

Computer Organization and Architecture: A Pedagogical Aspect
Prof. Jatindra Kr. Deka
Dr. Santosh Biswas
Dr. Arnab Sarkar
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 25
Direct-mapped Caches: Misses, Writes and Performance

(Refer Slide Time: 00:28)

Memory – Physical Issues

- Memory technologies – Access time, cost per GB
 - SRAM
 - 0.5 - 2.5 ns, \$2000 - \$5000 per GB
 - DRAM
 - 50 - 70 ns, \$20 - \$75 per GB
 - Magnetic disk
 - 5 - 20 ms, \$0.20 - \$2 per GB

Computer Organization and Architecture 11

Unit 1 part 2: We ended part 1 of unit 1 by saying that we have different memory technologies which vary in terms of their access times and cost per GB. For example, we said that SRAMs are very fast and its speed is about one 0.5 to 2.5 nanoseconds, its access time; that means, it is on an average about one-tenth as fast as the processor ok. However, the cost per GB of this type of memories is also very huge. The cost per GB is about 2000 dollars to 5000 dollars.

Then we have DRAMs which are about 150 to 100 times slower than SRAMs; that means, to bring a certain amount of data a data unit a word from DRAM the processor will require about hundreds of processor cycles to do so. The speed of a DRAM is typically in the range of 50 to 70 nanoseconds; that is the access time is in the range of 50 to 70 nanoseconds. But, it is also about hundred times cheaper than SDRAMs. So, the typical cost of DRAM units range in between 20 dollars to 75 dollars per GB.

Magnetic disks or hard disks are far more cheaper; about 1000 times cheaper than DRAMs being only about 0.2 to 2 dollars per GB. However, it is also about 1000 times slower than DRAM units. Its access times ranges in between 5 to 20 milliseconds. So, to bring a data word from the hard disk, the processor required tens of thousands of processor cycles.

(Refer Slide Time: 02:26)

Towards Memory Hierarchy

- **To achieve greatest performance**
 - Memory should be able to keep pace with the processor
 - Waiting for instructions/operands when processor executes instructions is not desirable
 - Hence we would like to use the fastest available memory technology
- **We need large capacity memory to hold all required information**
- **The cost of memory must be reasonable w.r.t other components**
- **Thus, we have a *design trade-off***

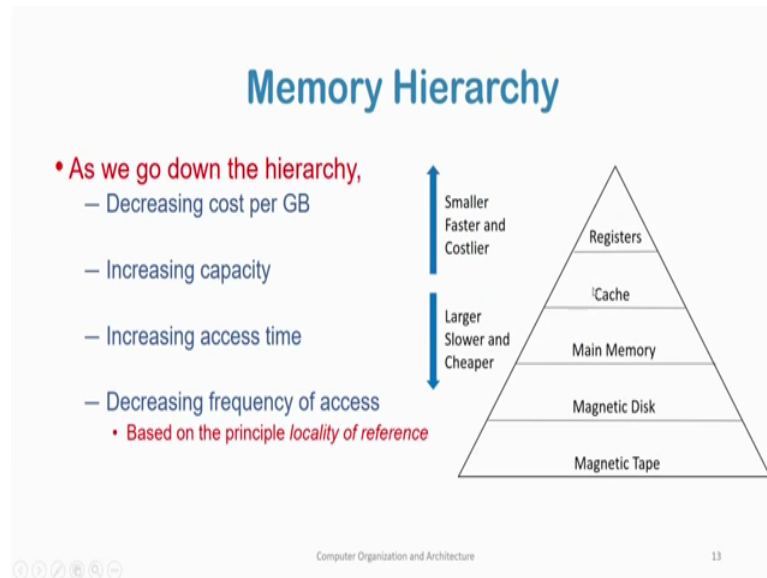
- **The solution**
 - *A memory hierarchy where smaller, more expensive and faster memories are supplemented by larger, cheaper and slower memories*

Computer Organization and Architecture 12

So, to achieve the best performance what would we desire? We would desire a very large capacity memory which can hold all our programs and data and which works at the pace of the processor. That means, if a processor requires a memory word in one cycle it is available in the processor from memory in the next cycle itself. However, in practice we saw the cost and performance parameters and it is difficult to achieve. So, to achieve the greatest performance memory should be able to keep pace with the processor. It is not desirable to wait for instruction slash operands when the processor executes instructions. And hence we would like to use the fastest available memory technology. We also need a large capacity memory to hold all our required information.

However, the cost of memory must be reasonable with respect to other components. Hence we understand that we have a design trade off. So, although the faster memories the mem although SRAMs are very fast in terms of access time, they are also very costly. The solution is to have memory hierarchy where smaller more expensive and faster memories are supplemented by larger, cheaper and slower memories.

(Refer Slide Time: 04:18)



Therefore, we have registers in the processor we typically have a few dozens of these registers and registers operate at the same speed as that of the processor. However, they are very expensive and we cannot have a large number of registers in the processor. Next in the hierarchy is cache. As I told it is about one tenth as fast as the processor speed; however, it is also very costly. Then we have the main memory which is a which is slower than cache memory about hundreds of times slower than the processor speed and; however, its cost is cheaper than that of the cache. We have magnetic disks which are much cheaper than the main memory. However, its access times are also much slower and so on.

So, as we go down the hierarchy we have decreasing cost per GB, increasing capacity because it is cheaper we can have more capacity, more amount of that memory. We, but we also have increasing access times as we go down the hierarchy memories become slower. And we have decreasing frequency of access based on and this phenomenon that we have that we are able to have decreasing frequency of access towards in memories which are down the hierarchy is based on the principle of the locality of reference.

(Refer Slide Time: 06:00)

Principle of Locality

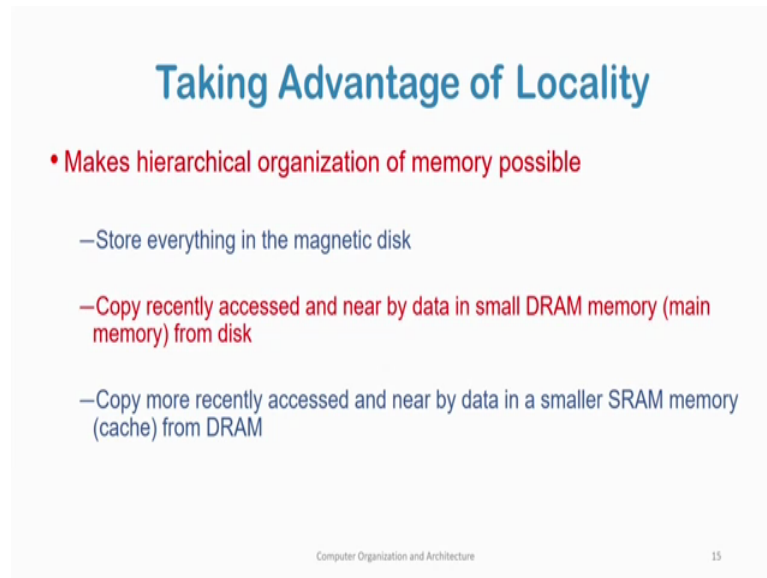
- **Programs access a small portion of the memory at a given time**
 - Programs typically contain a number of loops and subroutines
 - Within a loop/subroutine, a small set of instructions are repeatedly accessed
- **Temporal locality**
 - Items accessed recently are likely to be accessed again
 - eg. instructions in a loop
- **Spatial locality**
 - Items near to those accessed recently are likely to be accessed soon
 - eg. sequential access of data from array

Computer Organization and Architecture 14

Principle of the locality of reference is based on the fact that programs tend to access data and instructions and data in clusters, in the vicinity in the near vicinity at a of a given memory location. So, programs access a small portion of memory at a given time. Why? Because programs typically contain a large number of loops and subroutines, and within a loop order subroutine a small set of instructions are repeatedly accessed. These instructions again tend to access data in clusters. So, there are two distinct principles in the locality of reference. Temporal locality which says that items access recently are likely to be accessed again. For example, the instructions within a loop.

So, if the instructions within in one iteration of the loop will be again accessed in the next iteration of the loop. And special locality in items near those access recently are likely to be accessed soon; for example, sequential access of data from an array. So, if you have a big array we tend to access data one by one from the array in sequence. So, how do how does this locality how does this principle of the locality of reference helps to maintain this hierarchical memory organization.

(Refer Slide Time: 07:17)



Taking Advantage of Locality

- Makes hierarchical organization of memory possible
 - Store everything in the magnetic disk
 - Copy recently accessed and near by data in small DRAM memory (main memory) from disk
 - Copy more recently accessed and near by data in a smaller SRAM memory (cache) from DRAM

Computer Organization and Architecture 15

So, principle of locality makes hierarchical organization of memory possible. So, how can we do that? For example, we can store everything we stored everything in the magnetic disk. And then we copy recently accessed and nearby data in a small DRAM memory or the main memory. So, the main memory uses this technology of DRAM from the disk.

So, we have the magnetic disk which stores everything and whatever we require currently we access, we access it and store it in a DRAM or the main memory. Then whatever is still more recently accessed data and instructions are stored in an SRAM memory which is cache from the DRAM.

(Refer Slide Time: 08:20)

Cache Memory

- **Small amount of fast memory**
 - Sits between normal main memory and CPU
 - May be located on the CPU chip or as separate modules
- **When the processor attempts to read a memory word**
 - A check is made to determine if the word is in cache
 - If word is in cache we have a *cache hit*; otherwise we suffer a *cache miss*
 - Hit time*: Time to access a memory word in case of a *hit*
 - Fraction of memory accesses resulting in hits: *hit ratio (or hit rate)* = $\frac{\#hits}{\#accesses}$
 - Miss ratio (or miss rate)* = $1 - Hit\ ratio$

Computer Organization and Architecture 16

Cache memory: So now we begin our discussion on cache memory. So, cache memory as we said is based on the SRAM memory technology. It is a small amount of fast memory which sits between the main memory and the CPU and it may be located within the CPU chip or a separate modules which are plugged in on the motherboard. So, when the processor attempts to read a memory word from the main memory, it what does it do? It places the address of the memory word from where on the address bus. Then what is done? A check is made to determine if the word is in cache. If the word is in cache we have a cache hit otherwise we suffered a catch miss. What is the hit time? The time to access a memory word in memory word in case of a hit is the hit time. So, fraction of memory accesses resulting in hits is called the hit ratio or the hit rate and is defined as number of cache hits over a certain given number of accesses on the memory.

Miss ratio or miss rate is; obviously, 1 minus the hit ratio. In case of a cache miss a block of memory consisting of a fixed number of words is read into the cache and then the word is delivered to the processor. A block of memory is fetched instead of only the requested memory word to take advantage of the locality of reference. Future references may access other words in the block ok. A block of data is fetched instead of only the requested memory word to take advantage of the locality of reference because future references may access other words in the block. And in that case when those future references are made we will have a cache hit. Miss penalty the time to replace a cache block and delivered requested word to the processor is known as miss penalty.

(Refer Slide Time: 09:50)

Cache Memory

- **In case of a cache miss**
 - A block of memory consisting of a fixed number of words is read into cache and then the word is delivered to the processor
 - A block of data is fetched instead of only the requested memory word
 - To take advantage of the locality of reference
 - Future references may access other words in the block
 - **Miss penalty:** Time to replace cache block and deliver requested word to processor

The diagram illustrates two cache architectures. On the left, 'Single Cache' shows a CPU connected to a Cache, which is connected to Main Memory. A 'Word Transfer' arrow points from Cache to CPU, labeled 'Fast'. A 'Block Transfer' arrow points from Main Memory to Cache, labeled 'Slow'. On the right, 'A Three-level Cache Organization' shows a CPU connected to Level 1 (L1) cache, which is connected to Level 2 (L2) cache, which is connected to Level 3 (L3) cache, which is finally connected to Main Memory. The L1 cache is labeled 'Fastest', L2 'Fast', L3 'Less fast', and Main Memory 'Slow'. Red handwritten annotations include checkmarks and arrows pointing to the L1, L2, and L3 cache boxes.

Computer Organization and Architecture 17

So, here in the figure on the left we see that CPU asks for a word from memory and if that word is present in cache, we send it back to the CPU and this is word transfer. If this word is not present in cache we have a cache miss and then we fetch a block from main memory and this block contains the desired word also. So, between cache and main memory we have block transfer whereas, between CPU and cache we have word transfer. The figure on the right shows that we may have different levels of cache not only a single level of cache.

So, we have CPU followed by followed by a small a small very fast cache, followed by a level 2 cache which is lower than level one cache, but is also higher in capacity. We also may have level 3 cache which is higher in capacity than level 2 cache, but is also slower and then finally, we have the main memory.

(Refer Slide Time: 12:00)

Correction (Valid bit): The valid bit indicates whether data in an accessed cache line is valid. When the system is initialised, or the cache is flushed, valid bits of all cache lines are set to 0. When data is loaded into a particular cache line, the corresponding valid bit is set to 1.

- Main memory consists of 2^n addressable words
- For the purpose of mapping, the main memory is considered to consist of $M = 2^n / K$, fixed-length blocks of K words each
- The cache consists of m blocks, called *lines*
- Each line contains K words, a few tag bits and a valid bit
- The length of a line, not including the tag and valid bit, is the *line size*
- The no. of lines \ll no. of main memory blocks, i.e. $m \ll M$
- tag identifies which particular main memory block is currently in a line
- Valid bit indicates whether the line has been modified since being loaded in cache

Computer Organization and Architecture 18

Let us assume that we have an n bit address bus. Therefore, we have a main memory consisting of 2^n addressable words. For the purpose of mapping the main memory is considered to consist of M blocks of $2^n / K$ fixed length blocks of K words each. So, we have a main memory which consists of 2^n words or bytes and a block consisting of K words or bytes each. And each so the number of blocks we have in main memory is given by $2^n / K$. The cache contains m blocks called lines. Each line contains K words same as the same as the block size plus a few tag bits and a valid bit.

The length of a line not including the tag and valid bit is the line size. The number of lines in cache is much less than the main memory block size; that is small m is much much less than capital M . The tag identifies which particular main memory block is currently in a line; so therefore, right. The valid bit indicates whether the line has been modified since being loaded in cache.

(Refer Slide Time: 13:38)

Mapping Function

- $m \ll M$
 - we need a mechanism for mapping main memory blocks to cache lines
- **Direct Mapping – The simplest**
 - Each main memory block may be mapped to a single unique cache line
 - The mapping: $i = j \text{ modulo } m$
 - $i = \text{cache line no.}, j = \text{main memory block no.}, m = \text{no. of cache lines}$

Computer Organization and Architecture

19

Since, small m is much much less than capital M ; that is the number of lines in cache is much much less than the number of blocks in the main memory we need a mechanism for mapping main memory blocks to cache lines. Therefore, we have a mapping function. The simplest mapping function is called direct mapping. In this each main memory block may be mapped to a single unique cache line and the mapping function is given by i equals to j modular m ; where i is the cache line number, j is the main memory block number and m is the number of cache lines.

So, in this example that we the figure that we have at the bottom the cache has 8 lines and the main memory has 16 blocks. And we see that blocks 0 and block number 16, block number 0 and blocks number 16 maps both map to cache line number 0. Similarly, block number 15 as well as block number similarly block number 7 and block number 15 both map to line number 7.

(Refer Slide Time: 15:03)

Direct Mapping

- For purposes of cache access
 - Each main memory address may be viewed as consisting of $(s + w)$ bits
 - w LSBs = identify unique word (or byte) within a main memory block
 - Block size = Line size = 2^w bytes
 - s MSBs = block id; one of 2^s main memory blocks
 - Given size of cache as $m = 2^r$
 - r = Determines *line no.* or *cache index no.*
 - $(s - r)$ MSBs of main memory address = size of tag
 - Thus the main memory address has *three* parts:

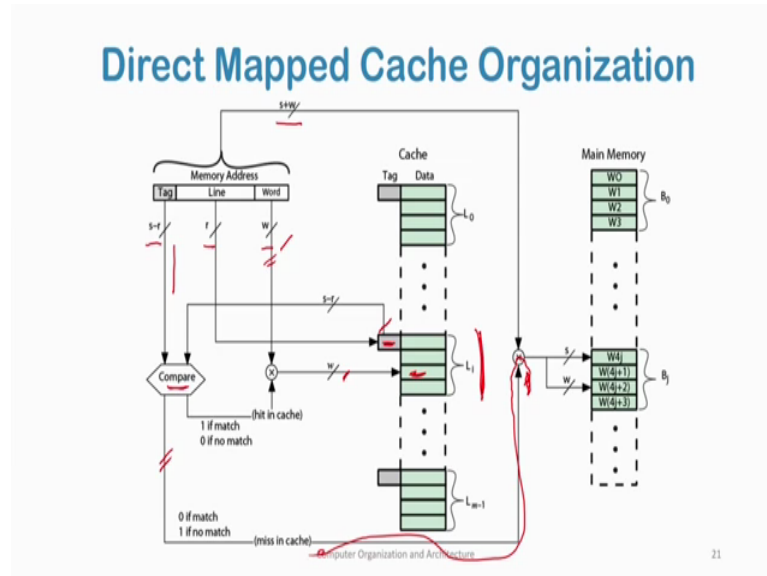
Tag	Index	Word
$s-r$ bits	r bits	w bits

Computer Organization and Architecture

For the purposes of cache access, when we want to read the cache, each main memory address may be viewed as consisting of s plus w bits. So, we have a main memory address consisting of s plus w bits. So, here this is s and this is w . Each main memory address may be viewed as consisting of s plus w bits. In which the w LSBs, the least significant bits identify a unique word within or byte within a main memory block. The block size is equal to the line size and we have 2^w addressable bytes within a block or line. Because the w LSBs, there are w LSBs, we have 2^w addressable bytes within a block or line. The s MSBs equals to is the block id, the most significant s bits are the block id. So, it identifies one of 2^s main memory blocks. Given the size of cache equals to $m = 2^r$. So, we have let the number of lines in cache equals to m and this m it is equals to 2^r ; r determines the, determines a line number or the cache index number.

So, r bits are used to determine the line number or cache index number. So, s minus r bits or the MSBs s minus r MSBs of the main memory address gives the size of the tag field. And thus the main memory address has 3 parts; the tag field which is the s minus r MSBs, the next r bits identify a line in cache and the least significant w bits identify a word in the main memory.

(Refer Slide Time: 17:19)



So, this figure shows the organization of a direct mapped cache. We see that the memory address consists of s plus w bits. The tag is s minus r bits long. The cache line index, the cache index the cache is indexed by an r length r bit length quantity and the each word within a particular block or line is identified by the by this word offset here by this word offset ok.

So, to identify whether a particular a particular line is in cache or not what do we do? We first match the; we first come to the line. We come to the line which is identified by identified by these r bits and then we compare the tag field. This is the tag field within the cache, we compare the tag field with the s minus r main memory bits. If this comparison says is if this comparison is 1, we have a match and a hit in cache. When we have a hit in cache, we read the word we read the corresponding word in the cache and we retrieve it ok.

So, the corresponding word is identified by these least significant w bits. And so, this identifies which word within this block or line which word within this line is in cache ok. Now if there is a miss; that means, the tag in the in the particular cache line in the tag in that particular cache line does not match with the main memory address tag; that is these s minus r when we have a mismatch here, we have a cache miss and then we go to the main memory. We go to the main memory and find the particular block in main memory containing the word and then retrieve it into the cache.

(Refer Slide Time: 19:49)

Direct Mapped Cache – Example 1

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

8 blocks, 1 word/block, direct mapped
Initial state

Sequence of main memory accesses are:
22, 26, 16, 3, 16, 18

Computer Organization and Architecture 22

Now we take an example of a very simple example of a direct mapped cache. For this cache we only have 8 blocks or 8 lines in the cache. We have 1 word per block so every word is a block. We have a direct mapped cache and the initial state is all blank. So, we see that all the valid bits are N; that means, nothing has been accessed, the tag field is empty, data is empty and there is nothing in the cache basically. And we have let us say we have the sequence of memory accesses 22, 26, 16, 3, 16, 18.

(Refer Slide Time: 20:36)

Direct Mapped Cache – Example 1

Index	V	Tag	Data	Addr	Binary addr	H/M	Cache block
000	Y	10	M[10000]	22	10110	M	110
001	N			26	11010	M	010
010	Y	11	M[11010]	16	10000	M	000
011	Y	00	M[00011]	3	00011	M	011
100	N			16	10000	H	000
101	N						
110	Y	10	M[10110]				
111	N						

Computer Organization and Architecture 25

So, when the first address 22 is accessed, the corresponding binary address of 22 is 10110. We have 8 lines in cache. So, the least 3 significant bits identify the cache line, the 2 most significant bits become the tag bits. We have a miss in cache because the cache is initially empty. We would retrieve the, we retrieve the cache we retrieve it from the main memory and put it at line put it at line 110 and the tag is 10 as we see. So, when the next address 26 is accessed, we again have a miss. The corresponding binary addresses is the corresponding binary address is 010. So, we put it at line 010 with the tag 11 which is the most 2 significant bits right and we take it from memory and put it in this cache line.

Next 3 memory accesses we access first we access 16. So, the line index is 000, the tag field is 10, we have a miss in cache right we have a miss in cache and we put it back into the memory. Next we access 3. So, the line number is 011, we again have a miss in cache and the tag is 00. Next we access 16 again.

(Refer Slide Time: 22:32)

Direct Mapped Cache – Example 1

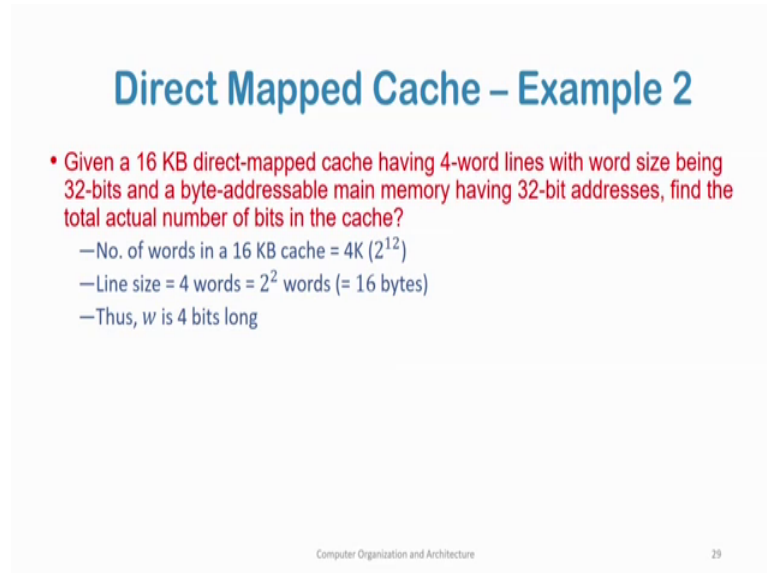
Index	V	Tag	Data	Addr	Binary addr	H/M	Cache block
000	Y	10	M[10000]	22	10110	M	110
001	N			26	11010	M	010
→ 010	Y	10	M[10010]	16	10000	M	000
011	Y	00	M[00011]	3	00011	M	011
100	N			16	10000	H	000
101	N			18	10010	M	010
110	Y	10	M[10110]				
111	N						

Computer Organization and Architecture 26

Now, 16 is already there in the cache, 16 is already there in the cache. We have a hit, we have a hit right after that when we access 18 we see that the line is 010. So, when we have 010 we already had 010 which is 26, 26 previously in this position we had we had 010; that means, the tag was 11, there was a mismatch in the tag. And therefore, there was a miss. There was a mismatch in the tag and therefore, there was a miss. We

replaced the cache block 26 with 18 and put it back into the cache with the new tag and word.

(Refer Slide Time: 23:16)



Direct Mapped Cache – Example 2

- Given a 16 KB direct-mapped cache having 4-word lines with word size being 32-bits and a byte-addressable main memory having 32-bit addresses, find the total actual number of bits in the cache?
 - No. of words in a 16 KB cache = $4K (2^{12})$
 - Line size = 4 words = 2^2 words (= 16 bytes)
 - Thus, w is 4 bits long

Computer Organization and Architecture 29

We come to a second example: given, a 16 KB direct mapped cache, having 4-word blocks with word size being 32 bits. A byte addressable main memory having 32 bit addresses we need to find the total actual number of bits in the cache. The first important thing to remember here is that line size is given by the number of words in each cache line. However, the number of bits in each line is given by the word itself along with the number of bits represented by the tag as well as the valid bit.

(Refer Slide Time: 24:39)

Direct Mapped Cache – Example 2

- Given a 16 KB direct-mapped cache having 4-word lines with word size being 32-bits and a byte-addressable main memory having 32-bit addresses, find the total actual number of bits in the cache?

- No. of words in a 16 KB cache = $4K (2^{12})$
- Line size = 4 words = 2^2 words (= 2^4 bytes; thus, w is 4 bits long)
- No. of Lines = $2^{12-2} = 2^{10}$ (thus, r is 10 bits long)
- No. of blocks in main memory = $2^{32-4} = 2^{28}$ (thus, s is 28 bits long)
- No. of tag bits = $s - r = 28 - 10 = 18$

Tag	Index	Byte offset
18 bits	10 bits	4 bits

So firstly, the number of words in the cache is given by in the in this in the in the 16 KB cache is 4K. Why? Because each word is 32 bits, so we have a 4 byte word and we have a 16 KB cache. So, we have 4K words in the cache. Line size equals to 4 words. So, each line contains 4 words; that means 2 to the power 2 words. So, w in this case is given by is 4 bits long. So, number of lines in the cache is given by the number of words divided by the number the number of words divided by the number of words in each line. So, total number of words divided by the number of words in each line, therefore, it is equal to 2 to the power twelve divided by 2 equals to the power 10. So, in our case we need 10 bits to address each line in the cache.

Number of blocks in main memory is given by 32. So, at the total number of blocks in main memory this total number of words in main memory is 2 to the power 32. Each block each block contains 2 to the power 4, 2 to the power 4 bytes. So, the number of blocks in main memory is equal to the total number of bytes divided by the number of bytes in each block and that is equals to 2 to the power 32 divided by 2 to the power 4 equals 2 to the power 28. So, therefore, we have s equals 2 to the power s is given by 2 to the power sorry 28 bits because we have 2 to the power 28 blocks in main memory. Therefore, the number of tag, so the number of tag bits in each line is s minus r equals to 28 minus 10 equals to 18 ok.

(Refer Slide Time: 26:20)

Direct Mapped Cache – Example 2

- Given a 16 KB direct-mapped cache having 4-word lines with word size being 32-bits and a byte-addressable main memory having 32-bit addresses, find the total actual number of bits in the cache?
 - No. of words in a 16 KB cache = $4K (2^{12})$
 - Line size = 4 words = 2^2 words (= 2^4 bytes; thus, w is 4 bits long)
 - No. of Lines = $2^{12-2} = 2^{10}$ (thus, r is 10 bits long)
 - No. of blocks in main memory = $2^{32-4} = 2^{28}$ (thus, s is 28 bits long)
 - No. of tag bits = $s - r = 28 - 10 = 18$
 - Each line contains: 4×32 bits of data + 18 tag bits + 1 valid bit = 147 bits
 - Total number of bits in the 2^{10} available lines: $2^{10} \times 147 = 147$ Kbits = 18.4 KB
- *For this cache, the total actual number of bits in cache (18.4 KB) is about 1.15 times as many bits needed just for data storage (16 KB)*

Computer Organization and Architecture 34

Now, each line therefore, contains 4 into 32 bits of data. So, we have 4 words each containing 32 bits. So, we have 4 in to 32 bits of data plus we have 18 tag bits plus 1 valid bit. So, we have one 47 bits of data in each line. So, the total number of bits in the 2 to the power 10 available lines is given by 2 to the power 10 into 147 equals to 147K bits equals to 18.4 kilobytes. Hence for this cache the total actual number of bits in cache which is 18.4KB is about 1.5 times as many bits needed for data storage which is 16 kilobytes.

(Refer Slide Time: 27:14)

Direct Mapped Cache – Example 3

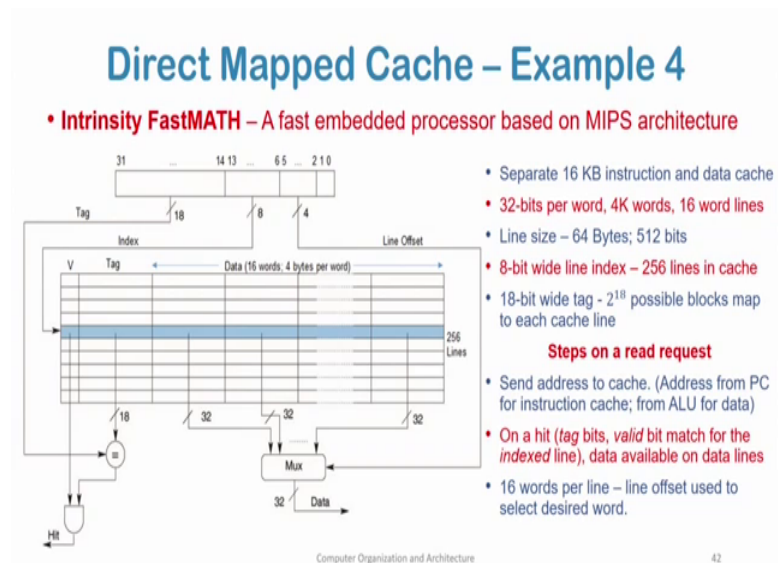
- Consider a cache with 64 blocks and a block size of 16 bytes? To what line number does byte address 1200 map?
 - Main memory block number in which byte 1200 belongs: $\left\lfloor \frac{1200}{16} \right\rfloor = 75$
 - Therefore, cache line number is given by: $75 \text{ modulo } 64 = 11$
- *This 75th block maps all byte addresses between 1200 and 1215*

Computer Organization and Architecture 37

Now we take a third example: Consider a cache with 64 blocks and a block size of 16 bytes. To what line number does byte address 1200 map? So, the main memory block number in which byte 1200 belongs is given by 1200 divided by 16. Why? We have 16 bytes in each block and the block id is 1200. So, to which block number will this is will this byte 1200 belong it is given by 1200 divided by 16 which is 75.

Now, therefore, the cache line number is given by 75 modulo 64. Why because we have 64 lines in the cache. So, blocks have been mistakenly said we have 64 lines in the cache and therefore, the cache line number is given by 75 modulo 64 which is 11. So, this 75th block of the main memory or this 11th line in the cache will contain all addresses between 1200 and between 1200 and 1215. The line in the cache may so, the 75th block rather of the main memory will contain 1200 to 1215 addresses.

(Refer Slide Time: 28:54)



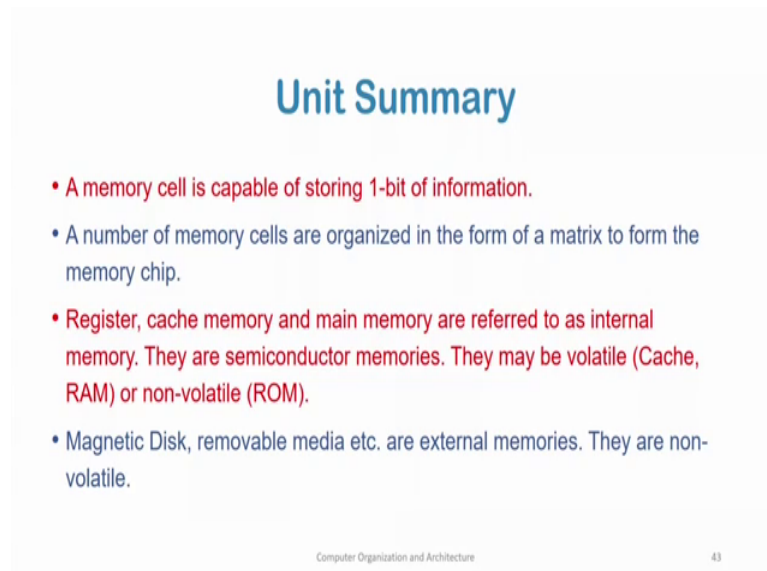
As a fourth and last example we take the example of a real word processor which uses direct mapped cache. So, we take the example of in Intrinsity FastMATH processor which is a fast embedded processor based on MIPS architecture. The direct mapped cache organization of this of this processor is shown in the figure here. So, the cache uses separate 16 KB instruction and data caches there is the mem the organization has 16 KB instruction and data caches separate. We have 32 bits per word. So, therefore, we have 4 byte words. We have 4Kwords in the cache and we have 16 word lines. So, each line contains 16 words. So, line size is 64 bytes. So, 16 words, each containing 4 bytes is 64

bytes and 64 bytes or 512 bits. We have a 8 bit wide line index. So, therefore, we have 256 lines in the cache. We have a 18 bit wide tag field. So, 2 to the power 18 possible blocks can map to each cache line ok.

So, we have 2 to the power 18 possible blocks that can map to each cache line. What are the steps to further read request on this? We send the address to cache, either the instruction cache or the data cache. Addresses are sent from the PC for the instruction cache and from the ALU for the data cache. On a hit, that means, the tag bits and valid bits match the tag bits and the valid bits match. On a hit when we have the tag bits and the valid bits matching, the data is made available on the data lines. We have 16 words per line; that means, a line offset 16 words per line.

So, we need to identify which word in the line is required. So, what we have? We have a line offset which is used to select which word in the line is desired by the memory. So, this line offset is used as a selector in a 16 cross 1 mux and we have a 4 bit line access because we have 16 words in the line. And based on this selection mechanism from the mux we get the required data.

(Refer Slide Time: 32:00).

A slide titled "Unit Summary" with a blue header. It contains four bullet points in red text. The first bullet point states: "A memory cell is capable of storing 1-bit of information." The second bullet point states: "A number of memory cells are organized in the form of a matrix to form the memory chip." The third bullet point states: "Register, cache memory and main memory are referred to as internal memory. They are semiconductor memories. They may be volatile (Cache, RAM) or non-volatile (ROM)." The fourth bullet point states: "Magnetic Disk, removable media etc. are external memories. They are non-volatile." At the bottom of the slide, there is a footer that reads "Computer Organization and Architecture" on the left and "43" on the right.

Unit Summary

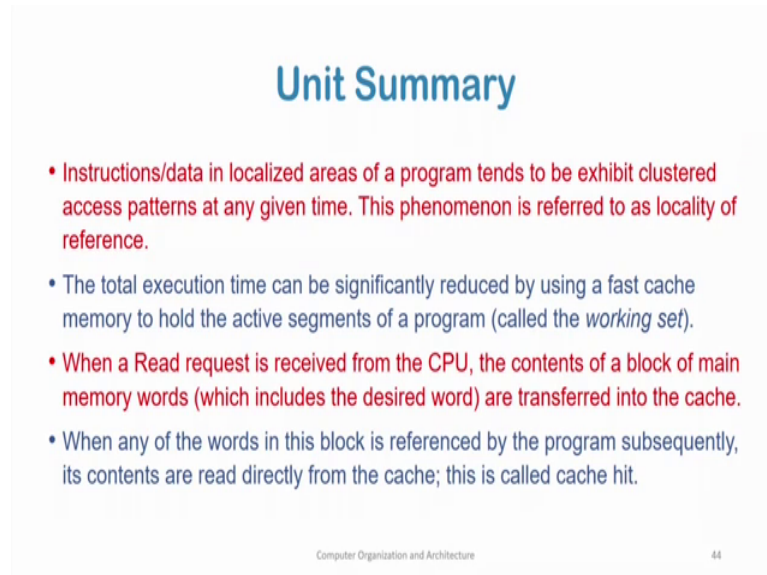
- A memory cell is capable of storing 1-bit of information.
- A number of memory cells are organized in the form of a matrix to form the memory chip.
- Register, cache memory and main memory are referred to as internal memory. They are semiconductor memories. They may be volatile (Cache, RAM) or non-volatile (ROM).
- Magnetic Disk, removable media etc. are external memories. They are non-volatile.

Computer Organization and Architecture 43

With this we come to the end of this unit the summary. In summary what we studied in the unit is as follows. A main memory cell is capable of storing 1-bit of information. A number of memory cells are organized in the form of a matrix to form the memory chip, Register, cache memory and main memory are referred to as internal or inboard memory.

These are semiconductor memories. They may be volatile, for example, for caches and RAMs or non-volatile in case of ROM. Magnetic disk removable media etcetera are external memories. They are non-volatile.

(Refer Slide Time: 32:46)



The slide is titled "Unit Summary" in blue text. It contains four bullet points: 1. Instructions/data in localized areas of a program tends to exhibit clustered access patterns at any given time. This phenomenon is referred to as locality of reference. 2. The total execution time can be significantly reduced by using a fast cache memory to hold the active segments of a program (called the working set). 3. When a Read request is received from the CPU, the contents of a block of main memory words (which includes the desired word) are transferred into the cache. 4. When any of the words in this block is referenced by the program subsequently, its contents are read directly from the cache; this is called cache hit. At the bottom, it says "Computer Organization and Architecture" and "44".

Unit Summary

- Instructions/data in localized areas of a program tends to exhibit clustered access patterns at any given time. This phenomenon is referred to as locality of reference.
- The total execution time can be significantly reduced by using a fast cache memory to hold the active segments of a program (called the *working set*).
- When a Read request is received from the CPU, the contents of a block of main memory words (which includes the desired word) are transferred into the cache.
- When any of the words in this block is referenced by the program subsequently, its contents are read directly from the cache; this is called cache hit.

Computer Organization and Architecture 44

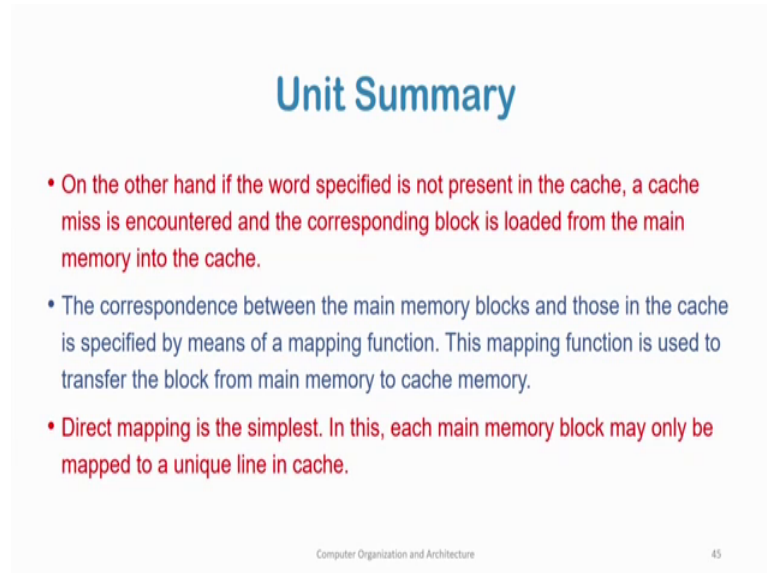
Instruction slash data in localized area of a program tends to exhibit clustered access patterns at any given time. This phenomenon is referred to as the locality of reference. The total execution time can be significantly reduced by using a fast cache. So, the total execution time of a program can be significantly reduced by using a fast cache memory to hold active segments of a program which is called the working set in OS parlance.

So, basically, because the memory is slower than the processor, we can to make things faster the processor has to wait for data to come from the memory to make things faster we can put a fast cache, which will hold active segments of the memory. We can only hold the active segment or not the whole program because this fast cache memory is very expensive and we cannot have a very large cache. Also when the when the size of the memory tends to grow the access times also tend to grow.

When a read request is received from the CPU, the contents of a block of main memory are transferred to the cache which includes the desired word. When and we were and we bring a block instead of a single word from the main memory to take advantage of the locality of reference. Due to which subsequent accesses may be near the vicinity of this memory word and therefore, subsequent accesses may result in hits. When any of the

words in this block is referenced by the program subsequently its contents are read directly from the cache and this is called cache hit.

(Refer Slide Time: 34:54)

A slide titled "Unit Summary" with three bullet points. The first bullet point is in red text, the second is in blue text, and the third is in red text. At the bottom, there is a footer with the text "Computer Organization and Architecture" and the number "45".

Unit Summary

- On the other hand if the word specified is not present in the cache, a cache miss is encountered and the corresponding block is loaded from the main memory into the cache.
- The correspondence between the main memory blocks and those in the cache is specified by means of a mapping function. This mapping function is used to transfer the block from main memory to cache memory.
- Direct mapping is the simplest. In this, each main memory block may only be mapped to a unique line in cache.

Computer Organization and Architecture 45

On the other hand, if the word specified is not present in a cache, a cache miss is encountered and the corresponding block is loaded from the main into the cache. The correspondence between the main memory blocks and those of the cache is specified by means of a mapping function. This mapping function is used to transfer the block from main memory to cache memory. Direct mapping is the simplest. In this each memory block can only be mapped to a unique line in the cache. There are other more complex forms of mapping as fully associative mapping and set associative mapping which we will study later.

With this we end the unit 1 of the module memory system.