Last time in Lecture 5

- Control hazards (branches, interrupts) are most difficult to handle as they change which instruction should be executed next.
- Speculation commonly used to reduce effect of control hazards (predict sequential fetch, predict no exceptions).
- Branch delay slots make control hazard visible to software.
- Precise exceptions: stop cleanly on one instruction, all previous instructions completed, no following instructions have changed architectural state.
- To implement precise exceptions in pipeline, shift faulting instructions down pipeline to “commit” point, where exceptions are handled in program order.
Early Read-Only Memory Technologies

- Punched cards, From early 1700s through Jaquard Loom, Babbage, and then IBM
- Punched paper tape, instruction stream in Harvard Mk 1
- Diode Matrix, EDSAC-2 μcode store
- IBM Card Capacitor ROS
- IBM Balanced Capacitor ROS
Early Read/Write Main Memory Technologies

Babbage, 1800s: Digits stored on mechanical wheels

Williams Tube, Manchester Mark 1, 1947

Also, regenerative capacitor memory on Atanasoff-Berry computer, and rotating magnetic drum memory on IBM 650

Mercury Delay Line, Univac 1, 1951
Core Memory

- Core memory was first large scale reliable main memory
  - invented by Forrester in late 40s/early 50s at MIT for Whirlwind project
- Bits stored as magnetization polarity on small ferrite cores threaded onto a 2-dimensional grid of wires
- Coincident current pulses on X and Y wires would write cell and also sense original state (destructive reads)
- Robust, non-volatile storage
- Used on space shuttle computers until recently
- Cores threaded onto wires by hand (25 billion a year at peak production)
- Core access time $\sim 1\mu s$

DEC PDP-8/E Board,
4K words x 12 bits, (1968)
Semiconductor Memory

- Semiconductor memory began to be competitive in early 1970s
  - Intel formed to exploit market for semiconductor memory
  - Early semiconductor memory was Static RAM (SRAM). SRAM cell internals similar to a latch (cross-coupled inverters).

- First commercial Dynamic RAM (DRAM) was Intel 1103
  - 1Kbit of storage on single chip
  - charge on a capacitor used to hold value

- Semiconductor memory quickly replaced core in ‘70s
One Transistor Dynamic RAM
[Dennard, IBM]

1-T DRAM Cell

- word
- access transistor
- bit

Storage capacitor (FET gate, trench, stack)

- TiN top electrode ($V_{REF}$)
- Ta$_2$O$_5$ dielectric
- poly word line
- W bottom electrode
- access transistor
Modern DRAM Structure

[Samsung, sub-70nm DRAM, 2004]
DRAM Architecture

- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays
DRAM Operation

Three steps in read/write access to a given bank

• Row access (RAS)
  – decode row address, enable addressed row (often multiple Kb in row)
  – bitlines share charge with storage cell
  – small change in voltage detected by sense amplifiers which latch whole row of bits
  – sense amplifiers drive bitlines full rail to recharge storage cells

• Column access (CAS)
  – decode column address to select small number of sense amplifier latches (4, 8, 16, or 32 bits depending on DRAM package)
  – on read, send latched bits out to chip pins
  – on write, change sense amplifier latches which then charge storage cells to required value
  – can perform multiple column accesses on same row without another row access (burst mode)

• Precharge
  – charges bit lines to known value, required before next row access

Each step has a latency of around 15-20ns in modern DRAMs
Various DRAM standards (DDR, RDRAM) have different ways of encoding the signals for transmission to the DRAM, but all share same core architecture
Double-Data Rate (DDR2) DRAM

**200MHz Clock**

- **Row**: BA0, BA1, A10, ADDRESS, COMMAND
- **Column**: Row, Column
- **Precharge**: PRE

**Data Rate**:
- 200MHz Clock
- 400Mb/s Data Rate

**[Micron, 256Mb DDR2 SDRAM datasheet]**
DRAM Packaging

- DIMM (Dual Inline Memory Module) contains multiple chips with clock/control/address signals connected in parallel (sometimes need buffers to drive signals to all chips)
- Data pins work together to return wide word (e.g., 64-bit data bus using 16x4-bit parts)
CPU-Memory Bottleneck

Performance of high-speed computers is usually limited by memory *bandwidth* & *latency*

- **Latency** (time for a single access)
  Memory access time >> Processor cycle time

- **Bandwidth** (number of accesses per unit time)
  if fraction $m$ of instructions access memory,
  ⇒ $1 + m$ memory references / instruction
  ⇒ CPI = 1 requires $1 + m$ memory refs / cycle
  (assuming MIPS RISC ISA)
Processor-DRAM Gap (latency)

Processor-Memory Performance Gap:
- 60% yearly for µProc
- 7% yearly for DRAM (grows 50% / year)

Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!
Physical Size Affects Latency

- Signals have further to travel
- Fan out to more locations
Memory Hierarchy

- **capacity**: Register << SRAM << DRAM  why?
- **latency**: Register << SRAM << DRAM  why?
- **bandwidth**: on-chip >> off-chip  why?

On a data access:
- if data ∈ fast memory ⇒ low latency access (*SRAM*)
- If data ∉ fast memory ⇒ long latency access (*DRAM*)
Relative Memory Cell Sizes

1. Memory cell in 0.5μm processes
   a) Gate Array SRAM
   b) Embedded SRAM
   c) Standard SRAM (6T cell with local interconnect)
   d) ASIC DRAM
   e) Standard DRAM (stacked cell)

<table>
<thead>
<tr>
<th>Memory</th>
<th>Process</th>
<th>Cell size (μm²)</th>
<th>Cell efficiency</th>
<th>Bits in mm² (10⁶)</th>
<th>Gate size (μm²)</th>
<th>Gate utilization (%)</th>
<th>Gates in 100mm² (10⁶)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gate array SRAM</td>
<td>3-metal</td>
<td>370</td>
<td>80%</td>
<td>216</td>
<td>185</td>
<td>70%</td>
<td>378</td>
</tr>
<tr>
<td>Embedded SRAM</td>
<td>3-metal</td>
<td>67</td>
<td>70%</td>
<td>1045</td>
<td>185</td>
<td>70%</td>
<td>378</td>
</tr>
<tr>
<td>Standard SRAM</td>
<td>2-metal 6T local int.</td>
<td>43</td>
<td>65%</td>
<td>1512</td>
<td>245</td>
<td>40%</td>
<td>163</td>
</tr>
<tr>
<td>Embedded ASIC-DRAM</td>
<td>3-metal</td>
<td>23</td>
<td>60%</td>
<td>2609</td>
<td>185</td>
<td>70%</td>
<td>378</td>
</tr>
<tr>
<td>Standard DRAM</td>
<td>2-metal stacked cell</td>
<td>3.2</td>
<td>50%</td>
<td>15625</td>
<td>411</td>
<td>40%</td>
<td>97</td>
</tr>
</tbody>
</table>

Table 1: Memory and logic density for a variety of 0.5μm implementations.

References:
[ Foss, “Implementing Application-Specific Memory”, ISSCC 1996 ]
CS152 Administrivia

- Class accounts available today
- Handed out in Section at 2pm
Management of Memory Hierarchy

• Small/fast storage, e.g., registers
  – Address usually specified in instruction
  – Generally implemented directly as a register file
    » but hardware might do things behind software’s back, e.g., stack management, register renaming

• Larger/slower storage, e.g., main memory
  – Address usually computed from values in register
  – Generally implemented as a hardware-managed cache hierarchy
    » hardware decides what is kept in fast memory
    » but software may provide “hints”, e.g., don’t cache or prefetch
Real Memory Reference Patterns

Typical Memory Reference Patterns

- Instruction fetches
- Stack accesses
- Data accesses

- n loop iterations
- subroutine call
- subroutine return
- argument access
- vector access
- scalar accesses
Common Predictable Patterns

Two predictable properties of memory references:

- **Temporal Locality**: If a location is referenced it is likely to be referenced again in the near future.

- **Spatial Locality**: If a location is referenced it is likely that locations near it will be referenced in the near future.
Memory Reference Patterns

Caches

Caches exploit both types of predictability:

– Exploit temporal locality by remembering the contents of recently accessed locations.

– Exploit spatial locality by fetching blocks of data around recently accessed locations.
Inside a Cache

- Processor
- CACHE
- Main Memory

Address
Data

copy of main memory location 100

Copy of main memory location 101

Address Tag

100
304
6848
416

Data Block

Data Byte

Line
Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either

- Found in cache, a.k.a. HIT
  - Return copy of data from cache

- Not in cache, a.k.a. MISS
  - Read block of data from Main Memory
  - Wait ...
  - Return data to processor and update cache

Q: Which line do we replace?
Placement Policy

Memory

Cache

Set Number

Block Number

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3</td>
</tr>
<tr>
<td>(12 mod 4)</td>
<td>(12 mod 8)</td>
</tr>
</tbody>
</table>

Block 12 can be placed anywhere in set 0 or only into block 4.
Direct-Mapped Cache

- Tag
- Index
- Block Offset
- V
- Tag
- Data Block

2^k lines

HIT

Data Word or Byte
Direct Map Address Selection

*higher-order vs. lower-order address bits*

Diagram showing the structure of a direct map with index, tag, and block offset fields. The diagram illustrates how address bits are used to select a tag, and the tag is then compared with the data block to determine if a hit occurs. The diagram also shows the relationship between the number of blocks and the number of lines.
2-Way Set-Associative Cache

Tag  Index  Block Offset

V Tag  Data Block

V Tag  Data Block

Data Word or Byte

HIT
Fully Associative Cache
Replacement Policy

In an associative cache, which block from a set should be evicted when the set becomes full?

• Random

• Least Recently Used (LRU)
  • LRU cache state must be updated on every access
  • true implementation only feasible for small sets (2-way)
  • pseudo-LRU binary tree often used for 4-8 way

• First In, First Out (FIFO) a.k.a. Round-Robin
  • used in highly associative caches

• Not Least Recently Used (NLRU)
  • FIFO with exception for most recently used block or blocks

This is a second-order effect. Why?

Replacement only happens on misses
Acknowledgements

• These slides contain material developed and copyright by:
  – Arvind (MIT)
  – Krste Asanovic (MIT/UCB)
  – Joel Emer (Intel/MIT)
  – James Hoe (CMU)
  – John Kubiatowicz (UCB)
  – David Patterson (UCB)

• MIT material derived from course 6.823
• UCB material derived from course CS252