Introduction

- Computer performance depends on:
  - Processor performance
  - Memory system performance

Memory Interface

Processor

Memory

MemWrite

Address

WriteData

CLK

 CLK

WE

ReadData

Chapter 8 :: Memory Systems

Digital Design and Computer Architecture
David Money Harris and Sarah L. Harris

Chapter 8 :: Topics

- Introduction
- Memory System Performance Analysis
- Caches
- Virtual Memory
- Memory-Mapped I/O
- Summary

Introduction

- Up until now, assumed memory could be accessed in 1 clock cycle
- But that hasn’t been true since the 1980’s
Memory System Challenge

- Make memory system appear as fast as processor
- Use a hierarchy of memories
- Ideal memory:
  - Fast
  - Cheap (inexpensive)
  - Large (capacity)

But we can only choose two!

Memory Hierarchy

<table>
<thead>
<tr>
<th>Technology</th>
<th>cost / GB</th>
<th>Access time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>~ $10,000</td>
<td>~ 1 ns</td>
</tr>
<tr>
<td>DRAM</td>
<td>~ $100</td>
<td>~ 100 ns</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>~ $1</td>
<td>~ 10,000,000 ns</td>
</tr>
</tbody>
</table>

Locality

- Exploit locality to make memory accesses fast
- **Temporal Locality:**
  - Locality in time (e.g., if looked at a Web page recently, likely to look at it again soon)
  - If used data recently, likely to use it again soon
  - **How to exploit:** keep recently accessed data in higher levels of memory hierarchy
- **Spatial Locality:**
  - Locality in space (e.g., if read one page of book recently, likely to read nearby pages soon)
  - If used data recently, likely to use nearby data soon
  - **How to exploit:** when access data, bring nearby data into higher levels of memory hierarchy too

Memory Performance

- **Hit:** is found in that level of memory hierarchy
- **Miss:** is not found (must go to next level)
  - **Hit Rate** = \# hits / \# memory accesses
  - Miss Rate = 1 – Hit Rate
  - **Miss Rate** = \# misses / \# memory accesses
  - Average memory access time (AMAT): average time it takes for processor to access data
    \[ AMAT = t_{cache} + MR_{cache}(t_{MAM} + MR_{MAM}(t_{VAM})) \]
Memory Performance Example 1

- A program has 2,000 load and store instructions
- 1,250 of these data values found in cache
- The rest are supplied by other levels of memory hierarchy
- What are the miss and hit rates for the cache?

Hit Rate = \( \frac{1250}{2000} = 0.625 \)

Miss Rate = \( \frac{750}{2000} = 0.375 = 1 - \text{Hit Rate} \)

Memory Performance Example 2

- Suppose processor has 2 levels of hierarchy: cache and main memory
- \( t_{\text{cache}} = 1 \) cycle, \( t_{\text{MM}} = 100 \) cycles
- What is the AMAT of the program from Example 1?

AMAT = \( t_{\text{cache}} + MR_{\text{cache}}(t_{\text{MM}}) \)

= \( [1 + 0.375(100)] \) cycles

= 38.5 cycles
Gene Amdahl, 1922 -

- **Amdahl’s Law**: the effort spent on increasing the performance of a subsystem is wasted unless the subsystem affects a large percentage of the overall performance.
- Cofounded three companies, including one called Amdahl Corporation in 1970.

Cache

*A safe place to hide things*

- Highest level in memory hierarchy
- Fast (typically ~ 1 cycle access time)
- Ideally supplies most of the data to the processor
- Usually holds most recently accessed data

Cache Design Questions

- What data is held in the cache?
- How is data found?
- What data is replaced?

We’ll focus on data loads, but stores follow same principles.

What data is held in the cache?

- Ideally, cache anticipates data needed by processor and holds it in cache
- But impossible to predict future
- So, use past to predict future – temporal and spatial locality:
  - **Temporal locality**: if processor accesses data not held in cache, copy data from lower level of hierarchy into cache. Next time, data is available in cache.
  - **Spatial locality**: copy neighboring data into cache too. Block size = number of bytes copied into cache at once.
**Cache Terminology**

- **Capacity** ($C$):
  - the number of data bits a cache stores
- **Block size** ($b$):
  - bits of data brought into cache at once
- **Number of blocks** ($B = C/b$):
  - number of blocks in cache: $B = C/b$
- **Degree of associativity** ($N$):
  - number of blocks in a set
- **Number of sets** ($S = B/N$):
  - each memory address maps to exactly one cache set

**How is data found?**

- Cache organized into $S$ sets
- Each memory address maps to exactly one set
- Caches categorized by number of blocks in a set:
  - **Direct mapped**: 1 block per set
  - **N-way set associative**: N blocks per set
  - **Fully associative**: all cache blocks are in a single set

**Examine each organization for a cache with**:
- Capacity ($C = 8$ words)
- Block size ($b = 1$ word)
- So, number of blocks ($B = 8$)

---

**Direct Mapped Cache**

- **Address**
  - Memory organization
  - Set Number
  - 8-entry x (1 + 27 + 32)-bit SRAM
  - 27 Word Main Memory
  - 2^b Word Cache

**Direct Mapped Cache Hardware**

- Memory Address
  - Tag
  - Byte
  - V
  - Data
  - Hit
  - Data
  - 8-entry x (1 + 27 + 32)-bit SRAM
### Direct Mapped Cache Performance

#### MIPS assembly code

```assembly
addi $t0, $0, 5
loop: beq $t0, $0, done
lw $t1, 0x4($0)  
lw $t2, 0xC($0)
lw $t3, 0x8($0)
addi $t0, $t0, -1
j    loop
done:
```

<table>
<thead>
<tr>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### Direct Mapped Cache: Conflict

#### MIPS assembly code

```assembly
addi $t0, $0, 5
loop: beq $t0, $0, done
lw $t1, 0x4($0)
lw $t2, 0xC($0)
lw $t3, 0x8($0)
addi $t0, $t0, -1
j    loop
```

<table>
<thead>
<tr>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### Miss Rate

- **Temporal Locality**: Compulsory Misses
  - Miss Rate = $\frac{3}{15} = 20\%$

- **Conflict Misses**
  - Miss Rate = $\frac{10}{10} = 100\%$
N-Way Set Associative Cache

![Diagram of N-Way Set Associative Cache]

N-Way Set Associative Performance

```
# MIPS assembly code
addi $t0, $0, 5
loop: beq $t0, $0, done
lw $t1, 0x4($0)
lw $t2, 0x24($0)
addi $t0, $t0, -1j    loop
done:
```

Miss Rate = 2/10 = 20%
Associativity reduces conflict misses

Fully Associative Cache

```
Miss Rate = 2/10 = 20%
Associativity reduces conflict misses
```

No conflict misses
Expensive to build

N-way Set Associative Performance

```
# MIPS assembly code
addi $t0, $0, 5
loop: beq $t0, $0, done
lw $t1, 0x4($0)
lw $t2, 0x24($0)
addi $t0, $t0, -1j    loop
done:
```

Set 3
Set 2
Set 1
Set 0
Spatial Locality?

- Increase block size:
  - Block size, $b = 4$ words
  - $C = 8$ words
  - Direct mapped (1 block per set)
  - Number of blocks, $B = C/b = 8/4 = 2$

Direct Mapped Cache Performance

```
addi $t0, $0, 5
loop: beq $t0, $0, done
lw $t1, 0x4($0)
lw $t2, 0xC($0)
lw $t3, 0x8($0)
addi $t0, $t0, -1
j loop

done:
```

Miss Rate = 1/15
= 6.67%

Larger blocks reduce compulsory misses through spatial locality
Cache Organization Recap

- Capacity: $C$
- Block size: $b$
- Number of blocks in cache: $B = C/b$
- Number of blocks in a set: $N$
- Number of Sets: $S = B/N$

<table>
<thead>
<tr>
<th>Organization</th>
<th>Number of Ways ($N$)</th>
<th>Number of Sets ($S = B/N$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Mapped</td>
<td>1</td>
<td>$B$</td>
</tr>
<tr>
<td>N-Way Set Associative</td>
<td>$1 &lt; N &lt; B$</td>
<td>$B / N$</td>
</tr>
<tr>
<td>Fully Associative</td>
<td>$B$</td>
<td>1</td>
</tr>
</tbody>
</table>

Capacity Misses

- Cache isn’t big enough to hold all data of interest at one time
- If the cache is full and program tries to access data X that is not in cache, cache must evict data Y to make room for X
- Capacity miss occurs if program then tries to access Y again
- X will be placed in a particular set based on its address
- In a direct mapped cache, there is only one place to put X
- In an associative cache, there are multiple ways where X could go in the set.
- How to choose Y to minimize chance of needing it again?
- Least recently used (LRU) replacement: the least recently used block in a set is evicted when the cache is full.

LRU Replacement

# MIPS assembly

1w $t0, 0x04($$0$$)
1w $t1, 0x24($$0$$)
1w $t2, 0x54($$0$$)

(a) V U Tag Data V Tag Data

<table>
<thead>
<tr>
<th>Set Number</th>
<th>3 (11)</th>
<th>2 (10)</th>
<th>1 (01)</th>
<th>0 (00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V U Tag Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) V U Tag Data V Tag Data

<table>
<thead>
<tr>
<th>Set Number</th>
<th>3 (11)</th>
<th>2 (10)</th>
<th>1 (01)</th>
<th>0 (00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V U Tag Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Caching Summary**

- What data is held in the cache?
  - Recently used data (temporal locality)
  - Nearby data (spatial locality, with larger block sizes)
- How is data found?
  - Set is determined by address of data
  - Block within set also determined by address of data
  - In associative caches, data could be in one of several ways
- What data is replaced?
  - Least-recently used way in the set

**Miss Rate Data**

- Bigger caches reduce capacity misses
- Greater associativity reduces conflict misses

**Miss Rate Data**

- Bigger blocks reduce compulsory misses
- Bigger blocks increase conflict misses

**Multilevel Caches**

- Larger caches have lower miss rates
- But also have longer access times
- Expand the memory hierarchy to multiple levels of caches
  - Level 1: small and fast (e.g. 16 KB, 1 cycle)
  - Level 2: larger and slower (e.g. 256 KB, 2-6 cycles)
- Even more levels are possible
Virtual Memory

- Gives the illusion of a bigger memory without the high cost of DRAM
- Each program uses virtual addresses
  - The entire virtual address space is stored on a hard disk, but DRAM acts as a cache for the most commonly accessed parts
  - These virtual addresses are translated by the CPU into physical addresses in DRAM. If the data is not in DRAM, it is fetched from the hard disk.
- Each program can have its own virtual to physical mapping
  - Two programs can use the same virtual address for different data
  - Programs don’t need to be aware that other ones are running
  - One program (or virus) can’t corrupt the memory used by another
  - This is called memory protection

The Memory Hierarchy

<table>
<thead>
<tr>
<th>Technology</th>
<th>cost / GB</th>
<th>Access time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>~ $10,000</td>
<td>~ 1 ns</td>
</tr>
<tr>
<td>DRAM</td>
<td>~ $100</td>
<td>~ 100 ns</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>~ $1</td>
<td>~ 10,000,000 ns</td>
</tr>
</tbody>
</table>

- Physical Memory: DRAM (Main Memory)
- Virtual Memory: Hard disk
  - Slow, Large, Cheap

The Hard Disk

- Takes milliseconds to seek correct location on disk
- Magnetic Disks
- Read/Write Head
Cache/Virtual Memory Analogues

Physical memory acts as cache for virtual memory

<table>
<thead>
<tr>
<th>Cache</th>
<th>Virtual Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>Page</td>
</tr>
<tr>
<td>Block Size</td>
<td>Page Size</td>
</tr>
<tr>
<td>Block Offset</td>
<td>Page Offset</td>
</tr>
<tr>
<td>Miss</td>
<td>Page Fault</td>
</tr>
<tr>
<td>Tag</td>
<td>Virtual Page Number</td>
</tr>
</tbody>
</table>

Virtual Memory Definitions

• Page size: amount of memory transferred from hard disk to DRAM
• Address translation: determining the physical address from the virtual address
• Page table: lookup table used to translate virtual addresses to physical addresses

Virtual Memory Definitions

Virtual and Physical Addresses

Most accesses hit in physical memory
But programs have the large capacity of virtual memory

Address Translation

Virtual Address

30 29 28 ... 14 13 12 11 10 9 ... 2 1 0

VPN | Page Offset

PPN | Page Offset

Physical Address
Virtual Memory Example

- **System with**
  - Virtual memory size: 2 GB = $2^{31}$ bytes
  - Physical memory size: 128 MB = $2^{27}$ bytes
  - Page size: 4 KB = $2^{12}$ bytes

- **Organization**
  - Virtual addresses use 31 bits
  - Physical addresses use 27 bits
  - Page offset = 12 bits
  - # Virtual pages = $2^{31}/2^{12} = 2^{19}$ (VPN = 19 bits)
  - # Physical pages = $2^{27}/2^{12} = 2^{15}$ (PPN = 15 bits)

What is the physical address of virtual address 0x247C?

- VPN = 0x2
- VPN 0x2 maps to PPN 0x7FFF
- The lower 12 bits (page offset) is the same for virtual and physical addresses (0x47C)
- Physical address = 0x7FFF47C
How do we translate addresses?

- Page table
  - Has an entry for each virtual page
  - Each entry indicates:
    - Valid: whether the virtual page is located in physical memory (if not, it must be fetched from the hard disk)
    - Physical page number where the page is located

Page Table Example

VPN is index into page table

Page Table Example 1

What is the physical address of virtual address 0x5F20?

- VPN = 5
- Entry 5 in page table indicates VPN 5 is in physical page 1
- Physical address is 0x1F20
Page Table Example 2

What is the physical address of virtual address 0x73E0?

Page Table Challenges

- Page table is large
  - usually located in physical memory
- Each load/store requires two memory accesses:
  - one for translation (page table read)
  - one to access data (after translation)
- Cuts memory performance in half
  - Unless we get clever...

Translation Lookaside Buffer (TLB)

- Use a translation lookaside buffer (TLB)
  - Small cache of most recent translations
  - Reduces number of memory accesses required for most loads/stores from two to one
Translation Lookaside Buffer (TLB)

- Page table accesses have a lot of temporal locality:
  - Data accesses have temporal and spatial locality
  - Large page size, so consecutive loads/stores likely to access same page
- TLB
  - Small: accessed in < 1 cycle
  - Typically 16 - 512 entries
  - Fully associative
  - >99% hit rates typical
  - Reduces # of memory accesses for most loads and stores from 2 to 1

Example Two-Entry TLB

Memory Protection

- Multiple programs (processes) run at once
- Each process has its own page table
- Each process can use entire virtual address space without worrying about where other programs are
- A process can only access physical pages mapped in its page table – can’t overwrite memory from another process

Virtual Memory Summary

- Virtual memory increases capacity
- A subset of virtual pages are located in physical memory
- A page table maps virtual pages to physical pages – this is called address translation
- A TLB speeds up address translation
- Using different page tables for different programs provides memory protection
Memory-Mapped Input/Output (I/O)

- Processor accesses I/O devices (like keyboards, monitors, printers) just like it accesses memory
- Each I/O device assigned one or more address
- When that address is detected, data is read from or written to I/O device instead of memory
- A portion of the address space dedicated to I/O devices (for example, addresses 0xFFFF0000 to 0xFFFFFFFF in reserved segment of memory map)

Memory-Mapped I/O Hardware

- **Address Decoder**:  
  - Looks at address to determine which device/memory communicates with the processor
- **I/O Registers**:  
  - Hold values written to the I/O devices
- **ReadData Multiplexer**:  
  - Selects between memory and I/O devices as source of data sent to the processor

The Memory Interface

- Processor
- Memory
- Address Decoder
- I/O Device 1
- I/O Device 2

Memory-Mapped I/O Hardware

- Processor
- Memory
- Address Decoder
- I/O Device 1
- I/O Device 2
Memory-Mapped I/O Code

- Suppose I/O Device 1 is assigned the address 0xFFFFFFF4
  - Write the value 42 to I/O Device 1
  - Read the value from I/O Device 1 and place it in $t3

Recall that the 16-bit immediate is sign-extended to 0xFFFFFFF4

Example I/O Device: Speech Chip

- Allophone: fundamental unit of sound, for example:
  - “hello” = HH1 EH LL AX OW
- Each allophone assigned a 6-bit code, for example:
  - “hello” = 0x1B 0x07 0x2D 0x0F 0x20

See www.speechchips.com
Speech Chip I/O

- **A6:1**: allophone input
- **ALD**: allophone load (low-asserted, i.e. loads the address when ALD goes low)
- **SBY**: standby, indicates when the speech chip is standing by waiting for the next allophone

Driving the Speech Chip

1. Set **ALD** to 1
2. Wait until the chip asserts **SBY** to indicate that it has finished speaking the previous allophone and is ready for the next one
3. Write a 6-bit allophone to **A6:1**
4. Reset **ALD** to 0 to initiate speech

Memory-Mapping the I/O Ports

- **A6:1**: 0xFFFFFF00
- **ALD**: 0xFFFFFF04
- **SBY**: 0xFFFFFF08

Software Driver for the Speech Chip

```
init: addi $t1, $0, 1       # $t1 = 1
    addi $t2, $0, 20      # $t2 = array size * 4
    lui $t3, 0x1000      # $t3 = array base address
    addi $t4, $0, 0       # $t4 = 0 (array index)

start: sw $t1, 0xFF04($0)  # ALD = 1
    loop: lw $t5, 0xFF08($0)  # $t5 = SBY
        beq $0, $t5, loop   # loop until SBY == 1
        add  $t5, $t3, $t4    # $t5 = address of allophone
        lw $t5, 0($t5)      # $t5 = allophone
        sw $t5, 0xFFFF04($0)  # A6:1 = allophone
        sw $t5, 0xFFFF00($0)  # ALD = 0 to initiate speech
        addi $t4, $t4, 4      # increment array index
        beq $t4, $t2, done   # last allophone in array?
        j start            # repeat
    done:
```
**Memory System Review**

- The memory interface
- Memory hierarchy
- Memory-mapped I/O

**Review: The Memory Interface**
**Review: The Memory Hierarchy**

<table>
<thead>
<tr>
<th>Technology</th>
<th>cost / GB</th>
<th>Access time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>~ $10,000</td>
<td>~ 1 ns</td>
</tr>
<tr>
<td>DRAM</td>
<td>~ $100</td>
<td>~ 100 ns</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>~ $1</td>
<td>~ 10,000,000 ns</td>
</tr>
</tbody>
</table>

Emulate memory that is: fast, large, cheap

**Review: Memory-Mapped I/O Hardware**

**Course Summary**

- You have learned about:
  - Combinational and sequential logic
  - Schematic and HDL design entry
  - Digital building blocks: adders, ALUs, multiplexers, decoders, memories, etc.
  - Assembly language – computer architecture
  - Processor design – microarchitecture
  - Memory system design
- The world is an increasingly digital place
- You have the tools to design and build powerful digital circuits that will shape our world
- Use this power wisely and for good!