

Computer Architecture

- or -

Everything you ever wanted to know about computer architecture but were afraid to ask

- or -

What to do with a billion transistors?

- or -

CSE371 in one lecture

CSE240 vs CSE371

Why didn't we cover this in CSE240?

CSE240

- "How it works"
- Simplifications, but fundamental functions
- Big picture of all of systems

CSE371

- "How to make it fast"
 - Fast, cheap, low-power, reliable
- Hardware design
- Engineering class to the core
 - "An engineer can do for a dime what anyone else can do for a dollar"

Four Big Ideas

Caches

- Problem: memory is *really* slow
- Solution: remember recently accessed data on-chip

Virtual Memory

- Goal: isolate programs from each other
- Give programs the illusion of a single large address space

Instruction-level parallelism

- Execute instructions in parallel

Thread-level parallelism

- Execute instruction from many threads
- Multiprocessors and "hyper-threading"

Memory is Slow

Accessing memory is ~100ns

- Slow?

Slow relative to computation

- 2 Ghz processor (2 billion clocks cycles per second)
- An "add" takes ~0.5ns

Result:

- In the past, memory wasn't as slow (relatively speaking)
- Today, accessing memory is 100s of times slower than an ADD
- Getting worse all the time

Not All Memory Is Slow

“Main” memory

- Off-chip
 - Speed of light issues
- Made of dense DRAM (lots of bits per unit area)
- Optimized for cost per bit (not speed)

On-chip memory

- Faster
 - Physically closer to processor
- Made of less-dense, faster SRAM (fewer bits per unit area)
- For example, registers are really fast

How can we exploit fast on-chip memory?

CSE 240

5

Exploiting Fast On-Chip Memory: Caches

Software managed

- Some address range is “fast”
- Rest of memory is “slow”
- Software explicitly moves data between slow and fast memory

Hardware managed

- Fast memory “caches” slower memory
- Hardware dynamically replicates recently-access data
- **Library analogy**
- Exploits spatial and temporal locality
- 95% of the time, data found on-chip

Today, hardware managed caches are ubiquitous

- For example:
 - Two first-level 32KB caches (instructions and data)
 - Second-level 4MB, memory 2GB

CSE 240

6

Virtual Memory

Consider executing two LC-3 programs

- Perhaps both have `.orig x3000`
- How do we load them both at once?
- How do we isolate them from each other?
 - Recall LC-3’s memory protection register (MPR)

Solution: virtual memory vs physical memory

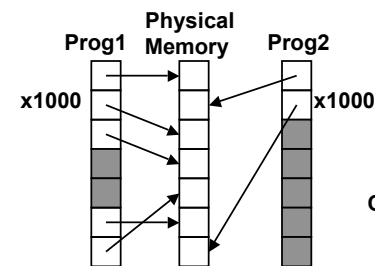
- Programs run in a “virtual” memory space
- Each section (or “page”) of virtual memory can map to a page of physical memory
- Managed by the operating system
 - Relying on hardware support

CSE 240

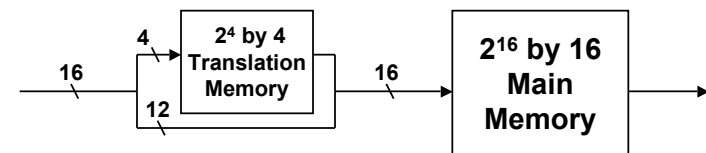
7

Virtual Memory

Consider a “page size” of 2^{12} words:



- Contents of translation memory
- Controlled by operating system
 - Different mapping for each program



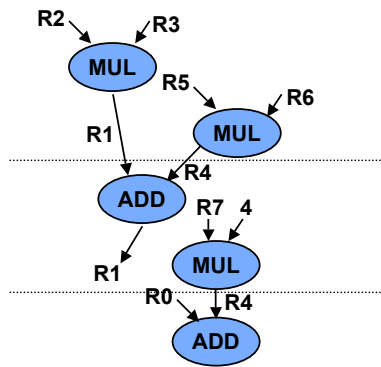
CSE 240

8

Instruction-Level Parallelism

Consider the following LC-3 snippet:

```
MUL R1, R2, R3
MUL R4, R5, R6
ADD R1, R4, R1
MUL R4, R7, #4
ADD R2, R0, R4
```



Why can't the processor execute some of these in parallel?

- It can!

Super-Scalar Execution and Dynamic Scheduling

What about executing them out-of-order?

- Yep, that too

Can even “rename” registers to uncover more parallelism

- Notice “name dependency” on R4 in example
- Additional “invisible” registers in processor

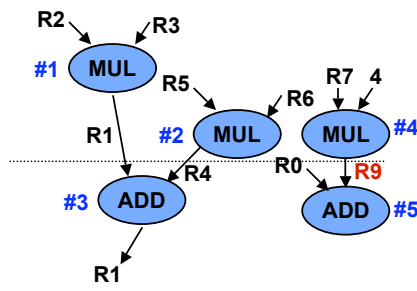
Modern processors

- Can execute 3 or 4 instructions per cycle
- Look at ~100 instructions to extract instruction-level parallelism
- Look beyond branches using *branch prediction*

Instruction-Level Parallelism

Remove false “name” dependence on R4:

```
MUL R1, R2, R3
MUL R4, R5, R6
ADD R1, R4, R1
MUL R9, R7, #4
ADD R2, R0, R9
```



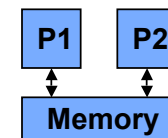
Modern processors

- Can execute 3 or 4 instructions per cycle
- Look at ~100 instructions to extract instruction-level parallelism
- Look beyond branches using *branch prediction*

Thread-Level Parallelism

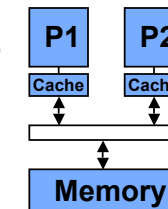
Two programs running at once

- Run them on multiple processors



Multiprocessing

- Multiple processors, one physical memory space
- Can communicate by loads and stores
 - What about caches?
 - Needs a hardware “cache coherence protocol”
- Yesterday: connect multiple chips
- Today and future: many “cores” per chip



Thread-Level Parallelism

Multithreading

- (Called “Hyperthreading” by Intel)
- One processor
- But, two PCs and two sets of registers
- Execute instructions from multiple programs or “threads”
- Advantages:
 - Find work when one thread is stalled on memory or dependent instructions
 - Share caches (no cache-coherence problem)

Which is better? Do both!

- Sun Microsystem’s Niagra
 - 8 processor cores, 4 threads each = 32 threads on a chip

Prediction (2003): in five year you won’t be able to buy a uniprocessor

A Final Caveat About Computer Architecture

Implementation technology keeps changing

- Vary rates of advance of each component
- Fixed costs vs. variable costs

Result:

- Today’s engineering tradeoffs different from yesterday’s
- Different from tomorrow’s, too

Qualifier Exam Story

Designs change over time...

- ...but design concepts change much more slowly