Sample Cache Access Diagram

- 64-bit address, 2-way set associative, cache size 128KB (only for the data part), block size 16B

- \(128\text{KB}/16B = 2^{13}\) records, \(2^{12}\) sets (two records per set)

Data address

- \(63\) bits
- \(39\) bits
- \(48\) bits

Index

- \(12\) bits

Tag

- \(48\) bits

Block offset

- \(4\) bits

One set

- \(4\) bits

4 bytes

64-offset-index = 48 bits

To CPU

Choose a byte where to start the access

Block size = 2

4

bytes

Index is 12 bits

Tag is 64

Offset is 4 bits

Choose a byte where to start the access

63

0

Sample Cache Access Diagram

- Choose a byte where to start the access

- \(63\) bits
- \(39\) bits
- \(48\) bits

Index

- \(12\) bits

Tag

- \(48\) bits

Block offset

- \(4\) bits

One set

- \(4\) bits

4 bytes

64-offset-index = 48 bits

To CPU

Choose a byte where to start the access

Block size = 2

4

bytes

Index is 12 bits

Tag is 64

Offset is 4 bits

Choose a byte where to start the access

63

0

### Improving Cache Performance

- There are several possible directions for improvement:
  - Reduce miss penalty
  - Reduce miss rate
  - Increase parallelism
  - Reduce hit time

1. Reducing Miss Penalty: Multilevel Caches

- There is a trade-off between the size of the cache and the access time
- Small caches are faster but cannot hold much data, therefore having larger miss rate
- Large caches can hold enough data but are slower, thus increasing hit time
- We can add another, smaller cache (L1) between the current cache (L2) and the CPU
- Since everything that is in L1 is also in L2, it only makes sense to add L2 if it is much bigger than L1

#### Example

Assume that in 1000 memory references there are 40 misses in L1 cache and 20 misses in L2 cache. Assume the miss penalty for L2 is 100 clock cycles, hit time of L2 is 10 clock cycles, hit time of L1 is 1 clock cycle and there are 1.5 memory references per instruction. What are various miss rates, the average memory access time and the average memory stalls per instruction?

#### Example

Given the data below, what is the impact of second-level cache associativity on its miss penalty:

- Hit time for direct mapped L2 is 10 clock cycles
- Two-way associativity increases hit time by 0.1 clock cycles
- Local miss rate in L2 for direct mapped is 25%
- Local miss rate in L2 for two-way set associative is 20%
- Miss penalty for L2 is 100 clock cycles

### Multilevel Caches

#### Average Memory Access Time = Hit Time + Miss_Rate \* Miss Penalty

**Miss Penalty** = Hit Time + Miss_Rate \* Miss Penalty

- We now have **local** and **global** miss rate
- Local miss rate is number of misses in the given cache level divided by number of accesses to this level and it is equal to Miss_Rate_L1 for L1 and Miss_Rate_L2 for L2
- Global miss rate is the number of misses in the given cache level divided by total number of accesses generated by the CPU. It is equal to Miss_Rate_L1 for L1, but it is Miss_Rate_L1 * Miss_Rate_L2 for L2

**Average Memory Stalls per Instruction = Misses per Instruction \* Hit Time + Misses per Instruction \* Miss Penalty**
2. Reducing Miss Penalty:
Critical Word First and Early Restart

- CPU is usually stalled waiting for one word, not the whole block of data
- **Critical word first**: Request from the memory and send this one word first to CPU thus reducing miss penalty, then continue loading the block
- **Early restart**: Request block from the beginning from the memory, but when the desired word arrives send it immediately to CPU
- These techniques do not help considerably as there is high chance that subsequent CPU accesses will be to other words in the block

Example
Assume that a computer has 64B block and L2 cache takes 11 clock cycles to get critical 8 bytes and then 2 clock cycles per 8 bytes to fetch the rest of the block. Calculate the average miss penalty with and without critical word first technique. What if the following instructions wait for the block to load then read the words following the first one, 8 bytes each.

3. Reducing Miss Penalty:
Giving Priority to Read Miss over Write

- If we have a write-through cache we can avoid stalls on write by writing data into a write buffer and then releasing CPU
- What happens if we do this but the subsequent read miss needs the data from the write buffer than has not been written yet?

```assembly
SW R3, 512(R0)  /* cache index 0 */
LW R1, 1024(R0) /* cache index 0 */
LW R2, 512(R0)  /* cache index 0 */
```
- The solution is to check the write buffer before going to memory
- If we have write-back cache, we can store the dirty block into write buffer, read data from memory into cache and release CPU, then perform the write from write buffer to memory

4. Reducing Miss Penalty:
Merging Write Buffer

- Writes usually modify one word in a block
- If a write buffer already contains some words from the given data block we will merge current modified word with the block parts already in the buffer

5. Reducing Miss Penalty:
Victim Caches

- Victim cache holds data that has been deleted from the cache, in case it is needed again
- This is fully associative cache and can help to reduce misses with direct-mapped or set associative caches

Reducing Miss Rate

- Miss categories:
  - Compulsory – first time when we want to access a block
  - Capacity – if cache cannot hold all blocks needed in a program
  - Conflict – if we use direct-mapped or set-associative strategy two blocks may map to the same record in cache
1. Reducing Miss Rate: Larger Block Size
   - Reduce number of compulsory misses
   - Larger blocks take advantage of spatial locality
   - But increase miss penalty
     - Larger block size means that fewer blocks will be in cache – this increases capacity misses and conflict misses

2. Reducing Miss Rate: Larger Caches
   - Reduce number of capacity misses
   - But increase hit time and have higher cost

3. Reducing Miss Rate: Higher Associativity
   - Experiments show that:
     - 8-way set associative cache has almost the same miss rate as fully associative cache
     - Direct mapped cache of size N has about the same miss rate as 2-way set associative cache of size N/2 (2:1 cache rule of thumb)
     - Greater associativity can come at the cost of increased hit time

Example
Assume higher associativity would increase clock cycle time over direct mapped cache as follows: 2-way 1.36 times, 4-way 1.44 times, 8-way 1.52 times. Hit time is 1 clock cycle. Miss penalty for direct mapped cache is 25 clock cycles. Using the miss rates from figure 5.14, find the average memory access times for different cache sizes.
   for solution see page 430

Example
Memory system takes 80 clock cycles of overhead and then delivers 16 bytes every 2 clock cycles. Miss rates for various block sizes are as follows.

<table>
<thead>
<tr>
<th>Block size</th>
<th>Cache size 4K</th>
<th>Cache size 16K</th>
<th>Cache size 64K</th>
<th>Cache size 256K</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0.51%</td>
<td>2.04%</td>
<td>0.19%</td>
<td>0.00%</td>
</tr>
<tr>
<td>32</td>
<td>7.44%</td>
<td>2.87%</td>
<td>1.32%</td>
<td>0.73%</td>
</tr>
<tr>
<td>64</td>
<td>7.93%</td>
<td>2.93%</td>
<td>1.08%</td>
<td>0.51%</td>
</tr>
<tr>
<td>128</td>
<td>7.79%</td>
<td>2.77%</td>
<td>1.03%</td>
<td>0.48%</td>
</tr>
<tr>
<td>256</td>
<td>9.31%</td>
<td>3.00%</td>
<td>1.15%</td>
<td>0.45%</td>
</tr>
</tbody>
</table>

Which block size gives us the smallest average memory access time?
   for solution see page 429

4. Reducing Miss Rate: Way Prediction and Pseudoassociative Caches
   - Way prediction: High associativity increases hit time
     - Keep predictors with each set to predict which block in the set will be needed on next access, then compare the tag of this block with the tag we are looking for
     - On correct prediction hit time is greatly reduced
     - Simplest prediction is to remember which block was requested last time and change the prediction if we are wrong
   - Pseudoassociativity
     - For two-way set associative cache, always check the first block
     - If we miss, check the second block before going to memory
     - Upon miss, swap blocks

for solution see page 430
5. Reducing Miss Rate:
Compiler Optimizations

- Make accesses to same block rather than across blocks

- Loop interchange:
  for(i=0; i<5000; i++)
  for(j=0; j<100; j++)
  x[i][j]=2*x[i][j]

- Blocking
  for(i=0; i<N; i++)
  for(j=0; j<N; j++)
  {     r=0;
    for(k=0; k<N; k++)
      r=r+y[i][k]*z[k][j];
    x[i][j]=x[i][j]+r; }

\[\Rightarrow\]

for(i=0; i<5000; i++)
for(j=0; j<100; j++)
\[x[i][j]=2*x[i][j]\]

\[\Rightarrow\]

for(i=0; i<N; i++)
for(j=0; j<N; j++)
{     r=0;
  for(k=0; k<N; k++)
    r=r+y[i][k]*z[k][j];
  x[i][j]=x[i][j]+r; }

\[x[i][j]=x[i][j]+r; \]