Improving Memory Access 1/3
The Cache and Virtual Memory
Principle of Locality

- **Principle of Locality** states that programs access a relatively small portion of their address space at any instance of time.

- **Two types of locality**
  - **Temporal locality** (locality in time)
    - If an item is referenced, then the same item will tend to be referenced soon.
      - “the tendency to reuse recently accessed data items”
  
  - **Spatial locality** (locality in space)
    - If an item is referenced, then nearby items will be referenced soon.
      - “the tendency to reference nearby data items”
Cache Terminology

A **hit** if the data requested by the CPU is in the upper level

**Hit rate** or **Hit ratio**

is the fraction of accesses found in the upper level

**Hit time**

is the time required to access data in the upper level

= <detection time for hit or miss> + <hit access time>

A **miss** if the data is not found in the upper level

**Miss rate** or **(1 – hit rate)**

is the fraction of accesses **not** found in the upper level

**Miss penalty**

is the time required to access data in the lower level

= <lower access time> + <reload processor time>
Direct Mapped Cache

- **Direct Mapped:** assign the cache location based on the address of the word in memory

  \[
  \text{cache_address} = \text{memory_address} \mod \text{cache_size};
  \]

Observe there is a Many-to-1 memory to cache relationship
Direct Mapped Cache: MIPS Architecture

![Diagram of Direct Mapped Cache]

Figure 7.7
Bits in a Direct Mapped Cache

How many total bits are required for a direct mapped cache with 64KB (= $2^{16}$ KiloBytes) of data and one word (=32 bit) blocks assuming a 32 bit byte memory address?

Cache index width = $\log_2$ words

= $\log_2 \frac{2^{16}}{4} = \log_2 2^{14}$ words = 14 bits

Block address width = <byte address width> – $\log_2$ word

= 32 – 2 = 30 bits

Tag size = <block address width> – <cache index width>

= 30 – 14 = 16 bits

Cache block size = <valid size> + <tag size> + <block data size>

= 1 bit + 16 bits + 32 bits = 49 bits

Total size = <Cache word size> $\times$ <Cache block size>

= $2^{14}$ words $\times$ 49 bits = 784 $\times$ $2^{10}$ = 784 Kbits = 98 KB

= 98 KB/64 KB = 1.5 times overhead
The DECStation 3100 cache

**write-through cache**
Always write the data into both the cache and memory and then wait for memory.

**DECStation uses a write-through cache**
- 128 KB total cache size (=32K words)
  - = 64 KB instruction cache (=16K words)
  - + 64 KB data cache (=16K words)
- 10 processor clock cycles to write to memory

In a gcc benchmark, 13% of the instructions are stores.

- Thus, CPI of 1.2 becomes 1.2+13%\times 10 = 2.5
- Reduces the performance by more than a factor of 2!
Cache schemes

**write-through cache**
Always write the data into both the cache and memory and then wait for memory.

**write buffer**
write data into cache and write buffer. If write buffer full processor must stall.

No amount of buffering can help if writes are being generated faster than the memory system can accept them.

**write-back cache**
Write data into the cache block and only write to memory when block is modified but complex to implement in hardware.
Hits vs. Misses

• **Read hits**
  – this is what we want!

• **Read misses**
  – stall the CPU, fetch block from memory, deliver to cache, and restart.

• **Write hits**
  – write-through: can replace data in cache and memory.
  – write-buffer: write data into cache and buffer.
  – write-back: write the data only into the cache.

• **Write misses**
  – read the entire block into the cache, then write the word.
The DECStation 3100 miss rates

- A split instruction and data cache increases the bandwidth

<table>
<thead>
<tr>
<th>Benchmark Program</th>
<th>gcc</th>
<th>spice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction miss rate</td>
<td>6.1%</td>
<td>1.2%</td>
</tr>
<tr>
<td>Data miss rate</td>
<td>2.1%</td>
<td>1.3%</td>
</tr>
<tr>
<td>Effective split miss rate</td>
<td>5.4%</td>
<td>1.2%</td>
</tr>
<tr>
<td>Combined miss rate</td>
<td>4.8%</td>
<td></td>
</tr>
</tbody>
</table>

Why a lower miss rate? Numerical programs tend to consist of a lot of small program loops.

split cache has slightly worse miss rate
Spatial Locality

- **Temporal only cache**
  cache block contains only one word (**No spatial locality**).

- **Spatial locality**
  Cache block contains multiple words.
  - When a miss occurs, then fetch multiple words.
  - **Advantage**
    Hit ratio increases because there is a **high probability** that the adjacent words will be needed shortly.
  - **Disadvantage**
    Miss penalty increases with block size
Spatial Locality: 64 KB cache, 4 words

- 64KB cache using four-word (16-byte word)
- 16 bit tag, 12 bit index, 2 bit block offset, 2 bit byte offset.

Figure 7.10
Use split caches because there is more spatial locality in code:

<table>
<thead>
<tr>
<th>Program Block size</th>
<th>gcc =1</th>
<th>gcc =4</th>
<th>spice =1</th>
<th>spice =4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction miss rate</td>
<td>6.1%</td>
<td>2.0%</td>
<td>1.2%</td>
<td>0.3%</td>
</tr>
<tr>
<td>Data miss rate</td>
<td>2.1%</td>
<td>1.7%</td>
<td>1.3%</td>
<td>0.6%</td>
</tr>
<tr>
<td>Effective split miss rate</td>
<td>5.4%</td>
<td>1.9%</td>
<td>1.2%</td>
<td>0.4%</td>
</tr>
<tr>
<td>Combined miss rate</td>
<td>4.8%</td>
<td>4.8%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Temporal only split cache: has slightly worse miss rate

Spatial split cache: has lower miss rate
Increasing the block size tends to decrease miss rate.
Designing the Memory System

- Make reading multiple words easier by using banks of memory
1-word-wide memory organization

Suppose we have a system as follows

- 1-word-wide memory organization
- 1 cycle to send the address
- 15 cycles to access DRAM
- 1 cycle to send a word of data

If we have a cache block of 4 words

Then the miss penalty is

\[= (1 \text{ address send}) + 4 \times (15 \text{ DRAM reads}) + 4 \times (1 \text{ data send})\]

= 65 clocks per block read

Thus the number of bytes transferred per clock cycle

\[= 4 \text{ bytes/word} \times 4 \text{ words}/65 \text{ clocks} = 0.25 \text{ bytes/clock}\]
Interleaved memory organization

Suppose we have a system as follows

- 4-bank memory **interleaving** organization
- 1 cycle to send the address
- 15 cycles to access DRAM
- 1 cycle to send a word of data

If we have a cache block of 4 words

Then the miss penalty is
\[= (1 \text{ address send}) + 1 \times (15 \text{ DRAM reads}) + 4 \times (1 \text{ data send})\]
\[= 20 \text{ clocks per block read}\]

Thus the number of bytes transferred per clock cycle
\[= 4 \text{ bytes/word} \times 4 \text{ words/17 clocks} = 0.80 \text{ bytes/clock}\]

we improved from 0.25 to 0.80 bytes/clock!
Wide bus: 4-word-wide memory organization

Suppose we have a system as follows

- 4-word-wide memory organization
- 1 cycle to send the address
- 15 cycles to access DRAM
- 1 cycle to send a word of data

If we have a cache block of 4 words

Then the miss penalty is

\[ = (1 \text{ address send}) + 1 \times (15 \text{ DRAM reads}) + 1 \times (1 \text{ data send}) \]

\[ = 17 \text{ clocks per block read} \]

Thus the number of bytes transferred per clock cycle

\[ = 4 \text{ bytes/word} \times 4 \text{ words/17 clocks} = 0.94 \text{ bytes/clock} \]

we improved from 0.25 to 0.80 to 0.94 bytes/clock!
Memory organizations

**One word wide memory organization**

**Advantage**
Easy to implement, low hardware overhead

**Disadvantage**
Slow: 0.25 bytes/clock transfer rate

**Interleave memory organization**

**Advantage**
Better: 0.80 bytes/clock transfer rate
Banks are valuable on writes: independently

**Disadvantage**
more complex bus hardware

**Wide memory organization**

**Advantage**
Fastest: 0.94 bytes/clock transfer rate

**Disadvantage**
Wider bus and increase in cache access time