Caches
Multiprocessor systems

Chapter 7 P&H
Chapter 9 P&H (CD-ROM)

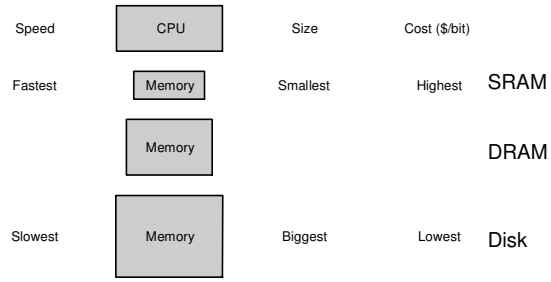
Introduction

- Desire is to have unlimited amounts of fast memory
- Can exploit *locality of reference* to appear to have lots of fast memory
- Two types of locality
  - Temporal Locality
  - Spatial Locality

Mem Tech	access time	$/MB (1997)
SRAM	5-25ns	$100-$250
DRAM	60-120ns	$5-$10
Disk	10-20million ns	$0.10-$0.20

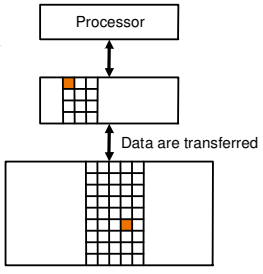
Mem Tech	access time	$/GB (2004)
SRAM	0.5-5ns	$100-$250
DRAM	50-70ns	$100-$200
Disk	5-20million ns	$0.50-$2

Memory Hierarchy



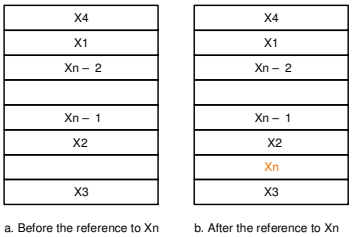
Memory Hierarchy

- data only copied between two levels at a time
- minimum unit is called a block
- *hit rate* is fraction of memory accesses found in upper level
- *miss rate* = (1 – hit rate)
- *hit time* Time to access upper level
- *miss penalty* time to replace block in upper level with corresponding block from lower level



Cache Basics

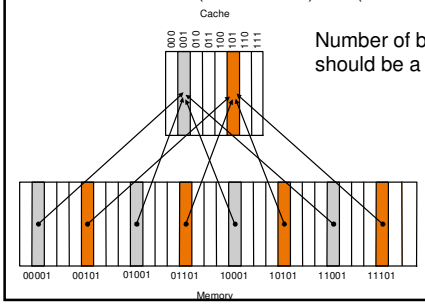
- Caches first appear in early 1960's
- following cache has single word sized block
- How do we know if the requested item is in the cache?
- If it is where is it?



a. Before the reference to Xn b. After the reference to Xn

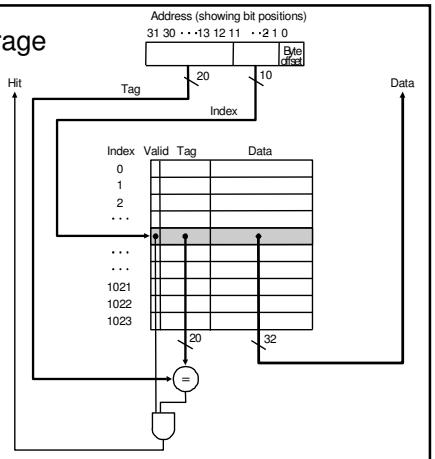
Direct Mapped Cache

- cache address = (block address) mod (No. blocks in cache)



Number of blocks in cache should be a power of two

Tagged Storage



Example #1

- How many bits in total are required for a direct mapped cache with 64 KB of Data and one word blocks, assuming a 32 bit address?

Example #1

- 64KB = 16K words
- 16K = 16384 = 2^{14} words
- One word per block = 2^{14} blocks
- Tag size: 32 – 14 – 2 = 16 bits
  - (word alignment means last 2 bits are implicit)
- 2^{14} x (32 + 16 + 1) = 784 Kbits
- 784 Kbits = 98KB for 64KB cache
  - 1.53

Example #2

- How many bits in total are required for a direct mapped cache with 64 KB of Data and four word blocks, assuming a 32 bit address?

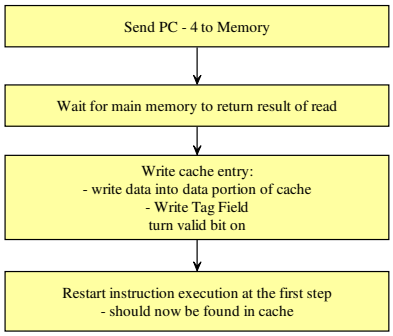
Example #2

- 64KB = 16K words
- 16K = 16384 = 2^{14} words
- Four words per block = 2^{12} blocks
- Tag size: 32 – 12 – 2 = 18 bits
  - (word alignment means last 2 bits are implicit)
- 2^{12} x (128 + 18 + 1) = 588 Kbits
- 588 Kbits = 73.5KB for 64KB cache
  - 1.14

Handling Misses

- No Modifications required to our pipelined data-path to handle hits
- On a miss a basic approach is to stall the *entire* data-path until the value is fetched
- Separate controller used for the caches

Example – Miss on Instruction Cache



Multiprocessors

- Make faster computers by adding multiple CPUs per computer
- Hard to build software to use added resources as multiple places it can go wrong
  - Poor operating system support
  - Poor application design

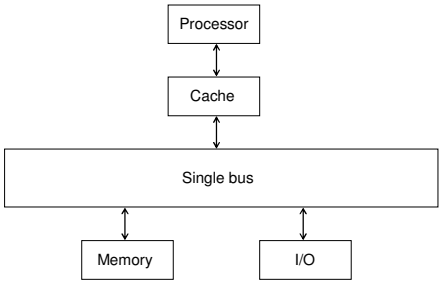
Multiprocessors

- 1990s:
  - motherboards came with a separate socket for each CPU
- 2004 onwards:
  - CPUs come with multiple cores built in
  - motherboards may come with additional CPU sockets

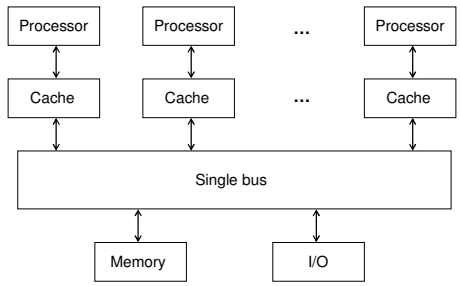
Multiprocessors

- To benefit, applications must be written to be multi-threaded
  - Have multiple points of simultaneous execution
  - Each using the same process address space

Single Processor system



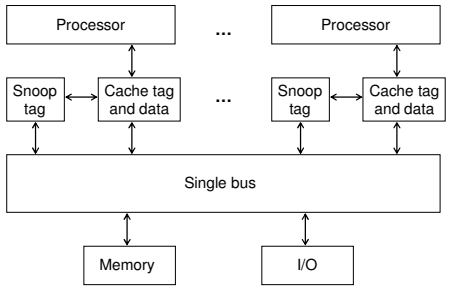
Multiprocessor system



Multiprocessor systems

- Caches reduce contention for the system bus by replicating data
  - Multiple CPUs using a read-only copies of data not a problem
- Challenge: keep data in cache coherent amongst processors when data is written to
  - Could force all reads to go to memory, but
    - Require method to indicate data is shared
    - Going to memory is slow

Cache coherence

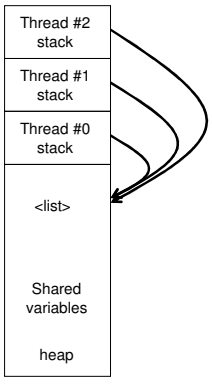


Cache coherence

- Snooping:
  - Processors connected using single bus
  - Snoop other CPUs accesses to main memory
  - Check address being read or written
- When snooping on a write, if in cache, can:
  - Invalidate cache block. When value in memory location next needed value will be read from memory
  - Update cache block with new value

Synchronisation

- Write-invalidate does not prevent multiple processors working on some item of data
  - each believing they have the up-to-date version of it
- To provide synchronisation, processors have to provide some atomic operation
- Applications build locking structures with this CPU support



Multiple threads deal with shared variables stored in the heap

e.g. some list

Operating system runs multiple threads simultaneously.

Threads will be put to sleep if CPU is busy.

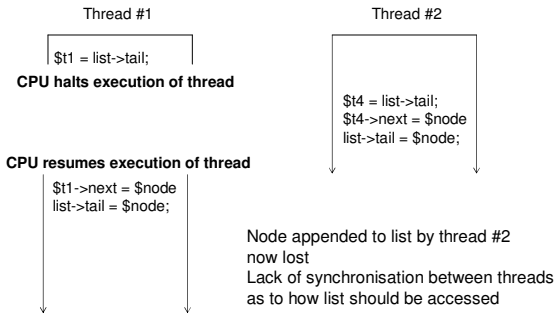
Unable to predict how CPU will schedule your threads to execute.

Example

```
struct list_node {
        void *item;
        struct list_node *next;
};
struct list {
        struct list_node *head;
        struct list_node *tail;
};

Append list_node items to tail to record items in the list:
        list->tail->next = node;
        list->tail = node;
```

Example



Java

```
/* wait (block) until list can be locked */
synchronized(list) {
    /* use list variable */
}
```

C / pthreads

pthread_mutex_init
pthread_mutex_lock
pthread_mutex_unlock
pthread_mutex_trylock

MIPS atomic operation

- MIPS uses Load Linked / Store Conditional
- Load Linked (LL)
  - ll $rt, offset($rs)
  - Special type of load word.
  - Processor remembers the memory address loaded.
  - Can only remember one address per processor
  - Processor monitors writes over shared bus to that address
- Store conditional (SC)

MIPS atomic operation

- Load linked
  - ll $rt, offset($rs)
- Store Conditional
  - sc $rt offset($rs)
  - Special type of store word.
  - Write occurs only if the value in memory has not been updated since ll was executed
  - If the value has been updated, $rt is set to zero

MIPS atomic operation

```
L1:
   LL      T1, (T0)     # load counter
   ADDI    T2, T1, 1    # increment
   SC      T2, (T0)     # store counter conditionally
   BEQ     T2, 0, L1    # if unsuccessful (0), try again
   NOP                  # branch delay
```

MIPS IV instruction set, rev 3.2, page 136.

LL/SC

- MIPS (II and above)
- ARM (v6 and above)
- Alpha
- PowerPC

- Intel: compare and swap
  - Implemented atomically, fairly similar to LL/SC