MEMORY ORGANIZATION

- Memory Hierarchy
- Main Memory
- Auxiliary Memory
- Associative Memory
- Cache Memory
- Virtual Memory
Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system.
RAM and ROM Chips

Typical RAM chip

<table>
<thead>
<tr>
<th>CS1</th>
<th>CS2</th>
<th>RD</th>
<th>WR</th>
<th>Memory function</th>
<th>State of data bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>Inhibit</td>
<td>High-impedence</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>Inhibit</td>
<td>High-impedence</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Inhibit</td>
<td>High-impedence</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Write</td>
<td>Input data to RAM</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>Read</td>
<td>Output data from RAM</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>Inhibit</td>
<td>High-impedence</td>
</tr>
</tbody>
</table>

Typical ROM chip

<table>
<thead>
<tr>
<th>Chip select 1</th>
<th>Chip select 2</th>
<th>9-bit address</th>
<th>Memory function</th>
<th>State of data bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS1</td>
<td>CS2</td>
<td>AD 9</td>
<td>512 x 8 ROM</td>
<td></td>
</tr>
</tbody>
</table>
Address space assignment to each memory chip

Example: 512 bytes RAM and 512 bytes ROM

<table>
<thead>
<tr>
<th>Component</th>
<th>Hexa address</th>
<th>Address bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM 1</td>
<td>0000 - 007F</td>
<td>0 0 0 x x x x x x x x</td>
</tr>
<tr>
<td>RAM 2</td>
<td>0080 - 00FF</td>
<td>0 0 1 x x x x x x x x</td>
</tr>
<tr>
<td>RAM 3</td>
<td>0100 - 017F</td>
<td>0 1 0 x x x x x x x x</td>
</tr>
<tr>
<td>RAM 4</td>
<td>0180 - 01FF</td>
<td>0 1 1 x x x x x x x x</td>
</tr>
<tr>
<td>ROM</td>
<td>0200 - 03FF</td>
<td>1 x x x x x x x x x x</td>
</tr>
</tbody>
</table>

Memory Connection to CPU

- RAM and ROM chips are connected to a CPU through the data and address buses

- The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs
CONNECTION OF MEMORY TO CPU

Address bus: 16-11 10 9 8 7-1

Decoder: 3 2 1 0

CPU: RD WR

Data bus

Memory Organization

Main Memory

[Diagram showing connections between components like CS1, CS2, RD, WR, AD7, RAM 1, RAM 2, RAM 3, RAM 4, and ROM, with address and data buses]
AUXILIARY MEMORY

Information Organization on Magnetic Tapes

Organization of Disk Hardware
Moving Head Disk  Fixed Head Disk

Prof. H. Yoon
ASSOCIATIVE MEMORY

- Accessed by the content of the data rather than by an address
- Also called Content Addressable Memory (CAM)

Hardware Organization

- Compare each word in CAM in parallel with the content of A (Argument Register)
- If CAM Word[i] = A, M(i) = 1
- Read sequentially accessing CAM for CAM Word(i) for M(i) = 1
- K (Key Register) provides a mask for choosing a particular field or key in the argument in A (only those bits in the argument that have 1’s in their corresponding position of K are compared)
Internal organization of a typical cell $C_{ij}$
MEMORY ORGANIZATION

MATCH LOGIC

\[
\begin{align*}
F'_{i1} & \quad F_{i1} \\
F'_{i2} & \quad F_{i2} \\
\cdots & \quad \cdots \\
F'_{in} & \quad F_{in}
\end{align*}
\]

\[K_1 \quad A_1 \quad K_2 \quad A_2 \quad K_n \quad A_n\]

\[M_i\]
CACHE MEMORY

Locality of Reference
- The references to memory at any given time interval tend to be confined within a localized areas
- This area contains a set of information and the membership changes gradually as time goes by
- **Temporal Locality**
  The information which will be used in near future is likely to be in use already (e.g. Reuse of information in loops)
- **Spatial Locality**
  If a word is accessed, adjacent(near) words are likely accessed soon (e.g. Related data items (arrays) are usually stored together; instructions are executed sequentially)

Cache
- The property of Locality of Reference makes the Cache memory systems work
- Cache is a fast small capacity memory that should hold those information which are most likely to be accessed
PERFORMANCE OF CACHE

Memory Access

All the memory accesses are directed first to Cache
If the word is in Cache; Access cache to provide it to CPU
If the word is not in Cache; Bring a block (or a line) including that word to replace a block now in Cache

- How can we know if the word that is required is there?
- If a new block is to replace one of the old blocks, which one should we choose?

Performance of Cache Memory System

Hit Ratio - % of memory accesses satisfied by Cache memory system
\[ T_e: \text{ Effective memory access time in Cache memory system} \]
\[ T_c: \text{ Cache access time} \]
\[ T_m: \text{ Main memory access time} \]

\[ T_e = T_c + (1 - h) T_m \]

Example: \( T_c = 0.4 \mu s, T_m = 1.2\mu s, h = 85\% \)

\[ T_e = 0.4 + (1 - 0.85) \times 1.2 = 0.58\mu s \]
Memory and Cache Mapping - Associative Mapping

Mapping Function
Specification of correspondence between main memory blocks and cache blocks

- Associative mapping
- Direct mapping
- Set-associative mapping

Associative Mapping
- Any block location in Cache can store any block in memory
  ⇒ Most flexible
- Mapping Table is implemented in an associative memory
  ⇒ Very Expensive
- Mapping Table
  Stores both address and the content of the memory word

Address (15 bits)

Argument register

CAM

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0</td>
<td>0 3 4 5 0</td>
</tr>
<tr>
<td>0 2 7 7 7</td>
<td>0 6 7 1 0</td>
</tr>
<tr>
<td>2 2 2 3 5</td>
<td>0 1 2 3 4</td>
</tr>
</tbody>
</table>
MEMORY AND CACHE MAPPING - DIRECT MAPPING

- Each memory block has only one place to load in Cache
- Mapping Table is made of RAM instead of CAM
- n-bit memory address consists of 2 parts; k bits of Index field and n-k bits of Tag field
- n-bit addresses are used to access main memory and k-bit Index is used to access the Cache

Addressing Relationships

Direct Mapping Cache Organization
DIRECT MAPPING

Operation

- CPU generates a memory request with (TAG;INDEX)
- Access Cache using INDEX ; (tag; data)
  Compare TAG and tag
- If matches ⇒ Hit
  Provide Cache[INDEX](data) to CPU
- If not match ⇒ Miss
  M[tag;INDEX] ← Cache[INDEX](data)
  Cache[INDEX] ← (TAG;M[TAG; INDEX])
  CPU ← Cache[INDEX](data)

Direct Mapping with block size of 8 words

<table>
<thead>
<tr>
<th>Index</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 0</td>
<td>000</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>007</td>
<td>01</td>
</tr>
<tr>
<td>Block 1</td>
<td>010</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>017</td>
<td>01</td>
</tr>
<tr>
<td>Block 63</td>
<td>770</td>
<td>02</td>
</tr>
<tr>
<td></td>
<td>777</td>
<td>02</td>
</tr>
</tbody>
</table>
- Each memory block has a set of locations in the Cache to load

Set Associative Mapping Cache with set size of two

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Data</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0 1</td>
<td>3 4 5 0</td>
<td>0 2</td>
<td>5 6 7 0</td>
</tr>
<tr>
<td>777</td>
<td>0 2</td>
<td>6 7 1 0</td>
<td>0 0</td>
<td>2 3 4 0</td>
</tr>
</tbody>
</table>
Many different block replacement policies are available

LRU (Least Recently Used) is most easy to implement

Cache word = (tag 0, data 0, $U_0$); (tag 1, data 1, $U_1$), $U_i = 0$ or 1 (binary)
CACHE WRITE

Write Through

When writing into memory

If Hit, both Cache and memory is written in parallel
If Miss, Memory is written
For a read miss, missing block may be overloaded onto a cache block

Memory is always updated
-> Important when CPU and DMA I/O are both executing

Slow, due to the memory access time

Write-Back (Copy-Back)

When writing into memory

If Hit, only Cache is written
If Miss, missing block is brought to Cache and write into Cache
For a read miss, candidate block must be written back to the memory

Memory is not up-to-date, i.e., the same item in Cache and memory may have different value
VIRTUAL MEMORY

Give the programmer the illusion that the system has a very large memory, even though the computer actually has a relatively small main memory.

Address Space (Logical) and Memory Space (Physical)

Address Mapping

Memory Mapping Table for Virtual Address -> Physical Address
ADDRESS MAPPING

Address Space and Memory Space are each divided into fixed size group of words called blocks or pages

1K words group

Organization of memory Mapping Table in a paged system

Address space
\[ N = 8K = 2^{13} \]

Memory space
\[ M = 4K = 2^{12} \]
ASSOCIATIVE MEMORY PAGE TABLE

Assume that
Number of Blocks in memory = m
Number of Pages in Virtual Address Space = n

Page Table
- Straight forward design -> n entry table in memory
  Inefficient storage space utilization
  <- n-m entries of the table is empty

- More efficient method is m-entry Page Table
  Page Table made of an Associative Memory
  m words; (Page Number:Block Number)

Page Fault
Page number cannot be found in the Page Table
Processor architecture should provide the ability to restart any instruction after a page fault.
PAGE REPLACEMENT

Decision on which page to displace to make room for an incoming page when no free frame is available

Modified page fault service routine
1. Find the location of the desired page on the backing store
2. Find a free frame
   - If there is a free frame, use it
   - Otherwise, use a page-replacement algorithm to select a *victim* frame
     - Write the victim page to the backing store
3. Read the desired page into the (newly) free frame
4. Restart the user process

---

<table>
<thead>
<tr>
<th>frame</th>
<th>valid/invalid bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>i</td>
</tr>
<tr>
<td>f</td>
<td>v</td>
</tr>
</tbody>
</table>

page table

- Change to invalid
- Reset page table for new page
### PAGE REPLACEMENT ALGORITHMS

#### FIFO

<table>
<thead>
<tr>
<th>Reference string</th>
<th>Page frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1</td>
<td>7 7 7 0 0 0 1 1 1 0 0 0 3 3 3 2 2 2 1 1 1 3 2 2 2 1</td>
</tr>
</tbody>
</table>

FIFO algorithm selects the page that has been in memory the longest time. Using a queue - every time a page is loaded, its identification is inserted in the queue.

Easy to implement
May result in a frequent page fault

#### Optimal Replacement (OPT) - Lowest page fault rate of all algorithms

Replace that page which will not be used for the longest period of time.
LRU

- OPT is difficult to implement since it requires future knowledge
- LRU uses the recent past as an approximation of near future.

Replace that page which has not been used for the longest period of time

Reference string

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 7 | 7 | 7 | 2 | 2 | 4 | 4 | 4 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Page frames

- LRU may require substantial hardware assistance
- The problem is to determine an order for the frames defined by the time of last use
PAGE REPLACEMENT ALGORITHMS

LRU Implementation Methods

- Counters
  - For each page table entry - time-of-use register
  - Incremented for every memory reference
  - Page with the smallest value in time-of-use register is replaced
- Stack
  - Stack of page numbers
  - Whenever a page is referenced its page number is removed from the stack and pushed on top
  - Least recently used page number is at the bottom

Reference string
4 7 0 7 1 0 1 2 1 2 7 1 2

LRU Approximation

- Reference (or use) bit is used to approximate the LRU
- Turned on when the corresponding page is referenced after its initial loading
- Additional reference bits may be used