UNIT 6 ADVANCE MEMORY ORGANISATION

Structure **Page Nos.**

6.0 INTRODUCTION

In the last unit, the concept of Memory hierarchy was discussed. The Unit also discussed different types of memories including RAM, ROM, flash memory, secondary storage technologies etc. The memory system of a computer uses variety of memories for program execution. These memories vary in size, access speed, cost and type, such as volatility (volatile/ non-volatile), read only or read-write memories etc. As you know, a program is loaded in to the main memory for execution. Thus, the size and speed of the main memory affects the performance of a computer system. This unit will introduce you to concepts of cache memory, which is small memory between the processing unit and main memory. Cache memory enhances the performance of a computer system. Interleaved memory and associative memories are also used as faster memories. Finally, the unit discusses the concept of virtual memory, which allows programs larger than the physical memory.

ignou THE PEOPLE'S UNIVERSITY

6.1 OBJECTIVES

After going through this Unit, you will be able to:

- explain the concept of locality of reference;
- explain the different cache organisation schemes;
- explain the characteristics of interleaved and associative memories;
- explain the concept of virtual memory.

6.2 LOCALITY OF REFERENCE

Memory system is one of the important component of a computer. A program is loaded in to the main memory for execution. Therefore, a computer should have a main memory, which should be as fast as its processor and should have large size. In general, the main memory is constructed using DRAM technology which is about 50 to 100 times slower than the processor speed. This may slow down the process of instruction execution of a computer. Using SRAM may change this situation as it is almost as fast as a processor, however, it is a costly memory. So, what can you do? Is it possible to use large main memory as DRAM, but use a faster small memory between processor and main memory? Will such a configuration enhance performance of a computer? This section will try to answer these questions.

The important task of a computer is to execute instructions. It has been observed that on an average 80-85 percent of the execution time is spent by the processor in accessing the instruction or data from the main memory. The situation becomes even worst when instruction to be executed or data to be processed is not present in the main memory.

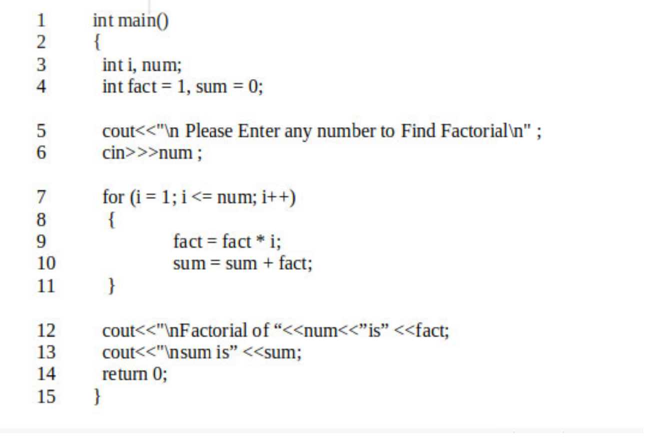
Another factor which has been observed by analysing various programs is that during the program execution, the processor tends to access a section of the program instructions or data for a specific time period. For example, when a program enters in a loop structure, it continues to access and execute loop statements as long as the looping condition is satisfied. Similarly, whenever a program calls a subroutine, the subroutine statements are going to execute. In another case, when a data item stored in an array or array like structure is accessed then it is very likely that either next data item or previous data item will be accessed by the processor. All these phenomenons are known as *Locality of Reference* or *Principle of Locality*.

So, according to the principle of locality, for a specific time period, the processor tends to make memory references closed to each other or accesses the same memory addresses again and again. The earlier type is known as *spatial locality*. Spatial locality specifies if a data item is accessed then data item stored in a nearby location to the data item just accessed may be accessed in near future. There can be special case of spatial locality, which is termed as sequence locality. Consider a program accesses the elements of a single dimensional array, which is a linear data structure, in the sequence of its index. Such accesses will read/write on a sequence of memory locations one after the other. This type of locality, which is a case of spatial locality, is referred to as sequence locality.

Another type of locality is the *temporal locality*, if a data item is accessed or referenced at a particular time, then the same data item is expected to be accessed for some time in near future. Typically it is observed in loop structures and subroutine call.

As shown in Figure 6.1, when the program enters in the loop structure at line 7, it will execute the loop statements again and again multiple times till the loop terminates. In this case, processor needs to access instructions 9 and 10 frequently. On the other hand, when a program accesses a data item store in an array, then in the next iteration it accesses a data item stored in an adjacent memory location to the previous one.

THE PEOPLE'S UNIVERSITY



```
1     int main()
2     {
3       int i, num;
4       int fact = 1, sum = 0;

5       cout<<"\n Please Enter any number to Find Factorial\n" ;
6       cin>>>num ;

7       for (i = 1; i <= num; i++)
8        {
9               fact = fact * i;
10              sum = sum + fact;
11       }

12      cout<<"\nFactorial of “<<num<<”is” <<fact;
13      cout<<"\nsum is” <<sum;
14      return 0;
15    }
```

Figure 6.1: Loop structure

The locality of reference, be it spatial or temporal, suggests that in most cases accesses to a program instruction and data confines to a locality, hence, a very fast memory that captures the instructions and data nearer to the current instructions and data accesses can potentially enhance the overall performance of a computer. Thus, attempts are continuously made to utilize the precious time of the processor efficiently A high speed memory, called cache memory, was developed. Cache memory utilises the principle of locality to reduce the memory references to the main memory by keeping not only the currently referenced data item but also the nearby data items. The cache memory and its organisation is discussed in the next sections.

6.3 CACHE MEMORY

A processor makes many memory references to execute an instruction. A memory reference to the main memory is time consuming as main memory is slower compared to the processing speed of the processor. These memory references, in general, tends to form a cluster in the memory, whether it is a loop structure, execution of a subroutine or an access to a data item stores in an array.

If you keep the content of the cluster of expected memory references in a small, extremely fast memory then processing time of an instruction can be reduced by a significant amount. Cache memory is a very high speed and expensive memory as compared to the main memory and its access time is closer to the processing speed of the processor. Cache memory act as a buffer memory between the processor and the main memory.

Because cache is an expensive memory so its size in a computer system is also very small as compared to the main memory. Thus, cache stores only those memory clusters containing data/ instructions, which have been just accessed or going to be accessed in near future. Data in the cache is updated based on the principle of locality explained in the previous section.

How data access time is reduced significantly by using cache memory?

Data in main memory is stored in the form of fixed size blocks/pages. Cache memory contains some blocks of the main memory. When processor wants to read a data item from the main memory, a check is made in the cache whether data item to be accessed is present in the cache or not. If data item to be accessed is present in the cache then it is read by the processor from the cache. If data item is not found in the cache, a memory reference is made to read the data item from the main memory, and a copy of the block containing data item is also copied into the cache for near future references as explained by the principle of locality. So, whenever processor attempts to read the data item next time, it is likely that the data item is found in the cache and saves the time of memory reference to the main memory.

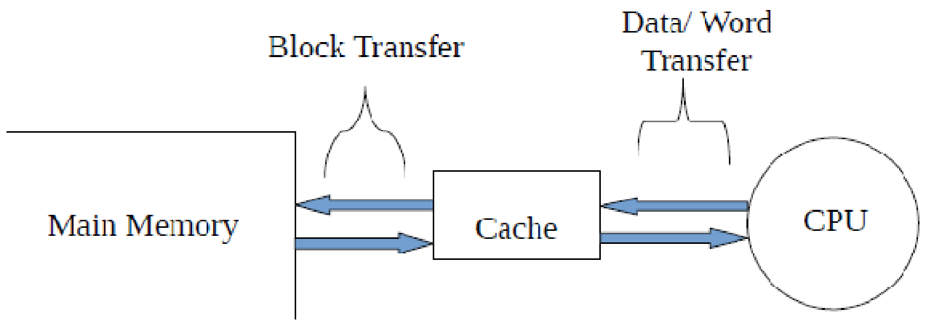


Figure 6.2: Cache Memory

As shown in the Figure 6.2, if requested data item is found in the cache it is called as ***cache hit*** and data item will be read by the processor from the cache. And if requested data item is not found in cache, called ***cache miss***, then a reference to the main memory is made and requested data item is read and block containing data item will also be copied into the cache.

Average access time for any data item is reduced significantly by using cache then without using cache. For example, if a memory reference takes 200 ns and cache takes 20 ns to read a data item. Then for five continuous references will take:

Time taken with cache : 20 (for cache miss) + 200 (memory reference)
+ 4 x 20 (cache hit for subsequent access)
= 300 ns

Time without cache : 5 x 200 = 1000 ns

In the given example, the system first looks into the cache for the requested data item. As it is the first reference to the data item it will not be present in the cache, called as cache miss, and thus, requested data item will be read from the main memory. For subsequent requests of the same data item, the data item will be read from the cache only and no references will be made to the main memory as long as the requested data remains in the cache.

Effective access time is defined as the average access time of memory access, when a cache is used. The access time of memory access is reduced in case of a cache hit, whereas it increases in case of cache miss. In the above mentioned example processor takes 20 + 200 ns for a cache miss, whereas it takes only 20 ns for each cache hit. Now suppose, we have a hit ratio of 80%, i.e. 80 percent of times a data item would be found in the cache and 20 % of the times it would be accessed from the main memory. So effective access time (EAT) will be computed as :

effective access time = (cache hit x data access time from cache only)
+(cache miss x data access time from cache and main memory)

effective access time = 0.8 (hit ratio) x 20 (cache hit time)
+ 0.2(miss ratio) x 220 (cache miss and memory reference)

effective access time = 0.8 x 20 + 0.2 x 220
= 16 + 44
= 60 ns

From the example it is clear that cache reduces the average access time and effective access time for a data item significantly and enhance the computer performance.

Check Your Progress 1

1. What is the importance of locality of reference?

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

2. What is block size of main memory for cache transfer?

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

3. Hit ration of computer system is 90%. The cache has an access time of 10ns, whereas the main memory has an access time of 50ns. Computer the effective access time for the system.

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

6.4 CACHE ORGANISATION

The main objective of using cache is to decrease the number of memory references to a significant level by keeping the frequently accessible data/ instruction in the cache. Higher the hit ratio (number of times requested data item found in cache / total number of times data item is requested), lower would be the references to the main memory. So there are number of questions that need to be answered while designing the cache memory. These Cache design issues are discussed in the next subsection.

6.4.1 Issues of Cache Design

In this section, we present some of the basic questions that should be asked for designing cache memory.

What should be the size of cache memory?

Cache is an extremely fast but very expensive memory as compared to the main memory. So large cache memory may shoot up the cost of the computer system and too small cache might not be very useful in real time. So, based on various statistical analyses, if a computer system has 4 GB of main memory then the size of the cache may go up to 1MB.

What would be the block size for data transfer between cache and main memory?

Block size directly affects the cache performance. Higher block size would ensure only fewer number of blocks in cache, whereas small block size contains fewer data items. As you increase the block size, the hit ratio first increases but it decreases as you further increase the block size. Further increase in block size will not necessarily result in access of newer data items, as probability of accessing data items in the block with larger number of data items tends to decrease. So, optimal size of the block should be chosen to maximise the hit ratio.

How blocks are going to be replaced in cache?

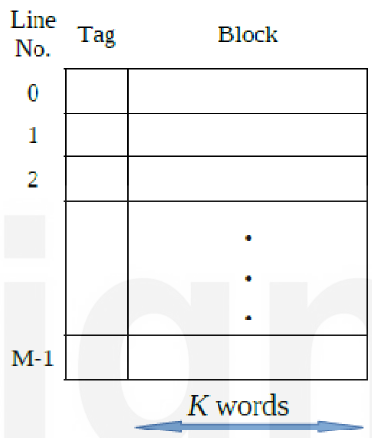
As execution of the process continues, the processor requests for new data items. For new data items and thus, new blocks to be present in the cache, the blocks containing old data items must be replaced. So there must be a mechanism which may select the block to be replaced which is least likely to be needed in near future.

When changes in the blocks will be written back on to the main memory?

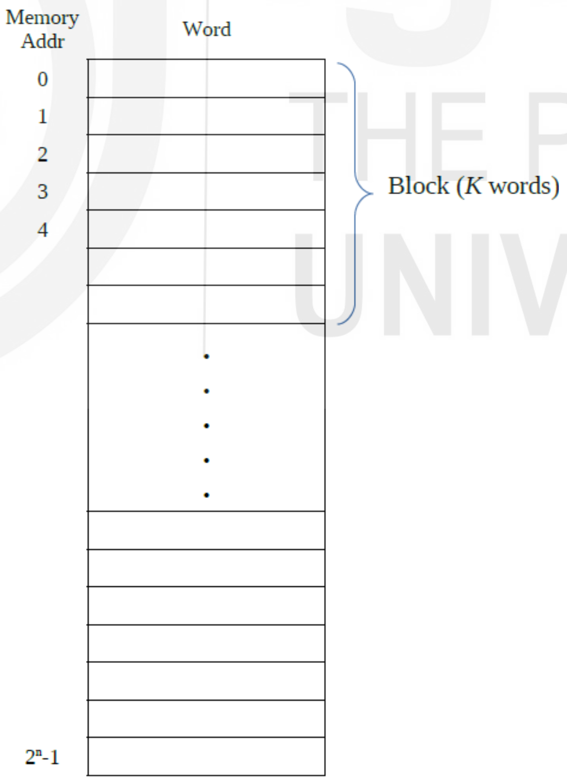
During the program execution, the value of a data item in a cache block may get changed. So the changed block must be written back to the main memory in order to reflect those changes to ensure data consistency. So there must be a policy, which may decide when the changed cache block is written back to the main memory.

In certain computer organisations, the cache memory for data and instruction are placed separately. This results in separate address spaces for the instructions and data. These separate caches for instructions and data are known as **instruction cache** and **data cache** respectively. If processor requests an instruction, then it is provided by the instruction cache, whereas requested data item is provided by the data cache. Using separate cache memories for instruction and data enhances computer performance. While some computer systems implements different cache memories for data and instructions other implements multiple level of cache memories. Two level cache popularly known as **L1 cache** and **L2 cache** is most commonly used. Size of level 1 cache or L1 cache is smaller than the level 2 or L2 cache. Comparatively more frequently used data/ instructions are stored in L1 cache.

As discussed earlier, the main memory is divided into blocks/ frames/ pages of *k words* each. Each word of the memory unit has a unique address. A processor requests for read/write of a memory word. When a processor's request of a data item cannot be serviced by cache memory, i.e. a *cache miss* occurs, the block containing requested data item is read from the main memory and a copy of the same is stored in cache memory. A cache memory is organised as a sequence of line. Each cache line is identified by a cache line number. A cache line stores a tag and a block of data. Cache and main memory structure is shown in Figure 6.3. General structure of cache memory having M lines and $N=2^n$ main memory size is shown in figure 6.3(a) and figure 6.3(b) respectively.



(a) Cache structure



(b) Main Memory structure

Figure 6.3: Structure of Cache and Main Memory

An example of cache memory of size 512 words is shown in Figure 6.4. The example shown in Figure 6.4 has a main memory of 64 K words of 16 bits each and cache

ignou THE PEOPLE'S UNIVERSITY

memory can have 512 words of 16 bits each. To read a data item processor sends a 16 bit address to the cache and if cache misses then the data item/ word is fetched from the main memory and accessed data item/ word is also copied into the cache. Please note that the size of block is just 1 memory word in this case.

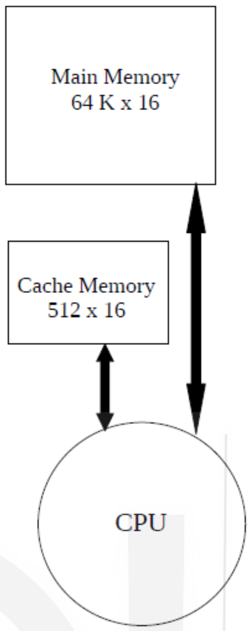


Figure 6.4: Example of Cache and Main Memory

6.4.2 Cache Mapping

As discussed earlier, a request of processor to access a main memory is to checked in cache memory. Where can a block of main memory be places in cache and how processor can determine, if the data requested is present in cache? Answer to these questions are provided by cache mapping scheme. A mapping mechanism maps the block from the main memory to a cache line. The mapping is required as cache is much smaller in size than the size of the main memory. So only few blocks from the main memory can be stored in cache. There are three types of mapping in cache:

Direct Mapping:

Direct mapping is the simplest amongst all three mapping schemes. It maps each block of main memory into only one possible cache line. Direct mapping can be expresses as a *modulo M* function, where *M* is the total number of cache lines as shown:

$$i = j \; modulo \; M$$

where, i = number of cache line to which main memory block would be mapped.

j = the block address of main memory, which is being requested

M = total number of cache lines

So, line 0 of the cache will store block 0, M, 2M….. and so on. Similarly, line 1 of cache will store block 1, M + 1, 2M + 1, and so on.

An address of main memory word, as shown in Figure 6.3(b), consists of *n bits*. This address of each word of main memory has two parts: block number (*n-k bits*) and word number within the block (*k bits*). Here, each block of the main memory contains 2^k number of words. The cache memory interprets *n-k bit* block number in to two parts as: tag and line number. As indicated in Figure 6.3(a), the cache memory contains M lines. Assuming *m* address bits (2^m = M) are used to identify each line, then most significant *(n-k)* - *m* bits of *(n-k) bit* block number are interpreted as tag and *m* bits are used as line number of the cache. Please note the tag bits are used to identify, which of the main memory block is presently in that cache line. The following diagram summarizes the bits of main memory address and related cache address:

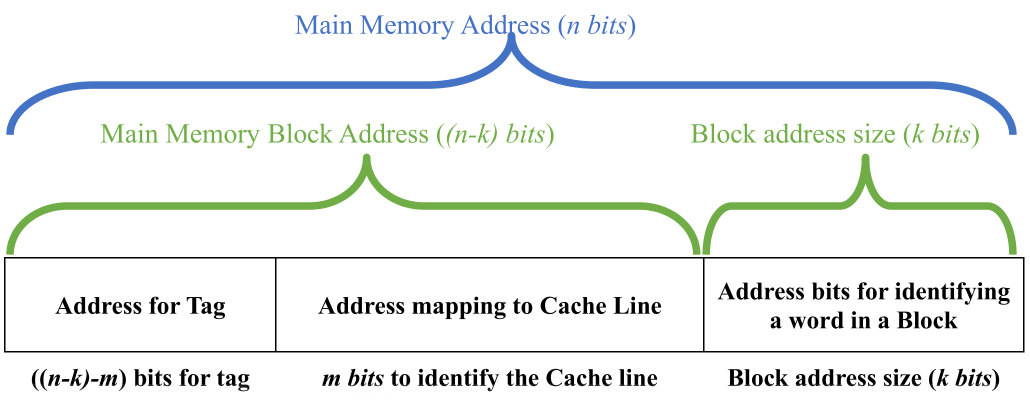

Main Memory Address (*n bits*)
Main Memory Block Address (*(n-k) bits*)
Block address size (*k bits*)
Address for Tag
Address mapping to Cache Line
Address bits for identifying a word in a Block
((*n-k*)-*m*) bits for tag
***m bits* to identify the Cache line**
Block address size (*k bits*)

Figure 6.5: Main Memory address to Cache address mapping in Direct mapping Scheme

Please note the following points in the diagram given above.

The size of Main Memory address: n bits

Total number of words in main memory: 2^n

The size of a Main memory block address: most significant (n-k) bits

Number of words in each block: 2^k

In case, a referenced memory address is in cache, bits in the tag field of the main memory address should match the tag filed of the cache.

***Example*: ** Let us consider a main memory of 16 MB having a word size of a byte and a block size is of 128 bits or 16 words (one word is one byte in this case). The cache memory can store 64 KB of data. Determine the following:

a) Size of various addresses and fields

c) how will you determine that a hexadecimal address is in cache memory, assuming a direct mapping cache is used?

Solution:

a) Size of main memory= 16MB = 2^{24} Bytes

Each block consists of 16 words, thus total number of blocks in main memory would be = $2^{24}/16 = 2^{20}$ blocks, thus $n = 24$ and $k = 4$ (as 2^4=16). Therefore, main memory block address $(n\text{-}r) = 20$

Data size of cache is 64 KB = 2^{16} Bytes

Total number of cache lines (M) = cache size/ block size = $2^{16} / 2^4 = 2^{12}$, therefore, number of cache lines = 2^{12} and m = 12

Length of address field to identify a word in a Block (k) = 4 bits

Length of address to identify a Cache line (m) = 12 bits

Length of Tag field = (24- 4) - 12 = 8 bits.

Thus, a main memory address to cache address mapping for the given example would look like this:

Main Memory Address n = 24 bits		
Address of a Block of data = 20 bits		***k=4 bits***
Address mapping for direct cache mapping scheme		
Tag = 8 bits	***Cache line number address m = 12 bits***	***k=4 bits***

Figure 6.6: Direct Cache mapping

ignou THE PEOPLE'S UNIVERSITY

b) Consider a 24 bit main memory address in hexadecimal as FEDCBA. The following diagram will help in identifying, if this address is in the cache memory or not in case direct mapping scheme is used.

Main Memory Address n = 24 bits = 6 hex digits					
F	E	D	C	B	A
Address of a Block of data = 20 bits					***k=4 bits***
F E D C B					A
Address mapping for direct cache mapping scheme					
Tag = 8 bits		***Cache line number address m = 12 bits***			***k=4 bits***
F E		D C B			A
1111 1110		1101 1100 1011			1010

Figure 6.7: Direct Cache mapping example

Now, the following steps will be taken by the processing logic of processing unit and hardware of Cache memory:

1. The tag number (FE in this case) is compared against the Tag number of data stored in the cache line (DCB in this case).
2. In case both are identical
   then (this is the case of cache hit): Ath word from the cache line DCB is accessed by the processing logic.
   else (this is a case of cache miss): The cache line 16 words data is read to cache memory line (DCB) and its tag number is now FE. The required Ath word is now accessed by the processing logic

Direct mapping is very easy to implement but has a disadvantage as location in which a specific block is to be stored in cache is fixed. This arrangement leads to low hit ratio as when processor wants to read two data items belongs to two different blocks, which map to single cache location, then each time other data item is requested, the block in the cache must be replaced by the requested one. This phenomenon is also known as *thrashing.*

Associative Mapping:

Associative mapping is the most flexible mapping in cache organisation as it allows to store any block of the main memory in any of the cache line/or location. It uses complete (*n-k*) *bits* of block address field as a tag field. Cache memory stores (*n-k*) *bits* of Tag and (2^k × Word Size in bit) data. When a data item/ word is requested, (*n-k*) *bit* tag field is used by the cache control logic to search the all the tag fields stored in the cache simultaneously. If there is a match (cache hit) then corresponding data item is read from the cache, otherwise (cache miss) the block of data that contains the word to be accessed is read from the main memory. It replaces any of the cache line. In addition, the block address of the accessed block from the main memory replaces the tag of the cache line. It is also the fastest mapping amongst all types. Different block replacement policies are used for replacing the existing cache content by newly read data, however, those are beyond the scope of this unit. This mapping requires most complex circuitry, as it requires all the cache tags to be checked simultaneously with the block address of the access request.

Main Memory Address :

Address of a block of data is same as Tag	**Address bits for identifying a word in a Block**
(n-k) bits	*k bits*

Every line of Associative Cache has the following format:

ignou THE PEOPLE'S UNIVERSITY

Tag	Data Block of *k* words
(n-k) bits	Data bits

Figure 6.8: Associative mapping

The following example explains the set associative mapping.

***Example*: ** Let us consider a main memory of 16 MB having a word size of a byte and a block size is of 128 bits or 16 words (one word is one byte in this case). The cache memory can store 64 KB of data. Determine the following size of various addresses and fields, if associative mapping is used

Solution:

Size of main memory= 16MB = 2^{24} Bytes

Each block consists of 16 words, thus total number of blocks in main memory would be = $2^{24}/16 = 2^{20}$ blocks, thus $n = 24$ and $k = 4$ (as 2^4=16). Therefore, main memory block address (n-r) = 20

Data size of cache is 64 KB = 2^{16} Bytes

Total number of cache lines (M) = cache size/ block size = $2^{16} / 2^4 = 2^{12}$, therefore, number of cache lines = 2^{12} and m = 12

Length of Tag field = (24- 4) = 20 bits.

Size of data = $2^k \times$ Word Size in bit = $2^4 \times 8$ = 128 bits.

Thus, size of one line of cache = 128+20=148 bits.

Set Associative Mapping:

The major disadvantage of direct mapping is that location of the cache line onto which a memory block is going to be mapped is fixed which results in poor hit ratio and unused cache locations. The associative mapping removes these hurdles and any block of memory can be stored anywhere in cache location. But associative cache uses complex matching circuit and big tag size. Set Associative mapping reduces the disadvantages of both the above mentioned cache mapping techniques and is built on their strengths. In set associative mapping scheme, cache memory is divided into *v sets* where each set contains *w cache lines*. So, total number of cache lines *M* is given as:

$$M = v\ x\ w$$

where v is the number of sets and w is the number of cache lines in v

The cache mapping is done using the formula:

$$i = j\ modulo\ \ v$$

where *i* is the set number and *j* is the block address of word to be accessed.

Cache control logic interprets the address field as a combination of *tag* and *set* fields as shown:

Tag	Set	Word
((n-k)-d) bits	*d bits*	*k bits*

Figure 6.8: Set Associative mapping

ignou THE PEOPLE'S UNIVERSITY

Cache mapping logic uses *d-bits* to identify the set as $v = 2^d$ and ((*n-k*)-*d*)) *bits* are used to represent the *tag field.* In set-associative mapping, a *block j* can be stored at any of the cache line of *set i.* To read a data item, the cache control logic first simultaneously looks into all the cache lines using ((*n-k*)-*d*)) *bits* of *tag field* of the *set* identified by *d-bits* of the *set field,* otherwise a data item is read from the main memory and corresponding data is copied into the cache accordingly. Set associative mapping is also known as *w*-way set-associative mapping. It uses lesser number of bits (((*n-k*)-*d*) *bits*) as compare to (*n-k*) *bits* in associative mapping in *tag field*.
A comprehensive example showing possible locations of main memory blocks in Cache for different cache mapping schemes is discussed next.

Example: Assume a main memory of a computer consists of 256 bytes, with each memory word of one byte. The memory has a block size of 4 words. This system has a cache which can store 32 Byte data. Show how main memory content be mapped to cache if (i) Direct mapping (ii) Associative mapping and (iii) 2 way set associative memory is used.
Solution:

Main memory size = 256 words (a word = one byte) = 2^8 ⇒ n=8 bits
Block Size = 4 words = 2^2 ⇒ k=2 bits

The visual representation of this main memory:

Block Number of memory in equivalent decimal	**Memory Location Address**		**Assume data stored in the location**
	Block Address	**Word Address**	
0	000000	00	1001010
	000000	01	1101010
	000000	10	0001010
	000000	11	0001010
1	000001	00	1111010
	000001	01	0101010
	000001	10	1001010
	000001	11	1101010
2	000010	00	1101010
	000010	01	0001010
	000010	10	0101010
	000010	11	0011010
…	…	…	…
7	000111	00	0000010
	000111	01	0000011
	000111	10	0000011
	000111	11	0001110
…	…	…	…
63	111111	00	1111010
	111111	01	1111011
	111111	10	0101011
	111111	11	0101110

Figure 6.9: An example of Main Memory Blocks

(i) Direct Mapping Cache:
The size of cache = 32 bytes
The block size of main memory = words in one line of cache =4 ⇒ k=2 bits
The cache has = 32 /4 = 8 lines with each line storing 32 bits of data (4 words)
Therefore, m=3 as 2^3 = 8

Thus, Tag size = (*n*-*k*) - *m* = (8 - 2) - 3 = 3

The address mapping for an address: 11111101

Block Address of Main Memory		**Address of a word in a Block**
111 111		01
111	111	01
Tag	**Line Number**	

Line Number = 111 = 7 in decimal
Tag = 111

The address mapping for an address: 00001011

Block Address of Main Memory		**Address of a word in a Block**
000 010		11
000	010	11
Tag	**Line Number**	

Line Number = 010 = 2 in decimal
Tag = 000

The following cache memory that uses direct mapping shows these two words (along with complete block in the cache)

Line Number of Cache in Decimal	**Contents of Cache Memory**				
	Tag of Data	**Data in Cache = 4 words = 32 bits**			
		Word 11	**Word 10**	**Word 01**	**Word 00**
0					
1					
2	000	0011010	0101010	0001010	1101010
3					
4					
5					
6					
7	111	0101110	0101011	1111011	1111010

Figure 6.10: An example Cache memory with Direct mapping

The access for an address: 00011110

Block Address of Main Memory		**Address of a word in a Block**
000 111		10
000	111	10
Tag	**Line Number**	

In case, a word like 00011110 is to be accessed, which is not in the cache memory and as per mapping should be mapped to line number 111 =7, the cache access logic will compare the tags, which are 000 for this address, and 111 in the cache line 7. This is the situation of cache miss, so accordingly this block will replace the content stored in line 7, which after replacement is shown below:

7	000	0001110	0000011	0000011	0000010

Please note the change in data value in the cache line 7.

(ii) Associative Mapping Cache:
The size of cache = 32 bytes
The block size of main memory = words in one line of cache =4 ⇒ *k*=2 bits
Therefore, cache has = 32 /4 = 8 lines with each line storing 32 bits of data (4 words)
Tag size = *n*-k = (8 - 2) = 6
The address mapping for an address: 11111101

Block Address of Main Memory	**Address of a word in a Block**
111111	01
111111	01
Tag	

The address mapping for an address: 00001011

Block Address of Main Memory	**Address of a word in a Block**
000010	11
000010	11
Tag	

The following associative cache shows these two words.

Line Number of Cache in Decimal	**Contents of Cache Memory**				
	Tag of Data	**Data in Cache = 4 words = 32 bits**			
		Word 11	**Word 10**	**Word 01**	**Word 00**
0	111111	0101110	0101011	1111011	1111010
1	000010	0011010	0101010	0001010	1101010
2	000111	0001110	0000011	0000011	0000010
3					
4					
5					
6					
7					

Figure 6.11: An example Cache memory with Associative mapping

The access for an address: 00011110

Block Address of Main Memory	**Address of a word in a Block**
000111	10
000111	10
Tag Number	

A word like 00011110 can in any cache line, for example, in the cache memory shown above it is in line 2 and can be accessed.
(iii) 2way set associative Mapping:
The size of cache = 32 bytes
The block size of main memory = words in one line of cache =4 ⇒ *k*=2 bits
The number of lines in a set (*w*) = 2 (this is a 2 way set associative memory)
The number of sets (*v*) = Size of cache in words/(words per line × *w*)
= 32/(4×2) =4
Thus, set number can be identified using 2 bits as $2^2 = 4$

Tag size = (*n*-*k*)-*v* = (8 - 2) - 2 = 4

The address mapping for an address: 11111101

Block Address of Main Memory		Address of a word in a Block
1111 11		01
1111	11	01
Tag	**Set Number**	

Set number = 11 = 3 in decimal

Tag = 1111

The address mapping for an address: 00001011

Block Address of Main Memory		Address of a word in a Block
0000 10		11
0000	10	11
Tag	**Set Number**	

Set number = 10 = 2 in decimal

Tag = 0000

Contents of Cache Memory Way 0					Set #	Contents of Cache Memory Way 1				
Tag						Tag	Data in Cache = 4 words = 32 bits			
	Word 11	Word 10	Word 01	Word 00			Word 11	Word 10	Word 01	Word 00
					0					
					1					
					2	0000	0011010	0101010	0001010	1101010
1111	0101110	0101011	1111011	1111010	3	0001	0001110	0000011	0000011	0000010

Figure 6.12: An example Cache memory with Set Associative mapping

The access for an address: 00011110

Block Address of Main Memory		Address of a word in a Block
0001 11		10
0001	11	10
Tag	**Set Number**	

Set number = 11= 3 in decimal

Tag = 0001

Word 00011110 can be stored and accessed from the cache set 11 at the second line (way 1).

6.4.3 Write Policy

Many processes read and write data in cache and main memory either by the processor or by the input/ output devices. Multiple read possess no challenge to the state of the data item, as you know, cache maintains a copy of frequently required data items to improve the system performance. Whenever a process writes/ updates the values of the data item in cache or in main memory, it must be updated in the copy as well. Otherwise it will lead to an inconsistent data and cache content may become invalid. Problems associated with writing in cache memories can be summarised as:

- Caches and main memory can be altered by multiple processes which may result in inconsistency in the values of the data item in cache and main memory.

- If there are multiple CPUs with individual cache memories, data item written by one processor in one cache may invalidate the value of the data item in other cache memories.

These issues can be addressed in two different ways:

1. **Write through:** This writing policy ensures that if a CPU updates a cache, then it has to write/ or make the changes in the main memory as well. In multiple processor systems, other CPUs-Cache need to keep an eye over the updates made by other processor's cache into the main memory and make suitable changes accordingly. It creates a bottleneck as many CPUs try to access the main memory.

2. **Write Back:** Cache control logic uses an ***update bit.*** Changes are allowed to write only in cache and whenever a data item is updated in the cache, the update bit of the block is set. As long as data item is in the cache no update is made in the main memory. All those blocks whose update bit is set is replaced in the main memory at the time when the block is being replaced in the cache. This policy ensures that all the accesses to the main memory are only through cache, and this may create a bottleneck.

You may refer to further readings for more details on cache memories.

Check Your Progress 2

1. Assume that a Computer system have following memories:
   RAM 64 words with each word of 16 bits
   Cache memory of 8 Blocks (block size of cache is 32 bits)
   Find in which location of cache memory a decimal address 21 can be found if Associative Mapping is used.

   ……………………………………………………………………………………

   ……………………………………………………………………………………

2. For the system as given above, find in which location of cache memory a decimal address 27 will be located if Direct Mapping is used.

……………………………………………………………………………………

   ……………………………………………………………………………………

3. For the system as given above, find in which location of cache memory a decimal address 12 will be located if two way set associative Mapping is used.

   ……………………………………………………………………………………

   ……………………………………………………………………………………

6.5 MEMORY INTERLEAVING

As you know that cache memory is used as a buffer memory between processor and the main memory to bridge the difference between the processor speed and access time of the main memory. So, when processor requests a data item, it is first looked into the cache and if data item is not present in the cache (called cache miss), only then main memory is accessed to read the data item. To further enhance the performance of the computer system and to reduce the memory access time of the main memory, in case of cache miss, the concept of memory interleaving is used. Memory interleaving is of three type, viz. lower order memory interleaving, higher order memory interleaving and hybrid memory interleaving. In this section we will discuss the lower order memory interleaving only. Discussion on other memory interleaving techniques is beyond the scope of this unit.

In memory interleaving technique, main memory is partitioned into *n* number of equal sized modules called as memory banks and technique is known as *n-way memory* *interleaving*. Where each memory module has its own memory address register, base register and instruction register, thus each memory bank can be accessed individually and simultaneously. Instructions of a process are stored in successive memory banks. So, in a single memory access time *n* successive instructions of the process can be accessed from *n* memory banks. For example, suppose main memory is divided into four modules or memory banks denoted as M1, M2, M3 and M4 then first n instructions of a process will be stored as: first instruction in M1, second instruction in M2, third instruction in M3, fourth instruction in M4 and again fifth instruction in M1 and so on.

When processor issues a memory fetch command during the execution of the program, memory access system creates *n* consecutive memory addresses and places them in respective memory address register of all memory banks in the right order. Instructions are read from all memory modules simultaneously and loads them into *n* instruction registers. Thus, each fetch for a new instruction results in the loading of *n* consecutive instructions in *n* instruction registers of the CPU, in the time of a single memory access. Figure 6.13 shows the structure of 4-way memory interleaving. The address is resolved by interpreting the least significant bits to select the memory module, and rest of the most significant bits are the address in the memory module. For example, in an 8-bit address and 4-way memory interleaving, two least significant bits will be used for module selection and six most significant bits will be used as an address in the module.

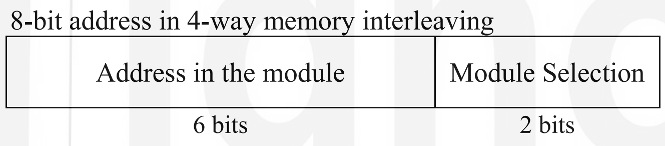


Figure 6.13: Address mapping for Memory interleaving

The following example demonstrates how the main memory words be distributed to different interleaved memory modules. For this example, only a four bit address of main memory is used.

Main Memory	
Address	**Data**
0000	10
0001	20
0010	30
0011	40
0100	50
0101	60
0110	80
0111	76
1000	46
1001	25
1010	58
1011	100
1100	23
1101	78
1110	35
1111	11

Module 00	
Address	**Data**
00	10
01	50
10	46
11	23

Module 01	
Address	**Data**
00	20
01	60
10	25
11	78

Module 10	
Address	**Data**
00	30
01	80
10	58
11	35

Module 11	
Address	**Data**
00	40
01	76
10	100
11	11

Figure 6.14: Example of Memory interleaving

ignou THE PEOPLE'S UNIVERSITY

Please note in the figure above how various data values are distributed in the modules.

6.6 ASSOCIATIVE MEMORY

Though cache is a high speed memory but still it needs to search the data item stored in it. Many search algorithms have been developed to reduce the search time in a sequential or random access memory. Searching time of a data item can be reduced further to a significant amount of data item is identified by its content rather by the address. Associative memory is a content addressable memory (CAM), that is memory unit of associative memory is addressed by the content of the data rather by the physical address. The major advantage of this type of memory is that memory can be searched in parallel on the basis of data. When a data item is to be read from an associative memory, the content of the data item, or part of it, is specified. The memory locates all data items, which matches the specified content, and marks them for reading. Because of the architecture of the associative memory, complete data item or a part of it can be searched in parallel.

Hardware Organization

Associative memory consists of a memory array and logic for *m* words with *n* bits per word as shown in block diagram in Figure 6.15. Both argument register (A) and key register (K) have *n* bits each. Each bit of argument and key register is for one bit of a word. The match register M has *m* bits, one each for each memory word.

The key register provides a mask for choosing a particular field or key in the argument word. The entire argument is compared with each memory word only if the key register contains all 1s. Otherwise, only those bits in the argument that have 1s in their corresponding positions of the key register are compared. Thus, the key provides a mask or identifying information, which specifies how reference to memory is made.

The content of argument register is simultaneously matched with every word in the memory. Corresponding bits in the mach register is set by the words that have match with the content of the argument register. Set bits of the matching register indicates that corresponding words have a match. Thereafter, memory is accessed sequentially, to read only those words whose corresponding bits in the match register have been set.

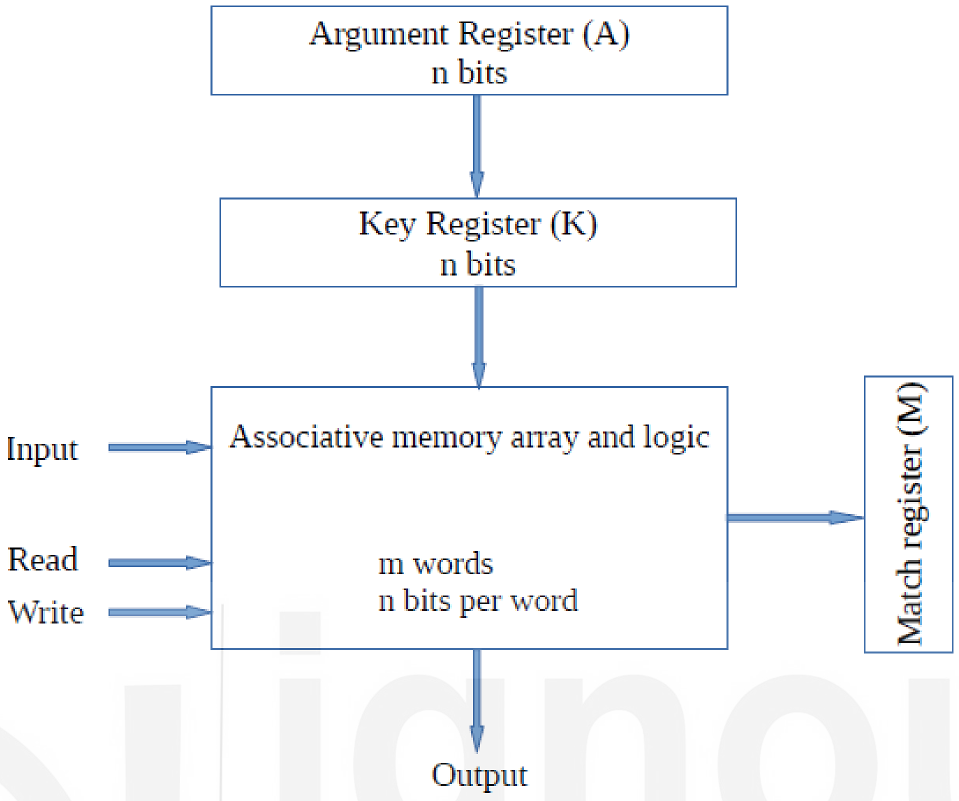


Figure 6.15: Block diagram of associative memory

Example: Consider an associative memory of just 2 bytes. The content register and argument registers are also shown in the diagram.

Description	The content of associative Memory								Match Word
Argument Register	0	1	1	0	0	0	0	1	
Key Register	1	1	1	1	0	0	0	0	
Bits to be matched	0	1	1	0					
Word 1	0	1	1	0	0	1	1	0	Match
Word 2	1	0	0	1	1	0	0	0	Not machted

Figure 6.16: An Example of Associative matching

Please note as four most significant bits of key register are 1, therefore only they are matched.

6.7 VIRTUAL MEMORY

As we know that a program is loaded into the main memory for execution. The size of the program is limited by the size of the main memory i.e. cannot load a program in to the main memory whose size is larger than the size of the main memory. Virtual memory system allows users to write programs even larger than the main memory. Virtual memory system works on the principle that portions of a program or data are loaded into the main memory as per the requirement. This gives an illusion to the programmer that they have very large main memory at their disposal. When an address is generated to reference a data item, virtual address generated by the processor is mapped to a physical address in the main memory. The translation or mapping is handled automatically by the hardware by means of a mapping table.

Let us say, you have a main memory of size 256K (2^{18})words. This requires 18-bits to specify a physical address in main memory. A system also has an auxiliary memory as large as the capacity of 16 main memories. So, the size of the auxiliary memory is 256K ×16 = 4096 K which requires 24 bits to address the auxiliary memory. A 24-bit virtual address will be generated by the processor which will be mapped into an 18-bit physical address by the address mapping mechanism as shown in Figure 6.17.

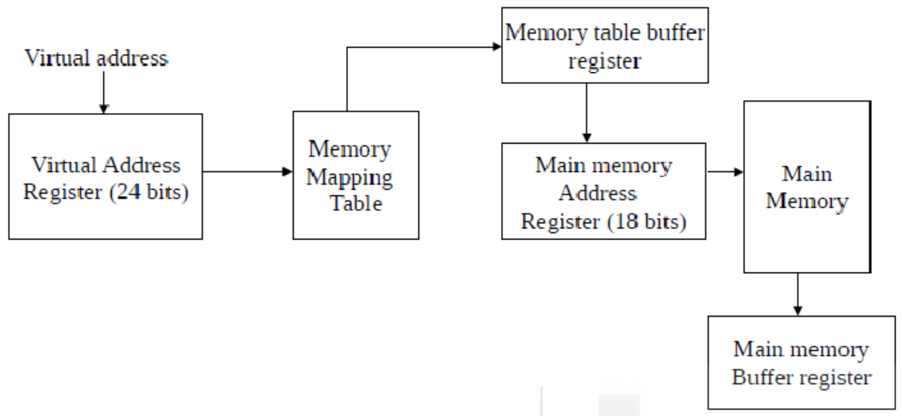

Virtual address
Virtual Address Register (24 bits)
Memory Mapping Table
Memory table buffer register
Main memory Address Register (18 bits)
Main Memory
Main memory Buffer register

Figure 6.17: Virtual Address Mapping to Physical Address

In a multiprogramming environment, programs and data are transferred to and from auxiliary memory and main memory based on demands imposed by the processor. For example program 1 is currently being executed by the CPU. Only program 1 and a portion of its associated data as demanded by the processor are loaded from secondary memory into the main memory. As programs and data are continuously moving in and out of the main memory, space will be created and thus both program or its portions and data will be scattered throughout the main memory.

Check Your Progress 3

1. How can interleaved memory can be used to improve the speed of main memory access?

   ………………………………………………………………………………………

   ………………………………………………………………………………………

2. Explain the advantages of using associative memory?

   ………………………………………………………………………………………

   ………………………………………………………………………………………

3. What is the need of virtual memory.

   ………………………………………………………………………………………

   ………………………………………………………………………………………

6.8 SUMMARY

This unit introduces you to the concept relating to cache memory. The unit defines some of basic issues of cache design. The concept of cache mapping schemes were explains in details. The direct mapping cache uses simple modulo function, but has limited use. Associative mapping though allows flexibility but uses complex circuitry and more bits for tag field. Set-associative mapping uses the concept of associative and direct mapping cache. The unit also explain the use of memory interleaving, which allows multiple words to be accessed in a single access cycle. The concept of content addressable memories are also discussed. The cache memory, memory interleaving and associative memories are primarily used to increase the speed of memory access. Finally, the unit discusses the concept of virtual memory, which allows execution of programs requiring more than physical memory space on a computer. You may refer to further readings of the block for more details on memory system.

6.9 ANSWERS

Check Your Progress 1

1. While executing a program during a period of time or during a specific set of instructions, it was found that memory reference to instructions and data tend to cluster to a set of memory locations, which are accessed frequently. This is referred to as locality of reference. This allows you to use a small memory like cache, which stores the most used instructions and data, to enhance the speed of main memory access.
2. Typical block size of main memory for cache transfer may be 1, 2, 4, 8, 16, 32 words.
3. *effective access time = 0.9 (hit ratio) x 10 (cache hit time)*
   + 0.1(miss ratio) x (50+10) (cache miss and memory reference)
   effective access time = 0.9 x 10 + 0.1x 60
   = 9 + 6
   = 15 ns

Check Your Progress 2

1. Main memory size = 64 words (a word = 16 bits) = $2^6 \Rightarrow n$=6 bits
Block Size = 32 bits = 2 words = $2^1 \Rightarrow k$=1 bit
The size of cache = 8 blocks of 32 bits each = 8 lines
Tag size for associative mapping = n-k = (6 - 1) = 5
The address mapping for an address: 21 in decimal that is 010101

Block Address	**Address of a word in a Block**
01010	1
01010	1
Tag	

In set associative memory the given tag can be stored in any of the 8 lines.

2. Main memory size = 64 words (a word = 16 bits) = $2^6 \Rightarrow n$=6 bits
Block Size = 32 bits = 2 words = $2^1 \Rightarrow k$=1 bit
The size of cache = 8 blocks of 32 bits each = 8 lines $\Rightarrow m$=3 bits
Tag size for direct mapping = (n-k) - m = (6 - 1) - 3 = 2
The address mapping for an address: 27 in decimal that is 011011

Block Address of Main Memory		**Address of a word in a Block**
01 101		1
01	101	1
Tag	**Line Number**	

The required word will be found in line number 101 or 5 (decimal)

3. Main memory size = 64 words (a word = 16 bits) = $2^6 \Rightarrow n$=6 bits
Block Size = 32 bits = 2 words = $2^1 \Rightarrow k$=1 bit

ignou THE PEOPLE'S UNIVERSITY

The number of sets (*v*) = 4 sets of 2 lines each, thus, *d* = 2
Tag size for direct mapping = (*n*-*k*) - *d* = (6 - 1) - 2 = 3
The address mapping for an address: 12 in decimal that is 001100

Block Address of Main Memory		**Address of a word in a Block**
001 10		0
001	10	0
Tag	**Set Number**	

Set number = 10 = 2 in decimal
Thus, required word can be in any of the line in set number 2.

Check Your Progress 3

1. Memory interleaving divides the main memory into modules. Each of these module stores the words of main memory as follows (example uses 4 modules and 16 word main memory.
   Module 0: Words 0, 4, 8, 12 Module 1: Words 1, 5, 9, 13
   Module 2: Words 2, 6, 10, 14 Module 3: Words 3, 7, 11, 15
   Thus, several consecutive memory words can be fetched from the interleaved memory in one access. For example, in a typical access words 4, 5, 6, and 7 can be accessed simultaneously from the Modules 0, 1, 2 and 3 respectively.

2. Associative memory do not use addresses. They are accessed by contents. They are very fast.

3. Virtual memory is useful, when large programs are to be executed by a computer having smaller physical memory.

ignou THE PEOPLE'S UNIVERSITY