Shuttle can’t fly over Jan 1? ⇒

A computer bug has come up for the shuttle – its computers don’t reset to day 1 of the new year, because to them it’s just day 366. Joking? We wish.
Review: Pipelining

• Pipeline challenge is hazards
  • Forwarding helps with many data hazards
  • Delayed branch helps with control hazard in our 5 stage pipeline
  • Data hazards w/Loads ⇒ Load Delay Slot
    ▪ Interlock ⇒ “smart” CPU has HW to detect if conflict with inst following load, if so it stalls

• More aggressive performance:
  • Superscalar (parallelism)
  • Out-of-order execution
The Big Picture

Computer

Processor (active)

Control (“brain”)

Datapath (“brawn”)

Memory (passive) (where programs, data live when running)

Devices

Input

Output

Keyboard, Mouse

Disk, Network

Display, Printer

Keyboard, Mouse

Disk, Network

Display, Printer
Memory Hierarchy

Storage in computer systems:

• Processor
  • holds data in register file (~100 Bytes)
  • Registers accessed on nanosecond timescale

• Memory (we’ll call “main memory”)
  • More capacity than registers (~Gbytes)
  • Access time ~50-100 ns

• Disk
  • HUGE capacity (virtually limitless)
  • VERY slow: runs ~milliseconds
Motivation: Why We Use Caches

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

- 1989 first Intel CPU with cache on chip
- 1998 Pentium III has two levels of cache on chip
Memory Caching

• Mismatch between processor and memory speeds leads us to add a new level: a memory cache

• Implemented with same IC processing technology as the CPU (usually integrated on same chip): faster but more expensive than DRAM memory.

• Cache is a copy of a subset of main memory.

• Most processors have separate caches for instructions and data.
Memory Hierarchy

As we move to deeper levels the latency goes up and price per bit goes down.
Memory Hierarchy

- If level closer to Processor, it is:
  - smaller
  - faster
  - subset of lower levels (contains most recently used data)

- Lowest Level (usually disk) contains all available data (or does it go beyond the disk?)

- Memory Hierarchy presents the processor with the illusion of a very large very fast memory.
Memory Hierarchy Analogy: Library (1/2)

• You’re writing a term paper (Processor) at a **table** in **Doe**

• **Doe** Library is equivalent to **disk**
  • essentially limitless capacity
  • very slow to retrieve a book

• **Table** is **main memory**
  • smaller capacity: means you must return book when table fills up
  • easier and faster to find a book there once you’ve already retrieved it
Memory Hierarchy Analogy: Library (2/2)

• Open books on table are cache
  • smaller capacity: can have very few open books fit on table; again, when table fills up, you must close a book
  • much, much faster to retrieve data

• Illusion created: whole library open on the tabletop
  • Keep as many recently used books open on table as possible since likely to use again
  • Also keep as many books on table as possible, since faster than going to library
Memory Hierarchy Basis

- Cache contains copies of data in memory that are being used.
- Memory contains copies of data on disk that are being used.
- Caches work on the principles of **temporal and spatial locality**.
  - Temporal Locality: if we use it now, chances are we’ll want to use it again soon.
  - Spatial Locality: if we use a piece of memory, chances are we’ll use the neighboring pieces soon.
Cache Design

• How do we organize cache?

• Where does each memory address map to?
  (Remember that cache is subset of memory, so multiple memory addresses map to the same cache location.)

• How do we know which elements are in cache?

• How do we quickly locate them?
Direct-Mapped Cache (1/2)

• In a **direct-mapped cache**, each memory address is associated with one possible **block** within the cache
  • Therefore, we only need to look in a single location in the cache for the data if it exists in the cache
  • Block is the unit of transfer between cache and memory
Direct-Mapped Cache (2/2)

- Cache Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ...
  - 4 blocks ⇒ any memory location that is multiple of 4
Issues with Direct-Mapped

- Since multiple memory addresses map to same cache index, how do we tell which one is in there?

- What if we have a block size > 1 byte?

- Answer: divide memory address into three fields

  | ttttttttttttttttt | iiiiiiiiiiiii | 0000 |
  |
  tag | index | byte
  to check | to | offset
  if have | select | within
  correct block | block | block
Direct-Mapped Cache Terminology

• All fields are read as unsigned integers.

• **Index**: specifies the cache index (which “row”/block of the cache we should look in)

• **Offset**: once we’ve found correct block, specifies which byte within the block we want

• **Tag**: the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to the same location
**TIO** Dan’s great cache mnemonic

\[
\text{AREA (cache size, B)} = \text{HEIGHT (# of blocks)} \times \text{WIDTH (size of one block, B/block)}
\]

\[
2^{(H+W)} = 2^H \times 2^W
\]

![Diagram of cache memory with tag, index, and offset labels, along with dimensions for height and width.](image)
Direct-Mapped Cache Example (1/3)

• Suppose we have a 16KB of data in a direct-mapped cache with 4 word blocks

• Determine the size of the tag, index and offset fields if we’re using a 32-bit architecture

• Offset
  • need to specify correct byte within a block
  • block contains 4 words
    = 16 bytes
    = 2^4 bytes
  • need **4 bits** to specify correct byte
Direct-Mapped Cache Example (2/3)

- Index: (~index into an “array of blocks”)
  - need to specify correct block in cache
  - cache contains 16 KB = $2^{14}$ bytes
  - block contains $2^4$ bytes (4 words)
  - # blocks/cache
    
    \[
    \text{bytes/cache} = \frac{\text{bytes/block}}{2^4} = 2^{14} \text{ bytes/cache} \quad \text{blocks/block} = 2^{10} \text{ blocks/cache}
    \]
  - need **10 bits** to specify this many blocks
Direct-Mapped Cache Example (3/3)

- Tag: use remaining bits as tag
  - tag length = addr length – offset - index
    = 32 - 4 - 10 bits
    = 18 bits
  - so tag is leftmost 18 bits of memory address

- Why not full 32 bit address as tag?
  - All bytes within block need same address (4b)
  - Index must be same for every address within a block, so it’s redundant in tag check, thus can leave off to save memory (here 10 bits)
Caching Terminology

• When we try to read memory, 3 things can happen:

1. **cache hit**: cache block is valid and contains proper address, so read desired word

2. **cache miss**: nothing in cache in appropriate block, so fetch from memory

3. **cache miss, block replacement**: wrong data is in cache at appropriate block, so discard it and fetch desired data from memory (cache always copy)
A. Mem hierarchies were invented before 1950. (UNIVAC I wasn’t delivered ‘til 1951)

B. If you know your computer’s cache size, you can often make your code run faster.

C. Memory hierarchies take advantage of spatial locality by keeping the most recent data items closer to the processor.
Assume 1 instr/clock, delayed branch, 5 stage pipeline, forwarding, interlock on unresolved load hazards (after $10^3$ loops, so pipeline full). Rewrite this code to reduce pipeline stages (clock cycles) per loop to as few as possible.

Loop:

```assembly
lw    $t0, 0($s1)
addu  $t0, $t0, $s2
sw    $t0, 0($s1)
addiu $s1, $s1, -4
bne   $s1, $zero, Loop
nop
```

• How many pipeline stages (clock cycles) per loop iteration to execute this code?
And in Conclusion…

• We would like to have the capacity of disk at the speed of the processor: unfortunately this is not feasible.

• So we create a memory hierarchy:
  • each successively lower level contains “most used” data from next higher level
  • exploits temporal & spatial locality
  • do the common case fast, worry less about the exceptions (design principle of MIPS)

• Locality of reference is a Big Idea