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.
CPU-Memory Bottleneck

Performance of high-speed computers is usually limited by memory \textit{bandwidth} & \textit{latency}

- Latency (time for a single access)
  Memory access time \gg Processor cycle time

- Bandwidth (number of accesses per unit time)
  if fraction $m$ of instructions access memory,
  \Rightarrow 1+m memory references / instruction
  \Rightarrow CPI = 1 requires 1+m memory refs / cycle

Core Memory

- Core memory was first large scale reliable main memory
  - invented by Forrester in late 40s at MIT for Whirlwind project
- Bits stored as magnetization polarity on small ferrite cores threaded onto 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, DRAM

- Semiconductor memory began to be competitive in early 1970s
  - Intel formed to exploit market for semiconductor memory

- First commercial 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

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

Wordline
**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 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)
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 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

Data Rate: 400Mb/s

[ Micron, 256Mb DDR2 SDRAM datasheet ]
**Processor-DRAM Gap (latency)**

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

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

---

**Little’s Law**

\[ \text{Throughput} \ (T) = \frac{\text{Number in Flight} \ (N)}{\text{Latency} \ (L)} \]

- Example:
  "Assume infinite bandwidth memory"
  "100 cycles / memory reference"
  "1 + 0.2 memory references / instruction"

\[ \Rightarrow \text{Table size} = 1.2 \times 100 = 120 \text{ entries} \]

120 independent memory operations in flight!
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.
Multilevel Memory

Strategy: Reduce average latency using small, fast memories called caches.

Caches are a mechanism to reduce memory latency based on the empirical observation that the patterns of memory references made by a processor are often highly predictable:

```
PC
...
loop: ADD r2, r1, r1  100
SUBI r3, r3, #1       104
BNEZ r3, loop        108
...
```

Memory Hierarchy

- **capacity**: Register $<$ SRAM $<$ DRAM  
  - why?

- **latency**: Register $<$ SRAM $<$ DRAM  
  - why?

- **bandwidth**: on-chip $>$ off-chip  
  - why?

On a data access:

- **hit** (data $\in$ fast memory)  $\Rightarrow$ low latency access
- **miss** (data $\notin$ fast memory)  $\Rightarrow$ long latency access (DRAM)

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

- **Large/slower storage**, e.g., memory
  - Address usually computed from values in register
  - Generally implemented as a cache hierarchy
    - hardware decides what is kept in fast memory
    - but software may provide “hints”, e.g., don’t cache or prefetch
CS152 Administrivia

• Krste, no office hours, Monday 2/18 (President’s Day Holiday)
  – email for alternate time
• Henry office hours, 511 Soda
  – 9:30-10:30AM Mondays
  – 2:00-3:00PM Fridays
• In-class quiz dates
  – Q1: Tuesday February 19 (ISAs, microcode, simple pipelining)
    » Material covered, Lectures 1-5, PS1, Lab 1
• Emailing PS/Lab questions
  – Email Henry & CC Krste, send only one email
• Watch website for updates/news
• Not sure how long book will take to appear in store, can buy online

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

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

Block Number

Memory

Set Number

Cache

block 12 can be placed

Fully Associative anywhere

(2-way) Set Associative anywhere in set 0 (12 mod 4)

Direct Mapped only into block 4 (12 mod 8)

Direct-Mapped Cache

Tag Index Block Offset

t k b

V Tag Data Block

2^k lines

HIT

Data Word or Byte
Direct Map Address Selection

higher-order vs. lower-order address bits

2-Way Set-Associative Cache
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?*
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