Advanced Microprocessors

Microprocessor Hardware
Memory & Memory Interfaces

EE 467/567
Winter 2012
Avinash Kodi

Background Materials

- Textbook:
  - See Website
- Other:
  - Articles
Memory Interfacing

Purpose

*Reprogrammable*
- Desktop, Laptop, Professional, Server, etc.
- Pentium, Pentium I, Celeron, Pentium II, III, IV, Extreme Edition, Core, Core 2, Core 2 Duo, etc.

*Embedded*
- Limited re-programmability required
- 8080, 8051, 186, 386, 486, Pentium, Pentium M
Memory Interfacing

8086
- \( A_0 - A_{19} \)
- \( D_0 - D_{15} \)
- Use BHE and BLE \( A_0 \) pin to select memory bank

80386/486
- \( A_0 - A_{31} \)
- \( D_0 - D_{31} \)
- Use BE0, BE1, BE2, BE3 pins to select memory bank

Pentium
- \( A_0 - A_{31} \)
- \( D_0 - D_{63} \)
- Use BE0, BE1, BE2, BE3, BE4, BE5, BE6, BE7 pins to select memory bank

Pentium II & Up
- \( A_0 - A_{63} \)
- \( D_0 - D_{63} \)
- Front-Side Bus

A

Memory Directly Connected to CPU

Memory Interfacing

8086
- \( A_0 - A_{19} \)
- \( D_0 - D_{15} \)
- Use BHE and BLE \( A_0 \) pin to select memory bank

80386/486
- \( A_0 - A_{31} \)
- \( D_0 - D_{31} \)
- Use BE0, BE1, BE2, BE3 pins to select memory bank

Pentium
- \( A_0 - A_{31} \)
- \( D_0 - D_{63} \)
- Use BE0, BE1, BE2, BE3, BE4, BE5, BE6, BE7 pins to select memory bank

Pentium II & Up
- \( A_0 - A_{63} \)
- \( D_0 - D_{63} \)
- Front-Side Bus

B

Memory Connected to CPU via Another Chip
Memory Interfacing (B)  
*Front-Side Bus*

**Memory Interfacing (B)  
Front-Side Bus (FSB)**

- **MCH = Memory Controller Hub**
- **ICH = I/O Controller Hub**

- **Intel Pentium® 4 Processor**
  - 4.2 or 3.2 GB/s

- **AGP4X**
  - >1 GB/s

- **ICH2**
  - 6 Channel Audio
  - 133 MB/s
  - 4 USB Ports

- **Intel® Hub Architecture**

- **ATA 100 MB/s 2 IDE Channels**
- **LAN Interface**
- **Flash BIOS**

- **Memory Interfacing (B)**
- **Front-Side Bus (FSB)**

- **Double Data Rate (DDR) DRAM**
**Memory Interfacing (B)**

*Dual Independent Bus (DIB)*

![Diagram](image)

- **CPU Core**
- **Back-Side Bus**
  - **L2 Cache**

**(Half the CPU Clock Speed)**

**Front-Side Bus**

(800/533/400/100/66.6MHz)

**NOTE:**

Earlier processors shared the bus between Cache and Memory on-die.

**Peak Bandwidth = frequency x (#bytes/bus-clock-cycle)**

Example: 800MHz FSB and 64-bit data bus.

What is the Peak Bandwidth?

-----

**CPU Core**

**Memory Interfacing**

@ 1.5 GHz and 64-bit bus: 48 Gbytes/seconds

**Mem1.9**

**Mem1.10**
The Memory Map

MOV AX, [1A 23 3B AF]

Segmentation

Linear Address

Paging

Physical Address

32-bit Address => \(2^{32} = 4\text{GB}\)

The Memory Map

Protection (CPL, DPL, RPL)

MOV AX, [1A 23 3B AF]

Segmentation

Linear Address

Paging

Physical Address

32-bit Address => \(2^{32} = 4\text{GB}\)
Figure 1. Intel® 875P Chipset System Block Diagram

Figure 9. Memory System Address Map
Memory Devices

Specifications

- \(2^n \times m\) memories:
  - \(2^n\) words of \(m\) bits,
  - \(n\) is the number of input lines (address),
  - \(m\) is the number of output lines (data).

- Example: \(128k \times 8\) SRAM
  - Static Random-Access Memory,
  - 8-bit (byte) data output,
  - 17-bit address input.
**Memory Interface (A)**

*How do we interface one or more memory devices to the CPU?*

- The **CPU** has a specified number of address and data lines,
- The **memory devices** have a specified number of address and data lines,
  - **Memory**: most byte-wide
- **Thus**:  
  - Data selectors and decoders are necessary to access multiple memory devices.

---

**Memory Map (A)**

*Via Direct Connection - Example*

![Memory Map Diagram](image)
Memory Interface (A)

x86 I/O and Memory Pin Examples

Memory Interfacing (A)

16-bit data bus (8086, 80186, 80286, 80386SX)
Memory Interfacing (A)
16-bit data bus - byte/word transfer

Even-address Byte Transfer

Odd-address Byte Transfer

Memory Interfacing (A)
16-bit data bus - byte/word transfer

Even-address Word Transfer
(Word is aligned)

x = least significant byte
x+1 = most significant byte
Memory Interfacing (A)

16-bit data bus - byte/word transfer

Odd-address Word Transfer
(Word is misaligned => 2 bus cycles required for reading and writing)

x = least significant byte
x+1 = most significant byte

16-Bit Data Bus Transfer

Summary

Two pins used: \( A_0 = \overline{BLE} \) and \( BHE \)

Even address byte transfer: \( A_0 = 0 \) and \( BHE = 1 \) One Bus Cycle
Odd address byte transfer: \( A_0 = 1 \) and \( BHE = 0 \) One Bus Cycle

Even address word transfer: \( A_0 = 0 \) and \( BHE = 0 \) One Bus Cycle
Odd address word transfer: \( A_0 = 0 \) and \( BHE = 1 \) followed by \( A_0 = 1 \) and \( BHE = 0 \) Two Bus Cycles
Memory Interfacing (A)

32-bit data bus (80386DX and 80486)

80486

A0 → BE3
A31 → D32
D0 → D31
D31 → D24
D23 → D16
D15 → D8
D7 → D0

Memory Interface (A)

Example

Selector between I/O and memory
Random Access Memory
(read/write volatile memory)

**Dynamic RAM**
Stores each bit as a charge on a capacitor
- Smaller and cheaper, but requires periodic refreshing of capacitors

**Static RAM**
Stores each bit in a flip-flop (like the registers)
- Larger and faster, does not require periodic refreshing

From: http://www.ocfaq.com/article.php/overclocking/62
Static RAM -- Dynamic RAM

Memory Cells

As long as the chip is powered, memory remains!

The capacitor requires a regular refreshing cycle b/c it will slowly discharge!

Memory Blocks

Dynamic RAM - 4

Each DRAM: 64k x 1

8 such DRAMs in parallel:

64k x 8 DRAM
Memory Interface (A)

Decoder Circuitry

- Each memory device must occupy a unique section in the memory address space (memory map).
- In order to attach one or more memory devices to the microprocessor, it is necessary to decode the address sent from the microprocessor.
- **Decoder circuitry** must be added to perform this task:
  - NAND gates
  - Dedicated chips such as 3-to-8 line decoders
  - Programmable Array Logic (PAL)
  - Etc.

Memory Interface (A)

Example - Again

Selector between I/O and memory

Memory Map:

- ROM 0: F8000 – F8FFF
- ROM 2: FA000 – FAFFFF
- ROM 4: FC000 – FCFFFF
- ROM 6: FE000 – FEFFFF
- ROM 1: F9000 – F9FFF
- ROM 3: FB000 – FBFFFF
- ROM 5: FD000 – FDFFFF
- ROM 7: FF000 – FFFFFF

These are physical addresses (PAs)
Decoder Circuitry

NAND Gate Decoder

- Use a NAND gate for decoding:

Memory Map:

![Memory Map Diagram]

Bits A_{19} through A_{11} must be equal to “1” for ROM 0 to be accessed:

PA = 1111 1111 1XXX XXXX XXXX

=> FF800 – FFFFF

In real-mode

MOV AX, [F8 12] with (DS) = F0 00

Memory Idealism

- Infinite Capacity
  - for storing large data sets and programs

- Infinite Bandwidth
  - for rapidly streaming data an programs from and to the processor

- Instantaneous or zero latency
  - to prevent the processor from stalling while waiting for data or program code

- Persistence and non-volatile
  - to allow data and programs to survive even when the power supply is cut off

- Zero- or very low implementation cost

From: “Modern Processor Design - Fundamentals of Superscalar Processors,” Shen and Lipasti

MEM2.34
**Memory Hierarchy**  
*Faster? Cheaper?*

- Registers
- SRAM
- DRAM
- Hard Disk

**Exploitation!**

- Large and slow memory (DRAM) combined with
- Small and fast memory (SRAM) that contains a portion of the large memory; this memory is referred to as cache.
Locality

**Temporal Locality:**
Refers to the access of the same memory locations that occur close together in time.

**Spatial Locality:**
Refers to the access of the nearby memory locations that occur relatively close together in time.

Temporal Locality

<table>
<thead>
<tr>
<th>Program sequence:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 0000000D 8B7508</td>
</tr>
<tr>
<td>B 00000010 8B7D0C</td>
</tr>
<tr>
<td>C 00000013 B905000000</td>
</tr>
<tr>
<td>D 00000018 660306</td>
</tr>
<tr>
<td>E 00000018 668907</td>
</tr>
<tr>
<td>F 0000001E 81C702000000</td>
</tr>
<tr>
<td>G 00000024 49</td>
</tr>
<tr>
<td>H 00000025 75EB</td>
</tr>
</tbody>
</table>

Same code (instruction) memory location ([ebp+8]) is accessed close together in time.
Spatial Locality

Suppose (ebp + 12) = M

Nearby data memory location are accessed in close together in time.

Program sequence:

A B C D E F G H D E F G H D E F G H D E F G H

Suppose (ebp + 12) = M

Memory Hierarchy

Exploitation!

- Large and slow memory (DRAM) combined with
- Small and fast memory (SRAM) that contains a portion of the large memory; this memory is referred to as cache.
Internal Memory

Cache

- **Large and slow memory (DRAM)** combined with
- **Small and fast memory (SRAM)** that contains a portion of the large memory; this memory is referred to as cache.

![Diagram of CPU, Cache, and Main Memory]

Word transfer

Block transfer

**Internal Memory**

Backside Bus - Cache & TLB’s

![Diagram of memory hierarchy including L2 and L3 caches, TLBs, and physical memory]

Cache Components

- Each cache is an SRAM memory.
- Consists of cache lines

Blocks of bytes

Examples:
- Pentium 4 L1 & L2 - cache line is 64 bytes wide
- P6 and Pentiums - cache line is 32 bytes wide

Cache Components

Cache Size: 256 kbytes = 2¹⁸ bytes
Cache Line: 64 bytes = 2⁶ bytes

\[
\frac{2^{18}}{2^6} = 2^{12} = 4k \text{ cache lines}
\]

Pentium 4 L2 Cache Example
**Cache Components**

Cache Lines are identified by a **Tag**

![Diagram of cache components with tags and lines]

**Pentium 4 L2 Cache Example**

**Main Memory**

Memory can also be divided into cache line size blocks

![Diagram of main memory address space with physical blocks]

MEM2.45

MEM2.46
Internal Memory

Cache Read Operation

1. Start
2. Receive physical address A from CPU
3. Is block containing A in cache?
   - Yes (Y): Fetch A word and deliver to CPU
   - No (N): Access main memory for block containing A -> Allocate Cache slot for main memory block
4. Load main memory block into Cache slot
5. Deliver A word to CPU
6. Done

Internal Memory

Cache Hits and Misses

- **Cache hit:**
  - Occurs when the addressed data or instruction is found in the cache.
  - *Hit ratio*: number of hits per total number of cache requests.

- **Cache miss:**
  - Occurs when the addressed data or instruction is not found in the cache.
  - After a cache miss the data is loaded from the memory.
Internal Memory
Cache Design Elements - 1

• **Mapping Function:**
  - How do we pick the blocks from the main memory?
    • Direct
    • Associative
    • Set Associative

• **Replacement Algorithm:**
  - If we read or write a new block, which block are we going to replace?
    • Least Recently Used (LRU)
    • First In First Out (FIFO)
    • Least Frequently Used (LFU)
    • Random

Internal Memory
Cache Design Elements - 2

• **Write Policy:**
  - If we are changing the content of a memory location, how are we going to do that?
    • Write through
    • Write back
    • Write once

• **Number of Caches:**
  • Single, two-level, three-level
  • Unified or split
Cache Mapping Function
Full Associative Mapping

Each memory block can be placed in each of the available cache lines
\[ B_j \rightarrow C_{B_i} \]

Number of blocks in the main memory is 'n', \((0 < j < n-1)\)

Number of tag bits required is \( \log_2 n \) and

Tag contains the highest 'n' bits of the physical address of the block in main memory.

\[ n = 16 \text{ blocks in main memory} \]
**Cache Mapping Function**

*Full Associative Mapping*

**Example:**

- 512 Mbytes main memory ($2^{29}$ bytes)
- 64-bytes per cache line ($2^6$ bytes)

$$n = \frac{2^{29}}{2^6} = 2^{23} = 8\text{Mblocks in main memory, thus } \log_2 2^{23} = 23 \text{ bits for tag}$$

**Physical Address:**

- tag = 23 bits
- 6 bits

00 1A 23 44 = 0 0000 0001 1010 0010 0011 0100 0100

**Cache Mapping Function**

*Direct Mapping*

Each memory block can be placed in available cache lines according to:

$$B_j \rightarrow C_{Bi} \text{ where } i = j \mod m$$

Number of cache lines is 'm' ($0 < i < m - 1$)

Number of blocks in the main memory is 'n', ($0 < j < n - 1$)

Number of tag bits required is $\log_2 (n/m)$
**Cache Mapping Function**

*Direct Mapping*

**Cache**

- Tag (2 bits)  
  - C_B0
  - C_B1
  - C_B2
  - C_B3

**Main Memory**

- B0
- B1
- B2
- B3
- B4
- B5
- B6
- B7
- B8
- B9
- B10
- B11
- B12
- B13
- B14

\[ m = 4 \text{ cache lines} \]

\[ \log_2(16/4) = 2 \]

**Example:**

- 512 Mbytes main memory \((2^{29} \text{ bytes})\)
- 64-bytes per cache line \((2^6 \text{ bytes})\)
- 256kbytes cache \((2^{18} \text{ bytes})\)

\[ m = 2^{18}/2^6 = 2^{12} = 4k \text{ cache lines in the cache,} \]
\[ n = 2^{29}/2^6 = 2^{23} = 8M\text{blocks in main memory,} \]
thus \[ 2^{23}/2^{12} = 2^{11} \text{ main memory blocks can be mapped onto each cache line} \]
thus \[ \log_2 (2^{11}) = 11 \text{ bits for tag} \]

**Physical Address:**

- \( \text{tag} = 11 \text{ bits} \)
- \( \text{line} = 12 \text{ bits} \)
- \( 6 \text{ bits} \)

To identify a byte in the cache line

\[ 00 1A 23 44 = 00000 0001 1010 0010 0011 0100 \]

\( \text{tag} \quad \text{line} \)

**MEM2.55**
### Cache Mapping Function

#### Direct Mapping

**Example:**

1. Cache can hold 64 kbytes data,
2. Cache has \( m = 16k = 2^{14} \) lines,
3. Each cache line consists of one block of 4 (=2²) bytes,
4. Main memory consists of 16 Mbytes (= 2²² bytes).

Main memory consists of \( 2^{24}/2 = 2^{22} \) blocks of 4 bytes.

How many bits are required for the tag?

The 4M blocks in main memory are mapped onto the 16k lines in cache.
Thus, each tag consists of \( \log_2 \left( \frac{2^{22}}{2^{14}} \right) = 8 \) bits

<table>
<thead>
<tr>
<th>Main Memory Address</th>
<th>Tag</th>
<th>Line</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>14</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Set Associative Mapping

Number of cache lines is \( 'm' \), \( 0 < i < m-1 \)

Number of blocks in the main memory is \( 'n' \), \( 0 < j < n-1 \)

Number of lines per sets is \( 'k' \)

\( k \)-way set associative cache

Number of sets is \( v \)

Number of tag bits required is \( \log_2 \left( \frac{nk}{m} \right) = \log_2 \left( \frac{n}{v} \right) \)
Cache Mapping Function

Set Associative Mapping

Example:

512 Mbytes main memory ($2^{29}$ bytes)
64-bytes per cache line ($2^{6}$ bytes)
256kbytes cache ($2^{18}$ bytes)
4-way set associative cache

$k = 4$ and $m = 4k$ cache lines in the cache,
thus $2^{12}/2^2 = 2^{10} = 1k$ sets in the cache

$n = 2^{29}/2^6 = 2^{23} = 8Mblocks$ in main memory,
thus $2^{23}/2^{10} = 2^{13}$ main memory blocks can be mapped onto each set
thus $\log_2 (2^{13}) = 13$ bits for tag

Physical Address:

\[ \begin{array}{c|c|c}
\text{tag} & \text{set} & \text{6 bits} \\
\hline
00 & 0000 & 0011 1010 \\
\end{array} \]

\[ 00 1A 23 44 = 0000 0001 1010 0010 0011 0100 \]

\[ \text{tag} \quad \text{set} \]

MEM2:59

MEM2:60
Cache Mapping Function

Set-Associative Mapping

**Example 1:**
1. Four-way set associative cache
2. Cache can hold 32 kbytes data,
3. Each cache line consists of one block of 16 (=2^4) bytes,
4. Main memory consists of 16 Mbytes (= 2^24 bytes).

Thus:
- Each set consists of 4 lines (because it is a four-way set-associative cache)
- Main memory consists of 2^24/2^4 = 2^20 = 1M blocks of 16 bytes,
- there are 32k (= 2^15) / 16 (= 2^4) = 2^11 = 2k cache lines, and 2k / 4 = 512 (= 2^9) sets.
- 24 bits required to address the main memory (2^24 =16M),
- 9 bits to address all the sets, and 4 bits to address the bytes in the blocks. That leaves 24-9-4 = 11 bits for the tag.

<table>
<thead>
<tr>
<th>Main Memory Address</th>
<th>Tag</th>
<th>Sets</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>

**Example 2:**
1. Two-way set associative cache
2. Cache can hold 256 kbytes of data,
3. Each cache line consists of one block of 8 (=2^3) bytes,
4. Main memory consists of 16 Mbytes (= 2^24 bytes).

Thus:
- Each set consists of 2 lines (because it is a two-way set-associative cache)
- Main memory consists of 2^24/2^3 = 2^21 = 2M blocks of 8 bytes,
- there are 256k (= 2^18) / 8 (= 2^3) = 2^15 = 32k cache lines, and 32k / 2 = 16k (= 2^14) sets.
- 24 bits required to address the main memory (2^24 =16M),
- 14 bits to address all the sets, and 3 bits to address the bytes in the blocks. That leaves 24-14-3 = 7 bits for the tag.

<table>
<thead>
<tr>
<th>Main Memory Address</th>
<th>Tag</th>
<th>Sets</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>14</td>
<td>3</td>
</tr>
</tbody>
</table>

Cache Memory

Replacement Policies

- **Replacement Policy:**
  - Policy used to choose the block that is going to be evicted (replaced) when a new block is brought into the cache
  - **Least Recently Used (LRU):**
    - Replace the block in the set that has been in the cache the longest with no reference to it.
    - Use ordered list
    - Easier to implement with Not-Most Recently-Used (NMRU)
  - **First-In First-Out (FIFO):**
    - Replace the block in the set that has been in the cache the longest.
    - Use circular queue
  - **Least Frequently Used (LFU):**
    - Replace the block in the cache that has experienced the fewest references.
  - **Random:**
    - Replace a randomly chosen block in the cache.
Cache Memory

Replacement Policy Example

Assume the following full associative cache:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data</th>
<th>Nanoseconds elapsed since last reference</th>
<th>Nanoseconds elapsed since the data was read into cache</th>
<th>Number of times referenced</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A0020</td>
<td>AA BB CC DD</td>
<td>120</td>
<td>121</td>
<td>2</td>
</tr>
<tr>
<td>1B0012</td>
<td>AA BB CC CD</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>234567</td>
<td>12 5D E9 FA</td>
<td>154</td>
<td>200</td>
<td>4</td>
</tr>
<tr>
<td>172E33</td>
<td>1A 2B 3C 4D</td>
<td>154</td>
<td>2,100</td>
<td>100</td>
</tr>
<tr>
<td>EDEEFA</td>
<td>AB CD EF 01</td>
<td>1,045</td>
<td>2,000</td>
<td>150</td>
</tr>
<tr>
<td>E23000</td>
<td>00 00 00 01</td>
<td>860</td>
<td>900</td>
<td>60</td>
</tr>
<tr>
<td>000001</td>
<td>01 01 A2 34</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1201AB</td>
<td>88 B8 88 B8</td>
<td>20</td>
<td>24</td>
<td>3</td>
</tr>
</tbody>
</table>

Determine the tag of the block will be replaced with each one of the four replacement policies.

Cache Memory

Write Policies

- **Write Policy:**
  - Before a block that is resident in cache can be replaced, it is necessary to consider whether it has been changed (due to writing to the cache) in the cache but not in the main memory.
  - **Write Through:**
    - When changing data using a write operation, the changes are made to the data in both the cache and the main memory.
    - **Disadvantage:** much memory traffic
  - **Write Back:**
    - Updates of the data are only made in cache. When an update occurs a specific UPDATE bit will be set. Then, when the data is replaced, it is written back to the main memory if and only if the UPDATE bit is set.
    - **Disadvantage:** at a particular time there may be an inconsistency between the data in the cache and the main memory.
Caches

- **Unified Cache:**
  - One cache for both instructions and data

- **Split Cache:**
  - A dedicated cache for instructions and a dedicated cache for data,

---

Cache Examples

- **Pentium M:**
  - **On-chip L1 cache:**
    - split
    - L1 instruction cache / L1 data cache
    - 32 kbytes / 32 kbytes
    - 8-way set associative / 8-way set associative
    - 64-bytes per line / 64-byte per line
  - **On-chip L2 cache:**
    - unified
    - 1 Mbytes
    - 8-way set associative
    - 64-bytes per cache line

See page 10-2 in Intel Volume 3
Cache Examples

- **Core/Core Duo (Core 2 Duo):**
  - **On-chip L1 cache:**
    - split
    - L1 instruction cache / L1 data cache
    - 32 kbytes / 32 kbytes
    - 8-way set associative / 8-way set associative
    - 64-bytes per line / 64-byte per line
  - **On-chip L2 cache:**
    - unified
    - 2 Mbytes (4 Mbytes)
    - 8-way set associative (16-way set associative)
    - 64-bytes per cache line

Memory Management

**Paging**

[Diagram of memory management with Linear Address and Physical Address Space]
Memory Management

Paging with TLB

To prevent the CPU from swapping data in and out of main memory unnecessarily, the CPU keeps a table of the 32 most recently used pages:
- Translation Lookaside Buffer (TLB),
- This way access to the page directory and table is not required.
Internal Memory

*Backside Bus - TLB's*

---

**Diagram:**

- Physical Memory
- System Bus (External)
- L2 Cache
- L3 Cache
- Bus Interface Unit
- Instruction Decoder
- Trace Cache
- Instruction TLBs
- Data TLBs
- Data Cache Unit (L1)
- Store Buffer

† Intel Xeon processors only

**Figure 10-1.** Cache Structure of the Pentium 4 and Intel Xeon Processors

*From: IA-32 Intel Architecture Software Developer's Manual - Volume 3*