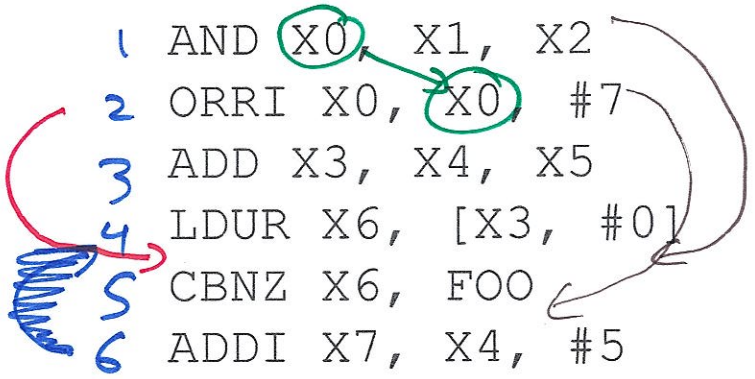
Review Problem 35

- What should we do to this code to run it on a CPU with delay slots?



ADD
LDUR
AND
CBNZ
ORRI
ADDI

The Problem

Cost vs. Performance

- Fast memory is expensive
- Slow memory can significantly affect performance

Design Philosophy

- Use a hybrid approach that uses aspects of both
- Keep frequently used things in a small amount of fast/expensive memory
  - "Cache"
- Place everything else in slower/inexpensive memory (even disk)
- Make the common case fast

Locality

Programs access a relatively small portion of the address space at a time

```
char *index = string;
while (*index != 0) { /* C strings end in 0 */
    if (*index >= 'a' && *index <= 'z')
        *index = *index +('A' - 'a');
    index++;
}
```

Types of Locality

- Temporal Locality – If an item has been accessed recently, it will tend to be accessed again soon index variable, instructions
- Spatial Locality – If an item has been accessed recently, nearby items will tend to be accessed soon string array, instructions

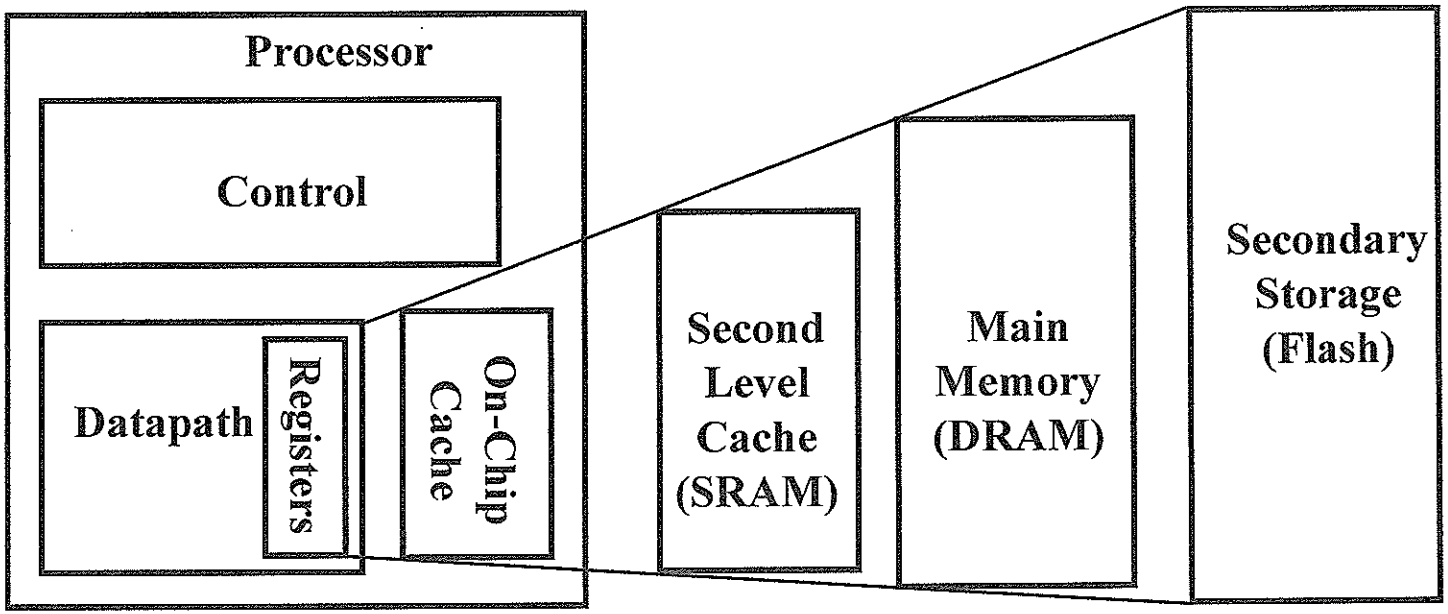
Locality guides caching

The Solution

By taking advantage of the principle of locality:

Provide as much memory as is available in the cheapest technology.

Provide access at the speed offered by the fastest technology.



Name	Register	Cache	Main Memory	Disk Memory
Speed	1 cycle	1-7 cycles	100 cycles	10,000 cycles
Capacity	1x (norm.)	64-4Kx	4Mx	1Gx

Cache Terminology

Block – Minimum unit of information transfer between levels of the hierarchy

Block addressing varies by technology at each level

Blocks are moved one level at a time

Upper vs. **lower** level – "upper" is closer to CPU, "lower" is futher away

Hit – Data appears in a block in that level

Hit rate – percent of accesses hitting in that level

Hit time – Time to access this level

Hit time = Access time + Time to determine hit/miss

Miss – Data does not appear in that level and must be fetched from lower level

Miss rate – percent of misses at that level = (1 – hit rate)

Miss penalty – Overhead in getting data from a lower level

Miss penalty = Lower level access time + Replacement time + Time to deliver to processor

Miss penalty is usually MUCH larger than the hit time

Cache Access Time

Average access time

Access time = (hit time) + (miss penalty)x(miss rate)

Want high hit rate & low hit time, since miss penalty is large

Average Memory Access Time (AMAT)

Apply average access time to entire hierarchy.

Cache Access Time Example

Level	Hit Time	Hit Rate	Access Time
L1	1 cycle	95%	$1 + .05 \times 65 = 4.25$
L2	10 cycles	90%	$10 + .1 \times 550 = 65$
Main Memory	50 cycles	99%	$50 + .01 \times 50{,}000 = 550$
Disk	50,000 cycles	100%	$50{,}000$

Note: Numbers are **local** hit rates – the ratio of access that go to that cache that hit (remember, higher levels filter accesses to lower levels)

$$AMAT = 1 + .05(10 + .1(50 + .01(50{,}000)))$$