Where We Are in This Course Right Now

- So far:
  - We know how to design a processor that can fetch, decode, and execute the instructions in an ISA
  - We have assumed that memory storage (for instructions and data) is a magic black box
- Now:
  - We learn why memory storage systems are hierarchical
  - We learn about caches and SRAM technology for caches
- Next:
  - We learn how to implement main memory

Readings

- Patterson and Hennessy
  - Chapter 7
  - Except 7.4 (for now)
Storage

- We have already seen some storage implementations
  - Individual registers
    - For singleton values: e.g., PC, PSR
    - For \mu\text{arch}/transient values: e.g., in multi-cycle design
  - Register File
    - For architectural values: e.g., ISA registers

- What else is there?

Storage Hierarchy

- [material in class]

(CMOS) Memory Components

- Interface
  - N-bit \textit{address} bus (on N-bit machine)
  - \textit{Data} bus
    - Typically read/write on same data bus
    - Can have multiple ports: address/data bus pairs
    - Can be \textit{synchronous}: read/write on clock edges
    - Can be \textit{asynchronous}: untimed “handshake”

- Performance
  - Access time proportional to (#ports) \times \sqrt{(#\text{bits})}
  - \sqrt{(#\text{bits})}? Proportional to max wire length
  - More about this a little later ...

- Only fundamental component is ... \textit{memory}
  - Registers not fundamental (e.g., memory-only and stack ISAs)
  - Only fundamental and desirable property of disks is non-volatility
  - But there is non-volatile memory technology (e.g., Flash)

- Registers vs. memory
  - Direct specification (fast) vs. address calculation (slow)
  - Few addresses (small & fast) vs. many (big & slow)
  - Not everything can be put into registers (e.g., arrays, structs)

- Memory vs. disk
  - Electrical (fast) vs. electro-mechanical (very slow)
  - Disk is so slow (relatively), it is considered I/O

- We will talk just about memory for \textit{instructions} and \textit{data}
Memory Performance Equation

- [material in class]

Memory Hierarchy

\[ t_{avg} = t_{hit} + \%miss \times t_{miss} \]

- Problem: hard to get low \( t_{hit} \) and \( \%miss \) in one structure
- Large structures have low \( \%miss \) but high \( t_{hit} \)
- Small structures have low \( t_{hit} \) but high \( \%miss \)
- Solution: use a hierarchy of memory structures
- A very old (by computer standards) idea:

"Ideally, one would desire an infinitely large memory capacity such that any particular word would be immediately available. ... We are forced to recognize the possibility of constructing a hierarchy of memories, each of which has a greater capacity than the preceding but which is less quickly accessible."

Burks, Goldstine, and Von Neumann

"Preliminary discussion of the logical design of an electronic computing instrument"
IAS memo 1946

Abstract Memory Hierarchy

- Hierarchy of memory components
  - Upper components (closer to CPU)
  - Fast ↔ Small ↔ Expensive
  - Lower components (further from CPU)
  - Slow ↔ Big ↔ Cheap

- Connected by buses
  - Which we will ignore for now

- Make average access time close to M1’s
  - How?
  - Most frequently accessed data in M1
  - M1 + next most frequently accessed in M2, etc.
  - Automatically move data up/down hierarchy

Why Hierarchy Works I

- [material in class]
Why Hierarchy Works II

- **Temporal locality**
  - Recently executed insns likely to be executed again soon
  - Inner loops (next iteration)
  - Recently referenced data likely to be referenced again soon
  - Data in inner loops, hot global data
  - Hierarchy can be "reactive": move things up when accessed

- **Spatial locality**
  - Insns near recently executed insns likely to be executed soon
  - Sequential execution
  - Data near recently referenced data likely to be referenced soon
  - Elements in an array, fields in a struct, variables in frame
  - Hierarchy can be "proactive": move things up speculatively

Abstract Hierarchy Performance

How do we compute $t_{avg}$?

\[
= t_{avg} - M_1 \\
= t_{hit} - M_1 + (\%miss - M_1 \times t_{miss} - M_1) \\
= t_{hit} - M_1 + (\%miss - M_1 \times (t_{hit} - M_2 + (\%miss - M_2 \times t_{miss} - M_2))) \\
= t_{hit} - M_1 + (\%miss - M_1 \times (t_{hit} - M_2 + (\%miss - M_2 \times (t_{hit} - M_3 + (\%miss - M_3 \times t_{avg} - M_4))))) \\
\]

Concrete Memory Hierarchy

- 1st level: L1 I$, L1 D$(L1 instr/data caches)
- 2nd level: L2 cache
  - Often on-chip, certainly on-package (with CPU)
  - Made of SRAM (same circuit type as CPU)
  - Managed in hardware
  - This unit of ECE 152
- 3rd level: main memory
  - Made of DRAM
  - Managed in software
  - Next unit of ECE 152
- 4th level: disk (swap space)
  - Made of magnetic iron oxide discs
  - Managed in software
  - Course unit after main memory

Note: some processors have off-chip L2S between L2S and memory

Chips today are 80+% cache by area → important!