{Loop1: ADD } R1, R2, R3 \\
\text{OR } R4, R1, R3 \\
\text{SUB } R5, R4, R3 \\
\text{AND } R1, R6, R5 \\
\text{BEQ R2, R1, Loop1}
\]

• Superscalar pipeline would send instructions one by one through EX, MEM, and WB
  – 1 ALU, 1 memory port, and 1 RF port would sit idle, perhaps through 10000+ loop iterations!

• How do improve the hardware utilization?
Thread-Level Parallelism (TLP)

- Refers to the parallelism among different *threads* (usually identified by the programmer)
  - A thread is a path of execution within a program
  - Each uses its own registers but they share memory

- Consider two threads that we want to run

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop1: ADD R1, R2, R3</td>
<td>Loop2: LW R7, 0(R1)</td>
</tr>
<tr>
<td>OR R4, R1, R3</td>
<td>ADD R4, R7, R2</td>
</tr>
<tr>
<td>SUB R5, R4, R3</td>
<td>SUBI R5, R4, 1</td>
</tr>
<tr>
<td>AND R1, R6, R5</td>
<td>SW R5, 0(R1)</td>
</tr>
<tr>
<td>BEQ R2, R1, Loop1</td>
<td>BGEZ R5, Loop2</td>
</tr>
</tbody>
</table>

- We can run them on separate cores, or create a superscalar pipeline that can run them both at the same time
Two threads share many of the pipeline resources
Each thread has its own PC and (physical) registers
This is one example of multithreading, called Simultaneous Multithreading (SMT)
Data-Level Parallelism

• Consider the following C code

```c
char A[4], B[4], C[4];
for (i = 0; i < 4; i++)
    A[i] = B[i] + C[i];
```

• Arrays a, b, and c contain four 8-bit elements

• Same operation is done for each data element

• Can replace the 4 add operations in the loop above by 1 SIMD add instruction
SIMD Instructions

• *Single Instruction, Multiple Data*

• Special instructions for *vector data* (arrays)

• Identical operation is performed on each of the corresponding data elements

• Data elements are stored contiguously
  – 1 load (store) can read (write) all the elements at once
  – Register file is wide enough to hold all the elements in one register
Implementing the *for* Loop Using SIMD

```c
char A[4], B[4], C[4];

for (i = 0; i < 4; i++)
    A[i] = B[i] + C[i];
```

; load b and c from memory
LW    R0, 0(R4) ; R4 points to B
LW    R1, 0(R5) ; R5 points to C
; vector add
ADD.V  R2, R0, R1 ; one inst does four 8-bit adds!
; store result
SW     R2, 0(R3) ; R3 points to A

Assuming 32-bit registers and that a memory word is 32 bits
ILP, TLP, and DLP

• Many processors exploit all three
  – Best performance/watt achieved with each in moderation rather than one/two to the extreme

• ILP
  – Typically 2 to 6-way superscalar pipeline
  – Performance improvement tapers off with wider pipelines while power may increase significantly

• TLP
  – Support for multiple threads may require small amount of additional hardware over single threaded SS pipeline
  – May improve HW efficiency compared to SS alone

• DLP
  – Many applications (graphics, video and audio processing, etc) make this worthwhile
Ongoing Trends in Computer Systems
Microprocessor Scaling

![Graph showing the scaling of transistors, frequency, typical power, and number of cores over time from 1975 to 2015. The graph includes data points for various processors such as MIPS R2K, Intel Pentium 4, DEC Alpha 21264, and AMD Opteron. The x-axis represents the year, and the y-axis represents the logarithmic scale for transistors (in thousands), frequency (in MHz), typical power (in Watts), and number of cores. The graph also includes a trend line for parallel processor performance and sequential processor performance. Data partially collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond.]
Typical Multicore Chip Organization

Lecture 25: 29

Do we expect a 8X speedup with 8 cores?
Amdahl’s Law

• Used to find the maximum expected improvement to an overall system when only part of the system is improved

\[
\text{Execution Time}_{\text{enhanced}} = \frac{\text{Execution Time affected}}{\text{Amount of improvement}} + \text{Execution Time unaffected}
\]

• Example
  – Feature improves multiply operations by factor of 10
  – Total execution time of a program is 100 sec
  – Multiply operations consume 5 sec of the total

\[
\text{Execution Time}_{\text{enhanced}} = \frac{5}{10} + 95 = 95.5 \text{ sec}
\]
Modern Computers are Power Constrained

\[ \text{Power} = \frac{\text{Energy}}{\text{Second}} = \frac{\text{Energy}}{\text{Op}} \times \frac{\text{Ops}}{\text{Second}} \]

To get increased performance given a fixed power budget => energy/op must be reduced

\[ \text{Performance (Ops/Second)} \]

\[ \text{Energy per Operation} \]

For different power constraints:
- <<1W/chip
- ~1W/chip
- ~15W/chip
- ~50W/chip
- ~100W/chip
- >100W/chip

Lecture 25: 31
Hardware Specialization in Mobile Chips

• Special-purpose hardware accelerators improve performance and energy efficiency
Evolution of Smartphone Graphics

- Example: Modern smartphones contain highly specialized processing units to accelerate gaming tasks
Hardware Specialization in Datacenter

- ASIC- and FPGA-based accelerators are being deployed for a rich mix of compute-intensive applications in cloud datacenters
Lecture 2: 35

Flexibility vs. Efficiency: A Vast Trade-off Space

- “A New Golden Age for Computer Architecture!”
  John Hennessy and David Patterson, 2017 ACM Turing Award Lecture

Source: Bob Broderson, Berkeley Wireless group
Follow-on Courses

- ECE 2400 / ENGRD 2140: Computer Systems Programming
- ECE 3140 / CS 3420 (Embedded Systems)
- ECE 3400 (Intelligent Physical Systems)
- ECE 4750 / CS 4420 (Computer Architecture)
- ECE 4740 (Digital VLSI Design)
- CS 4410 (Operating Systems)
- CS 4120 (Compilers)
• Fill out 2300 course evaluation

Next Time

Final Exam