Chapter 8: Main Memory
Implementation of Page Table

- Page table is kept in main memory
  - Page-table base register (PTBR) points to the page table
  - Page-table length register (PRLR) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
  - The two memory access problem can be solved by the use of a special fast-lookup hardware cache called translation look-aside buffers (TLBs)
Associative Memory

Associative memory – parallel search

<table>
<thead>
<tr>
<th>Page #</th>
<th>Frame #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Address translation (p, d)

- If p is in associative register, get frame # out
- Otherwise get frame # from page table in memory
Paging Hardware With TLB

Diagram showing the process of converting a logical address to a physical address using a combination of a TLB (Translation Lookaside Buffer), page table, and physical memory. The CPU generates a logical address which is then split into page (p) and directory (d) bits. These bits are used to look up a page frame number in the TLB. If there is a hit (TLB hit), the corresponding physical address is retrieved. If there is a miss (TLB miss), the page number is used to access the page table, which provides the frame number. This frame number is then combined with the directory bit to form the physical address.
Effective Access Time

- Associative Lookup = $\varepsilon$ time unit
- Assume memory cycle time is 1 microsecond
- Hit ratio $\alpha$ – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers

**Effective Access Time (EAT)**

$$EAT = (1 + \varepsilon) \alpha + (2 + \varepsilon)(1 - \alpha)$$

$$= 2 + \varepsilon - \alpha$$
Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table
Two-Level Page-Table Scheme
Two-Level Paging Example

- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset
- Thus, a logical address is as follows:

<table>
<thead>
<tr>
<th>page number</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
</tr>
</tbody>
</table>

12 10 10

where $p_1$ is an index into the outer page table, and $p_2$ is the displacement within the page of the outer page table.
Address-Translation Scheme

The logical address p₁p₂d is translated through an outer page table indexed by p₁ to a page of page table indexed by p₂. The page of page table then provides the actual address d.
### Three-level Paging Scheme

<table>
<thead>
<tr>
<th>outer page</th>
<th>inner page</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
<td>$d$</td>
</tr>
<tr>
<td>42</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2nd outer page</th>
<th>outer page</th>
<th>inner page</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
<td>$p_3$</td>
<td>$d$</td>
</tr>
<tr>
<td>32</td>
<td>10</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>
Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
  - A segment is a logical unit such as:
    - main program
    - procedure
    - function
    - method
    - object
    - local variables, global variables
    - common block
    - stack
    - symbol table
    - symbol table
    - arrays
Logical View of Segmentation

user space

physical memory space
Segmentation Architecture

- Logical address consists of a two tuple:
  \(<\text{segment-number}, \text{offset}>\),

- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - *base* – contains the starting physical address where the segments reside in memory
  - *limit* – specifies the length of the segment

- **Segment-table base register (STBR)** points to the segment table’s location in memory

- **Segment-table length register (STLR)** indicates number of segments used by a program;
  
  segment number \(s\) is legal if \(s < \text{STLR}\)
Segmentation Hardware
Example of Segmentation

Diagram showing a logical address space with segments labeled as follows:
- Segment 0: subroutine
- Segment 1: main program
- Segment 2: symbol table
- Segment 3: stack
- Segment 4: Sqrt

A table is also shown with two columns:
- limit
- base

The table values are:
- 0: limit 1000, base 1400
- 1: limit 400, base 6300
- 2: limit 400, base 4300
- 3: limit 1100, base 3200
- 4: limit 1000, base 4700

The physical memory is divided into segments:
- Segment 0
- Segment 1
- Segment 2
- Segment 3
- Segment 4
End of Chapter 8