Memory

- Memory technologies
- Memory hierarchy
  - Cache basics
  - Cache variations
  - Virtual memory
- Synchronization

Memory Technologies

- Read Only Memory (ROM)
- Static RAM (SRAM)
  - Basic cell: clocked D latch (vs. D flip flop)
  - Array of cells: 1-dimension and 2-dimension
  - RAM access: timing
- Dynamic RAM (DRAM)
  - Basic cell
  - Faster DRAM, e.g. synchronous DRAM
ROM

- *Random access* means that the access time (to read or write) is not dependent on the address -- different from hard disks
- ROM is random access but *read only*
- Programmable ROM (PROM)
- Erasable PROM (EPROM)
- Electrically EPROM (EEPROM)

RAM

- RAM is read and *writeable* random access memory
- Memory is erased when power goes off
  - ROM doesn’t lose memory when power goes off
- Static RAM: faster but more expensive
- Dynamic RAM: slower, smaller, and less expensive but it’s getting faster
**Components: CMOS**

**n-channel transistor**
- Source
- Gate
- Drain
- Gate = '1' → close
- Gate = '0' → open

**p-channel transistor**
- Source
- Gate
- Drain
- Gate = '1' → open
- Gate = '0' → close

**Switch and NOR**

**Switch:**
- When C = 1 then switch is closed, i.e., A = B.
- When C = 0, switch is open

**NOR Circuit:**
- High (H) or GND
- Low (L)
- A = L

- A = H
  - Open
  - Low (L)

- At least one is open

- At least one is closed

Galen Sasaki EE 361 University of Hawaii
Simple Memory Cell: D Latch

Similar to a D flip flop

Clock = L, it holds its value
Clock = H, Q = D,
i.e., it’s transparent from
input to output.

Clock = L

D → Q

Hold

Clock = H

D → Q

Transparent

D Flip Flop

Master

Slave

Clock = L

D → Q

Transparent

Clock = H

D → Q

Hold

last value of D
while Clock = L

Clock = H

D → Q

Transparent

Clock = H

D → Q

Hold

Clock = L

last value of D
while Clock = L

Galen Sasaki

EE 361 University of Hawaii
Static RAM

Common Type

Chip Select (CS):
Asserting it will turn on the RAM

R/W:
Asserting it, will allow you to read it
Unasserting it, will allow you to write to it

Address should be stable while CS = 1

1-Bit SRAM Cell

Let's build it in steps.
First consider just R/W

Add in CS
1-Bit SRAM

32-Bit SRAM
Dynamic RAM

- Memory cells can forget!
- Cells must be refreshed
- Less number of devices per bit
- Read destroys bits
  - Write bit back after a read

A cell

Word line
Passive transistor
Capacitor
Bit line

Sense/write amplifiers -- sense and amplifier data

R/W
CS
data

4M x 1 DRAM

Row decoder
11-to-2048

Address[10-0]

2048 x 2048 array

Column latches

Mux

Latch 2048 bits at a time
Throw away 2047

Dout
Memory Hierarchy

• Motivation, i.e., Why?
• Cache basics
  – Contents
  – Reading
    • Replacement policy
  – Writing
    • Data consistency
• Example: Direct Mapped
• Blocks of memory: better efficiency

Memory Hierarchy

• Making large fast memory cheaply.

<table>
<thead>
<tr>
<th>Memory Types</th>
<th>Access Times</th>
<th>Cost/Mbyte (93)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static RAM (SRAM)</td>
<td>fast (8-35 ns)</td>
<td>$100 - 400</td>
</tr>
<tr>
<td>Dynamic RAM (DRAM)</td>
<td>slower (90 - 120 ns)</td>
<td>$25-50</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>even slower (10-20 M ns)</td>
<td>$1-2</td>
</tr>
</tbody>
</table>

CPU  

cache (fast)  

main memory (slower)
Cache: Why does it work?

- **CPU**
- **cache** (fast)
- **main memory** (slower)

**Principle of locality:** observation about executing programs

- **Temporal locality (locality in time):** if an item is referenced, it will tend to be referenced soon, e.g., loops
- **Spatial locality (locality in space):** if an item is referenced, items that have nearly the same address values will tend to be referenced soon.

---

Temporal Locality

```
main:    add    $1,$2,$3
         andi   $2,$5,19
         ...
loop:    beq    skip
         ...
skip:    ...
```

**Spatial locality**

**Temporal locality**
Cache Basics

Cache holds copies of the contents of memory

Memory access: read/write

• hit: find element in cache
• miss: element not in cache

Replacement policy: if we store something new in the cache, where do we write it?

• Example: Least Recently Used (LRU) -- the “oldest” one.

Cache Contents

tag: main memory address-- it’s an identifier of where the data comes from

age: used by LRU policy this is updated every time the memory address is accessed

valid bit: used or unused

data: contents from memory
Read

**Step 1**
- Processor requests address 1024.
- Check if 1024 is in cache (check tags).

**Step 2**
- Processor obtains data for 1024.
- Update age of 1024.

Write

**Step 1**
- Processor writes address 1024.
- Check if 1024 is in cache (check tags).

**Step 2**
- Processor updates age of 1024.
- Replacement depends on write policy.
Memory Write

Three approaches:

1. **Write-through**: write to both cache and main memory. Simple, but large overhead.

2. **Write-buffer**: write-through, but there is a buffer that takes care of the writing to memory.
   - Buffer allows CPU to go onto the next operation.
   - If buffer is not finished before the next memory access then CPU stalls until the write-through is finished.

3. **Write-back**: write to cache
   - Update main memory when item in cache is replaced. More complicated.

---

Galen Sasaki
EE 361 University of Hawaii

23

Galen Sasaki
EE 361 University of Hawaii

24
Example: Direct Mapped

- Caching: Variations
  
  1. Where do you place a new entry? Or which do you replace?  
     E.g., Least Recently Used  
  2. In case of write, when do you write back into main memory?  
     E.g., write-through (always)

Direct Mapped deals with 1.

You want to place the contents of memory location A into the cache.

You use the value of A to determine this.

Thus, A is always placed into the same position in the cache.  
(Different than LRU)

Most common way is to use the last few bits of A.

Easy to find data in cache (don’t have to search entire contents)

Direct Mapped Example

Use the last two bits to determine the cache index to map memory contents to.

Why the last two bits, rather than the first two bits?
Direct Mapped Example

<table>
<thead>
<tr>
<th>Validation</th>
<th>Used</th>
<th>Unused</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tag:
Main memory address minus last bits

\[
\text{Main memory address} = \text{tag} + \text{cache index (e.g., 01)}
\]
Direct Mapped Hardware

Blocks of Memory

- Speed things up by organizing memory into “blocks” of words.
  - Access “blocks” instead of words
  - Take advantage of spatial locality of references

- Address
  - Block Number or Block Address
  - Block Offset (indicates where the word is in the block)
Memory Organization

* Access time of block is approx. access time of a single word.

Cache
Memory Hierarchy

- Bigger blocks mean smaller miss rates
- BUT
- Higher miss penalties
- TRADE OFF

Designing memory systems to support caches

- Example: 1 clock cycle to send address
- 15 clock cycles for each DRAM access initiated
- 1 clock cycle to send a word of data
Memory Access

READ: same as before

Step 1. Check if word is in cache
Step 2.
   HIT: Read word from cache
   MISS: Read block (and word) from main memory.
       Put block in cache

WRITE: different for write through

Step 1.
   Read block from main memory into cache
   Write into main memory and cache

Cache Variations

• Associative memory
• Set associate memory
• Two levels of caching
Associative Memory

- **Cache Contents**
  - Each element contains: tag, data, valid bit, age

- **Access Cache**
  - Check cache by checking all elements (check tags of all elements to see if there’s a hit)
  - Replacement policy: can be more intelligent to reduce misses
    - E.g., least recently used (LRU): need to update age field
    - E.g., random

- Lowers miss rate but increases hardware complexity (and maybe speed)

---

**Associative Memory Hardware**

![Diagram of Associative Memory Hardware]

- CPU
- byte offset (2 bits)
- 4-to-1 Multiplexer
- OR gate
- Hit
- Data
Set Associative

* **Associative**: Reduces the chance of a miss. Large complexity because all blocks must be searched.

* **Direct-Mapped**: Simpler complexity. A block is mapped to a set just as in direct mapped.

* **Set Associative** (Generalization of above): A block may be mapped anywhere within its set just as in associative.

n-way set associative means n blocks in a set.

A block is a "super word". Block #s are the addresses. Block size = \(2^k\) words.

Main Memory

<table>
<thead>
<tr>
<th>block 0</th>
<th>block 1</th>
<th>block 2</th>
<th>block 3</th>
<th>block 4</th>
<th>block 5</th>
<th>block 6</th>
<th>block 7</th>
</tr>
</thead>
</table>

Word address

\[00\]

\(2^m\) sets

m bits

set index

n-way set associative cache

set 0

set 1

direct map to the set using set index

associative with n blocks

set \(2^{m-1}\)
Cache Performance

A. CPU time = (CPU execution clock cycles + memory-stall clock cycles) x clock cycle time.

memory-stall clock cycles = read-stall cycles + write-stall cycles

B. Read-stall cycles = reads/program * read miss rate * read miss penalty

C. Write-stall cycles = writes/program * write miss rate * write miss penalty + write buffer stalls

usually negligible

D. Simplifying case: Write and read penalties are about the same, and write buffer stalls are negligible.

Memory-stall cycles = memory access/program * miss rate * miss penalty

Memory Hierarchy

![Memory Hierarchy Graph](image)

The graph shows the miss rate (%) for different associativities (One-way, Two-way, Four-way, Eight-way) and memory sizes (1 KB, 16 KB, 2 KB, 32 KB, 4 KB, 64 KB, 8 KB, 128 KB). The y-axis represents the miss rate, while the x-axis shows the associativity. The data points indicate a decrease in miss rate as the associativity increases, with variations for different memory sizes.
Improving Performance

Set associative memory
Choose set and block size so that
• hardware is fast (hit time)
• minimize miss rate

CPU

Level 1 cache

Min hit time

Min miss rate
Lowers miss penalty of L1 cache

Level 2 cache

Main memory

Two Level Cache

CPU

single chip
direct mapped

SRAM
set associative
LRU

Virtual Memory

• Really big memory
  – E.g., there are 32 bits in an address, addressing up to around 4 billion bytes. 64 bit addresses address much much more. Much bigger than main memory.
• Virtual memory is this really big memory but data is physically stored in all kinds of stuff
Virtual Memory

Really big memory

“Pages” instead of “blocks”

Pages are reasonably big chunks of bytes

Pages are stored in main memory (physical memory) or hard disk

How do you keep track of where the pages are?

Main memory

Hard disk

Virtual Memory

Address Translation

Main memory

“a cache for hard disk”

Hard disk

Processor page table register

Page Table does address translation

page #s location

5

7
Accessing Virtual Memory

Virtual Memory

- Disk accesses are slow
  - memory management can be done in software
  - page sizes are large (10s of Kbytes) to mitigate slow access times of disks
  - miss penalties are very large so
    - associative memory
    - least recently used or even more complicated strategies
    - write-back rather than write-through
Virtual Memory

• Programs do not have to be in contiguous portions of memory. They can be on pages located anywhere.
• Note that strategy for virtual memory is different than for caches because of the high miss penalties

Buses
Memory Hierarchy

Virtual address

TLB access

TLB hit? Yes

Physical address

Write? Yes

Write access bit on? Yes

Write data into cache, update the tag, and put the data and the address into the write buffer

Cache miss stall

No

Cache hit? Yes

Deliver data to the CPU

Try to read data from cache

No

Write protection exception

Cache miss stall

No

No

Yes

TLB miss exception

Memory Hierarchy

![Graph showing cache size versus miss rate for different cache capacities and ways.](image)

Galen Sasaki

EE 361 University of Hawaii

57
Memory Hierarchy

Galen Sasaki  
EE 361 University of Hawaii
Memory Hierarchy

Galen Sasaki
EE 361 University of Hawaii
61

Memory Hierarchy

Galen Sasaki
EE 361 University of Hawaii
62
Memory Hierarchy

- **a. One-word-wide memory organization**

- **b. Wide memory organization**

- **c. Interleaved memory organization**

---

### Memory Hierarchy

- **One-way set associative (direct mapped)**

- **Two-way set associative**

- **Four-way set associative**

- **Eight-way set associative (fully associative)**

---

Galen Sasaki

EE 361 University of Hawaii

65
Memory Hierarchy

Levels in the memory hierarchy

- Level 1
- Level 2
- ... 
- Level n

Increasing distance from the CPU in access time

Size of the memory at each level

Memory Hierarchy

<table>
<thead>
<tr>
<th>Speed</th>
<th>CPU</th>
<th>Size</th>
<th>Cost ($/bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fastest</td>
<td>Memory</td>
<td>Smallest</td>
<td>Highest</td>
</tr>
<tr>
<td>Slowest</td>
<td>Memory</td>
<td>Biggest</td>
<td>Lowest</td>
</tr>
</tbody>
</table>

Galen Sasaki
EE 361 University of Hawaii

34
Memory Hierarchy

a. Before the reference to Xn

\[
\begin{array}{c|c|c|c|c}
X4 & & & & \\
X1 & & & & \\
X_n - 2 & & & & \\
X_n - 1 & & & & \\
X2 & & & & \\
X3 & & & & \\
\end{array}
\]

b. After the reference to Xn

\[
\begin{array}{c|c|c|c|c}
X4 & & & & \\
X1 & & & & \\
X_n - 2 & & & & \\
X_n - 1 & & & & \\
X2 & & & & \\
Xn & & & & \\
X3 & & & & \\
\end{array}
\]
Memory Hierarchy

Data are transferred

Processor