Mapping Functions 2

- **Direct Mapping**

\[ \text{Cache#} = f(\text{main#}) \]

- **Associative Mapping:**

- **Set Associative Mapping**
Cache System Performance

Hit rate = \frac{\# \text{ of memory references found in cache}}{\text{total \# of memory references}}

Miss rate = 1 - Hit rate

\begin{align*}
(1-h) & \quad \text{cache capacity} \\
0.4 & \quad 512 \\
0.3 & \quad 1024 \\
0.2 & \quad 2048 \\
0.1 & \quad 4096 \\
\infty & \quad 8192 \\
\infty & \quad 16384
\end{align*}

\begin{align*}
\text{Block size (in words)} & \\
16 & \quad 32 \\
64 & \quad 128 \\
256 & \quad 512 \\
\infty & \quad 1024
\end{align*}

\begin{align*}
\text{(block size, cache capacity)} & \\
(128, 1k) & \\
(32, 1k) & \\
(32, 4k) & \\
(128, 4k) & \\
(32, 16k) & \\
(128, 16k) & \\
\infty & \quad \infty
\end{align*}

Set size
Associative Memory

- Capable of addressing items by content vice location
  - also termed “content-addressable memory”

- To be effective, must be able to search all memory locations in parallel
  - requires each bit to have integral logic
  - therefore, these are significantly more costly

- Typical operations:
  - Equal, unequal, less than greater than, min value, max value, between limits, etc.

- Example of “greater than” operation

```
input

Argument Register A

Mask Register B

key

Control

match

Storage Array

select

Output Register

Match Register C
```
Virtual Memory

- Blocking is the grouping of several words into a single managed unit
  - Paging: blocks are of equal length
  - Segmentation: blocks are of unequal length
  - Segmentation with paging: blocks (segments) are multiples of equal length pages

- Memory references consist of a block number and an offset (or displacement) within the block

- Direct Mapping

  \[
  \text{Virtual address} \quad \begin{array}{c|c}
  \text{page no.} & \text{line no.} \\
  \hline
  0 & \\
  1 & \\
  2 & \\
  i & \\
  \cdots & \\
  \text{total pages} & \\
  \end{array} \quad \text{frame no.} \quad \begin{array}{c|c}
  \text{line no.} \\
  \end{array}
  \]

  \[
  \text{Real address}
  \]
Paging Systems 1

- **Associative Mapping**
  
  Virtual address

  page no. | line no.
  --------|--------
  i       | j      
  k       | l      
  ...     | ...    
  total memory

  frame no. | line no.
  --------|--------

- **Combination Mapping**
  
  Virtual address

  page no. | line no.
  --------|--------

  match

  Associative map

  frame no. | line no.
  --------|--------

  Real address

  PP

  1
  2
  ...

  no-match

  Direct Mapping Table

  N_{max}
Paging Systems 2

• Sharing of code and data
  – dynamic, if without indication at load time which pages are sharable
  – segmented system provide better handling for sharing both code and data
    • sized can be logically tied to program structures
    • paged sized are arbitrary and generally a compromise
Paging Systems 3

- Page size constraints
  - efficiency of the secondary storage device
    - most influential
  - space required for page tables
  - average size of the logical programs entities

- As page size increases, the amount of wasted space in the last page in virtual space increases, “internal fragmentation”

- For a fixed associative map size, as page size decreases, the percent of information directly accessible decreases and the direct map table grows, “table fragmentation”

- Research indicates that most logical blocks within programs are smaller that 1000 words, often less than 100 words
  - Therefore pages and segments tend to be in the range of 1238-1024 words

Graph showing the relationship between M(p) and p.
Replacement Algorithms 1

• Local vs global replacement strategy
  – local: consider only the faulting task and select only a frame from one already allocated to that task
  – consider all tasks in current residence for potential replacement

• Random Replacement
  – (local or global, fixed or variable partitioning)
  – contrary to the locality principle
  – simple, minimal overhead
  – provides a good performance baseline

• First-In-First-Out (FIFO)
  – (local and fixed partitioning; can be extended to global and variable partitioning)
  – must maintain a queue of loaded pages
  – frequently used pages can be replaced too often

• “Clock” or First-In-Not-Used-First-Out (FINUFO)
  – (local and fixed partitioning; can be extended to global and variable partitioning)
  – maintains a circular FIFO queue
  – tag each page with a “use” bit which is turned on if that page is used after initial loading
  – no consideration of frequency of use
Replacement Algorithms 2

“Stack Algorithms”

• Least-Recently-Used (LRU)
  – (local and fixed partitioning; can be extended to global and variable partitioning)
  – uses a stack, ordered on use, vice a queue
  – when a page is used shift its position downwards
  – pure LRU has significant overhead due a potential change in stack position with every reference

• Minimal Faulting or Optimal Replacement in Fixed Partition (MIN)
  – (local and fixed partitioning)
  – remove the page which will be referenced furthest in the future
  – required knowledge of future references
  – not realizable in real-time
  – good benchmark for performance comparison

• Average number of page faults
  \[ f_{\text{RAND}} > f_{\text{FIFO}} > f_{\text{CLOCK}} > f_{\text{LRU}} > f_{\text{MIN}} \]
Replacement Algorithms 3

- "Parachor curve" compares the number of faults to the amount of primary memory allocated

- Belady's anomaly appears in some replacement algorithms (e.g. FIFO)
  - page fault rate may increase as the number of allocated frames increase
  - not present in stack-type algorithms

- Trashing is very high, inefficient paging activity
  - a process is trashing if it spending more time paging than executing
  - reflects too few frames & too slow swapping
  - effects can be limited by using local or priority-based replacement algorithms
Segmented Systems 1

- Division of task into equal sized block (pages) is not logically natural

- Unequally-sized Segments provide a more logical alternative
  - can more directly associate blocks with procedures, arrays, or other program elements

- A Segment Table is associated with each task
  - a register contains a pointer to each table
  - must store segment location and length data
  - a flag indicates if the referenced segment is in memory
  - protection bits prevent an out-of-range call

Segment table pointer

Virtual address

real address

Segment table

flag

length

base

protection bits
Segmented Systems 2

- After a fault, a new segment would need to be moved into primary memory
  - requires a check if sufficient space is available
  - two linked lists: Reserved (uses) Segments and List of Available Space (LAVS)

- Algorithms to place a new segment
  - First-Fit: place segment into the first empty space of sufficient size
  - Best-Fit: place segment into an empty space of sufficient size with minimum excess
  - First-Best Fit: place segment into the first empty space of sufficient size with excess of less than a pre-specified amount

- If no fit is found, a replacement algorithm needs to find a loaded segment to remove
Segmented Systems 3

- As the system operates, available memory space tends to fragment
  - resulting numerous, but small areas of available space
  - eventually forcing segment replacement sooner than necessary
  - Compaction involves moving loaded segments
    - takes time (overhead)
    - using smaller segments required memory and time overhead

- Requested segments could also be unduly large
  - a considerable amount of primary memory would be storing code/data not currently being used
  - eventually forcing segment replacement (of other segments) sooner than necessary
  - one solution would be to place limit on the maximum segment size
  - somewhat defeats the purpose of segmentation
  - second solution is to break large segments up into pages
Segmentation and Paging

- Segmentation and Paging can be combined in two ways
  - both approaches involve segment which are divided into one or more pages
  - address translation requires two indirect references
  - use Translation Lookup Buffers (TLB) to speed up the address translation

- “Linear Segmentation”
  - paging is the dominant factor
  - protection bits are stored in the page table
  - no program-related meaning to the segments

```
Segment table pointer

Segment table

limit PT Pointer

Virtual address

s_i p_j o_k

flag

real address

prot. bits

page table for segment s_i

Real address
```
Segmentation and Paging 2

• “Linear Segmentation” (cont.)
  – reduces the number of page entries for a given program over a simple paged system
  – requires a TLB large enough to avoid most segment and page references
  – more common approach

• “Segmented Name Space”
  – segmentation is dominant
  – protection bits are stored in the segment table
  – each segment is a task within a set of associated tasks or process
  – each segment has a page table
  – address translation and range checking is more complex
  – less common approach
Limits of Memory Organizations

• To increase the processing power in computer
  – faster components
  – smaller components
  – parallelism, increase the number of instructions that can be executed in a time period
  – pipelining, improve the instruction throughput
  – organization of the memory system
    • required to insure that the memory can keep up and provide CPU with instructions and data at a sufficient rate for performance improvement to be significant

• The main problem in matching memory response to processor speed is the memory cycle time
  – time between two successive memory operations
  – processor cycle times are typically much shorter than memory cycle times
  – when a processor initiates a memory transfer at time \( t_0 \), the memory will be busy until \( t_0 + t_c \)
    where \( t_c \) is the memory cycle time
  – during this period no other device, I/O controller or even the processor can use the memory since it will be busy responding to the request
Interleaved Memory

• In an interleaved memory, the memory is divided into a set of banks
  – with \( n \) banks, the memory is “\( n \)-way interleaved”
  – consecutive addresses reside in different banks
    • memory location \( i \) is in bank number \( i \mod n \).
  – For example (4 banks each of 256 bytes):
    • in block-orientation, addresses 0 … 255 are in the first bank, 256 … 511 in the second bank, etc
    • in interleaved, addresses 0, 4, 8, are in the first bank, 1, 5, 9, in the second bank, etc.

• Requests are sent to two different banks and can be handled simultaneously
  – the processor can request a transfer from location \( i \) on one cycle, and on the next cycle request information from location \( j \)
  – the information will then be returned on successive cycles.

• The number of cycles a processor has to wait before receiving the contents \( i \), is not affected
  – but the bandwidth is improved
  – if there are enough banks the memory system can potentially send information at a rate of one word per processor cycle, regardless of what the memory cycle time is
Shared & Distributed Memories

- Memories for parallel systems
  - *shared memory system*
    - where is one large virtual memory, and all processors have equal access to data and instructions in this memory
  - *distributed memory*
    - where each processor has a *local memory* that is not accessible from any other processor

- The difference between these approaches is how the memory subsystem interprets an address generated by a processor
  - Example: a processor executes the instruction load \( R_0, \text{Mem}[i] \), which means `load register \( R_0 \) with the contents of memory location \( i \)`

- In a shared memory system, \( i \) is a global address, and \( \text{Mem}[i] \) to one processor is the same memory cell as \( \text{Mem}[i] \) to another processor
  - If both processors execute this instruction at the same time they will both load the same information into their \( R_0 \) registers
• In a distributed memory system, i is a local address
  – If both processors execute this instruction, they may end up with different values in their $R_0$ registers since $\text{Mem}[i]$ designates two different memory cells, one in the local memory of each processor

  ![Diagram](image)

  – The distinction between shared memory and distributed memory determines how different parts of a parallel program will communicate
• With shared memory, it is only necessary to build a data structure in memory and pass references to the data structure to parallel subroutines
  – e.g., a matrix multiplication routine that breaks matrices into quadrants only needs to pass the indices of each quadrant to the parallel subroutines
• With distributed memory, copies of shared data are created in each local memory
  – copies are created by sending a *message* containing the data to another processor
  – e.g., with matrix multiplication, the controlling process would have to send messages to three other processors
  – each message would contain the submatrices required to compute one quadrant of the result
  – a drawback is that these messages might have to be quite large; (e.g. half of each input matrix needs to be sent to each parallel subroutine)
Whether to allocate addresses as contiguous blocks or in interleaved fashion depends on how one expects information to be accessed
  – programs are compiled so there is a high probability that after a processor executes the sequential instructions

Compilers can also allocate vector elements to successive addresses, so operations on entire vectors can take advantage of interleaving
  – for these reasons, vector processors universally have some form of interleaved memory
  – shared memory multiprocessors use the block-oriented scheme since memory referencing patterns in an MIMD system are quite different. There the goal is to connect a processor to a
Interleaved Memory 3

• Systems often provide some flexibility in fetching vector elements
  – In some systems it is possible to load every $n^{th}$ element
    • e.g., when fetching elements of a vector $v$ that is
      stored in consecutive memory cells with $n=4$ the
      memory would return $v_0$, $v_4$, $v_8$, …
    • The interval between elements is known as the
      stride.

• One interesting use of this feature is in
  accessing matrices. If the stride is set to one
  more than the number of rows, a single memory
  request will return the diagonal elements
  (assuming column major layout and the columns
  are stored contiguously). Using a stride may
  cancel any benefits of interleaving if
  programmers are not careful. In an extreme
  case, setting the stride to the degree of
  time is fetched from
  the same bank and the time between successive
  memory cycle time.

(a) four processors sharing a bus
(b) Local caches reduce bus contention
(c) Performance curve. Caches move the
  “knee” to the right, but performance still
  levels off at around 30 processors
Shared Memory

- A straightforward way to connect several processors together to build a multiprocessor is shown in Figure 7. The physical connections are quite simple. Most bus structures allow an arbitrary (but not too large) number of devices to communicate over the bus. Bus protocols were initially designed to allow a single processor and one or more disk or tape controllers to communicate with memory. If the I/O controllers are replaced by processors, one has a small single-bus multiprocessor.

- The problem with this design is that processors must contend for access to the bus. If a processor $P_i$ is fetching an instruction, all other processors $P_{j\neq i}$ must wait until the bus is free. If there are only two processors they can perform close to their maximum rate since the bus can alternate between them: as one processor is decoding and executing an instruction, the other can be using the bus to fetch its next instruction. However, when a third processor is added performance begins to degrade. Usually by the time 10 processors are connected to the bus the performance curve has flattened out so that adding an 11th...
Distributed Memory

- In a distributed memory system the memory is associated with individual processors and a processor is only able to address its own memory. Some authors refer to this type of system as a *multicomputer*, reflecting the fact that the building blocks in the system are themselves small computer systems complete with processor and memory.

- There are several benefits of this organization. First, there is no bus or switch contention. Each processor can utilize the full bandwidth to its own local memory without interference from other processors. Second, the lack of a common bus means there is no inherent limit to the number of processors; the size of the system is now constrained only by the network used to connect processors to each other. Third, there are no cache coherency problems. Each processor is in charge of its own data, and it does not have to worry about putting copies of it in its own local cache and having another processor reference the original.

- The major drawback in the distributed memory
Exercise 10: Analysis of memory cycle time

Figure 21 Gantt chart for 8-way interleaved memory

A Gantt chart can be used to show how interleaved memory works by drawing one row for each memory bank. Mark the time line in units of processor cycles. If a processor requests an item from bank \( b \) at time \( t \), draw a line in row \( b \) starting at time \( t \) and continuing for \( n \) units, where \( n \) is the memory cycle time. Figure 21 shows the Gantt chart for an 8-way interleaved memory in a system where the processor cycle time is 10ns and the memory cycle time is 40ns.

The chart illustrates which memories are busy when the processor requests items from successive memory cells. Asterisks on the time line indicate when data items reach the processor (assuming data is delivered on the last memory cycle). Asterisks in every column indicate the memory is performing at its full potential because there are no bank conflicts.
0. Initialize register C to all 0s
1. Scan A for the first 0 bit, “target bit”
2. Compliment the target bit
3. Mask all bits right of the target bit
4. Perform match on equality OR ing matches in C
5. Restore target bit to 0
6. Find next 0 bit after the target, if none, end, else designate it as new target and go to step 2.