Chapter 7

MEMORY ORGANIZATION

Assistant Professor

Er. Shiva Ram Dam

Contents:

1. Introduction to Memory
2. Hierarchical Memory System
3. Cache Memory: Associative Memory
4. Cache Mapping techniques
5. Replacing data in cache, writing data to the cache, cache performance basics
6. Virtual Memory: Paging, segmentation and Memory protection

7.1 Memory

- A memory stores large number of items commonly referred as words.
- Each word consists of a specific number of bits, for e.g. 8 bits.
- Therefore, a memory can be viewed as:
  - ***m*** words of ***n*** bits
  - each for a total of ***m x n*** bits

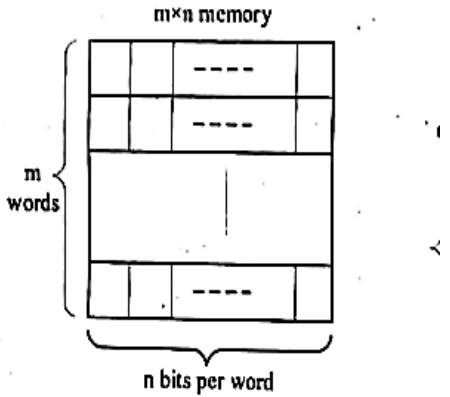


Figure: Words and bits per word in memory

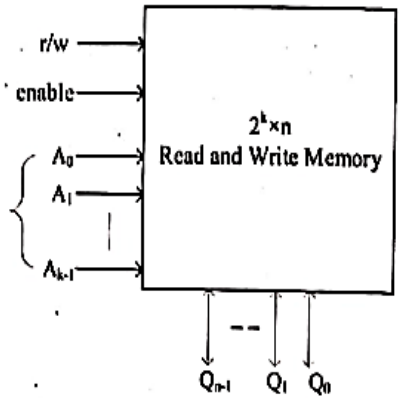
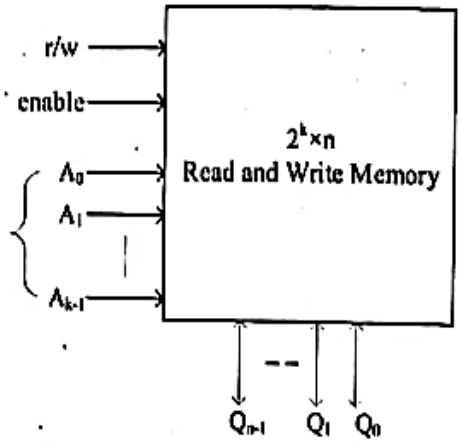


Figure: Memory Block diagram

- If a memory has *k* address inputs, it can have up to 2^k words ($m=2^k$) which implies that; $k=\log_2(m)$.
- This means that $\log_2(m)$ address input signals are required to identify a particular word.
- Also, n data signals are required to output a selected word.
- For example:

  4096 x 8 memory:
  - can store 32768 bits and
  - require 12 address signals (i.e $k= \log_2(4096) = 12$)
  - eight I/O data signals.



7.2 Memory Hierarchy

- There are three key characteristics of memory that can put them in a hierarchical order, namely: capacity, access time, and cost.
- The below relationship holds true for the memories:
  - Faster the access time, the greater is the cost per bit
  - Greater the storage capacity, smaller the cost per bit
  - Greater storage capacity, slower access time

- To meet performance requirements, the ES designer wishes to use expensive, relatively lower-capacity memories with short access times.
- But ultimately, the cost increases.
- So, the designer should employ a memory hierarchy.

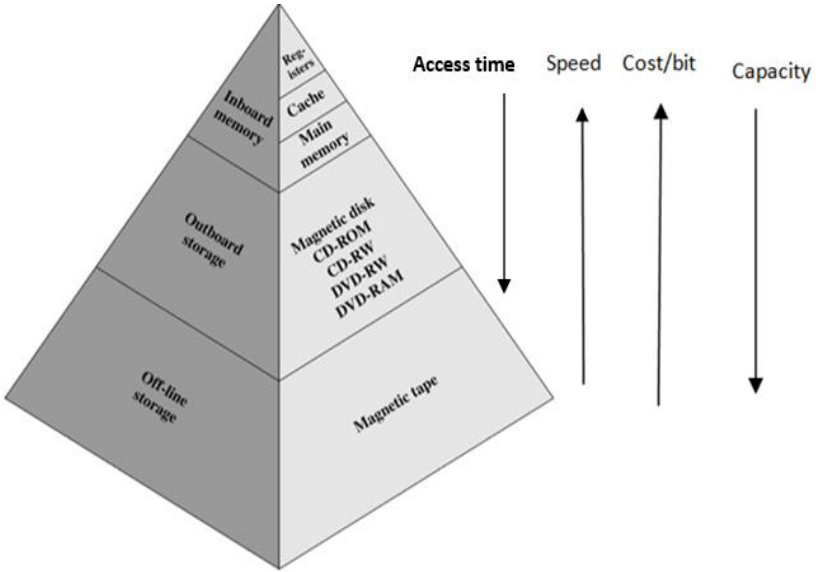


Figure: Memory hierarchy

- As one goes down the hierarchy, the following occur:
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access of the memory by the processor

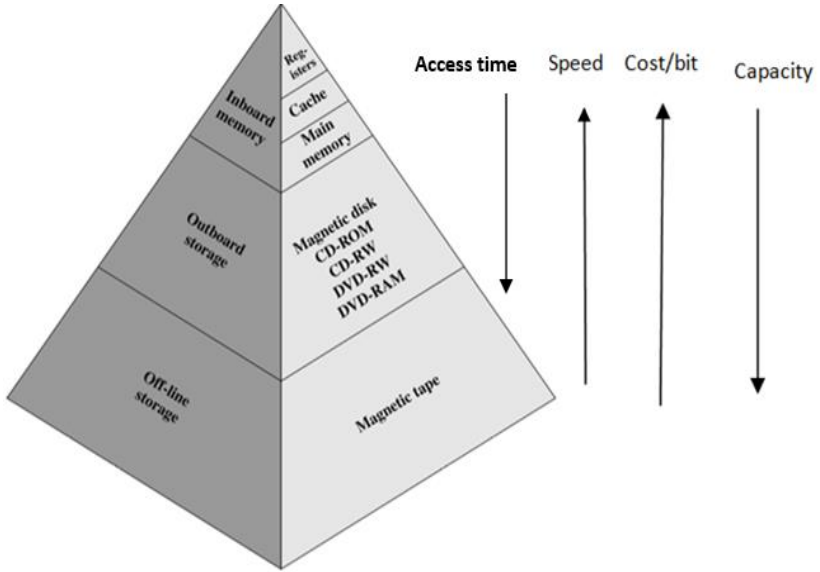


Figure: Memory hierarchy

7.2 Associative Memory

- Aka Content Addressable Memory
- Cache memory is constructed using either static RAM or associative memory, depending on mapping scheme used.
- SRAM receives an address and access the data at that address.
- Associative memory is accessed differently rather than other types of memory. A portion of data is specified to access data in associative memory.
- The associative memory searches all of its locations in parallel and marks the locations that match the specified data input. The matching data values are then read out sequentially.

7.2 Associative Memory

- An associative memory is more expensive than a RAM because each cell must have storage capability as well as logic circuit for matching its contents with an external argument.
- For this reason, associative memories are used in application where search time is very critical and short.

7.2 Associative Memory

- This type of memory is accessed simultaneously & in parallel on the basis of data content rather than by specific address or location
- When a word is written in an associative memory, no address is given.
- The CAM checks for location whose valid bit is 0; stores data in that location, sets valid bit to 1.
- if no location found, clears location(some algorithm) to store data.
- When a word is to be read from an associative memory, the content of word or part of word is specified. The CAM locates all words which match the specified content and mark them for reading.
- Uniquely suitable for parallel searches.
- More expensive than RAM.
- Therefore, CAM are used in application where search time is very critical and must be very short

7.2 Associative Memory

- Cache

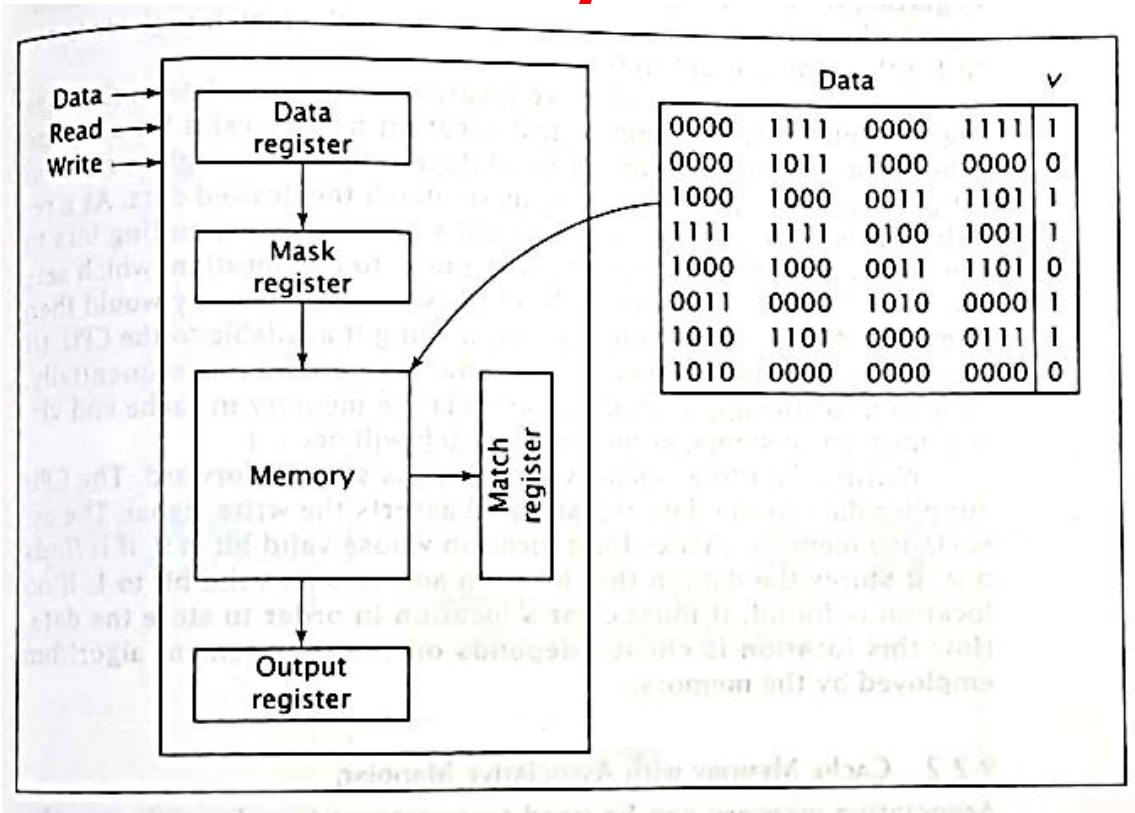
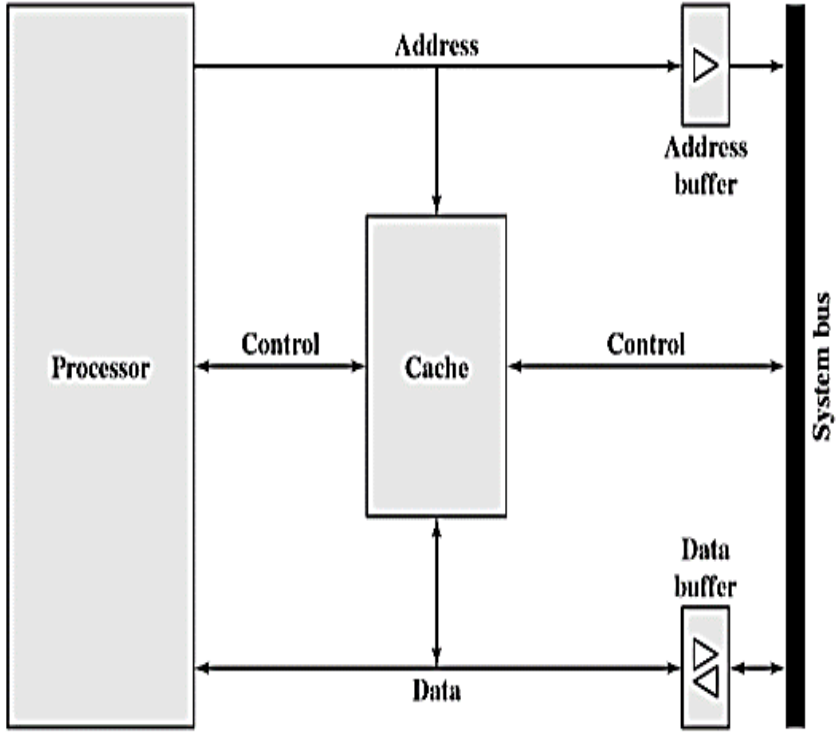


Fig: Internal organization of Associative memory

7.3 Cache

- Cache memory is a small-sized type of volatile computer memory that provides high-speed data access to a processor and stores frequently used computer programs, applications and data.
- It is the fastest memory in a computer, and is typically integrated onto the motherboard and directly embedded in the processor or main random-access memory **(RAM).**
- Cache is usually designed using static RAM rather than dynamic RAM, which is one reason that cache is more expensive but faster than main memory.
- When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache. If so, the word is delivered to the processor. If not, a block of main memory, consisting of fixed number of words is read into the cache and then the word is delivered to the processor.

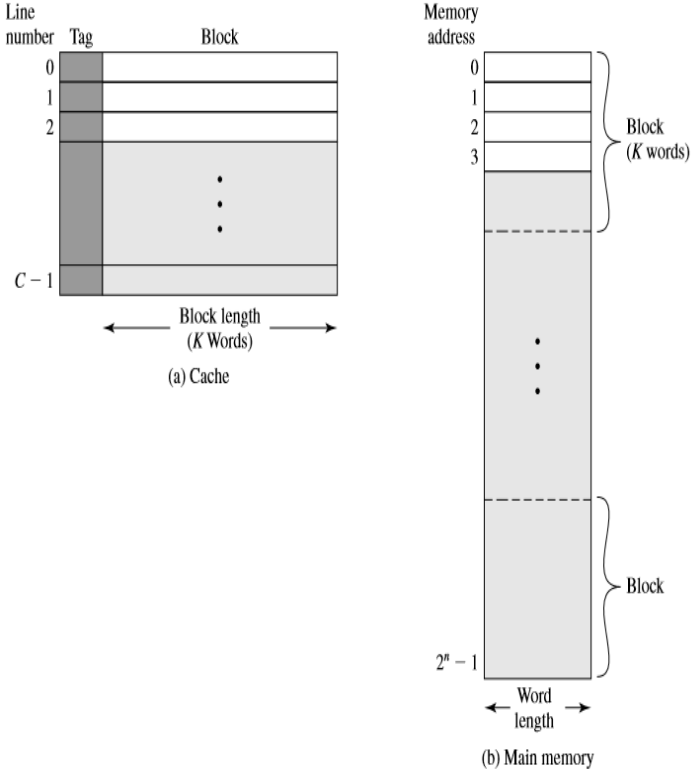
- Cache connects to the processor via data control and address line.
- The data and address lines also attached to data and address buffer which attached to a system bus from which main memory is reached.
- When a cache hit occurs, the data and address buffers are disabled and the communication is only between processor and cache with no system bus traffic.
- When a cache miss occurs, the desired word is first read into the cache and then transferred from cache to processor. For later case, the cache is physically interposed between the processor and main memory for all data, address and control lines.



Typical Cache Organization

Structure of Cache and Main memory:

- Main memory consists of 2^n addressable words, each word having a unique n bit address.
- This memory is considered to consist of a number of fixed length blocks of size = K words each. i.e. M = 2^n/ K blocks.



- Cache consists of C lines of K words each.
- Each line contains K words, plus a tag of a few bits.
- The tag is usually a portion of the main memory address.
- The number of lines (C) of cache is less than the number of main memory blocks (M).

7.3 Cache Mapping Techniques:

- The transformation of data from main memory to cache memory is referred to as memory mapping process.
- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.
- There are three different types of mapping functions in common use and are:
  - direct, associative and set associative.

1. Direct Mapping

- It is the simplest mapping technique.
- It maps certain block of main memory into only one possible cache line, i.e. a given main memory block can be placed in one and only one place on cache.
- It implements one-to-many function, i.e., one cache line can map many blocks of memory.
- The cache line no can be calculated as:

  i = j MOD m

  Where,

  i = cache line number; j = main memory block number;

  m = number of lines in the cache

- For e.g., if there are 16 blocks of main memory and 4 cache lines, and if we need to identify which cache line maps the block no 14 of the main memory.
  - Here, i = 14 MOD 4 =2
  - So, the block no 14 of the main memory is mapped by cache line no 2.
- Each physical address generated by the CPU is viewed as three fields

Cache tag bits	Cache line no	Word

- For eg, if CPU generates a 6 bit address 000101, it is viewed as:

Tag bits	Line no	Word
00	01	01

  - This tells to look at cache line no 1 tagged as 00 at its second index if desired address is present.
- So, a 2-bit tag is enough to identify the block of memory in this case.
- If the search is matched, this is a cache hit. But if the search does not match, then it is a miss.
- **Advantage**: easy to implement
- **Disadvantage**: not flexible, contention problem

- Contention is **what happens when demand for a shared resource exceeds the supply**

- Here, memory consists of 16 blocks of each containing 4 words in an array (say Block 0 has words 0,1,2,3 and Block 1 has 4,5,6,7….Block 15 has 60,61,62,63).
- Similarly, each cache line maps 4 blocks of memory.

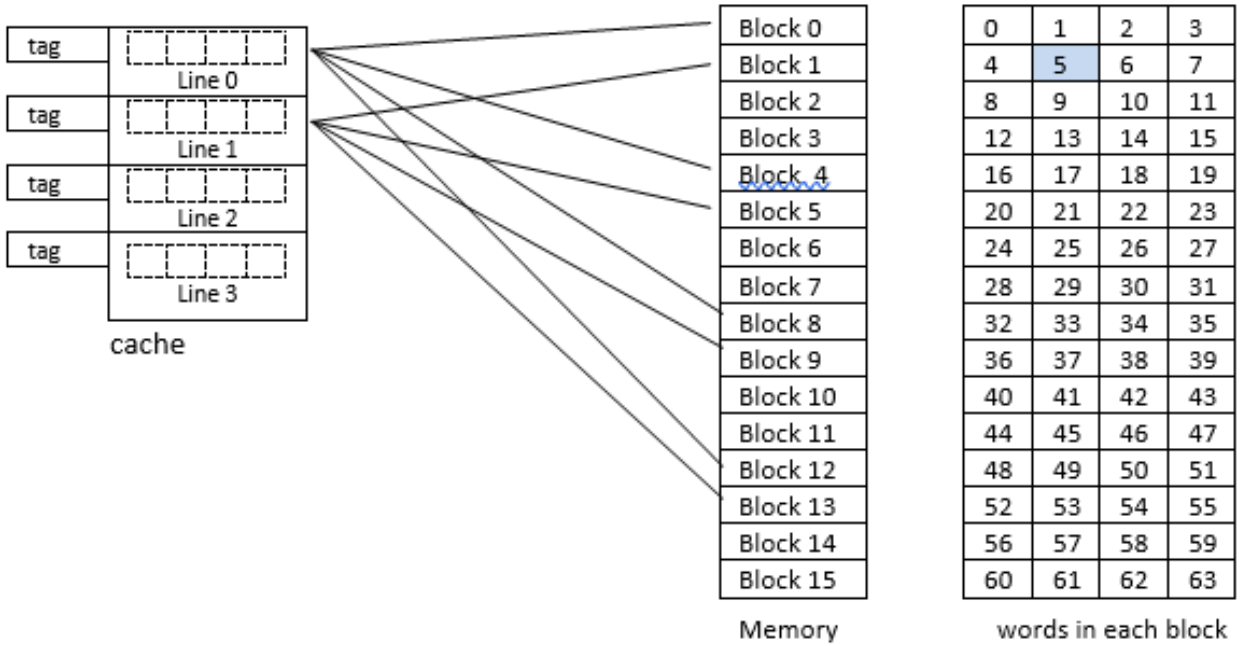
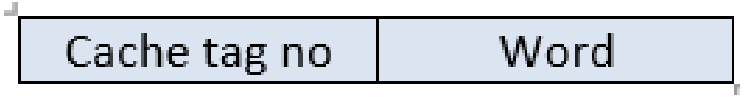


Figure: Direct cache mapping

Associative Mapping

- This is a much more flexible mapping technique.
- Any main memory block can be loaded to any cache line position.
- It implements many-to-many function.
- Each physical address generated by the CPU is viewed as two fields:



- For eg, if CPU generates a 6-bit address 000101, it is viewed as:

Tag no	word
0001	01

  - This tells to look at cache line tagged as 0001 if it contains the word 01 of the block 0001 of the main memory.
  - In this case, a 4-bit tags are required to identify those 16 blocks of main memory. The tag no represents the block no of the main memory.
- Here, we need to search for all the 16 tags from 0000 to 1111 to determine whether a given block is in the cache or not. Therefore, the cost of implementation of hardware (i.e comparator) will be higher.

- Advantage: flexibility
- Disadvantage: increased hardware cost, increased access time
- The figure below represents the associative mapping:

The figure below represents the associative mapping:

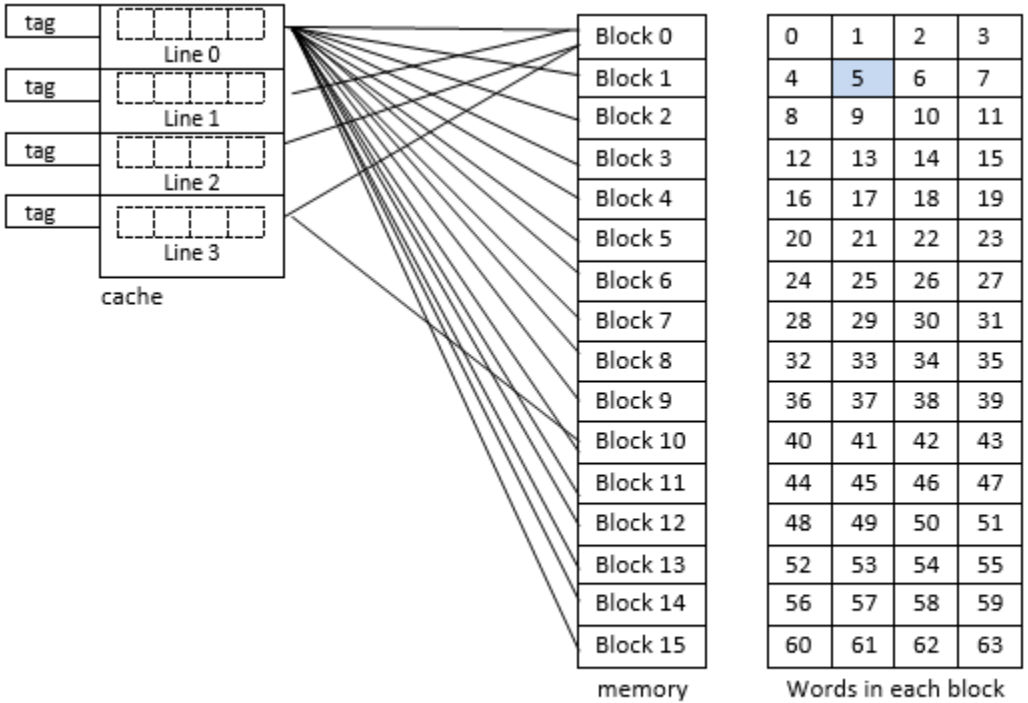


Figure: Associative mapping

Set-associative Mapping

- This is the combination of the direct and associative technique. Overcomes the limitation of contention and increased hardware cost.
- In this case, blocks of the cache are grouped into sets and the mapping allows to reside in any block of a particular set.
- It implements many-to-many function.
- Each physical address generated by the CPU is viewed as three fields:

For e.g., if CPU generates a 6-bit address 000101, it is viewed as:

Tag no	Set no	word
00	01	01

- This tells to look at cache line tagged as 00 and set 0 if it contains the word 01 of the block 0001 of the main memory.
- First, it compares set no 01 if it is present in SET 0 (i.e from 00 and 01) and then looks for tag no 00 and finally looks for the index 2 of the cache line 1.
- In this case, a 4-bit tags are required to identify those 16 blocks of main memory. The tag no represents the block no of the main memory.

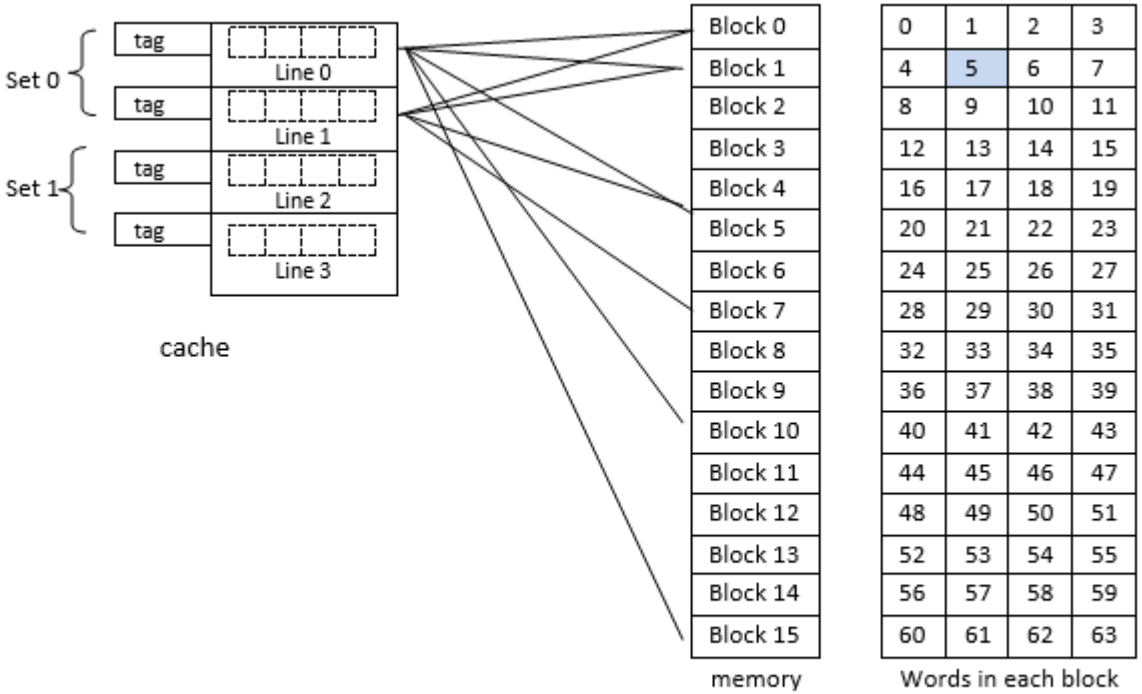


Figure: Set Associative cache mapping

7.4 Cache Replacement Policy:

- The cache replacement policy is the technique for choosing which cache block to replace when Full Associative cache is full, or when a Set-associative cache's line is full.
- It is to be noted that there is no choice in a direct-mapped cache; a main memory address always maps to the same cache address and thus replaces whatever block is already there.
- There are three common replacement policies:
  - Random
  - Least Recently Used
  - First In First Out

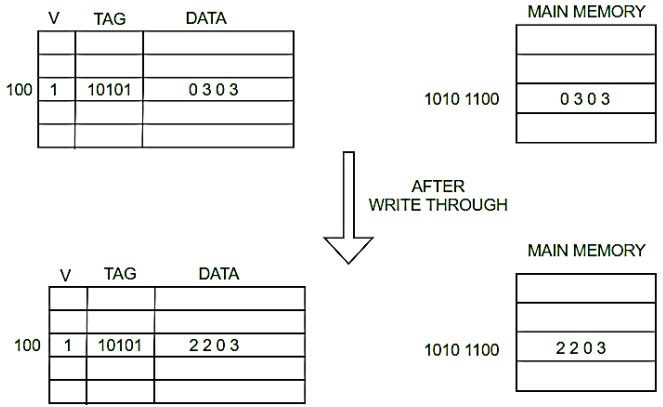
i)	Random	• This policy chooses the block to replace randomly. • It is simple to implement. • However, it does not prevent replacing a block that is likely to be used again soon.
ii)	Least Recently Used (LRU)	• This policy replaces the block that has not been accessed for the longest time, assuming that this means that it is least likely to be accessed in the near future. • This policy provides for an excellent hit/miss ratio but requires expensive hardware to keep track of the times blocks are accessed.
iii)	First In First Out (FIFO)	• Replace that block in the set which has been in the cache longest • Uses a queue of size N, pushing each block address onto the queue when the address is accessed, and then choosing the block to be replaced by popping the queue.

Cache Writing technique/policy:

- Cache is a technique of storing a copy of data temporarily in rapidly accessible storage memory.
- Cache stores most recently used words in small memory to increase the speed at which data is accessed.
- It acts as a buffer between RAM and CPU and thus increases the speed at which data is available to the processor.

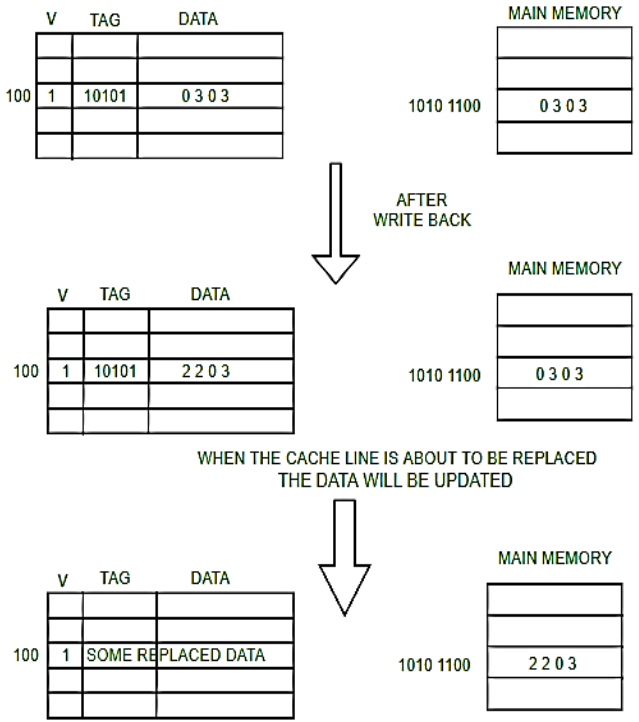
Write Through

- In write-through, data is simultaneously updated to cache and memory.
- This process is simpler and more reliable.
- This is used when there are no frequent writes to the cache (The number of write operations is less).



Write Back:

- The data is updated only in the cache and updated into the memory at a later time.
- Data is updated in the memory only when the cache line is ready to be replaced



Composing Memory:

Cases for Composing Memory:

a) To increase the number of bits per word (or column size), simply connect the available memories side by side as shown in figure.

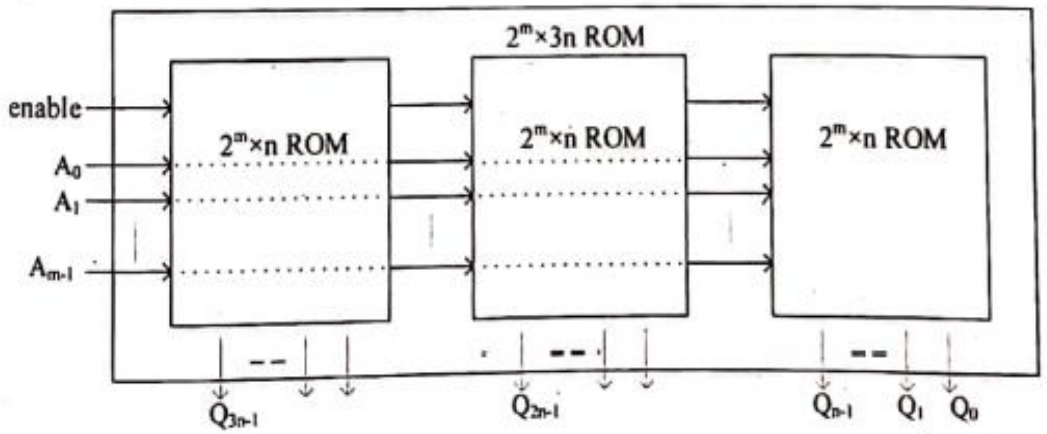


Figure 4.7 a: Increasing width of words

b) To increase the number of words (or row size), simply connect the available memories top to bottom as shown in figure below:

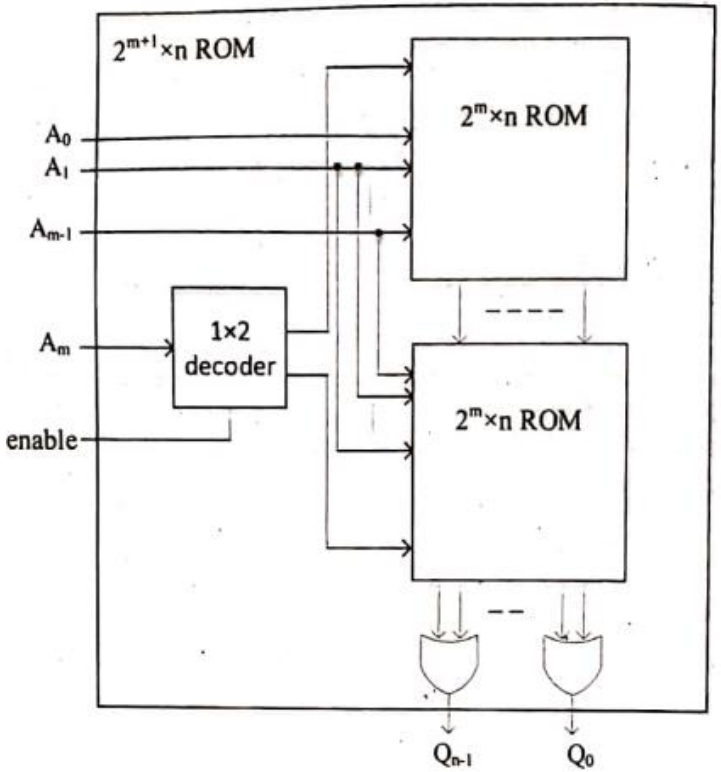


Figure 4.7 b: Increasing number of words

c) If the available memories have a smaller word width as well as fewer words than required, we then combine the above two techniques:

- First creating the number of columns of memories necessary to achieve the needed word width, and
- Then creating the number of columns of memories required to achieve the needed word, and then creating the number of rows of magnitude necessary, along with a decode to achieve the needed number of words as shown in figure below:

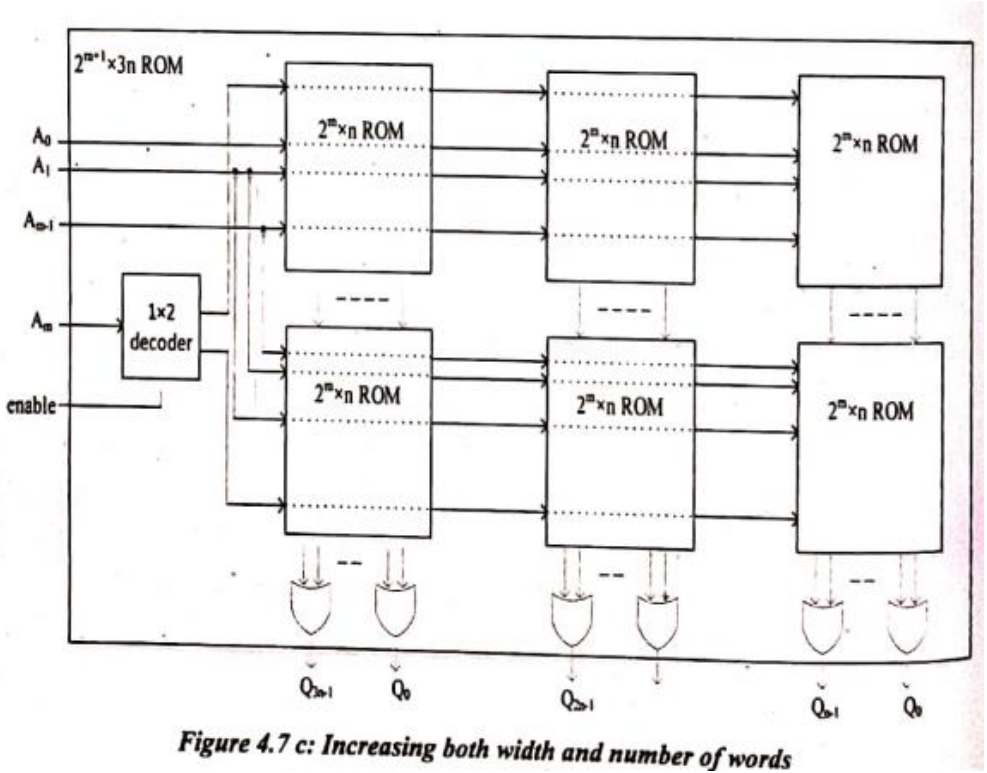


Figure 4.7 c: Increasing both width and number of words

Steps:

1. **Find no or required given ROMS:**

   No of required ROMs = $\frac{Size\ of\ required\ ROM}{size\ of\ given\ ROM}$

2. **Find the no of address lines of required ROM**

   No of address lines or required ROM = $\log_2$ (no of words in required ROM)

3. **Find no of data lines of required ROM**

   No of data lines = no of bits in a word.

4. **If multiplier is different, then align no of required given ROMs horizontally.**
5. **If multiplicand is different, then align no of required given ROMs vertically**
6. **If both are different, then align no of required given ROMs horizontally and vertically (in rows and column).**

1. Compose 1K x 8 ROM into 1K x 32 ROM

Here;

1K x 8 ------> 1K x 32

Now:

- No of required given ROMs = $\frac{Size\ of\ required\ ROM}{size\ of\ given\ ROM} = \frac{1K\ x\ 32}{1K\ x\ 8} = 4$
- No of required address lines = $\log_2$ (no of words in req^d ROM) = $\log_2$ (1024 x 1) =10
- No of required data lines = no of bits in a word = 32
- Here multiplier is different (i.e. 8 Vs 32), we make horizontal alignment of ROMs.

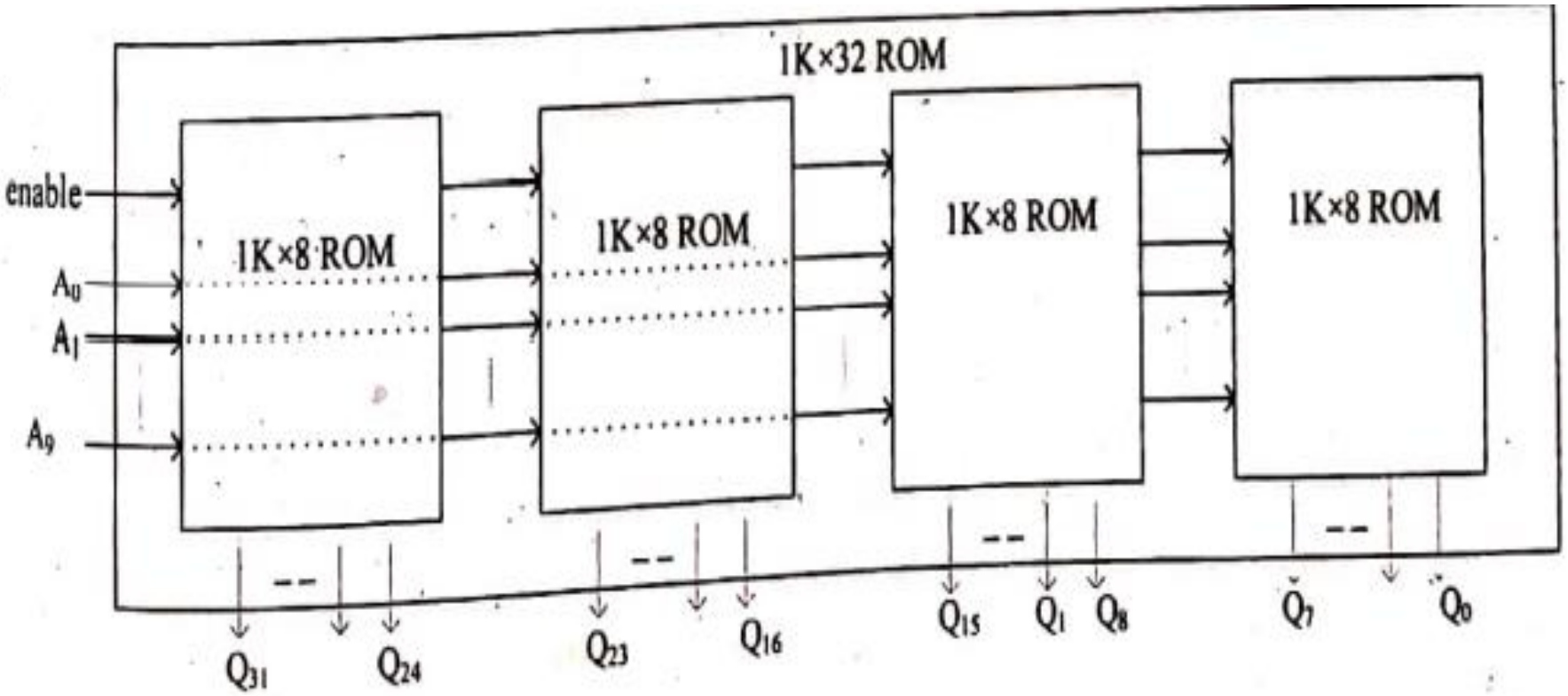


Figure 4.8 b: Composing **1K × 8** *into* **1K × 32** *ROM*

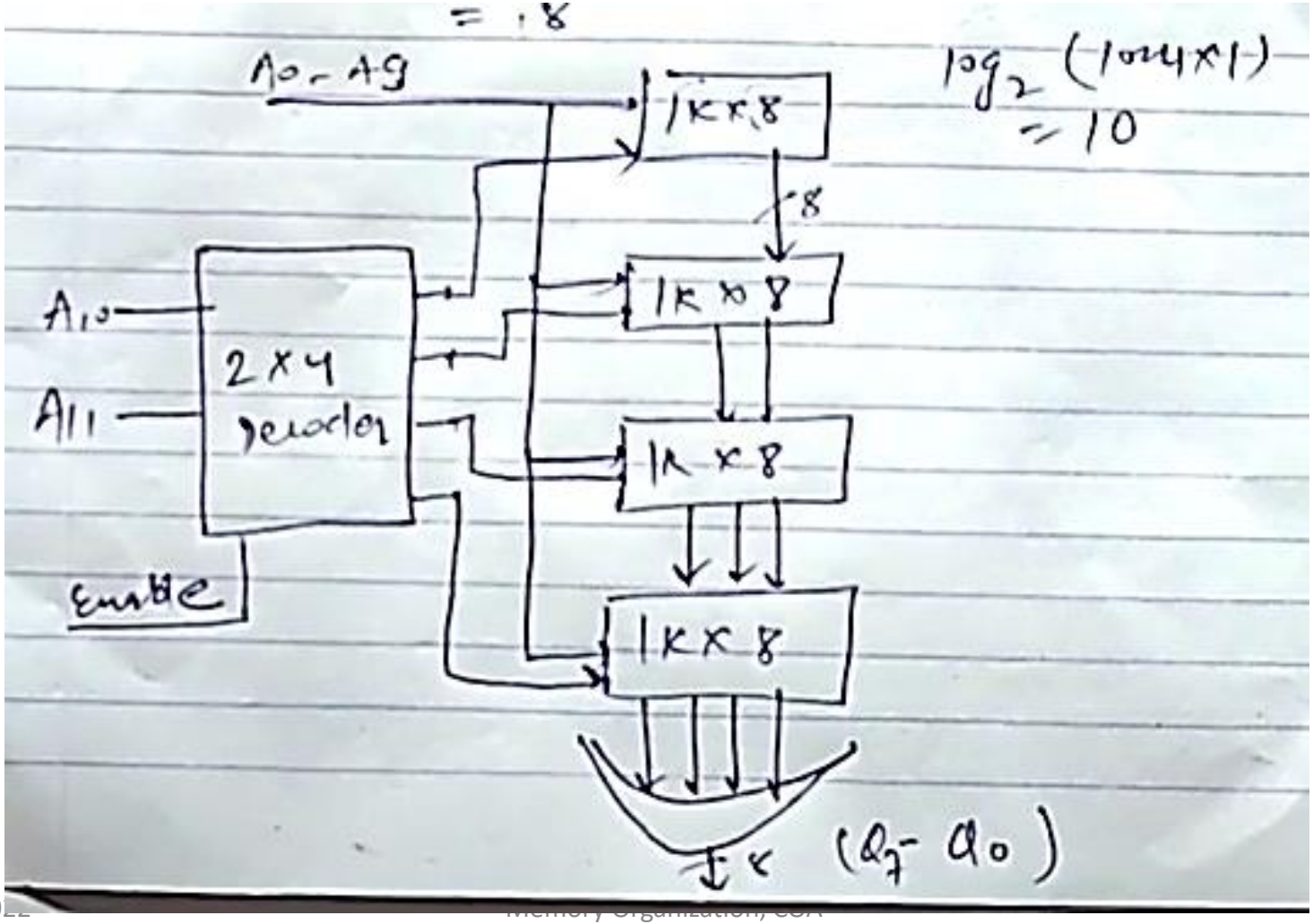
Compose 1K x 8 ROM into 4K x 8 ROM.

Here;

1K x 8 ------> 4K x 8

Now:

- No of required given ROMs = $\frac{Size\ of\ required\ ROM}{size\ of\ given\ ROM} = \frac{4K\ x\ 8}{1K\ x\ 8} = 4$
- No of required address lines = $\log_2$ (no of words in req^d ROM) = $\log_2$ (1024 x 4) =12
- No of required data lines = no of bits in a word = 8
- Here multiplicand is different (i.e. 1K Vs 4K), we make vertical alignment of ROMs.
- Hence, the required memory is:


= 18
A0 - A9
1K x 8
log2 (1024 x 1)
= 10
8
1K x 8
A10
A11
2 x 4
decoder
1K x 8
Enable
1K x 8
(Q7 - Q0)

3. Compose 1K x 8 ROM into 8K x 8 ROM.

Here;

1K x 8 ------> 8K x 8

Now:

- No of required given ROMs = $\frac{Size\ of\ required\ ROM}{size\ of\ given\ ROM} = \frac{8K \times 8}{1K \times 8} = 8$
- No of required address lines = $\log_2$ (no of words in req^d ROM) = $\log_2$ (1024 x 8) =13
- No of required data lines = no of bits in a word = 8
- Here multiplicand is different (i.e. 1K Vs 8K), we make vertical alignment of ROMs.
- Hence, the required memory is:

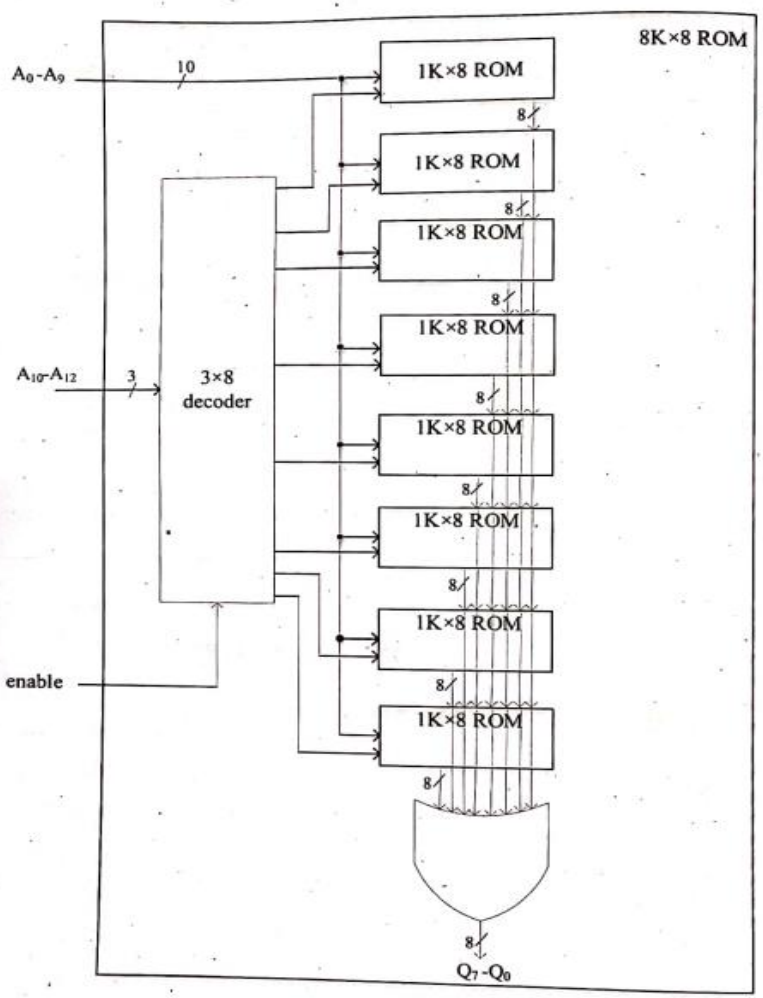


Figure 4.8 c: Composing $1K \times 8$ *into* $8K \times 8$ *ROM*

4. Compose 1K x 8 ROM into 2K x 16 ROM.

Here;

1K x 8 ------> 2K x 16

Now:

- No of required given ROMs = $\frac{Size\ of\ required\ ROM}{size\ of\ given\ ROM} = \frac{2K\ x\ 16}{1K\ x\ 8} = 4$
- No of required address lines = $\log_2$ (no of words in req^d ROM) = $\log_2$ (1024 x 2) =11
- No of required data lines = no of bits in a word = 16
- Here both multiplicand and multiplier are different (i.e. 1K Vs 4K and 8 Vs 16), we make both horizontal and vertical alignment of ROMs.
- No of rows = $\frac{2K}{1K} = 2$
- No of columns = $\frac{16}{8} = 2$
- Hence, the required memory is:

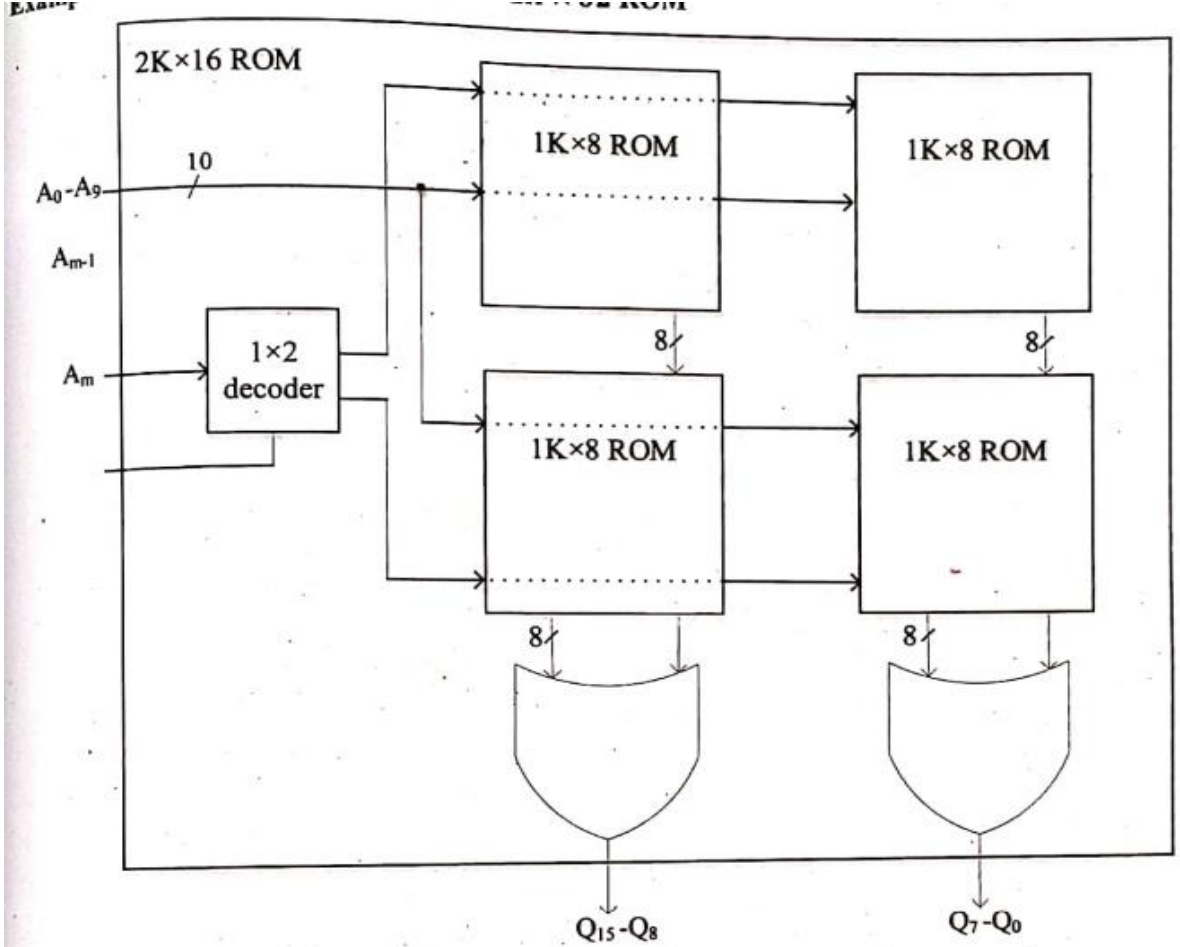


Figure 4.8 a: Composing 1K × 8 into 2K × 16 ROM

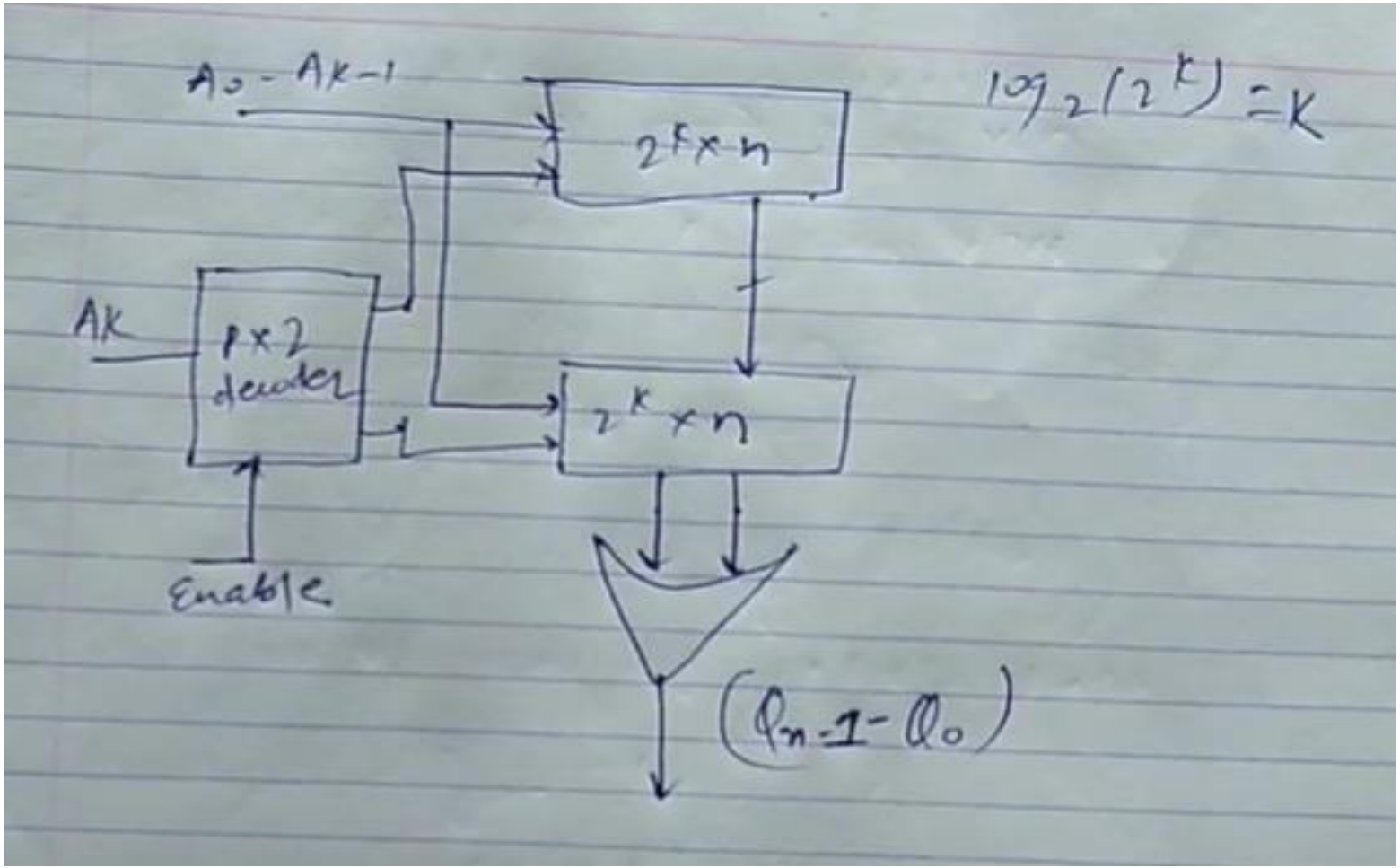
5. Construct $2^{(k+1)}$ x n and $2^{(k)}$ x 4n memories using $2^{(K)}$ x n memory modules.

- Here;
- **First Part:** 2^K x n ------> 2^{k+1} x n

Now:

- No of required given ROMs = $\frac{Size\ of\ required\ ROM}{size\ of\ given\ ROM} = \frac{2^{k+1}\ x\ n}{2^k x\ n} = 2$

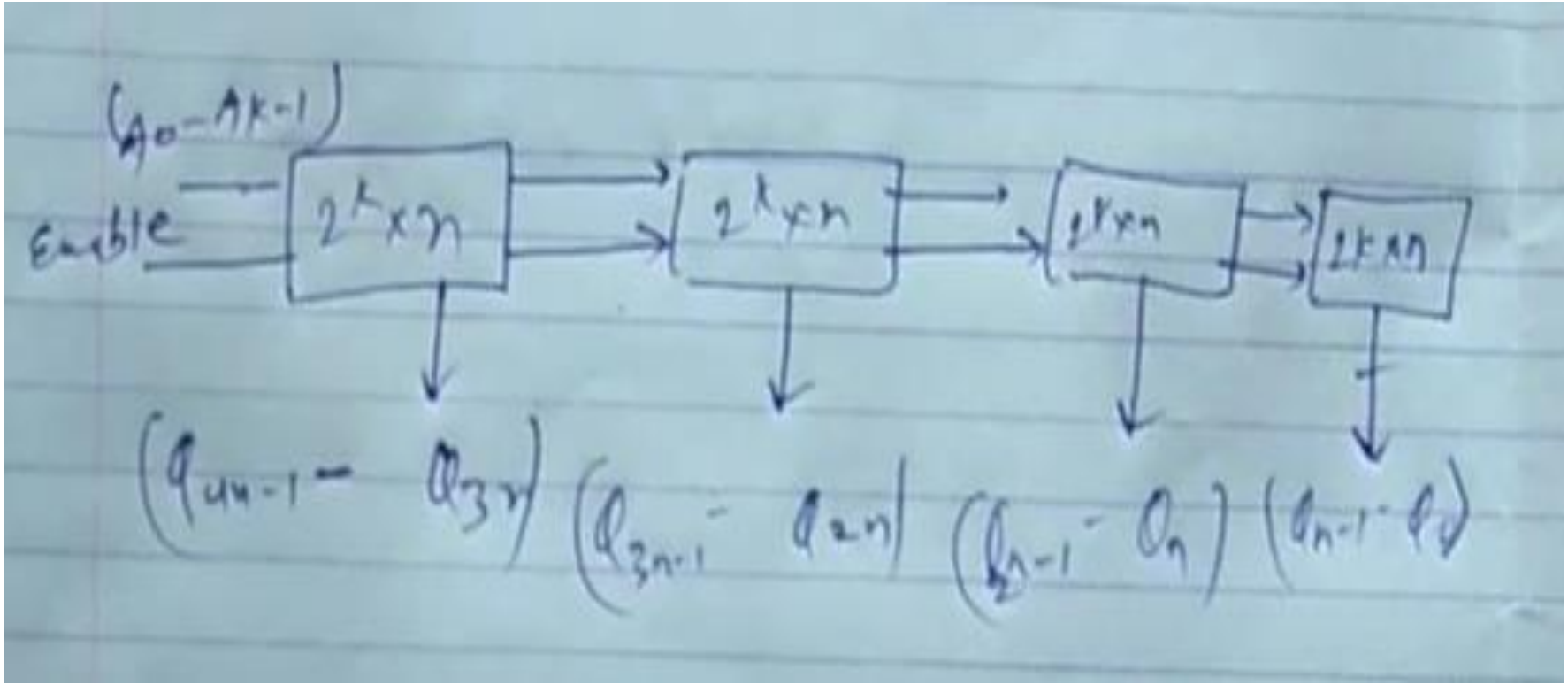
- No of required address lines = $\log_2$ (no of words in req^d ROM) = $\log_2$ (2^{k+1}) =(K+1)
- No of required data lines = no of bits in a word = n
- Here multiplicand is different (i.e. 2^k Vs 2^{k+1}), we make vertical alignment of ROMs.
- Hence, the required memory is:


A0 - AK-1
2^k x n
log2(2^k) = K
AK
1 x 2 decoder
2^k x n
Enable
(Qn-1 - Q0)

- **Second Part:** 2^K x n ------> 2^k x 4n

- No of required given ROMs = $\frac{Size\ of\ required\ ROM}{size\ of\ given\ ROM}$ = $\frac{2k+1\ x\ 16}{1K\ x\ 8}$ = $\frac{2^k \times 4\ n}{2^k \times n}$ = 4

- No of required address lines = $\log_2$ (no of words in req^d ROM) = $\log_2$ (2^k) =k
- No of required data lines = no of bits in a word = 4n
- Here multiplier is different (i.e. n Vs 4n), we make horizontal alignment of ROMs.
- Now required ROM is:


(A0-Ak-1)
Enable
2^k x n
2^k x n
2^k x n
2^k x n
(q4n-1 - q3n)
(q3n-1 - q2n)
(q2-1 - qn)
(qn-1 - q0)

Virtual Memory

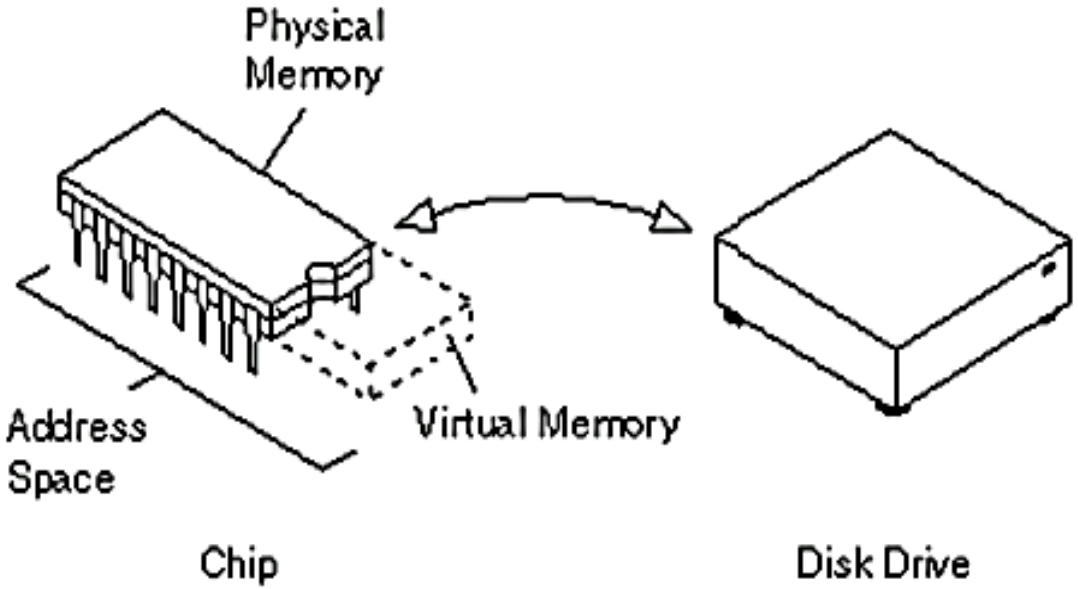


Figure: Conceptual view of virtual memory

Virtual Memory

- Cache is used to keep the most commonly used sections of RAM in the cache, where is can be accessed quickly.
- If we could access RAM at 3 Ghz, there wouldn’t be any need for cache.
- Since RAM cannot keep up the speed, we need to use cache.
- But what if we want more RAM than what we had.
- One way to extend the amount of memory accessible by a program is to use disk.
- In effect, the RAM acts like cache for disk.
- This idea of extending memory is called virtual memory.
- It is called virtual only because it’s not RAM.
- This non-physical memory, which is called virtual memory, is actually a section of a hard disk that is set up to emulate the computer’s RAM.

Virtual Memory

- Hence, virtual memory is a concept to optimize the use of the main memory (the higher speed portion) with hard disk (the lower speed portion).
- It is the technique to extend the apparent limited size of the physical memory beyond its actual size (physical).
- Those parts of program that are currently active are brought to the main memory while those parts that are not active will be stored in the magnetic disk.
- A virtual memory system provides a mechanism for translating program-generated addresses into correct main memory locations.
- This is done dynamically, while programs are being executed in the CPU.
- The translation or mapping is handled automatically by the hardware by means of a mapping table

Virtual Memory

- A virtual memory system provides a mechanism for translating program-generated addresses into correct main memory locations.
- This is done dynamically, while programs are being executed in the CPU.
- The translation or mapping is handled automatically by the hardware by means of a mapping table
- There are two uses of virtual memory:
  - Extension of main memory (RAM)
    - Paging
    - Segmentation
  - Memory protection

Extension of main memory

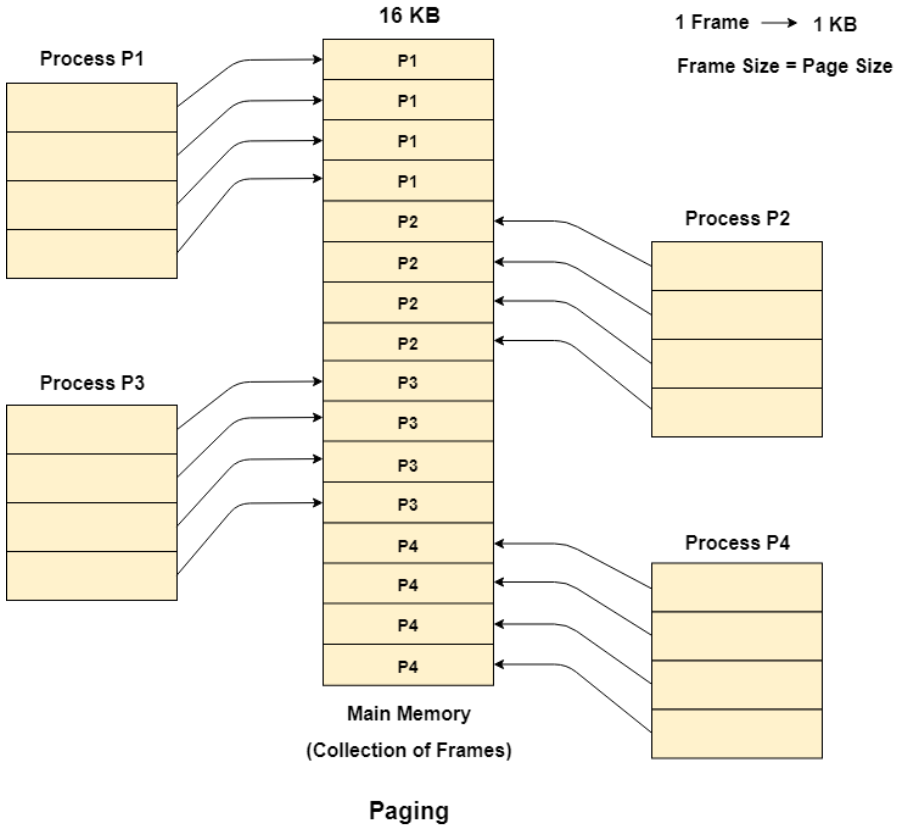
- Since a program actually cannot access and manipulate data within the virtual memory, there are two primary methods for transferring the data from virtual memory to physical:
  - paging and
  - segmentation.

Paging

- Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.
- The main idea behind the paging is to divide each process in the form of **pages.** The main memory will also be divided in the form of **frames.**
- One page of the process is to be stored in one of the frames of the memory.
- The pages can be stored at the different locations of the memory but the priority is always to find the **contiguous frames** or holes.
- Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

Paging

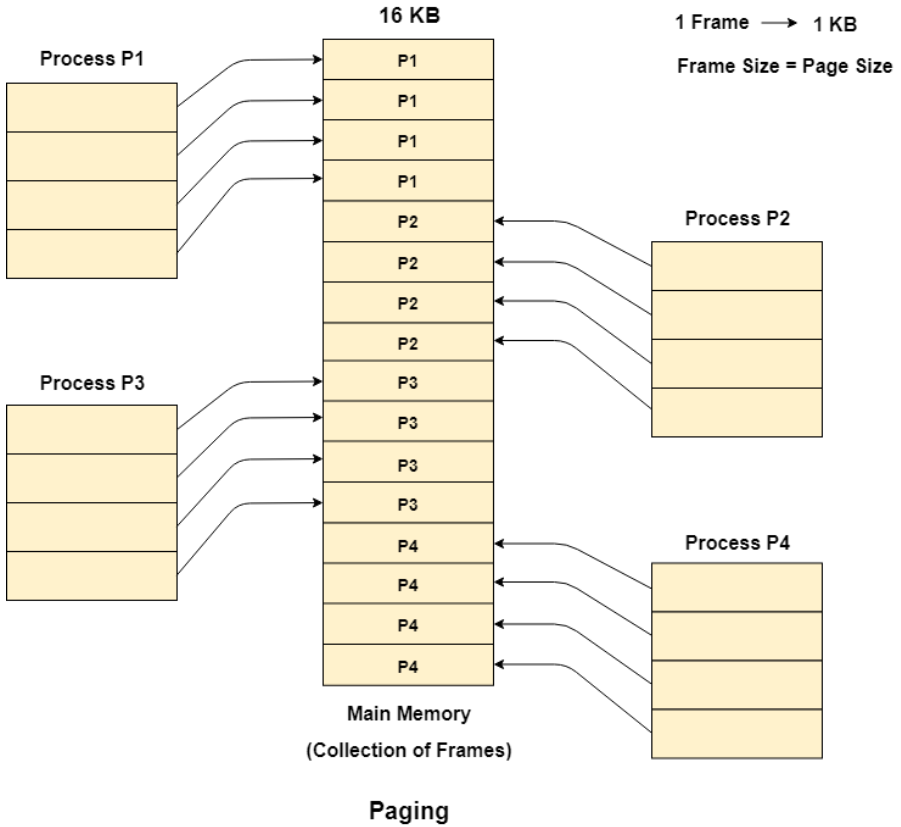
- Different operating system defines different frame sizes.
- The sizes of each frame must be equal.
- Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.



Paging

Paging

- Different operating system defines different frame sizes.
- The sizes of each frame must be equal.
- Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.



Paging

Paging

- **Advantage:**
  - Simple approach, allocating memory is easy and cheap
  - No external fragmentation
  - More efficient swapping
- **Disadvantages:**
  - Longer memory access times
  - Internal fragmentation

Address Mapping using Paging

- For implementing paging, the logical and physical memory space are divided into the same fixed sized blocks that holds processes.
- Physical memory is divided into frames and can store processes in a non-contiguous fashion.
- Logical memory is divided into pages.
- The page table holds the page no (p) and the frame no (f).
- Let us consider, CPU generates a logical address of 1000.
- The first two bits 10 indicates the page no, and the other bits 00 indicates the page-offset.
- As shown in figure, page 2 (ie. 10) is mapped and this page consists of frame 2 (ie. 010).
- The frame no 010 and offset 00 is combined together to map the physical address 01000 in the RAM.
- Thus, the logical address 1000 is converted to physical address 01000. This address 01000 is in the RAM.
- Thus, page mapping is done.

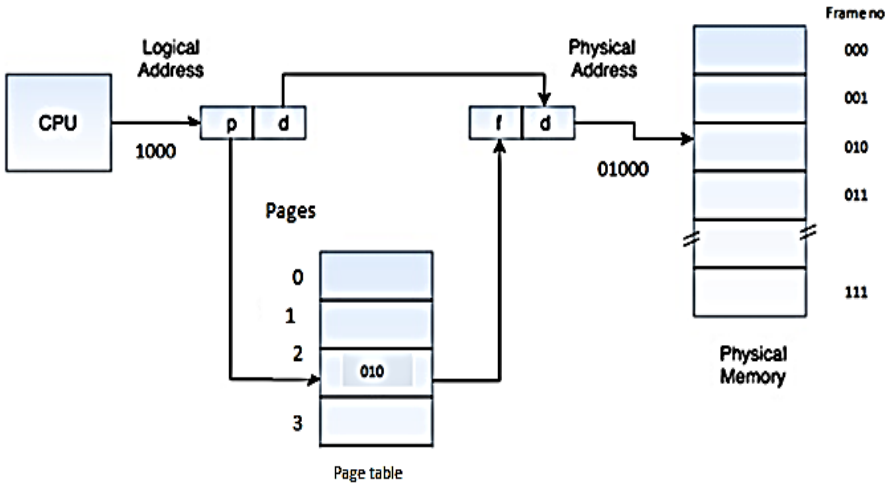


Figure: page mapping

Note: Refer this link for better understanding:
https://www.youtube.com/watch?v=0Rf5Jc61ArM
https://www.youtube.com/watch?v=xAvC-MJ_Sz8
https://www.studytonight.com/operating-system/paging-in-operating-systems

Segmentation

- Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions.
- Each segment is actually a different logical address space of the program
- When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation

- Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.
- This method **uses variable sized memory blocks.**
- This method is useful because it is very common for the size of objects in a program to change dynamically.
- If a single address space is used, as in the paging, after the memory has been allocated it cannot change size, resulting in wasted memory or not enough memory.
- To fix this problem, the computer system would set up many independent address spaces.
- **Each of these address spaces is called a segment.**
- However, unlike paging, the programmer actively maintains the segments.

Segmentation

Advantage:

- No internal fragmentation
- Segment table consumes less space than page table

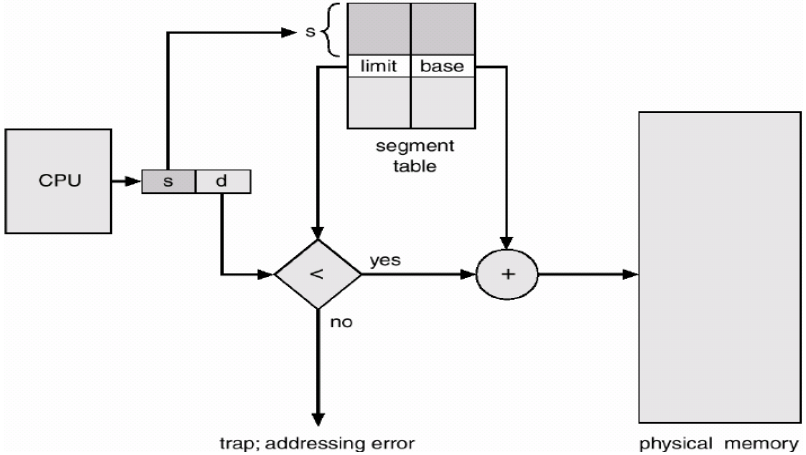
Disadvantage:

- Causes external fragmentation when removed from the memory

Address Mapping using Segmentation

- In case of segmentation, we will be having unequal size partition.
- The segment table consists of limit and base address.
  - Limit indicates the size of each segment. This limit value is stored in the STLR (Segment Table Limit Register).
  - Base address indicates from where the segment is loaded in the main memory. This Base Address value is stored in the STBR (Segment Table Base Register).
- CPU generates the logical address.
- This logical address is having two parts: segment no(s) and displacement value (d).
- The segment part maps the corresponding segment no in the segment table.
- The limit value in that segment no and the displacement value is compared.
- If displacement value is equal to or more than the limit, it is trapped as addressing error and handled appropriately.
- If displacement value is less than the limit, the displacement value and the base address is added to get the physical address.

- Let us consider, CPU generates a logical address 100101 in which **100 is segment no (s)** and **101 is displacement value (d).**
- Consider the segment table holds **limit=1010** and **base address=010** in the segment no 4 (ie 100)
- Now is 101 < 1010? It is Yes.
- Then BAV + Disp = 101 + 010 = 111
- So the physical address is 111 in the main memory.



Note: Refer this link for better understanding in memory segmentation:
https://www.youtube.com/watch?v=8_dDcNzogSg
https://www.youtube.com/watch?v=E7XQesMS0bg
*https://www.youtube.com/watch?v=EGzpLXKc_*

*BAV=Base Address Value

Memory Protection

- A multitasking operating system may have several different programs, or parts of programs, resident in memory simultaneously.
- These programs components may belong to one user or to different users.
- The OS may also have components resident in memory.
- Some mechanism is needed to ensure that one component does not overwrite other.
- No computer can read data from another that it should not be able to access.
- In addition, a component can be protected from its owner by restricting its mode of access. All of these fall under memory protection.
- The main purpose of memory protection is to prevent a process from accessing memory that has not been allocated to it.

Memory Protection

- Memory protection includes:
  - No other component overwrite other.
  - No component can read data from other.
  - Protection of component by restricting its modes of access.

Memory Protection

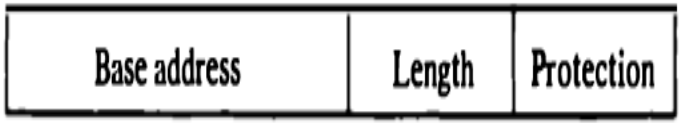
- How memory protection done?
  - Protect segment or page: Protection bits are added to each entry in segment or page table which signify segment is read only or write.
  - Keeping tack of segments and pages which are associated with users so that one user cannot read or overwrite another's data.

Memory Protection

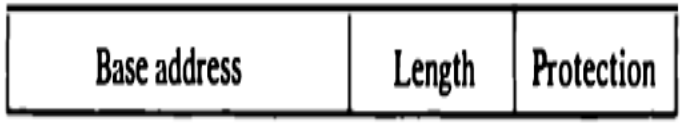
- Memory protection can be assigned to the physical address or the logical address.
- The protection of memory through the physical address can be done by using protection bits.
- Every time a page is moved from one block to another it would be necessary to update the block protection bits.
- A much better place to apply protection is in the logical address space rather than the physical address space.
- This can be done by including protection information within the segment table or segment register of the memory management hardware.
- The content of each entry in the segment table or a segment register is called a descriptor.

Memory Protection

- The content of each entry in the segment table or a segment register is called a descriptor.
- A typical descriptor would contain, in addition to a base address field, one or two additional fields for protection purposes.
- A typical format for a segment descriptor is shown in figure below:



Memory Protection



- The base address field is the address used in mapping from a logical to the physical address.
- The length field gives the segment size by specifying the maximum number of pages assigned to the segment. The length field is compared against the page number in the logical address. A size violation occurs if the page number falls outside the segment length boundary. Thus, a given program and its data cannot access memory not assigned to it by the operating system.
- The protection field in a segment descriptor specifies the access rights available to the particular segment such as:
  - Full Read and Write privileges
  - Read Only
  - Execute Only
  - System Only

Assignment:

- Differentiate between Segmentation and Paging.

Exam Questions:

1. What is cache performance? Describe in brief the strategies used to write data to the cache memory. [PU 2017 Fall]
2. What is memory hierarchy? Differentiate between different types of cache mapping. [PU 2017 Spring]
3. What is Memory hierarchy? Describe the significance of Cache and Virtual Memory. [PU 2018 Fall]
4. What is virtual memory? Differentiate between paging and segmentation. [PU 2018 Spring]
5. What is cache memory? Briefly explain Direct and Set-associative mapping. [PU 2019 Spring]
6. What is cache memory? Explain associative, set associative and direct mapping in cache. [PU 2020 Spring]

End of Chapter