Announcements

- **Reading Assignment**
  Chapter 7: The Basics of Caches
  - 7.1, 7.2

- **Homework 6:**
  - 6.31, 6.32
  - 7.2, 7.3, 7.4, 7.5, 7.9, 7.10, 7.12, 7.23

- **Quiz**
  **When:** Mon., May 23rd, First 10 minutes of the class
  **Topic:** Cache Chapter 7  **Need:** Paper, pen
Advanced Techniques

- Superpipelining
  - More pipeline stages
  - Operand forwarding becomes complicated
  - Branch penalty is high
    - Must use branch prediction scheme
    - Enables running the clock at higher frequency
- Superscalar
  - Multiple pipelines executing in parallel
  - Each pipeline may be dedicated to a particular task (integer, float, mem)
  - Challenge is finding instructions in parallel
  - Decreases CPI
Superscalar Issues

- Two instructions have to be fetched and decoded
  - 64-bits fetched at a given PC
- Additional ports are needed in the register file
  - Total 4 read ports, 2 write ports in our example
- Hardware resources have to be replicated
  - One ALU for arithmetic operation, another for MEM Address
  - Additional data forwarding paths, control logic, …
- Problems
  - How to find multiple instructions to issue at run time?
  - Compiler technology needs to statically schedule instructions
    - Breaks binary compatibility
  - How to deal with a stall between LD and arithmetic instruction?
    - In this case, next two instructions cannot use load result w/o stalling
Advanced Techniques
Dynamic Pipeline Scheduling

- **Dynamic pipelining**
  - Execute instruction out-of-order to avoid pipeline hazards/stalls
    - A stalled instruction should not hold other instructions
  - Retire instructions in execution order (i.e. commit result)
  - **Decreases CPI**
- **Three major sections**
  - Instruction fetch and issue
  - Execute units
    - Each unit has reservation station to hold operands and operations
    - Instructions held in the reservation station until ready to execute
  - Commit unit
    - Common approach is in-order completion
    - Must discard instructions as a result of a mis-predicted branch

**Dynamically Scheduled Pipeline**

```
<table>
<thead>
<tr>
<th>Instruction fetch and decode unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservation station</td>
</tr>
<tr>
<td>Reservation station</td>
</tr>
<tr>
<td>Functional units</td>
</tr>
<tr>
<td>Integer</td>
</tr>
<tr>
<td>Integer</td>
</tr>
<tr>
<td>Floating point</td>
</tr>
<tr>
<td>Load/Store</td>
</tr>
<tr>
<td>Reservation station</td>
</tr>
<tr>
<td>Reservation station</td>
</tr>
<tr>
<td>Commit unit</td>
</tr>
</tbody>
</table>
```

- Very complex to design and verify
Memory Hierarchy

Memory Systems

Computer

Control

Datapath

Memory

Input

Output
Pipelined Design: Datapath and Control

• Can arbitrarily large amount of I-MEM and D-MEM be accessed in a single cycle?

Memory Hierarchy in Computer Systems

- Processor
- Control
- Datapath
- Registers
- On-Chip Cache
- Second Level Cache (SRAM)
- Main Memory (DRAM)
- Secondary Storage (Disk)
- Tertiary Storage

Speed: 1 ns 10’s ns 100’s ns (10s ms)

Size (bytes): 100s ~ KBytes ~ M Bytes ~ G Bytes ~ Tera Bytes

Cycles (3 GHz): 1 1- 10 100’s 10’s of Millions
Memory Subsystem Challenge

- Conflicting goals to provide:
  - Largest possible memory
  - At fastest access time
  - With lowest cost
- Processor speeds now exceed 3 Ghz (0.3 ns cycle time)
- DRAM access times are still ~10s of ns
- Serious Memory access gap
  - Every instruction has to be accessed from memory
  - ~15% of the instructions are load/store

Static RAM Cell and Data Access

- **Write:**
  1. Drive bit lines to data
  2. Select row
- **Read:**
  1. Precharge bit and $\overline{\text{bit}}$ to Vdd
  2. Select row
  3. Cell pulls one line low
  4. Sense amp on column detects difference between bit and $\overline{\text{bit}}$

Fast access, large area (6 transistors per cell)
Dynamic RAM (DRAM) Cell and Data Access

- **Write:**
  - 1. Drive bit line to data
  - 2. Select row

- **Read:**
  - 1. Precharge bit line to Vdd
  - 2. Select row
  - 3. Cell and bit line share charges
    - Very small voltage changes on the bit line
  - 4. Sense voltage difference
    - Can detect changes of ~1 million electrons
  - 5. Write: restore the value

- **Refresh**
  - 1. Just do a dummy read to every cell

Slow access, small area (1 transistor per cell). Needs periodic refresh.

Magnetic Disk

Average access time =

- Average seek time +
- Average rotational delay +
- Data transfer time +
- Disk controller overhead

Slow access (~ ms), very large capacity (100’s GB)
Who Cares about Memory Hierarchy?

- Processor vs Memory Performance

- Memory technology has not kept pace with Processor Performance
- Memory access time is the performance bottleneck
Memory Hierarchy and Locality

- **Memory locality** is the principle that future memory accesses are near past accesses.
- There are two types of locality:
  - Temporal locality -- near in time
    - we will often access the same data again very soon
  - Spatial locality -- near in space/distance
    - our next access is often very close in address to recent access
- Types(s) of locality in following address sequence?
  1,2,3,4,7,8,9,10,8,8,4,8,9,8,10,8,8…
- Memory hierarchy is designed to take advantage of memory locality.
  - Cache is implemented with SRAM (fast, expensive)
  - Main memory is implemented with DRAM (cheap, slower)
  - Storage is disk and tape (very slow, cheap, vast)
What is a Cache?

- Dictionary meaning:
  - A hiding place used especially for storing provisions
- A cache is a small amount of fast memory
- Memory hierarchies exploit locality by *caching* (keeping close to the processor) data likely to be used again.
- It is impractical to build large, fast memories.
- Caches give an illusion of
  - Fast access time (of a SRAM)
  - With very large capacity (provided by a disk)

Localities and Caching

- A cache is a small amount of fast memory
- Memory hierarchies exploit locality by *caching* (keeping close to the processor) data likely to be used again.
- This is done because we can build large, slow memories and small, fast memories, but we can’t build large, fast memories.
- If it works, we get the illusion of SRAM access time with disk capacity

SRAM (static RAM) -- 5-20 ns access time, very expensive
DRAM (dynamic RAM) -- 60-100 ns, cheaper
disk -- access time measured in milliseconds, very cheap
Cache Terminology

- **Instruction cache**: cache that only holds instructions
- **Data cache**: cache that only holds data
- **Split cache**: instruction and data cache are separate
  - Provides increased bandwidth from the cache
  - Hit rate is lower (than unified cache)
  - Wins over unified cache due to higher bandwidth
- **Unified cache**: cache that holds both instructions and data
  - Hit rate is higher
  - Bandwidth is lower (than that of the split cache)

Cache Terminology

- **Cache hit**: an access where the data is found in the cache
- **Cache miss**: an access which is not found in the cache
- **Hit time**: time to access the cache
- **Miss penalty**: Time to process a cache miss
  - Move data from lower level memory to the cache and CPU
- **Hit ratio**: % of time the data is found in the cache
- **Miss ratio**: (1 - hit ratio)
- **Cache block size or cache line size**: the amount of data that gets transferred on a cache miss
- **Effective access time**:
  - \((\text{Hit Ratio} \times \text{Hit Time}) + (\text{Miss Ratio} \times \text{Miss Time})\)
Pipelined Design: I-Cache & D-Cache

- How is the cache organized and managed?

How are Cache Entries Made?

<table>
<thead>
<tr>
<th>X4</th>
<th>X4</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>X1</td>
</tr>
<tr>
<td>Xn-2</td>
<td>Xn-2</td>
</tr>
<tr>
<td>Xn-1</td>
<td>Xn-1</td>
</tr>
<tr>
<td>X2</td>
<td>X2</td>
</tr>
<tr>
<td>X3</td>
<td>X3</td>
</tr>
</tbody>
</table>

a. Before the reference to Xn

b. After the reference to Xn

- How is it determined whether the data for a given address is in the cache?
- In case of a miss, where is the data corresponding to the new address stored?
A Direct-mapped Cache

- If a data item is in the cache, how do we find it?
  
  \[ \text{Cache location} = (\text{block address}) \mod (\text{Number of cache blocks in the cache}) \]

- In the following case:
  
  - Number of cache blocks in the cache = 8

A cache that can put a line of data in exactly one place is called a direct-mapped cache.

An index is used to determine which line an address might be found in the cache.

The tag identifies a portion of address of the cached data.

Valid bit indicates that entry is valid.

Address trace:

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Valid</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>00010101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>00000101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00001010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>00001100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>00000100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>00001001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>00000111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>00001000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>00010101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>00011000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>00001110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>00001011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>00000100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How is a Block found in the Cache?

A 4 Kbyte Cache with a 1 word blocks
- Number of blocks = Cache Size/(Block Size) = 1 K
- Index bits = log₂(Number of blocks) = 10 bits
- Tag bits = (Total address - Index bits - Byte offset bits) = 32 - 10 - 2 = 20

• 4 Kbyte Cache direct mapped cache with 1 word (4-byte) blocks

Handling a Cache Read Miss

A mis-match on tag and/or Valid bit indicates a miss. Stall CPU.
- Make read request to memory (via memory controller)
- When memory returns the data write it into the cache
- Return the data to the CPU
Handling a Cache Write Miss

A mis-match on tag and/or Valid bit indicates a miss.
- Write tag, valid bit and data into the cache
  Works only if block size = word size
- Should the data be written to memory also?

Announcements

- **Reading Assignment**
  Chapter 7: The Basics of Caches
  - 7.1, 7.2

- **Homework 6:**
  - 6.31, 6.32
  - 7.2, 7.3, 7.4, 7.5, 7.9, 7.10, 7.12, 7.23

- **Quiz**
  **When:** Mon., May 23rd, First 10 minutes of the class
  **Topic:** Cache Chapter 7  **Need:** Paper, pen