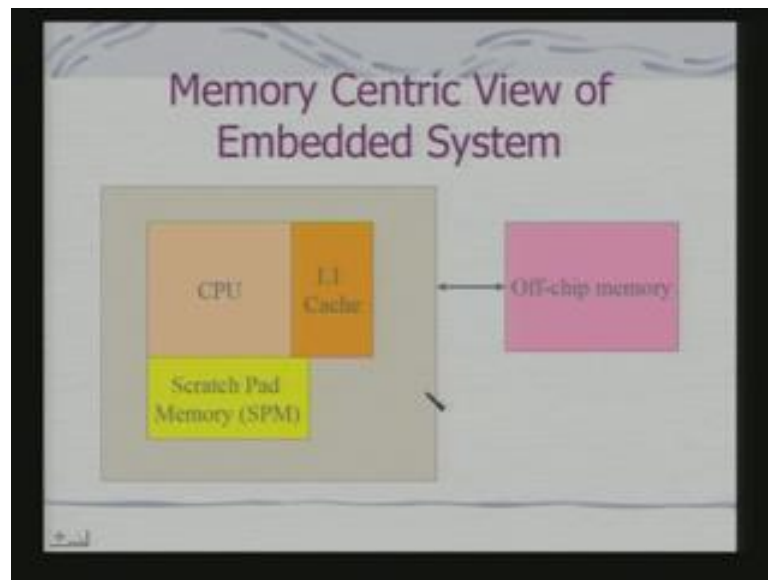


**Embedded Systems**  
**Prof. Dr. Santanu Chaudhury**  
**Department of Electrical Engineering**  
**Indian Institute of Technology, Delhi**

**Lecture - 12**  
**Memory Organization**

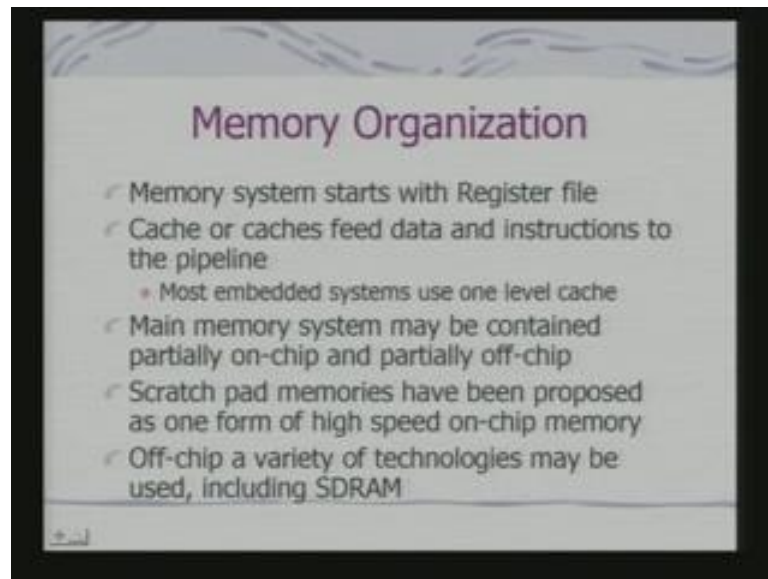
In the last class, we had discussed different types of memory and how they can be interfaced with microcontrollers like ARM. In today's class, we shall look at the memory organization, how these different types of memories are used to setup the complete memory system in an embedded system.

(Refer Slide Time: 01:47)



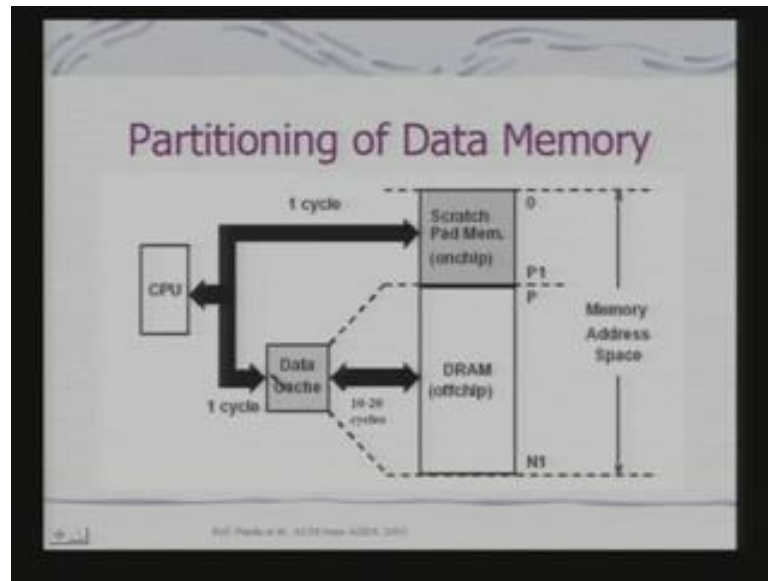
So, let us have a look at what we call a memory centric view of embedded system. We have the CPU here and this is which is typically would be SRAM static RAM and this is a level 1 cache that is why we call it a L1 cache ray. There is another ready of memory which is also on-chip and this is called scratch pad memory or SPM. This also would be typically RAM and not DRAM off-chip memory is the memory which is outside that of the chip foundry. So, these are the very basic components. Now, what we shall look at is, what are these components, how they can real be organized and used?

(Refer Slide Time: 02:42)



So, memory system in an embedded appliance starts with the register file, which is typical to that of the microcontroller which is being used in the embedded system. There may be one or more cache memory that fed data and instructions to the pipeline, because most of a microcontrollers that we have studied at pipelined processors. Main memory system may be contained partially on-chip and partially off-chip. In fact, scratch pad memories have been proposed as one form of what you call high speed on-chip memory. Off-chip memory can include a variety of technologies that is memory made with a variety of technologies including synchronous DRAM. Because synchronous DRAM gives you first data access particularly in burst mode synchronous with the system or bus clock.

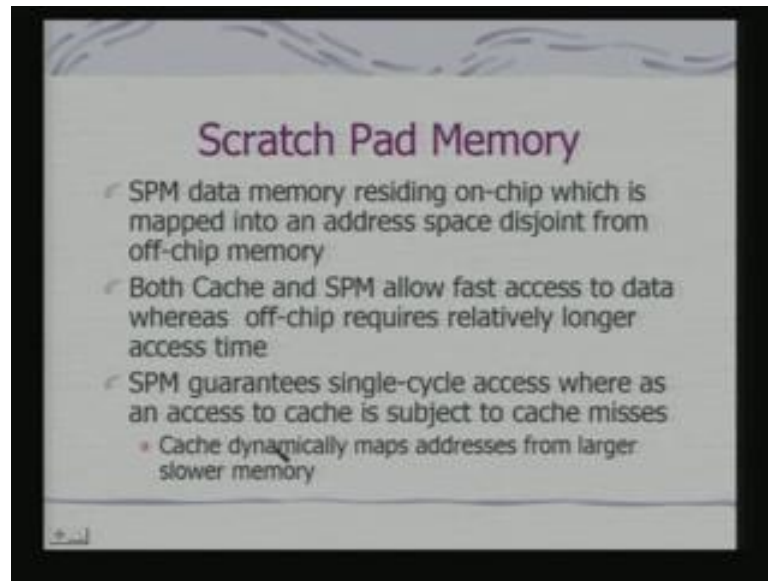
(Refer Slide Time: 03:53)



So, if you look at the memory map of the processor this is a complete memory map and this scratch pad memory is basically on-chip memory and you have seen that different processors have different amounts of on-chip RAM. So, these we can think in terms of a scratch pad memory. These memory DRAM is typically off-chip memory which occupies a portion of the address space, memory address space of the processor, but it is typically resident outside that of the chip. This is what we have shown as a data cache now it may have instruction as well as data. So, I have got an unified cache, there may be separate data an instruction cache.

So, that will be case with ((refer time: 04:54)) of the architecture. So, here we are simply showing at data cache and what is instructing that you will find that this data cache is mapped 1 to the address space of the DRAM that is typically the feature of cache memory. So, in terms of access speed your scratch pad memory should be typically synchronized with the one cycle, that is same cycle clock cycle as that of a CPU. So, that accessing this memory does not meet wait state. In case of DRAM if we have shown here typically 10 to 20 cycles. So, it might require wait state and that is why that memory is expected it to be mapped on to data cache depending on your cache management policies.

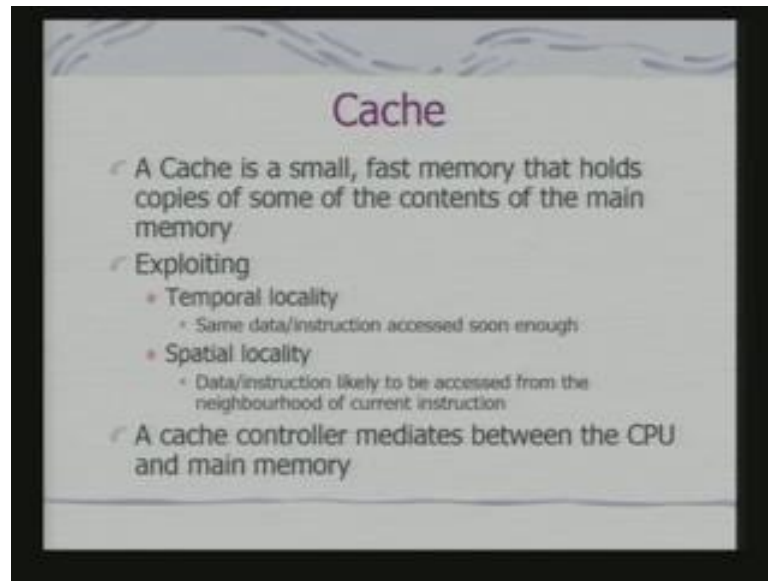
(Refer Slide Time: 05:50)



So, what is really therefore, scratch pad memory and what is the distinction with the cache? SPM data memory residing on-chip and it is mapped into an address space which is disjoint from off-chip memory. In as we have seen in the diagram both cache and scratch pad memory allows fast access whereas off-chip requires longer access time. But what is important to note is that scratch pad guarantees single sizes a single cycle access whereas access to cache is subject to cache misses because I got a small cache memory and the complete DRAM area is mapped onto cache area. So, obviously, the entire DRAM is not stored in cache.

So, there would be cache means when there is a cache means; obviously, my CPU is not accessing data in 1 cycle. So, this is not guaranteed 1 cycle, but since SPM is occupying, apart of the address space of the processor itself. So, the access to SPM is guaranteed to be completed in 1 cycle. So, the basic advantage of using SPM on-chip memory which is built with high speed RAM. The basic advantage is that it is a guaranteed access if a location is map to SPM there will be always a guaranteed access. So, typically your critically routines are mostly access routines is expected to be stored there as well as data which is to be used very frequently or to be used in a compute intensive fashion should be accommodated in SPM.

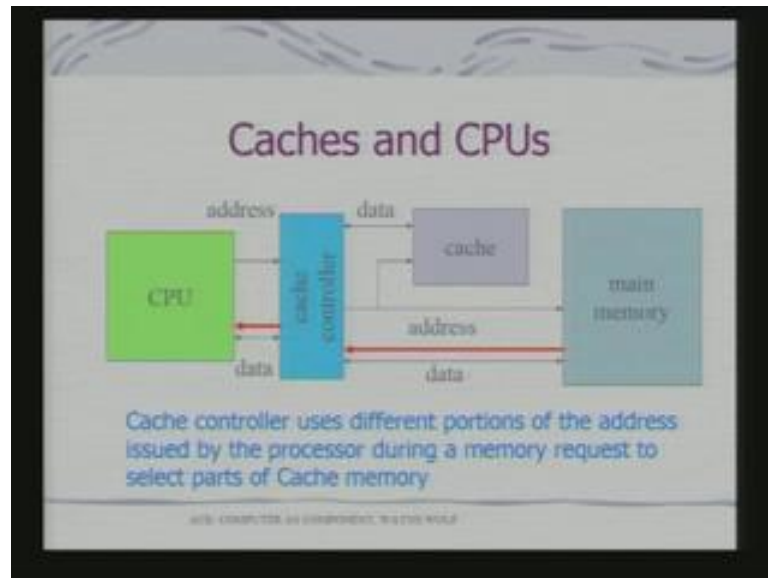
(Refer Slide Time: 08:01)



So, cache as I have already told you is a small fast memory that holds copies of some of the contents of main memory. And not definitely the entire content and the whole logic of having cache and today you will find cache many of these embedded processors and particularly embedded microcontroller chips. There is one chip cache they are being used to speed up the access exploiting what you called temporal locality as well as spatial locality. The basic principle of temporal locality is that if we are accessing at data are an instruction now it is very likely that I shall be accessing that instruction soon. So, once an instruction or a data element being loaded into the cache what is likely that CPU next time will access that the data are instruction from cache and hence that access will be much faster.

Associated with this is a principle of spatial locality; that means, when you are loading and data item are an instruction into the cache. You typically load not just a single instruction or a single did element. But a block of data element or a block of instructions which are located near to the current instruction in the memory address space, because it is very likely that these instructions would be currently needed for execution. So, typical example could be a loop. So, if you are executing a loop its very likely that as set of instructions will be executed which are in the neighborhood of the current instruction in the memory address spaces and these instructions would be accessed frequently in recent times. Therefore, they also satisfied the property of temporal locality. In fact, a cache controller mediates between the CPU and main memory.

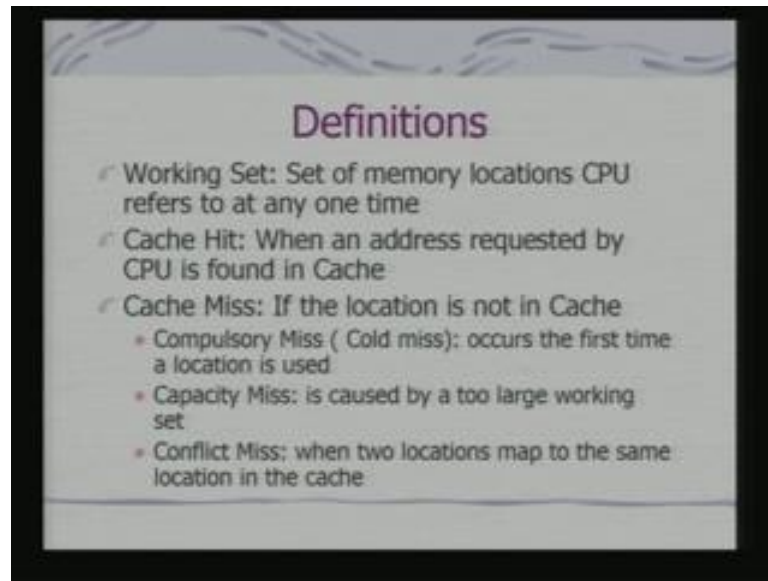
(Refer Slide Time: 10:22)



So, pictorially it looks something like this we have a CPU and we have cache controller. We have these as cache and this is your main memory. So, according to this model so, whatever address the CPU generates that is processed by the cache controller to find out if the data is typically in the cache or not and if it is shown if the data is there in the cache. The data would be passed on the CPU or there is a need to access main memory. So, data from the main memory is needed to be loaded onto the cache on the same time may be passed onto the CPU. So, basic sequence of operations if you look see this to CPU generates the address depending on where it is located either on the cache or main memory cache controller, because that out and when it gets a data in the cache that is returned.

If it is not sure what will happen is you have to get the data from the main memory and other same time the data gets loaded also into the cache. So, when you get the data in the cache what we say is cache hit when you do not get the data in the cache what you what happens is cache miss. So, when it is a cache miss; obviously, processor is not accessing data or an instruction in a single clock cycle. And therefore, what we say that when you will cache the performance of the processor is not completely predictable. So, cache controller uses different portions of the address you should by the processor during the memory request to select parts of cache memory and how it uses this addresses depends upon the organization of a cache that is being adopted. So, let us look at some of these definitions.

(Refer Slide Time: 12:26)

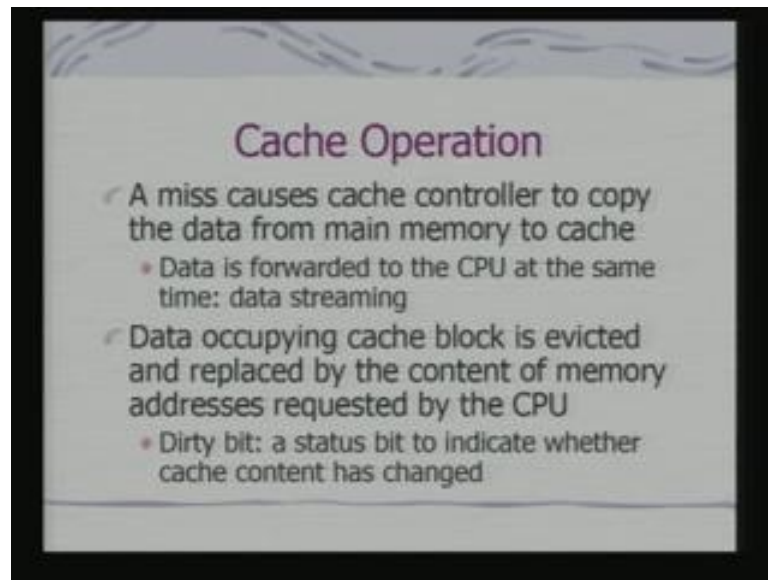


Working set; what is really working set is a set of memory location that is a CPU refers to at any 1 time. This one time; obviously, has associated with a concept of granularity that is a sequence of instructions is what is being refer to as one time cache hit. As I have already explained happens when an address requested by CPU is found in cache. The cache miss is when the location is not in cache. There are different types of cache miss the first one is called compulsory miss or cold miss it occurs the first time a location is used. So, whenever a location is effort in the memory. It has not been used before; obviously, that data is not expected to be cache, because a cache management policy is typically follows load only marked. So, that is what we call compulsory miss or cold miss? Capacity miss occurs when you have a too large working set that means, you accessing a set of instructions frequently.

But these set of instructions is large enough such that it cannot be accommodated completely to the cache. This may happen with a data area as well you may be processing a large enough a data block which cannot be accommodated in the cache. That means, that your working set is not small enough to be accommodated in the cache that you have implemented that is what is known as capacity miss. Conflict miss is when two locations map to the same location in the cache should happen, because you are mapping a DRAM area the main memory area which is pretty a large on to a small cache. Therefore, whatever be the cache in policy you follow and number of locations from the main memory could be mapped on to a single location in the cache. So, if you

are accessing two such locations there would be a conflict and cache miss. So, if you look at now cache operation when a miss occurs. The miss can be any one of these types of miss it would cause cache controller to copy the data from main memory to cache.

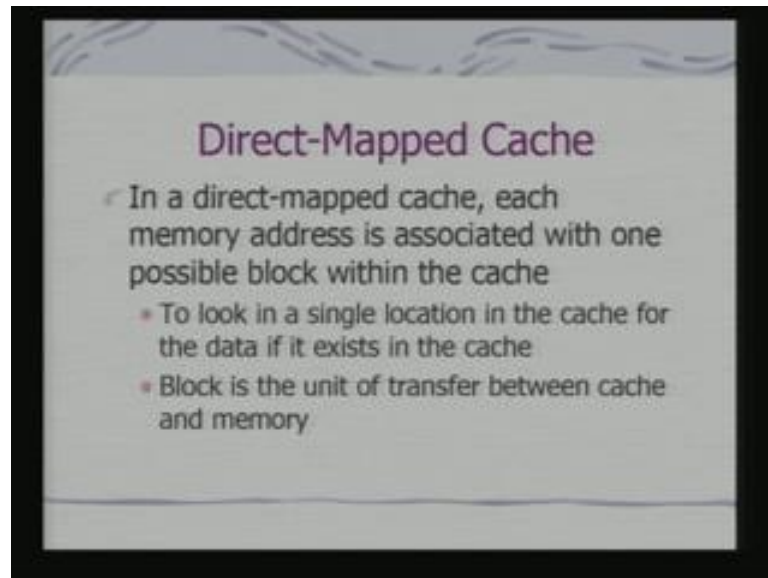
(Refer Slide Time: 15:28)



Data is forwarded to the CPU at the same time and this is this phenomena is known as data streaming. And if that block is already occupied then that data has to be evicted out and replaced by the content of memory addresses currently requested by CPU. And when such a thing happens you load just not a single location content of a single location, but content of a block to exploit what you call spatial locality. So, when you are evicting what you have to take care of we have to take care of the fact whether the data which was contained in the cache has been modified or not. If it is has been modified I need to figure out a mechanism to store back that data into the main memory or DRAM memory. So, for that purpose typically you associate a bit which is called dirty bit. It is a status bit associated with the cache memory to indicate whether the cache content has been changed or not. So; obviously, if the dirty bit is set; that means, the block has to be return back to the memory otherwise you need not write back an; obviously, write back would imply a performance over hidden. Select see different causing mechanism first we shall look at simplest causing scheme which is direct mapped cache.

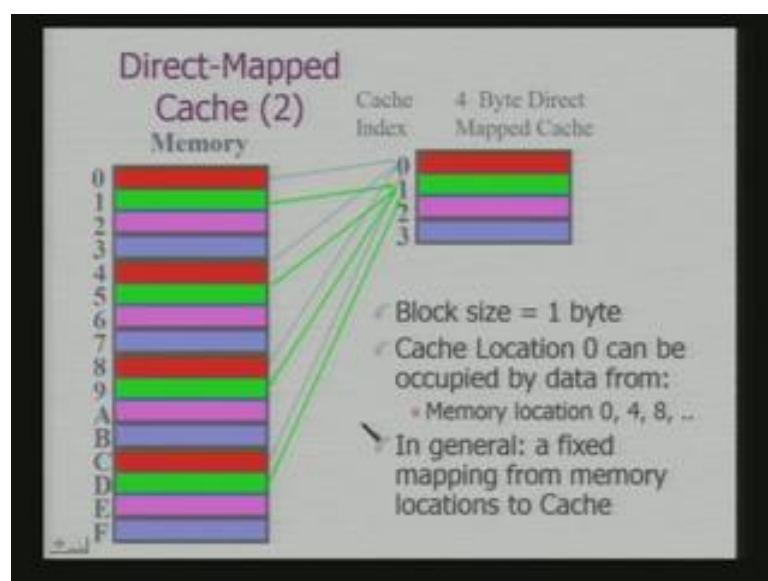


(Refer Slide Time: 16:53)



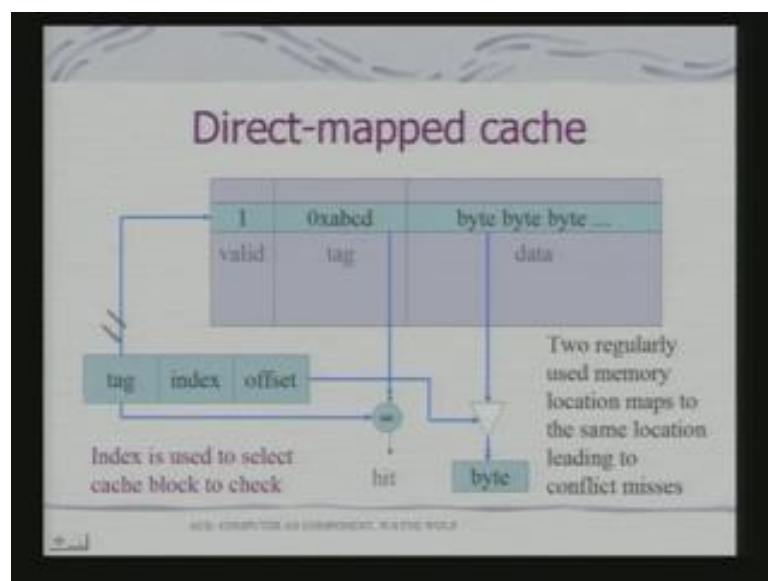
So, in direct mapped cache each memory address is associated with one possible block within the cache. Therefore what happens to look for a data or look for an instruction the cache controller needs to loop at a single location to find out whether the data or instruction exists in cache or not. I mean in fact, what we typically say that the whole transfer takes place in terms of block. So, you search for a block at a fix location in the cache. So, this is a very straight forward as set of memory locations to a single block in the cache. So, when a particular memory is accessed and that data is not there in the cache you fetched that data from the memory and load it at that location.

(Refer Slide Time: 17:56)



So, let say this is my memory this is just as called 16 locations and you have got your cache which has as got 4 locations. In a direct mapped cache what will happen is there is a mapping which is already defined. So, in this case I define a mapping that 0 goes to 0 4 goes to 0 4 goes to that is fourth location goes to 0. Similarly, that the every multiple of 4 that 0 goes to 0 th location 1 goes to 1 location 2 goes to the second 3 goes to 3 and now fourth location goes to 0. Similarly, your 8 location goes to 0. So, you can see that multiple locations a getting map to a single cache block. Now; obviously, what will happen is in these case when we are accessing a memory locations a 0 if these block is empty will load these data onto these block and not to any other block. So, it is very easy for the cache controller to find out whether the data is clear in the cache. So, here the assumption if you work with simple assumption that block size is 1 byte the cache location 0 can be occupied by the data from memory location 0 4 8 and so on. And in the general case a fixed mapping from the memory locations to that of the cache. So, how is it really done or how is really implemented by the cache control?

(Refer Slide Time: 19:28)

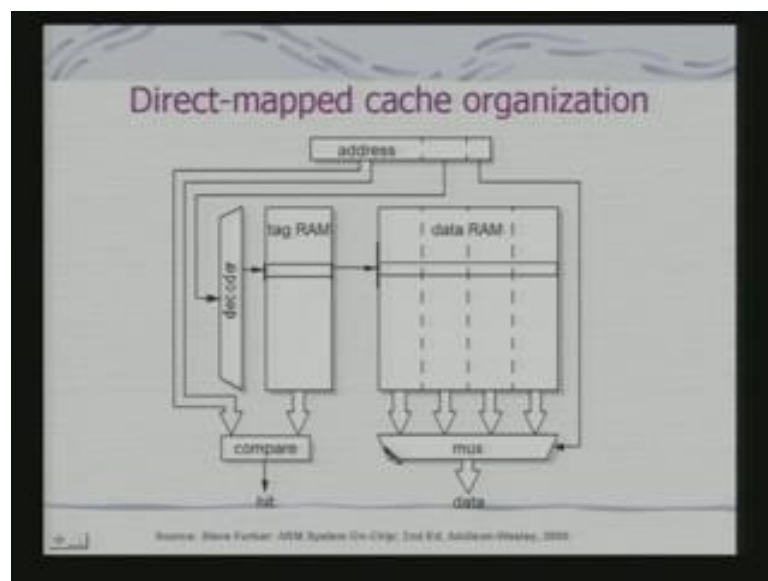


For that purpose actually the address that is being generated by the process service logically divided in to these blocks. We call cad the tag index offset block. So, this is the division of the memory address that is being generated by the processor. So, what you have this is your cache block in the cache block what you store you store data definitely. And there can be multiple bytes of data stored here in a block, but along with you stored the tag and store another bit of valid bit which tells you whether the cache block contains

really the data from the main memory or it is just a junk index I told you that address is divided into 3 portions. This is your index; index is used to select which cache block to check. In fact, there may be multiple such cache blocks. The tag information the tag part of the address is compared with the tag part of the cache memory which is already stored. So, if these two tags do match; that means, there is a cache hit if there is a cache hit the new get the value. So, this is what this is what happens in a direct mapped cache.

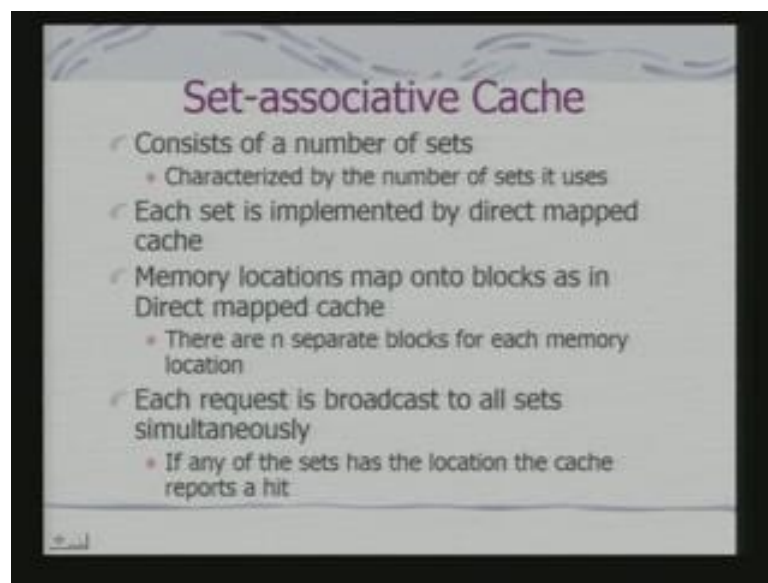
So, you realize the two regularly used memorial memory locations. If they maps to the same location then that would be to what you call conflict misses. So, this direct mapped cache suffers from the problem of what you call conflict misses because there are these locations, multiple locations mapped to a single location in cache. So, then if cache is other locations of cache is empty if I access these locations regularly then they will be a conflict miss and the worst case performance be really back. So, this is what we are trying to show here. So, you have a tag. So, this tag using the index I have got to a particular block looking at the tag I come to other tag. And then on the basis of that I sell ci whether it is a heat or not if these equals then there is a heat and then I shall get the data which is a byte. Now, this logic is what is implemented by the cache control. These logic is what is implemented by the cache control along with other things as well the whole policy part as well.

(Refer Slide Time: 22:08)



So, this is an example with respect to ARM architecture. So, if you got an address. So, the address part if you see that this is the tag part the index part is directly being fed to the decoder to select the cache block. Then what you have doing there is a tag RAM apart from, actually the data RAM which contains multiple bytes that is a block as a whole. So, if you got a tag RAM. So, this tag RAM is used to get the tag and this part is compared with this part of the address. So, this tells your hit and accordingly the data would be provided. So, this heat will be used for enabling the appropriate input line for the multiplexer.

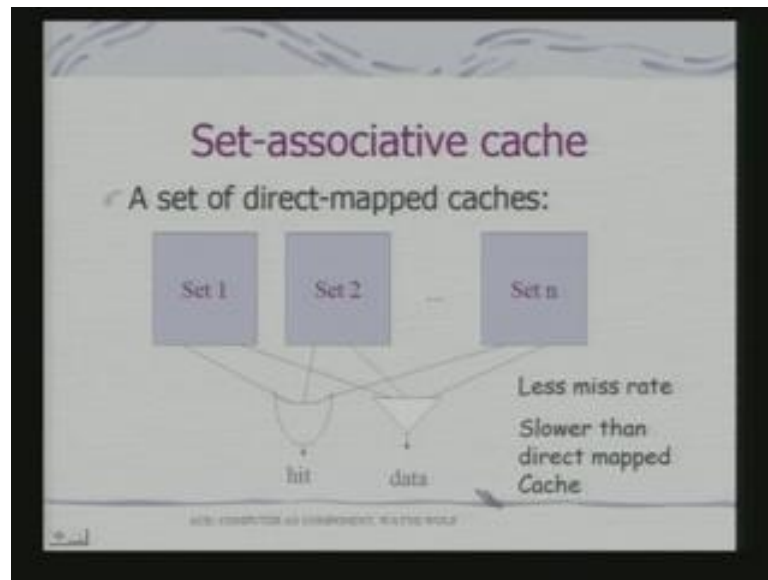
(Refer Slide Time: 22:57)



Next we get what we call Set-associative Cache. So, the basic issue of the Set-associative cache is to remove the problem of conflict misses that you frequently encounter in your direct mapped Cache. A set associative cache characterize by a number of sets there is not a single set or a single block. There are number of sets and each set is actually mapped or implemented by direct mapped cache mechanism that we have show for discussed. So, memory locations map onto blocks as in direct mapped cache and there are  $N$  separate blocks for each memory location. So, since there are multiple cache blocks or cache sets therefore, a memory location now gets mapped onto  $N$  different blocks. Because you have used  $N$  different cache sets or cache blocks and each request for an address it broadest it is broadcasted to all sets simultaneously if any of the sets has a location the cache reports a hit. So, I hope you have understood why these arrangements minimizes conflict misses, because multiple locations now gets mapped

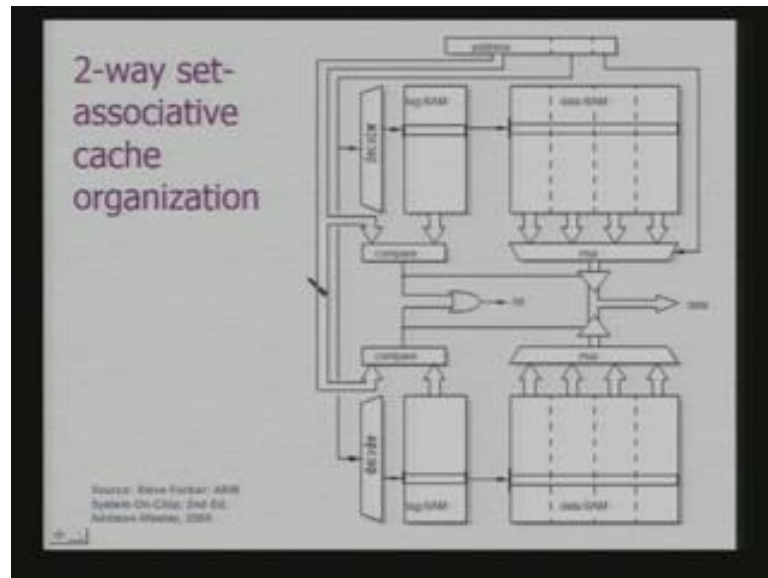
onto not a single location, but onto different cache blocks. So, even if we are accessing as these kind of locations which are having identical mapping on a regular basis. It is very likely that I shall find them in different cache blocks and I shall not generate cache miss false.

(Refer Slide Time: 24:57)



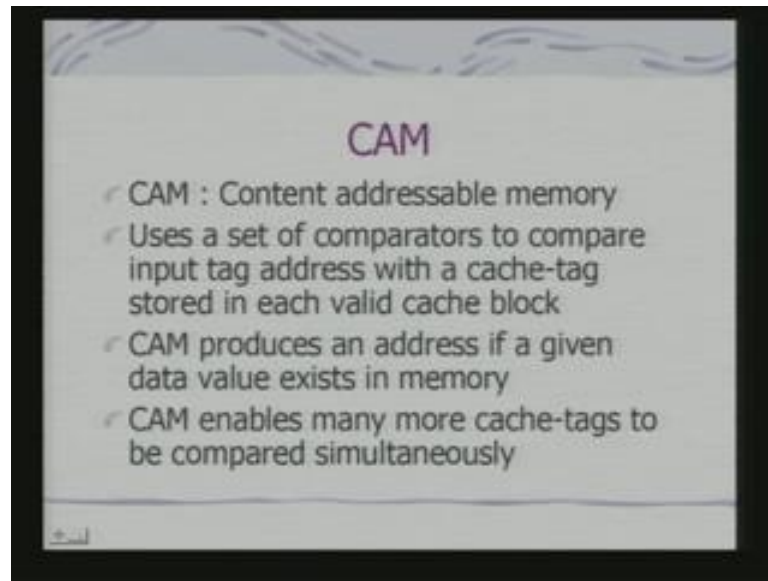
So, let us see its actual implementation. So, I have got a set of direct mapped cache these are different sets. So, my hit can come from any of these sets because a a given memory block gets mapped onto a block in this cache and since there are N such that is. So, it can get mapped 1 to any 1 of these blocks in any 1 of the sets. So, the hit logic will come from any 1 of these blocks. So, it has got less miss rate compare to that of your direct mapped Cache, but obviously, it will slower than direct mapped cache because you have to now take him to account. So, many comparisons.

(Refer Slide Time: 25:41)



In a typical ARM implementation if you have two way set associative cache you see that this organization is very similar. Just like that of your direct mapped cache instead of 1 block an away for 2 blocks. This is a 2 way that is why we are calling it 2 way set associative cache. So, the decoder the same index part is used to find out the block and these tag is know compared with the address path. So, it can be either here or it can be in these block. So, I do the comparison. So, there are two comparator blocks depending on the output of the comparison I enable the corresponding output of the multiplexer. So, one of them will have the data. So, I enable and the control signal for these multiplexer effectively comes from the result of these comparisons. So, this is how a two way set associative cache is implemented; obviously, your miss error in this case is expected to be much less.

(Refer Slide Time: 26:53)

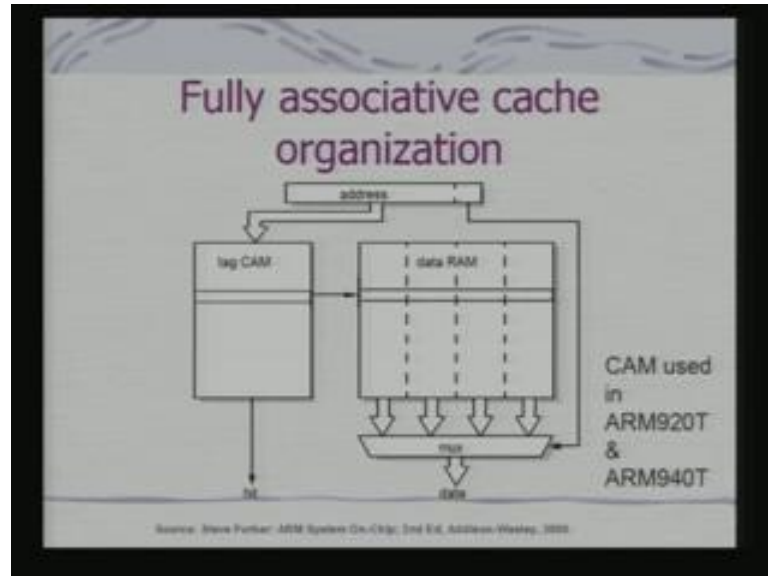


The other way of really looking into the cache is or increase its efficiency is to use what you call CAM or content addressable memory. Now, in a way CAM implements of fully associative cache; that means, without really mapping each location of the memory to one location in the cache instead of doing that in an intelligent way CAM implements almost a fully associative cache. The basic idea is that it uses now we can think in these terms that we have got a set of comparators and that comparator compares the input tag address with a cache tag store in each valid cache block. So, I am doing many such comparisons in ((refer time: 28:02)). Now, effectively what does that mean that if I can do that comparison an if that comparison is successful. I would like to have on the basis of the comparison the actual address. So, the organization now is slightly different. So, in principle if you just consider that I have got two way associative cache an if I increase the number that number of the sets that we using.

We increase that number effectively what I will getting a location can be mapped to any of these sets. So, the miss rate will be much less, but we cannot really have that much hardware on-chip, because having that hardware would essentially imply area as well as energy of a ray. So, you use what is called a CAM. So, CAM works effectively like that of your RAM, but slightly in a in a different way. CAM produces an address if a given a data value exists in memory. So, that is the basic reason why we call CAM a content addressable memory. So, the basic idea is that I take the tag just consider in this way. So, the moment I take the tag if I can use the tag to get the address which is really stored in

the cache and I use the tag to generate that address in the cache. I have effectively done almost a fully associated memory. So, let us see how it gets implemented.

(Refer Slide Time: 29:50)



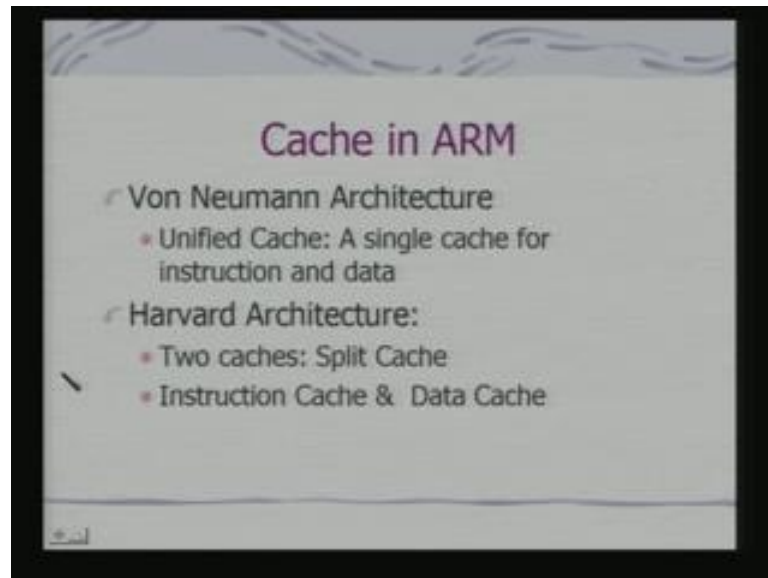
So, you have got the address now this address is fed to what is called a tag CAM fine. So, once it feeds to tag CAM so, this is something like your memory itself, but it takes in these as input and produces address; address of what address of the location in the cache which contains the data of these location. So, now it becomes fully associated we can see why what was the reason of going into N way set associative. We are trying to approximate the requirement that I would like to map any memory location to any free location in the cache that is my target. Now, if I have to do that using simple comparators have do what compare tag of each and every location of the cache. So, becomes a sequential comparison and that will not really help me in first access. So, to overcome that problem we are suggested N way associative cache; that means, I map to N such that and find out which set as got the data from my target location.

In case of CAM what we are doing your just reorganizing the whole RAM that is your tag RAM as well as your cache. So, you are using this address part this address part what we are showing here is a combination of both your tag as well as the index together. You are using that you can consider this as this is an address which is going into these tag can which is a content addressable memory. So, depending on these address what the tag CAM is generating it generating the address of location in the cache. So, effectively we



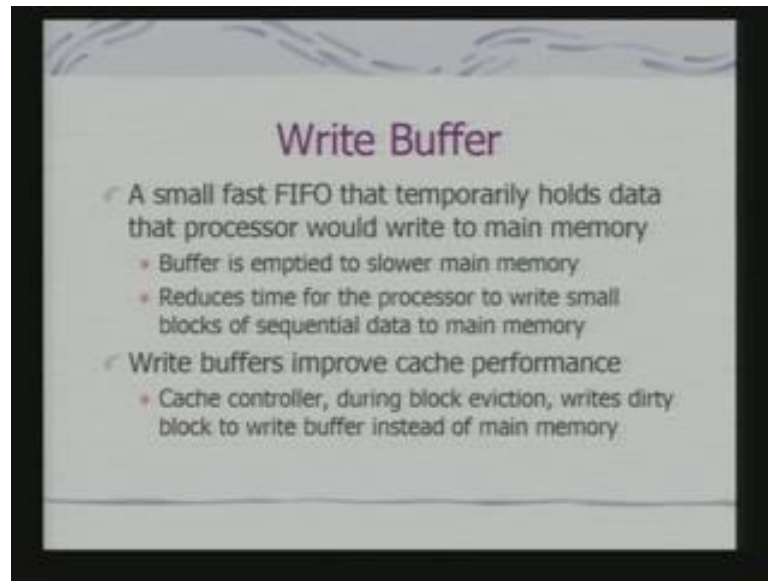
are doing many such comparisons in parallel, but instead of using many such comparators. We have intelligently use the address of these memory to generate address of the corresponding location in the cache. So, CAM in that way is an efficient organization and CAM is used in architectures ARM architectures like 920 T ARM 940 T. These are for cache implementations they use this kind of CAM based architecture.

(Refer Slide Time: 32:40)



So, if you look at now cache in a microcontroller. So, actually in a microcontroller core how is the cache really used? If I am having a Von Neumann Architecture; obviously, I shall be having an Unified cache a single cache what instruction as well as data. For Harvard Architecture which shall have 2 cases. So, these we call Split cache organization. I have Instruction cache and Data cache separately.

(Refer Slide Time: 33:17)

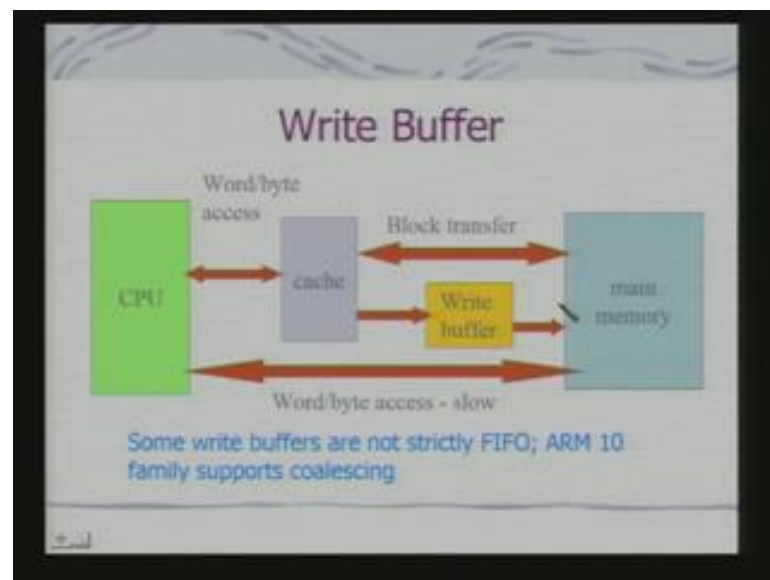


A small fast FIFO that temporarily holds data that processor would write to main memory is used in many a times along with cache. Now, why these buffer is used the basic idea of using this buffer is if you remember I talked about the basic problem of writing. So, when I am writing it back. So, write back me require a kind of an overhead. So, in order to take care of that overhead many a times this processors including ARM have got one chip write buffer which is the first memory, but it is not same as that of cache. So, buffer is emptied to slower main memory and it reduces time for processor to write small blocks of sequential data to main memory. So, instead of writing the data directly onto the main memory what processor does is that it writes data onto the write buffer. And from the write buffer the data is stored back on to the memory of the processor.

Obviously write buffers improve cache performance because cache controller during block eviction writes dirty block that is the block for which the dirty piece is set to the write buffer instead of writing in fact, to the main memory. So, processor is not suffering because see if should understand that while this write back is taking place from a dirty block of the cache to the main memory. The processor has to wet when there is a miss and a miss requires a block eviction. So, if I can use a write buffer which is the first memory. So, that eviction will take place first. So, processor now needs to wet only for the time required to load the data from the main memory to the cache another same time data is to being streamed in. If I would not have got a write buffer the time taken would

have been double when there is a block eviction because if I if I consider that access time for read and write is similar. So, if you have to first write back a block and then you up to read the block and after that only the processor can really start execution. So, that time can be reduced by using this write buffer.

(Refer Slide Time: 36:29)

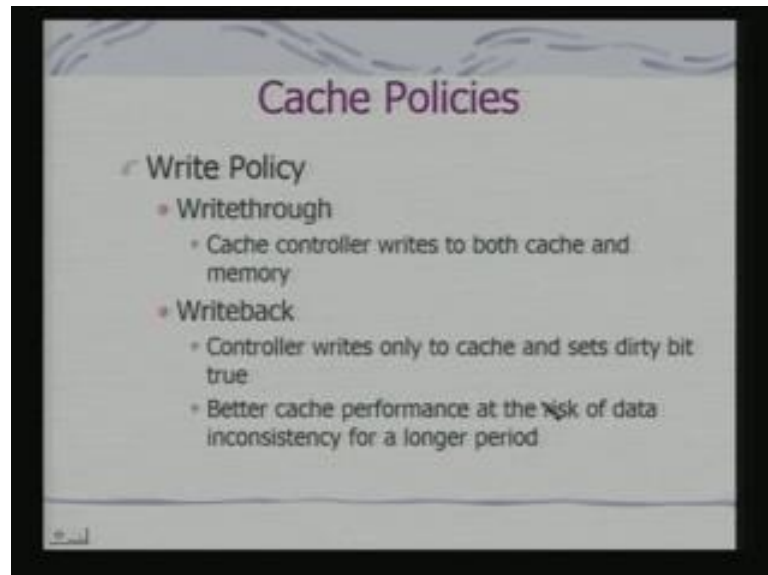


So, pictorially you have got a write buffer at this path I am not shown the other path from CPU. CPU can also have 1 path directly on to the write buffer if it is required and you have got this cache. So, cache transfer. So, this is the typical scenario which would take place. This would take place when there is transfer of the data from cache to main memory. In write buffers, in many cases are implemented as FIFO memory you have seen hardware stack memory where in peak where you can push the PC in case of a subroutine call or interrupt. This is an example where in many cases you have hardware FIFO memory so effectively a queue, why it is a queue? Simply, because I am storing the data sequentially. In many architectures these write buffers are not also implemented as a queue, but in a different way.

So, say for example, ARM 10 family supports what is called coalescing; that means, you can write the data and they are grouped together. And then the group for a block is transferred from the write buffer to the main memory. Now; obviously, the issue that comes up in this context is that the data in the main memory and the data in the write buffer or; obviously, not consistent why the data might have changed and that data is

written into the write buffer and the data in the main memory is not yet updated. So, that is an important concern when programming the system.

(Refer Slide Time: 38:20)

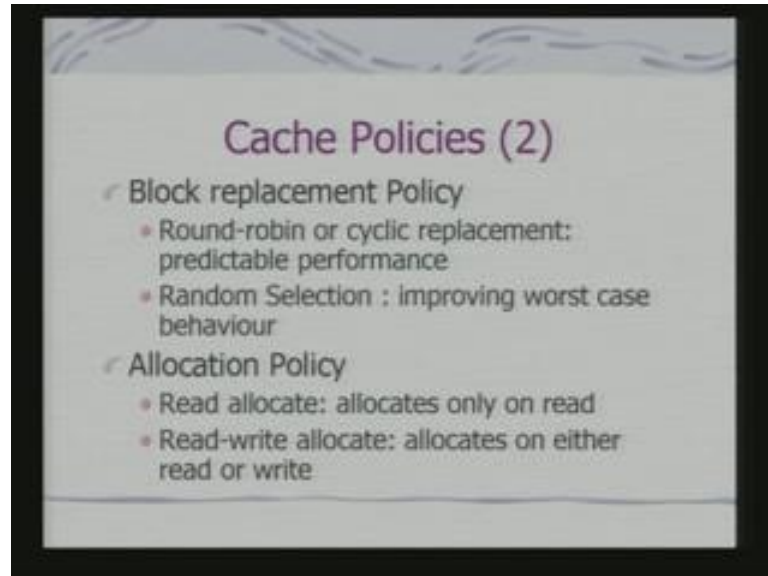


So, now; obviously, all these issues take as to different kinds of cache Policies which are to be adopted. So, write Policy I can have writethrough or writeback. So, if you really do not want to have inconsistency as a problem. I would like to implement what you call or writethrough policy in a writethrough policy cache controller writes to both cache and memory at the same time of variant of these policy would be writing to cache as well as to the write buffer. So, that that can be emptied later on, but it will not wait; that means, cache controller is not waiting for the write process to be initiated till a block eviction takes place. Obviously this writethrough policy would have overhead when you are writing onto the main memory and if you are frequently writing that can really slow down your execution.

Now, you should keep in mind also the other fact that of energy consumption because accessing of main memory that is you DRAM will consume more energy many cases because cache that although their static RAM. They are also in many cases optimize in terms of energy requirement for access in case of writeback the policy that we were taking about an writeback policy really requires these dirty bit. And in a writeback you actually wait till the block eviction takes place. When the block eviction takes place then only you will writeback the data either to the memory or to the write buffer as a case

menu. Obviously these as a problem I will be discussing earlier that of inconsistency of the data for a longer period.

(Refer Slide Time: 40:28)



Next cache block replacement policy if you are talking about a block eviction which block is to be evicted that is which block is to be replaced, because you may find that if you are talking about in way set associative cache. So, there can be  $N$  such locations where a set of memory locations can be accommodated. Now, block eviction would take place when  $N$  such blocks are filled up. So, for eviction the cache controller has to decide which block is to be evicted has to decide which block has to be evicted. The simple policy is round robin or cyclic replacement now; that means, the block which is loaded currently will; obviously, not be replaced. So, you go through a round robin procedure here and you start from the current block and you go back and find out the next block which is in line to be replaced and you replace it.

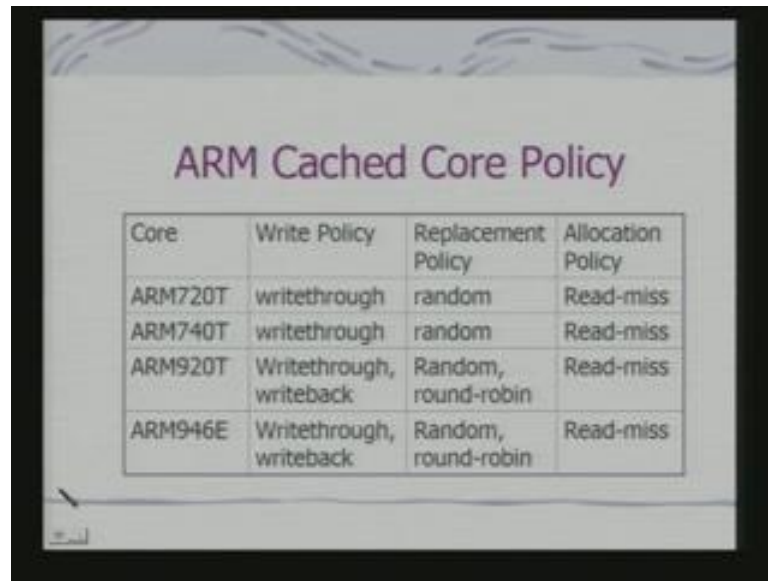
Now; obviously, this round robin or cyclic replacement gives you a predictable performance because you exactly know the sequence in which the blocks are to be replaced. But it can also have very bad worst case performance why because if you have say  $N$  blocks and these  $N$  blocks have been accessed and it is arranged in such a way. The next block that you will access becomes the next block according to the round robin policy to be replaced. Then you will be actually doing a fraction that is your loading a block onto the cache again evicting a block from the cache. So in fact, the round robin

policy can give rise to a very bad worst case performance for the cache. So, that is why in many cases this random selection is another policy which is implemented now you can realize that when you are talking about these block replacement policies. These policies have to be implemented in hardware in the cache controller. So, really I do not have an option of going into very sophisticated policies.

So, a block replacement using round robin is a simple of counter that is I start with the value and a modulus counter. Simply a modulus counter will give me an implementation of a round robin and a random selection is a random number generated effectively within a block that can also be very easily implemented in hardware. Then next thing is allocation policy when is really a cache block allocated; that means, when is that block from the main memory is loaded onto the cache. So, this is what is known as your allocation policy. So, you have typically a read allocate and read write allocate. A read allocate policy means that allocation of the cache takes place only on read when it is a write access, a write access may not really mean that that location would be accessed again and typically it is true when you are looking at a unified cache.

So, your executing a sequence of instructions. You might be writing the result on to a location in the memory. So, that may be written only once. It may not be true that the locations around that write memory area would be accessed very frequently in a future. So, read write allocation policy only tries to take care of the fact that if you are really writing onto that location; that means, the data area you are likely to access. So, very simple example could be an array access. So, if you are likely to modify one element in that array you are very likely to modify other elements of that. So, in that case even when there is a write operation. You would like to allocate that block in cache otherwise a typical instruction execution would necessitate a read allocate policy. Now, these policies are always a part of what I called hardware policies implemented in the hardware.

(Refer Slide Time: 45:24)



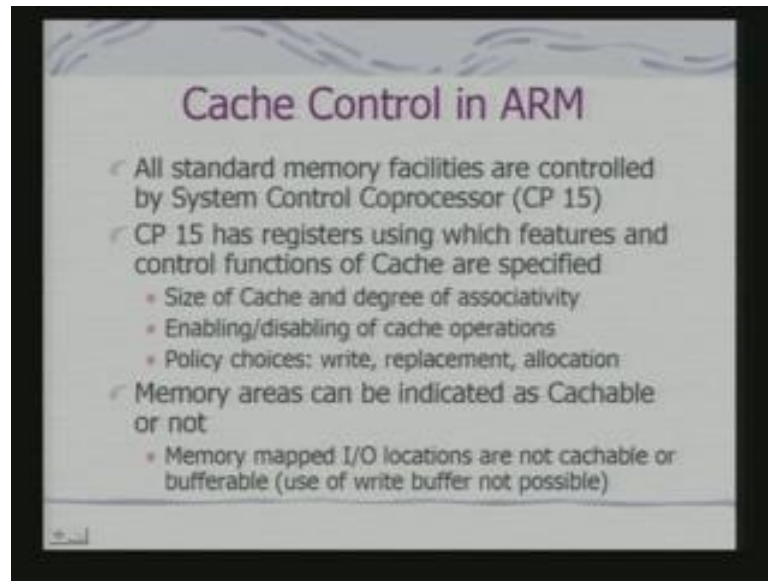
The image shows a slide titled "ARM Cached Core Policy" with a table listing the policies for four ARM cores. The table has four columns: Core, Write Policy, Replacement Policy, and Allocation Policy. The rows correspond to ARM720T, ARM740T, ARM920T, and ARM946E.

Core	Write Policy	Replacement Policy	Allocation Policy
ARM720T	writethrough	random	Read-miss
ARM740T	writethrough	random	Read-miss
ARM920T	Writethrough, writeback	Random, round-robin	Read-miss
ARM946E	Writethrough, writeback	Random, round-robin	Read-miss

So, if you look at now ARM this a typical example of a micro controller where these policies availability of the policies are there. So, these are cache core. So, ARM seven and ARM 9 I have picked up some examples of ARM 7 and ARM 9 that is what we have discussed. So, they implement writethrough policy writethrough the replacement is random and your allocation policy is replace and in case of this is same true is for ARM 740 T. If you look at ARM 920 T you have an option of writethrough as well as writeback random and round robin similar thing is true for ARM 946 E. Sum of the ARM advanced ARM 10 11 11 they have also read write these options. Now, these are actually options which are to be selected for the purpose cache into the memory and these are available as a as part of the core policy.

Now, how are these things really implemented in case of a microcontroller? Because these a policy available, but how they are main available and how they are to be really used? So, that lists to the issue of cache control enough. So, if you are looking at the cache management in an embedded System, you have to have this mechanism available on the embedded controller itself to manage the cache. In fact, typically today you have got what you call on-chip cache. So, at SOC designer who picks up the ARM code as an IP will make use of these option and configure the cache. Because in putting the cache along with the core and new as an external programmer will not really have option to play with them in many cases. So, these options are actually options which should be available to an SOC designer.

(Refer Slide Time: 47:36)



So, all standard for these purpose ARM core has got some facilities. In fact, we have discussed coprocessor instructions and use of coprocessor in ARM. So, the most important coprocessor that ARM uses as coprocessor 15 and these coprocessor is called system control coprocessor and whose job is to manage it standard memory facilities including cache. CP 15 has registers. So, you program this, a coprocessor by writing onto the registers by using appropriate instructions. The coprocessor instructions I had discussed earlier. So, CP 15 has registers using which different features and control functions of cache can be specified.

You can specify size of cache and degree of associativity. This should be depend on how the cache's actually getting implemented. You can enable or disable cache operations you can also execute policy choices like a write replacement an allocation. There are also other kind of control functions like flushing of cache which can be initiated by an instructions on to the cache coprocessor that is a your system control coprocessor. Some aspects of it is predetermined because if since they are determined by the additional hardware which have been put in place. Some aspects of it are available to the external programmer. So, that he can exercise the control on the cache management through this coprocessor. In fact, these coprocessor is actually what is the block that you have seen the cache controller.

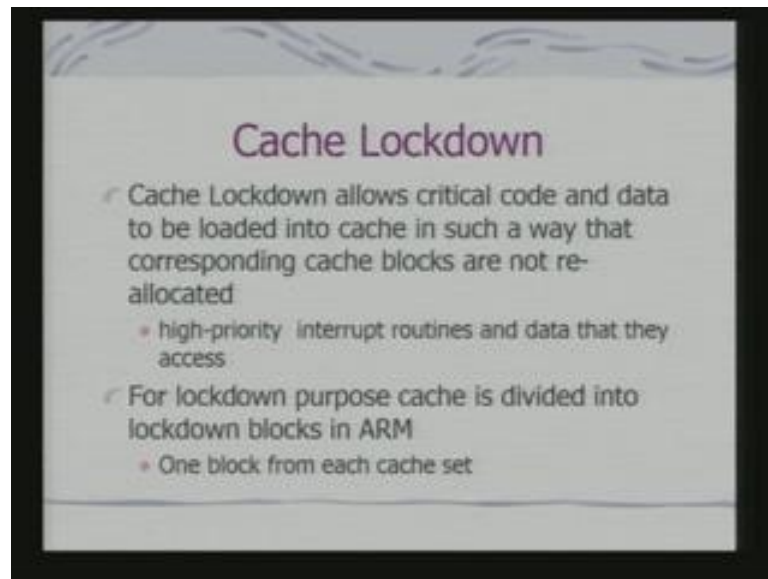


So, these cache controller manages the complete cache access for the case of ARM. In fact, ARM cache controller provides another interesting feature. In fact, what we talked about in case of SPM scratch pad memory that that occupies a part the address space of the processor itself. Since is occupy a part of the address space what is the advantage you get a guaranteed access and there is no question of cache hit or miss because you are not doing of any kind of a mapping. Now, ARM provides for a kind of idea which is called lockable cache . So, a part of the core can be locked into the cache. The other important issue related to the cache in ARM is that of memory areas thing indicated cachable or not. Because in my original diagram the first diagram I have shown that my entire memory space which is occupied by the DRAM is getting mapped on to the cache, but you if you remember the IO management of ARM.

In ARM what we use we use memory mapped IO. So, IO ports are located in the memory. Now, data in a memory port if it is an input port can change asynchronously can be changed, because of the external source. So, if I now make those memory locations cacheable. Actually I am getting an inconsistent data I am having a data which is not a true data. So, those locations typically will not be cacheable. In fact, the locations which I am mapped 1 to the peripheral processors that is IO IO devices. They would be typically not cacheable and it many cases the would not be also buffer able when we are using a write buffer say for example, when you have got an interrupt from a processor.

You have got an interrupt and would like to writeback to the processor acknowledging that yours servicing that interrupt from that IO device. Now, if that write is written on to the buffer then what will happen there will be a time expiry before that is written on to the device. So, in between device can again raise on interrupt which is not frequently call for. So, such locations may not be also buffer able. So, for memory mapped IO processors and memory mapped IO processors in particularly. These cacheable memory area or buffer able memory area is an important consideration and these cache controllers typically has got the provision for indicating whether some area can be cache on note.

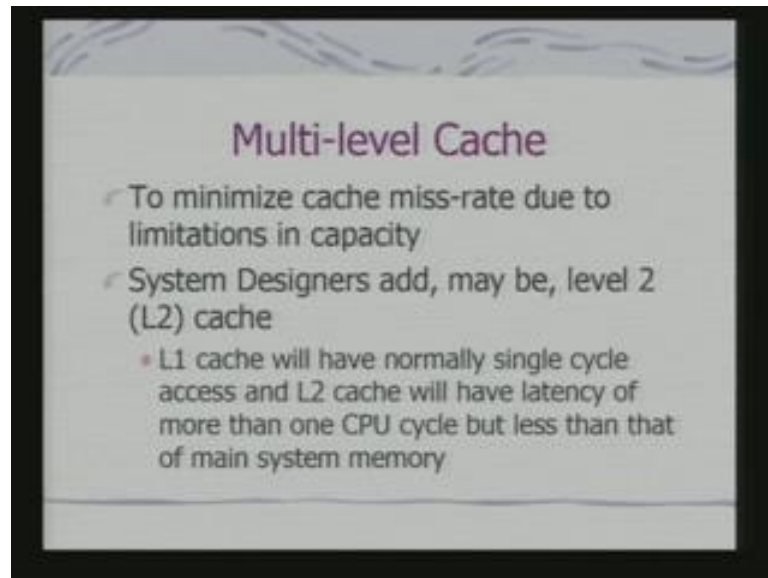
(Refer Slide Time: 52:49)



This cache lockdown is other point that I was discussing a ready that I would to like to have some critical code and data to be loaded into cache in such a way that corresponding cache blocks are not reallocated because these will be repeatedly used. In fact, the whole purpose of using SPM also is something similar in the SPM also we can have the critical code which will have be replaced, because it is part of your memory itself memory map itself. Now, in case of in case of ARM they provide this similar facility for cache. Because if I can do it in the cache then some critical part can already to be loaded on to the cache which should be repeatedly used and fall locked down purpose a cache in ARM is divided into what we call Lockdown blocks. So, one block from each cache said can be marked as lock block

So, your data from main memory can be loaded on to this blocks. So, that will not be really replaced. Now, if will fine if you remember we are talked about then in the DSP processors we have talked about loop cache; that means, frequently accessed loops being loaded onto the cache. So, that there is a minimum overhead for loop execution that also has got a similar idea that there is a dedicated memory where you stored a particular loop which is very frequently used and you may force it not to be replaced at all. So, there are various combinations of these, these may be locked for long time may be locked for a local time period depending upon the nature of the core or the application.

(Refer Slide Time: 54:42)

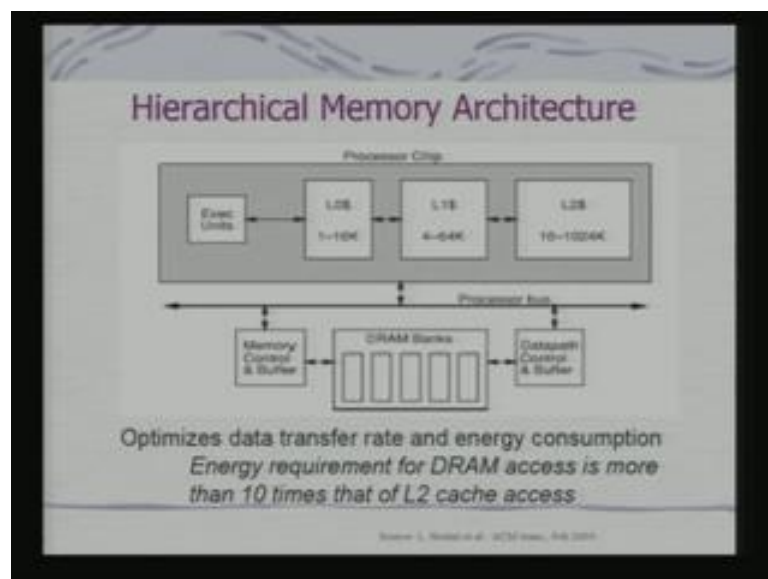


**Multi-level Cache**

- ✓ To minimize cache miss-rate due to limitations in capacity
- ✓ System Designers add, may be, level 2 (L2) cache
  - L1 cache will have normally single cycle access and L2 cache will have latency of more than one CPU cycle but less than that of main system memory

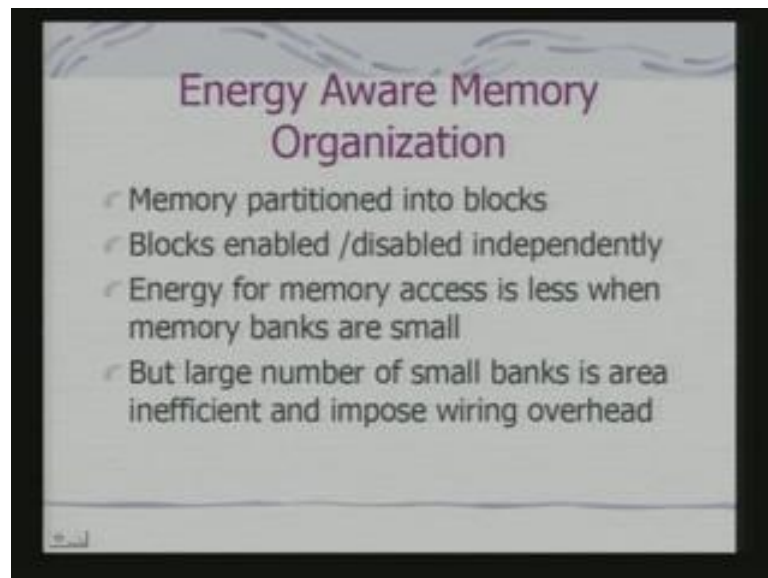
Next question is that of multilevel cache. Although today most of the embedded Systems come with level 1 cache, but advanced embedded processor architectures are supporting multiples levels of cache. So; obviously, this is to minimize what we call capacity miss to minimize cache miss rate due to limitations in capacity. System designers may add what is call level 2 or L2 cache L1 cache normally single cycle access and L2 cache will have latency of more than 1 CPU cycles, but less than that of main system memory. So, that would speed of the operations.

(Refer Slide Time: 55:23)



In fact, the interesting we look at this kind of a hierarchical memory architecture which have been propounded and proposed and actually being implemented in various SOCs. So, if you have you here you have shown 3 levels of cache L0 L1. So, we are not talking this L1 we are talking about L0 L1 L2. These are on processor cache and then you have got DRAM bands. So, these are all separated in 2 bands and why this architecture is being proposed; obviously, it would optimize data transfer rate optimizing data transfer rate would imply that the capacity miss which should be there if I just use 1 to 16 K and it is the first as memory and first less memory. This other optimization happens in terms of cost then that would be taken care by L1 as well as by L2 and even and then it is not really in L2. You will be really access in the DRAM max and energy requirement for DRAM access is more than the ten times of that of L2 access. So, in the in F8 what we are talking about is that your optimizing energy utilization as well.

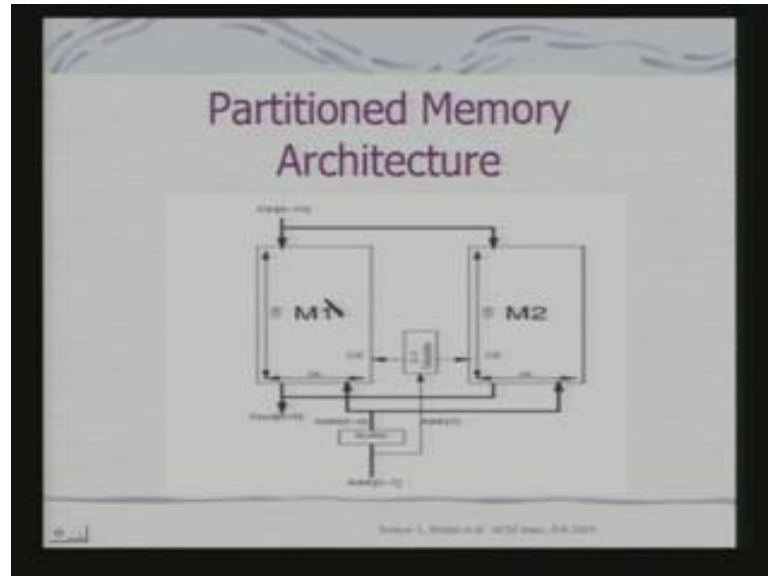
(Refer Slide Time: 56:41)



In fact, this concept of energy lists as to what we call energy aware memory organization in typical the memory is partition into blocks. Blocks are enabled or disabled independently and energy for memory access is less when the memory banks are small, but large number of small banks is area inefficient and can impose wiring overhead. So, energy is an important consideration for all embedded systems at particularly battery powered embedded systems. So, far we have looked at use of cache to speed up and we said that the cache using multilevel cache you can also have energy of optimization. But what about actually memory organization in terms of blocks, because I told you that your

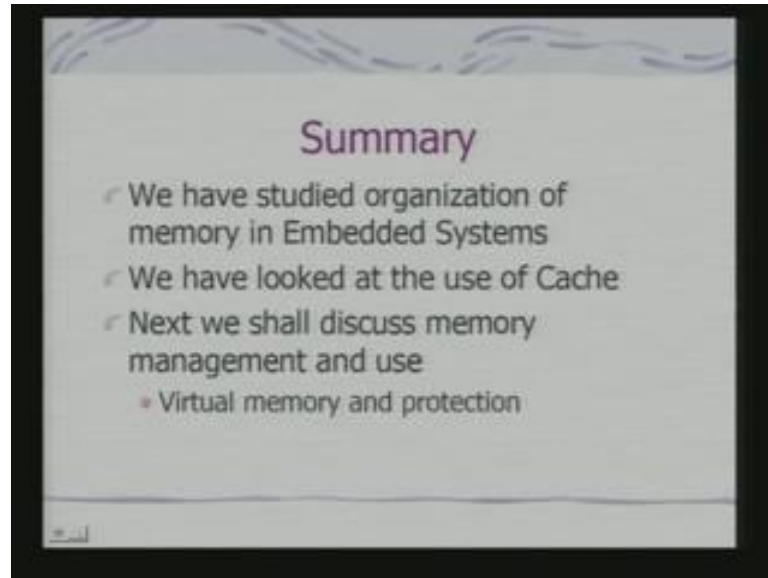
driving if you are driving a long bit line that would actually lead to more energy, because you need more current to drive a long is hit line.

(Refer Slide Time: 57:47)



So, if you divide your memory into blocks what happens just look at this example that this is a memory area which are using 0 to 7 address bits that is you are having a 8 bit address bus and these memory is divided into 2 blocks. So, the number of locations that to be driven at the given point in time is 128 it is half of that of the complete block. And these block would be deactivated, because you are using a 2 by 1 decoder depending on the address bit 7 that is the most significant address bit so effectively or energy consumption would be less. So, this is a suggestion for using smaller memory blocks. So, your energy consumption for accessing one of these banks at given point of time could be less. Because energy is a very important consideration for memory organization in embedded system as well.

(Refer Slide Time: 58:43)



So, what we have study today is organization of memory in embedded systems. We have looked at the use of cache and different policies being used for management of cache. In the next class, we shall discuss memory management in particular virtual memory and protection issues.