Lecture 17: Memory Systems

Prepared by: Professor David A. Patterson
Computer Science 152, 252
Edited and presented by: Prof. Kurt Keutzer
Computer Science 252, Spring 2000
The Five Classic Components of a Computer

- Processor
- Control
- Datapath
- Memory
- Input
- Output

Today’s Topics:
- SRAM Memory Technology
- DRAM Memory Technology
- Memory Organization
Technology Trends

<table>
<thead>
<tr>
<th>Year</th>
<th>Size</th>
<th>Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>64 Kb</td>
<td>250 ns</td>
</tr>
<tr>
<td>1983</td>
<td>256 Kb</td>
<td>220 ns</td>
</tr>
<tr>
<td>1986</td>
<td>1 Mb</td>
<td>190 ns</td>
</tr>
<tr>
<td>1989</td>
<td>4 Mb</td>
<td>165 ns</td>
</tr>
<tr>
<td>1992</td>
<td>16 Mb</td>
<td>145 ns</td>
</tr>
<tr>
<td>1995</td>
<td>64 Mb</td>
<td>120 ns</td>
</tr>
</tbody>
</table>

Logic: 2x in 3 years, 2x in 3 years

DRAM: 4x in 3 years, 2x in 10 years

Disk: 4x in 3 years, 2x in 10 years

1000:1!, 2:1!
Who Cares About the Memory Hierarchy?

Processor-DRAM Memory Gap (latency)

```

1982

Performance

1000

100

10

1

CPU

μProc

60%/yr.

(2X/1.5yr)

Processor-Memory Performance Gap:

(grows 50% / year)

“Moore’s Law”

DRAM

9%/yr.

(2X/10 yrs)

Lec17.4
```
Today’s Situation: Microprocessor

° Rely on caches to bridge gap

° Microprocessor-DRAM performance gap
  • time of a full cache miss in instructions executed
    1st Alpha (7000): \(340 \text{ ns}/5.0 \text{ ns} = 68 \text{ clks x 2 or} \ 136 \text{ instructions}\)
    2nd Alpha (8400): \(266 \text{ ns}/3.3 \text{ ns} = 80 \text{ clks x 4 or} \ 320 \text{ instructions}\)
    3rd Alpha (t.b.d.): \(180 \text{ ns}/1.7 \text{ ns} = 108 \text{ clks x 6 or} \ 648 \text{ instructions}\)
  • \(1/2\text{X latency x 3X clock rate x 3X Instr/clock} \ -5\text{X}\)
Suppose a processor executes at
- Clock Rate = 200 MHz (5 ns per cycle)
- CPI = 1.1
- 50% arith/logic, 30% ld/st, 20% control

Suppose that 10% of memory operations get 50 cycle miss penalty

CPI = ideal CPI + average stalls per instruction
= 1.1(cyc) + (0.30 (datamops/ins) \times 0.10 (miss/datamop) \times 50 (cycle/miss))
= 1.1 cycle + 1.5 cycle
= 2.6

58% of the time the processor is stalled waiting for memory!

A 1% instruction miss rate would add an additional 0.5 cycles to the CPI!
The Goal: illusion of large, fast, cheap memory

- **Fact:** Large memories are slow, fast memories are small

- **How do we create a memory that is large, cheap and fast (most of the time)?**
  - Hierarchy
  - Parallelism
An Expanded View of the Memory System

**Speed:** Fastest
**Size:** Smallest
**Cost:** Highest

**Speed:** Slowest
**Size:** Biggest
**Cost:** Lowest
Why hierarchy works

° The Principle of Locality:
  • Program access a relatively small portion of the address space at any instant of time.
Memory Hierarchy: How Does it Work?

° **Temporal Locality** (Locality in Time):
  => Keep most recently accessed data items closer to the processor

° **Spatial Locality** (Locality in Space):
  => Move blocks consists of contiguous words to the upper levels
**Memory Hierarchy: Terminology**

° **Hit**: data appears in some block in the upper level (example: Block X)
  - **Hit Rate**: the fraction of memory access found in the upper level
  - **Hit Time**: Time to access the upper level which consists of
    RAM access time + Time to determine hit/miss

° **Miss**: data needs to be retrieve from a block in the lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace a block in the upper level +
    Time to deliver the block the processor

° **Hit Time << Miss Penalty**

![Diagram showing memory hierarchy with blocks Blk X and Blk Y, and arrows indicating data flow from/to processor and between levels.](diagram_url)
By taking advantage of the principle of locality:

- Present the user with as much memory as is available in the cheapest technology.
- Provide access at the speed offered by the fastest technology.
How is the hierarchy managed?

° Registers <-> Memory
  • by compiler (programmer?)

° cache <-> memory
  • by the hardware

° memory <-> disks
  • by the hardware and operating system (virtual memory)
  • by the programmer (files)
Memory Hierarchy Technology

° **Random Access:**
  - “Random” is good: access time is the same for all locations
  - **DRAM**: Dynamic Random Access Memory
    - High density, low power, cheap, slow
    - Dynamic: need to be “refreshed” regularly
  - **SRAM**: Static Random Access Memory
    - Low density, high power, expensive, fast
    - Static: content will last “forever” (until lose power)

° **“Non-so-random” Access Technology:**
  - Access time varies from location to location and from time to time
  - Examples: Disk, CDROM

° **Sequential Access Technology:** access time linear in location (e.g., Tape)

° The next two lectures will concentrate on random access technology
  - The Main Memory: DRAMs + Caches: SRAMs
Main Memory Background

° Performance of Main Memory:
   • Latency: Cache Miss Penalty
     - Access Time: time between request and word arrives
     - Cycle Time: time between requests
   • Bandwidth: I/O & Large Block Miss Penalty (L2)

° Main Memory is DRAM: Dynamic Random Access Memory
   • Dynamic since needs to be refreshed periodically (8 ms)
   • Addresses divided into 2 halves (Memory as a 2D matrix):
     - RAS or Row Access Strobe
     - CAS or Column Access Strobe

° Cache uses SRAM: Static Random Access Memory
   • No refresh (6 transistors/bit vs. 1 transistor)
   Size: DRAM/SRAM - 4-8
   Cost/Cycle time: SRAM/DRAM - 8-16
Random Access Memory (RAM) Technology

Why do computer designers need to know about RAM technology?

• Processor performance is usually limited by memory bandwidth
• As IC densities increase, lots of memory will fit on processor chip
  - Tailor on-chip memory to specific needs
    - Instruction cache
    - Data cache
    - Write buffer

What makes RAM different from a bunch of flip-flops?

• Density: RAM is much denser
**Static RAM Cell**

6-Transistor SRAM Cell

- **Write:**
  1. Drive bit lines (bit=1, bit=0)
  2. Select row

- **Read:**
  1. Precharge bit and \( \overline{bit} \) to Vdd or Vdd/2 => make sure equal!
  2. Select row
  3. Cell pulls one line low
  4. Sense amp on column detects difference between bit and \( \overline{bit} \)

Diagram showing the 6-transistor SRAM cell with word (row select) and bit lines. The diagram also includes a note about the pullup replaced with pullup to save area.
Typical SRAM Organization: 16-word x 4-bit

Q: Which is longer: word line or bit line?
Write Enable is usually active low (WE_L)

Din and Dout are combined to save pins:
- A new control signal, output enable (OE_L) is needed
- WE_L is asserted (Low), OE_L is disasserted (High)
  - D serves as the data input pin
- WE_L is disasserted (High), OE_L is asserted (Low)
  - D is the data output pin
- Both WE_L and OE_L are asserted:
  - Result is unknown. Don’t do that!!!

Although could change VHDL to do what desire, must do the best with what you’ve got (vs. what you need)
Typical SRAM Timing

Write Timing:

Read Timing:

- **2^N** words x M bit SRAM
- Write Timing:
  - Data In
  - Write Address
  - Write Setup Time
  - Write Hold Time
- Read Timing:
  - Data Out
  - Read Address
  - Read Access Time
Problems with SRAM

° Six transistors use up a lot of area

° Consider a “Zero” is stored in the cell:
  • Transistor N1 will try to pull “bit” to 0
  • Transistor P2 will try to pull “bit bar” to 1

° But bit lines are precharged to high: Are P1 and P2 necessary?
1-Transistor Memory Cell (DRAM)

° **Write:**
  1. Drive bit line
  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 (fancy sense amp)
     - Can detect changes of ~1 million electrons
  5. Write: restore the value

° **Refresh**
  1. Just do a dummy read to every cell.
Classical DRAM Organization (square)

Row and Column Address together:
- Select 1 bit a time
DRAM logical organization (4 Mbit)

- Square root of bits per RAS/CAS

```
A0...A10

Address Buffer

Row Decoder

Column Decoder

Sense Amps & I/O

Memory Array
(2,048 x 2,048)

Word Line

Storage Cell

Data Out

Data In

D

Q
```
DRAM physical organization (4 Mbit)

Block 0  ...  Block 3

Row Address:
- Block Row Dec. 9:512
- 128K bits
- 512 S.A.
- 128K bits
- 128K bits
- 128K bits

Column Address:
- I/O
- I/O
- I/O
- I/O
- I/O

8 I/Os

I/O Select

D
Q
2

8 I/Os
Memory Systems

\[ T_c = T_{cycle} + T_{controller} + T_{driver} \]
Control Signals (RAS_L, CAS_L, WE_L, OE_L) are all active low

Din and Dout are combined (D):
- WE_L is asserted (Low), OE_L is disasserted (High)
  - D serves as the data input pin
- WE_L is disasserted (High), OE_L is asserted (Low)
  - D is the data output pin

Row and column addresses share the same pins (A)
- RAS_L goes low: Pins A are latched in as row address
- CAS_L goes low: Pins A are latched in as column address
- RAS/CAS edge-sensitive
Key DRAM Timing Parameters

° $t_{RAC}$: minimum time from RAS line falling to the valid data output.
  - Quoted as the speed of a DRAM
  - A fast 4Mb DRAM $t_{RAC} = 60$ ns

° $t_{RC}$: minimum time from the start of one row access to the start of the next.
  - $t_{RC} = 110$ ns for a 4Mbit DRAM with a $t_{RAC}$ of 60 ns

° $t_{CAC}$: minimum time from CAS line falling to valid data output.
  - 15 ns for a 4Mbit DRAM with a $t_{RAC}$ of 60 ns

° $t_{PC}$: minimum time from the start of one column access to the start of the next.
  - 35 ns for a 4Mbit DRAM with a $t_{RAC}$ of 60 ns
DRAM Performance

◦ A 60 ns ($t_{RAC}$) DRAM can
  - perform a row access only every 110 ns ($t_{RC}$)
  - perform column access ($t_{CAC}$) in 15 ns, but time between column accesses is at least 35 ns ($t_{PC}$).
    - In practice, external address delays and turning around buses make it 40 to 50 ns

◦ These times do not include the time to drive the addresses off the microprocessor nor the memory controller overhead.
  - Drive parallel DRAMs, external memory controller, bus to turn around, SIMM module, pins...
  - 180 ns to 250 ns latency from processor to memory is good for a “60 ns” ($t_{RAC}$) DRAM
DRAM Write Timing

° Every DRAM access begins at:
  - The assertion of the RAS_L
  - 2 ways to write: early or late v. CAS

Early Wr Cycle: WE_L asserted before CAS_L
Late Wr Cycle: WE_L asserted after CAS_L
DRAM Read Timing

° Every DRAM access begins at:
  - The assertion of the RAS_L
  - 2 ways to read: early or late v. CAS

256K x 8 DRAM

DRAM Read Cycle Time

RAS_L → CAS_L → WE_L → OE_L

Early Read Cycle: OE_L asserted before CAS_L
Late Read Cycle: OE_L asserted after CAS_L
Main Memory Performance

° **Simple:**
  - CPU, Cache, Bus, Memory same width (32 bits)

° **Wide:**
  - CPU/Mux 1 word; Mux/Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits)

° **Interleaved:**
  - CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is *word interleaved*
DRAM (Read/Write) Cycle Time >> DRAM (Read/Write) Access Time

- 2:1; why?

DRAM (Read/Write) Cycle Time:
- How frequent can you initiate an access?
- Analogy: A little kid can only ask his father for money on Saturday

DRAM (Read/Write) Access Time:
- How quickly will you get what you want once you initiate an access?
- Analogy: As soon as he asks, his father will give him the money

DRAM Bandwidth Limitation analogy:
- What happens if he runs out of money on Wednesday?
Increasing Bandwidth - Interleaving

Access Pattern without Interleaving:

- Start Access for D1
- D1 available
- Start Access for D2

Access Pattern with 4-way Interleaving:

- Access Bank 0
- Access Bank 1
- Access Bank 2
- Access Bank 3

We can Access Bank 0 again
Main Memory Performance

° Timing model
  • 1 to send address,
  • 4 for access time, 10 cycle time, 1 to send data
  • Cache Block is 4 words

° Simple M.P. = 4 x (1 + 10 + 1) = 48
° Wide M.P. = 1 + 10 + 1 = 12
° Interleaved M.P. = 1 + 10 + 1 + 3 = 15
Independent Memory Banks

- **How many banks?**
  
  number banks  number clocks to access word in bank
  
  • For sequential accesses, otherwise will return to original bank before it has next word ready

- **Increasing DRAM => fewer chips => harder to have banks**
  
  • Growth bits/chip DRAM : 50%-60%/yr
  
  • Nathan Myrvold M/S: mature software growth (33%/yr for NT) - growth MB/$ of DRAM (25%-30%/yr)
### Fewer DRAMs/System over Time

(from Pete MacWilliams, Intel)

<table>
<thead>
<tr>
<th>DRAM Generation</th>
<th>'86</th>
<th>'89</th>
<th>'92</th>
<th>'96</th>
<th>'99</th>
<th>'02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory per DRAM growth @ 60% / year</td>
<td>32 → 8</td>
<td>8 → 2</td>
<td>32 → 8</td>
<td>8 → 2</td>
<td>4 → 1</td>
<td>4 → 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Minimum PC Memory Size</th>
<th>4 MB</th>
<th>8 MB</th>
<th>16 MB</th>
<th>32 MB</th>
<th>64 MB</th>
<th>128 MB</th>
<th>256 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory per System growth @ 25%-30% / year</td>
<td>4 MB</td>
<td>8 MB</td>
<td>16 MB</td>
<td>32 MB</td>
<td>64 MB</td>
<td>128 MB</td>
<td>256 MB</td>
</tr>
</tbody>
</table>
Page Mode DRAM: Motivation

° Regular DRAM Organization:
  • N rows x N column x M-bit
  • Read & Write M-bit at a time
  • Each M-bit access requires a RAS / CAS cycle

° Fast Page Mode DRAM
  • N x M “register” to save a row
Fast Page Mode Operation

- **Fast Page Mode DRAM**
  - N x M “SRAM” to save a row

- **After a row is read into the register**
  - Only CAS is needed to access other M-bit blocks on that row
  - RAS_L remains asserted while CAS_L is toggled

![Diagram of DRAM access](image)

- N rows
- N cols
- N x M “SRAM”
- M-bit Output
- M bits

<table>
<thead>
<tr>
<th>1st M-bit Access</th>
<th>2nd M-bit</th>
<th>3rd M-bit</th>
<th>4th M-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS_L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAS_L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Row Address</td>
<td>Col Address</td>
<td>Col Address</td>
<td>Col Address</td>
</tr>
</tbody>
</table>
## DRAM v. Desktop Microprocessors Cultures

<table>
<thead>
<tr>
<th>Standards</th>
<th>pinout, package, refresh rate, capacity, ...</th>
<th>binary compatibility, IEEE 745, I/O bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sources</td>
<td>Multiple</td>
<td>Single</td>
</tr>
<tr>
<td>Figures of Merit</td>
<td>1) capacity, 1a) $/bit</td>
<td>1) SPEC speed</td>
</tr>
<tr>
<td></td>
<td>2) BW, 3) latency</td>
<td>2) cost</td>
</tr>
<tr>
<td>Improve Rate/year</td>
<td>1) 60%, 1a) 25%,</td>
<td>1) 60%,</td>
</tr>
<tr>
<td></td>
<td>2) 20%, 3) 7%</td>
<td>2) little change</td>
</tr>
</tbody>
</table>
DRAM Design Goals

° Reduce cell size 2.5, increase die size 1.5

° Sell 10% of a single DRAM generation
  • 6.25 billion DRAMs sold in 1996

° 3 phases: engineering samples, first customer ship (FCS), mass production
  • Fastest to FCS, mass production wins share

° Die size, testing time, yield => profit
  • Yield >> 60%
    (redundant rows/columns to repair flaws)
DRAM History

° DRAMs: capacity +60%/yr, cost –30%/yr
  • 2.5X cells/area, 1.5X die size in -3 years

° ‘97 DRAM fab line costs $1B to $2B
  • DRAM only: density, leakage v. speed

° Rely on increasing no. of computers & memory per computer (60% market)
  • SIMM or DIMM is replaceable unit
    => computers use any generation DRAM

° Commodity, second source industry
  => high volume, low profit, conservative
  • Little organization innovation in 20 years
    page mode, EDO, Synch DRAM

° Order of importance: 1) Cost/bit 1a) Capacity
  • RAMBUS: 10X BW, +30% cost => little impact
Today’s Situation: DRAM

° Commodity, second source industry
  high volume, low profit, conservative
  • Little organization innovation (vs. processors)
    in 20 years: page mode, EDO, Synch DRAM

° DRAM industry at a crossroads:
  • Fewer DRAMs per computer over time
    - Growth bits/chip DRAM: 50%-60%/yr
    - Nathan Myrvold M/S: mature software growth
      (33%/yr for NT) - growth MB/$ of DRAM (25%-30%/yr)
  • Starting to question buying larger DRAMs?
Today’s Situation: DRAM

- Intel: 30%/year since 1987; 1/3 income profit
Two Different Types of Locality:

- Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
- Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.

By taking advantage of the principle of locality:

- Present the user with as much memory as is available in the cheapest technology.
- Provide access at the speed offered by the fastest technology.

DRAM is slow but cheap and dense:

- Good choice for presenting the user with a BIG memory system

SRAM is fast but expensive and not very dense:

- Good choice for providing the user FAST access time.
Summary: Processor-Memory Performance Gap “Tax”

<table>
<thead>
<tr>
<th>Processor</th>
<th>% Area</th>
<th>% Transistors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha 21164</td>
<td>37%</td>
<td>77%</td>
</tr>
<tr>
<td>StrongArm SA110</td>
<td>61%</td>
<td>94%</td>
</tr>
<tr>
<td>Pentium Pro</td>
<td>64%</td>
<td>88%</td>
</tr>
</tbody>
</table>

- 2 dies per package: Proc/I$/D$ + L2$
- Caches have no inherent value, only try to close performance gap
Recall: Levels of the Memory Hierarchy

Upper Level

- Processor
  - Instr. Operands
    - Cache
      - Blocks
        - Memory
          - Pages
            - Disk
              - Files
                - Tape
- faster

Lower Level

- Larger

Lec17.47
Cache performance equations:

• Time = IC x CT x (ideal CPI + memory stalls/instr)

• memory stalls/instruction =
  Average accesses/inst x Miss Rate x Miss Penalty =
  (Average IFETCH/inst x MissRate_{Inst} x Miss Penalty_{Inst}) +
  (Average Data/inst x MissRate_{Data} x Miss Penalty_{Data})

• Assumes that ideal CPI includes Hit Times.

• Average Memory Access time =
  Hit Time + (Miss Rate x Miss Penalty)
Impact on Performance

° Suppose a processor executes at
  • Clock Rate = 200 MHz (5 ns per cycle)
  • CPI = 1.1
    • 50% arith/logic, 30% ld/st, 20% control

° Suppose that 10% of memory operations get 50 cycle miss penalty

° Suppose that 1% of instructions get same miss penalty

° CPI = ideal CPI + average stalls per instruction
  1.1(cycles/ins) +
  [ 0.30 (DataMops/ins) +
    x 0.10 (miss/DataMop) x 50 (cycle/miss)] +
  [ 1 (InstMop/ins) +
    x 0.01 (miss/InstMop) x 50 (cycle/miss)]
  = (1.1 + 1.5 + .5) cycle/ins = 3.1

° 58% of the time the proc is stalled waiting for memory!
Optimize the memory system organization to minimize the average memory access time for typical workloads.
Example: 1 KB Direct Mapped Cache with 32 B Blocks

° For a $2^N$ byte cache:
  - The uppermost $(32 - N)$ bits are always the Cache Tag
  - The lowest $M$ bits are the Byte Select (Block Size = $2^M$)

<table>
<thead>
<tr>
<th>Cache Tag</th>
<th>Example: 0x50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Index</td>
<td>Byte Select</td>
</tr>
<tr>
<td>Ex: 0x01</td>
<td>Ex: 0x00</td>
</tr>
</tbody>
</table>

Valid Bit Cache Tag

<table>
<thead>
<tr>
<th>Cache Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 31</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>Byte 63</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>Byte 1023</td>
</tr>
<tr>
<td>31</td>
</tr>
</tbody>
</table>
In general, larger block size take advantage of spatial locality **BUT:**

- Larger block size means larger miss penalty:
  - Takes longer time to fill up the block
- If block size is too big relative to cache size, miss rate will go up
  - Too few cache blocks

In general, **Average Access Time:**

\[ \text{Average Access Time} = \text{Hit Time} \times (1 - \text{Miss Rate}) + \text{Miss Penalty} \times \text{Miss Rate} \]
Extreme Example: single line

\[\begin{array}{c|c|c}
\text{Valid Bit} & \text{Cache Tag} & \text{Cache Data} \\
\hline
& & \\
\end{array}\]

\begin{itemize}
\item Block Size = 4 bytes
\end{itemize}

\[\begin{array}{c|c|c|c|c|}
\text{Byte 3} & \text{Byte 2} & \text{Byte 1} & \text{Byte 0} \\
\hline
0 & & & 0 \\
\end{array}\]

\begin{itemize}
\item Cache Size = 4 bytes
\item Only ONE entry in the cache
\item If an item is accessed, likely that it will be accessed again soon
  \begin{itemize}
  \item But it is unlikely that it will be accessed again immediately!!!
  \item The next access will likely to be a miss again
  \begin{itemize}
  \item Continually loading data into the cache but discard (force out) them before they are used again
  \item Worst nightmare of a cache designer: Ping Pong Effect
  \end{itemize}
  \end{itemize}
\item Conflict Misses are misses caused by:
  \begin{itemize}
  \item Different memory locations mapped to the same cache index
  \begin{itemize}
  \item Solution 1: make the cache size bigger
  \item Solution 2: Multiple entries for the same Cache Index
  \end{itemize}
  \end{itemize}
\end{itemize}
Another Extreme Example: Fully Associative

° Fully Associative Cache
  • Forget about the Cache Index
  • Compare the Cache Tags of all cache entries in parallel
  • Example: Block Size = 32 B blocks, we need N 27-bit comparators

° By definition: Conflict Miss = 0 for a fully associative cache

```
<table>
<thead>
<tr>
<th>Cache Tag (27 bits long)</th>
<th>Byte Select</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ex: 0x01</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Cache Tag</th>
<th>Valid Bit</th>
<th>Cache Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 0</td>
<td>**</td>
<td>Byte 63</td>
</tr>
<tr>
<td>X</td>
<td>**</td>
<td>Byte 33</td>
</tr>
<tr>
<td>X</td>
<td>**</td>
<td>Byte 32</td>
</tr>
<tr>
<td>X</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>**</td>
<td></td>
</tr>
</tbody>
</table>
```

Lec17.54
Set Associative Cache

- **N-way set associative**: N entries for each Cache Index
  - N direct mapped caches operate in parallel

- **Example: Two-way set associative cache**
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared to the input in parallel
  - Data is selected based on the tag result
Disadvantage of Set Associative Cache

° N-way Set Associative Cache versus Direct Mapped Cache:
  • N comparators vs. 1
  • Extra MUX delay for the data
  • Data comes AFTER Hit/Miss decision and set selection
° In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  • Possible to assume a hit and continue. Recover later if miss.
A Summary on Sources of Cache Misses

° **Compulsory (cold start or process migration, first reference):** first access to a block
  - “Cold” fact of life: not a whole lot you can do about it
  - Note: If you are going to run “billions” of instruction, Compulsory Misses are insignificant

° **Conflict (collision):**
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity

° **Capacity:**
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size

° **Coherence (Invalidation):** other process (e.g., I/O) updates memory
### Source of Cache Misses Quiz

Assume constant cost.

<table>
<thead>
<tr>
<th>Source of Cache Misses</th>
<th>Direct Mapped</th>
<th>N-way Set Associative</th>
<th>Fully Associative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Size: Small, Medium, Big?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compulsory Miss:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict Miss</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacity Miss</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coherence Miss</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Choices:** Zero, Low, Medium, High, Same
## Sources of Cache Misses Answer

<table>
<thead>
<tr>
<th></th>
<th>Direct Mapped</th>
<th>N-way Set Associative</th>
<th>Fully Associative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Size</td>
<td>Big</td>
<td>Medium</td>
<td>Small</td>
</tr>
<tr>
<td>Compulsory Miss</td>
<td>Same</td>
<td>Same</td>
<td>Same</td>
</tr>
<tr>
<td>Conflict Miss</td>
<td>High</td>
<td>Medium</td>
<td>Zero</td>
</tr>
<tr>
<td>Capacity Miss</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Coherence Miss</td>
<td>Same</td>
<td>Same</td>
<td>Same</td>
</tr>
</tbody>
</table>

**Note:**
If you are going to run “billions” of instruction, Compulsory Misses are insignificant.
Four Questions for Caches and Memory Hierarchy

° Q1: Where can a block be placed in the upper level? *(Block placement)*

° Q2: How is a block found if it is in the upper level? *(Block identification)*

° Q3: Which block should be replaced on a miss? *(Block replacement)*

° Q4: What happens on a write? *(Write strategy)*
Q1: Where can a block be placed in the upper level?

Block 12 placed in 8 block cache:

- Fully associative, direct mapped, 2-way set associative
- S.A. Mapping = Block Number Modulo Number Sets

- **Fully associative:** block 12 can go anywhere
- **Direct mapped:** block 12 can go only into block 4 (12 mod 8)
- **Set associative:** block 12 can go anywhere in set 0 (12 mod 4)
Q2: How is a block found if it is in the upper level?

- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativity shrinks index, expands tag
Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

<table>
<thead>
<tr>
<th>Associativity: 2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>LRU</td>
<td>Random</td>
</tr>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
</tr>
</tbody>
</table>
Q4: What happens on a write?

- **Write through**—The information is written to both the block in the cache and to the block in the lower-level memory.

- **Write back**—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - is block clean or dirty?

**Pros and Cons of each?**

- WT: read misses cannot result in writes
- WB: no writes of repeated writes

**WT always combined with write buffers so that don’t wait for lower level memory**
A Write Buffer is needed between the Cache and Memory

- Processor: writes data into the cache and the write buffer
- Memory controller: write contents of the buffer to memory

Write buffer is just a FIFO:
- Typical number of entries: 4
- Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle

Memory system designer’s nightmare:
- Store frequency (w.r.t. time) > 1 / DRAM write cycle
- Write buffer saturation
Write Buffer Saturation

° Store frequency (w.r.t. time) > 1 / DRAM write cycle
  - If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
    - Store buffer will overflow no matter how big you make it
    - The CPU Cycle Time <= DRAM Write Cycle Time

° Solution for write buffer saturation:
  - Use a write back cache
  - Install a second level (L2) cache: (does this always work?)
Write-miss Policy: Write Allocate versus Not Allocate

° Assume: a 16-bit write to memory location 0x0 and causes a miss

  • Do we read in the block?
    - Yes: Write Allocate
    - No: Write Not Allocate
Impact of Memory Hierarchy on Algorithms

° Today CPU time is a function of (ops, cache misses) vs. just f(ops):
  What does this mean to Compilers, Data structures, Algorithms?


° Quicksort: fastest comparison based sorting algorithm when all keys fit in memory

° Radix sort: also called “linear time” sort because for keys of fixed length and fixed radix a constant number of passes over the data is sufficient independent of the number of keys

° For Alphastation 250, 32 byte blocks, direct mapped L2 2MB cache, 8 byte keys, from 4000 to 4000000
Quicksort vs. Radix as vary number keys: Instructions

Set size in keys

Radix sort

Instructions/key

Quick (Instr/key)

Radix (Instr/key)
Quicksort vs. Radix as vary number keys: Instrs & Time

Radix sort

Quick sort

Time

Quick (Instr/key)
Radix (Instr/key)
Quick (Clocks/key)
Radix (clocks/key)

Set size in keys

Instructions
What is proper approach to fast algorithms?
How Do you Design a Cache?

° Set of Operations that must be supported
  • read:  data <= Mem[Physical Address]
  • write: Mem[Physical Address] <= Data

° Determine the internal register transfers

° Design the Datapath

° Design the Cache Controller
Impact on Cycle Time

Cache Hit Time:
- directly tied to clock rate
- increases with cache size
- increases with associativity

Average Memory Access time (AMAT) =
Hit Time + Miss Rate x Miss Penalty

Compute Time = IC x CT x (ideal CPI + memory stalls)

Example: direct map allows miss signal after data
What happens on a Cache miss?

- For in-order pipeline, 2 options:
  - Freeze pipeline in Mem stage (popular early on: Sparc, R4000)
    
    | IF | ID | EX | Mem | stall | stall | stall | … | stall | Mem | Wr |
    |----|----|----|-----|------|------|------|---|------|-----|----|
    | IF | ID | EX | stall | stall | stall | stall | … | stall | stall | Ex | Wr |

  - Use Full/Empty bits in registers + MSHR queue
    - MSHR = “Miss Status/Handler Registers” (Kroft)
      Each entry in this queue keeps track of status of outstanding memory requests to one complete memory line.
      - Per cache-line: keep info about memory address.
      - For each word: register (if any) that is waiting for result.
      - Used to “merge” multiple requests to one memory line
    - New load creates MSHR entry and sets destination register to “Empty”. Load is “released” from pipeline.
    - Attempt to use register before result returns causes instruction to block in decode stage.
    - Limited “out-of-order” execution with respect to loads. Popular with in-order superscalar architectures.

- Out-of-order pipelines already have this functionality built in… (load queues, etc).
Improving Cache Performance: 3 general options

Time = IC x CT x (ideal CPI + memory stalls/instruction)

memory stalls/instruction =
   Average memory accesses/inst x AMAT =
   (Average IFETCH/inst x AMAT_{Inst}) +
   (Average DMEM/inst x AMAT_{Data}) +

Average Memory Access time =
   Hit Time + (Miss Rate x Miss Penalty) =

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.
3Cs Absolute Miss Rate (SPEC92)

- Compulsory
- Conflict
- Capacity

Compulsory vanishingly small

Cache Size (KB)

1-way
2-way
4-way
8-way
miss rate 1-way associative cache size \( X \)
= miss rate 2-way associative cache size \( X/2 \)

2:1 Cache Rule
3Cs Relative Miss Rate

Flaws: for fixed block size
Good: insight => invention

Cache Size (KB)

Conflict

Capacity

1-way
2-way
4-way
8-way

Compulsory

1 2 4 8 16 32 64 128

Lec17.78
1. Reduce Misses via Larger Block Size

The graph illustrates the relationship between block size (in bytes) and miss rate. It shows that as the block size increases, the miss rate generally decreases. The block sizes considered are 16, 32, 64, 128, and 256 bytes. The graph includes lines for different block sizes: 1K, 4K, 16K, 64K, and 256K, with each line representing a specific block size. The y-axis represents the miss rate, ranging from 0% to 25%. The x-axis represents the block size in bytes. The graph suggests that larger block sizes can effectively reduce miss rates.
2. Reduce Misses via Higher Associativity

° 2:1 Cache Rule:
  • Miss Rate DM cache size N - Miss Rate 2-way cache size N/2

° Beware: Execution time is only final measure!
  • Will Clock Cycle time increase?
  • Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%,
    internal + 2%
**Example: Avg. Memory Access Time vs. Miss Rate**

- **Example:** assume $CCT = 1.10$ for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped

<table>
<thead>
<tr>
<th>Cache Size (KB)</th>
<th>1-way</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.33</td>
<td>2.15</td>
<td>2.07</td>
<td>2.01</td>
</tr>
<tr>
<td>2</td>
<td>1.98</td>
<td>1.86</td>
<td>1.76</td>
<td>1.68</td>
</tr>
<tr>
<td>4</td>
<td>1.72</td>
<td>1.67</td>
<td>1.61</td>
<td>1.53</td>
</tr>
<tr>
<td>8</td>
<td>1.46</td>
<td>1.48</td>
<td>1.47</td>
<td>1.43</td>
</tr>
<tr>
<td>16</td>
<td>1.29</td>
<td>1.32</td>
<td>1.32</td>
<td>1.32</td>
</tr>
<tr>
<td>32</td>
<td>1.20</td>
<td>1.24</td>
<td>1.25</td>
<td>1.27</td>
</tr>
<tr>
<td>64</td>
<td>1.14</td>
<td>1.20</td>
<td>1.21</td>
<td>1.23</td>
</tr>
<tr>
<td>128</td>
<td>1.10</td>
<td>1.17</td>
<td>1.18</td>
<td>1.20</td>
</tr>
</tbody>
</table>

(Red means A.M.A.T. not improved by more associativity)
3. Reducing Misses via a “Victim Cache”

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines
4. Reducing Misses via “Pseudo-Associativity”

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?

- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit)

- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Better for caches not tied directly to processor (L2)
  - Used in MIPS R1000 L2 cache, similar in UltraSPARC
5. Reducing Misses by **Hardware Prefetching**

° *E.g., Instruction Prefetching*
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in “stream buffer”
  - On miss check stream buffer

° *Works with data blocks too:*
  - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches

° *Prefetching relies on having extra memory bandwidth that can be used without penalty*
Data Prefetch

- Load data into register (HP PA-RISC loads)
- Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
- Special prefetching instructions cannot cause faults; a form of speculative execution

Issuing Prefetch Instructions takes time

- Is cost of prefetch issues < savings in reduced misses?
- Higher superscalar reduces difficulty of issue bandwidth
7. Reducing Misses by Compiler Optimizations

° McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software

° Instructions
  • Reorder procedures in memory so as to reduce conflict misses
  • Profiling to look at conflicts (using tools they developed)

° Data
  • Merging Arrays: improve spatial locality by single array of compound elements vs. 2 arrays
  • Loop Interchange: change nesting of loops to access data in order stored in memory
  • Loop Fusion: Combine 2 independent loops that have same looping and some variables overlap
  • Blocking: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows
Summary #1 / 3:

° The Principle of Locality:
  • Program likely to access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space

° Three Major Categories of Cache Misses:
  • Compulsory Misses: sad facts of life. Example: cold start misses.
  • Conflict Misses: increase cache size and/or associativity.
    Nightmare Scenario: ping pong effect!
  • Capacity Misses: increase cache size
  • Coherence Misses: invalidation caused by “external” processors or I/O

° Cache Design Space
  • total size, block size, associativity
  • replacement policy
  • write-hit policy (write-through, write-back)
  • write-miss policy
Summary #2 / 3: The Cache Design Space

° Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation

° The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost

° Simplicity often wins
Summary #3 / 3: Cache Miss Optimization

Lots of techniques people use to improve the miss rate of caches:

<table>
<thead>
<tr>
<th>Technique</th>
<th>MR</th>
<th>MP</th>
<th>HT</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger Block Size</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Higher Associativity</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Victim Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Pseudo-Associative Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>HW Prefetching of Instr/Data</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Compiler Controlled Prefetching</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Compiler Reduce Misses</td>
<td>+</td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>