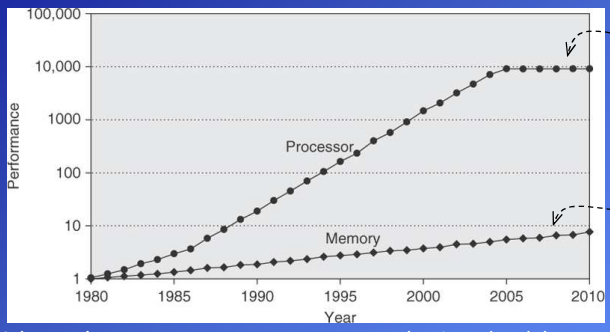
8 Cache Memory , The internal memory system

8.1 The performance gap between the CPU and the main memory:

Main memories are implemented based on DRAM (dynamic RAM) technology.

The speed of processors are much higher then the speeds of main memories.

Therefore main memory cannot meet bandwidth requirements of the CPUs.



Memory requests per second, on average
Single processor

Source: Hennesy, Patterson, *Computer Architecture*, 5/e

Access per second, 1/latency

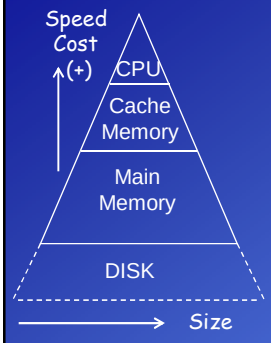
The multicore processors increase the bandwidth requirements.

Intel i7 with 4 cores and 3.2 GHz clock rate can achieve a total peak bandwidth of about 409.6 GB/s.

The peak bandwidth to DRAM main memory is only 6% of this (25 GB/s).

8.2 The memory hierarchy:

The memory hierarchy



There are memories of different speeds, costs and sizes.

Faster memory is more expensive.

To increase the average speed and (at the same time) to decrease the cost of the memory system, different types of memories are organized in the form of multilevel memory hierarchy.

A faster memory (SRAM) namely the cache memory is installed between the main memory and the CPU.

In cache systems we focus on main memory and cache memory not on disk.

Main memory and cache memory can be accessed by the CPU directly (internal).

But the data in the disk must be first fetched in to the main memory (or cache).

The goal: To provide a memory system with high speed close to the fastest level of memory and cost per byte low as the cheapest level.

This solution takes the advantage of **locality** of computer programs.

8.3 The principle of locality (Locality of reference):

Programs tend to reuse instructions and data they have used recently.

Observation: Most programs spend 90% of its execution time in only 10% of the code.

We can predict what instructions and data a program will use in the near future based on its access in the recent past.

There are two types of locality:

Temporal Locality (Locality in time): Recently accessed addresses are likely to be accessed in the near future.

Spatial Locality (Locality in space) : After an access to a memory address the next access will be likely to a near address.

Reasons for locality:

Structure of the program: Generally, related data is stored in nearby locations.

Loops

Arrays

8.4 Memory Technologies

RAM (Random Access Memory):

Actually, the phrase "random access" is a misuse of the term, because all of the semiconductor (electronic) memories used in computer systems are random access (not sequential).

Distinguishing characteristic of RAM:

- CPU can access the memory to read data and to write new data directly (easily and rapidly without an additional device).

  These operations are accomplished through the use of electrical signals.

  It could be better to name this type of memory as "*Read Write Memory*".
- RAM is *volatile*. If the power is interrupted, then the data are lost.

  Thus, RAM can be used only as temporary storage.

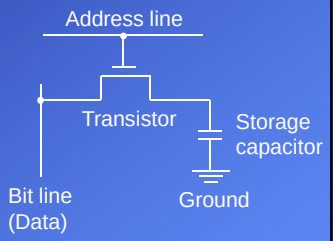
Memory latency measures:

- **Access time:** Time between read request and when desired word arrives
- **Cycle time:** Minimum time between two unrelated requests to memory

There are two types of RAM; DRAM (Dynamic RAM) and SRAM (Static RAM).

DRAM (Dynamic RAM):

- It is used to implement the **main memory**.
- A dynamic RAM (DRAM) is made with cells that store data as charge on capacitors.
- Because capacitors have a natural tendency to discharge, dynamic RAMs require periodic charge **refreshing** to maintain data storage (Every ~ 8ms).
- Called *dynamic*, because the stored charge leaks away, even with power continuously applied.



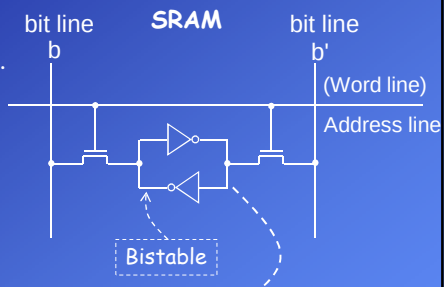
- During refreshing memory is unavailable (latency). (-)
- Must be re-written after being read. Reading destroys the information (latency). (-)
- Difference between access time and cycle time. Cycle time > access time (-)
- Cheap and dense: one transistor/bit. More bits can be placed in one chip. (+)

Some improvements:

- SDRAM (Synchronous DRAM): Added clock to DRAM interface. Burst mode.
- DDRAM (Double data rate DRAM): Data is transferred on both the rising edge and falling edge.

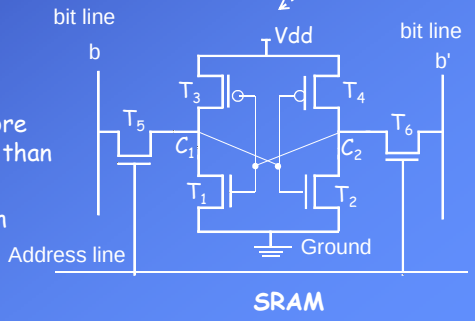
SRAM (Static RAM):

- It is used to implement the **cache memory**.
- SRAM uses flip-flops (or latches) to store bits (see Digital Circuits course).
- It requires 6 transistors/bit (more expensive, less bits per chip). (-)
- Refreshment is not necessary. (+)
- Access time ≈ cycle time. (+)



DRAM vs. SRAM:

- Both are volatile. Power must be continuously supplied.
- DRAM is more dense (smaller cells, more cells per unit area) and less expensive than a corresponding SRAM.
- DRAM requires the supporting refresh circuitry. Latency of DRAM is high.
- DRAM is cheap, SRAM is fast.



Associative Memory, Content Addressable Memory (CAM):

- Random access memory (SRAM) + circuitry for search
- It consists of SRAM to store data and digital circuits for parallel search.
- It is used in high speed searching applications.
  We will use it to search a data in the cache memory.
- A word is retrieved based on a portion of its contents rather than its address.
- The user supplies a data word (Argument A) (not the address) and the CAM searches its entire memory simultaneously (not sequential) to see if that word is stored anywhere in it.
- The user also supplies a key value (K) to determine the portion of data to search for.
- If the search data (or the required portion) is found in a row of the memory then the corresponding match bit is set and the output is the stored data in this row.

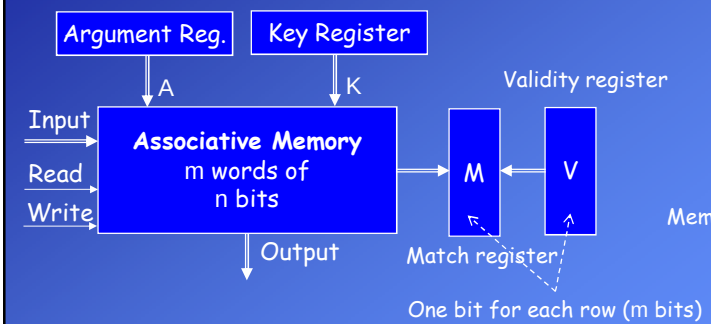
Associative Memory, Content Addressable Memory (CAM) (cont'd):

On power-up the memory may include random values.

An additional flag (valid bit V) is necessary to indicate whether or not a memory line has been loaded with valid data.

On power-up, the hardware sets all the valid bits of the all lines to "invalid" (V=0).

When data is written to line i then corresponding valid bit is set (V_i=1).



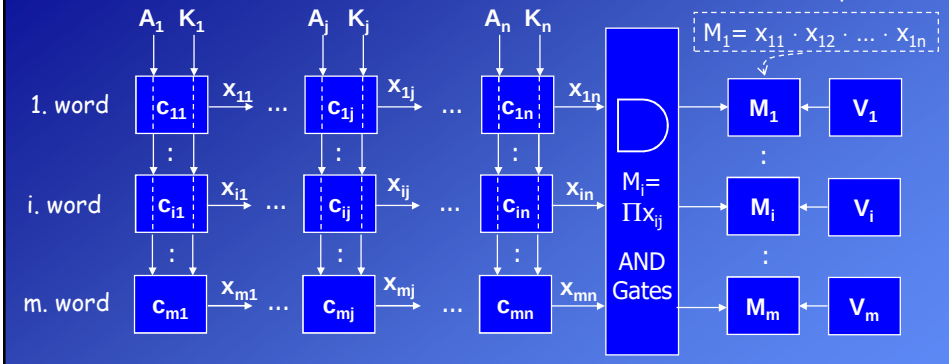
Example:

A: 1011 1111
K: 1111 0000

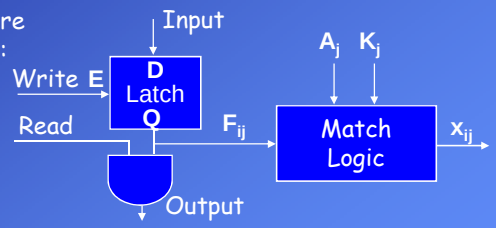
Memory: 1100 1111 M=0
1011 1010 M=1

Out: 1011 1010

Internal Structure: A_i and K_i values are broadcasted to all cells in parallel.



The structure of a cell (c_{ij}):



$X_{ij} = \overline{K_j} + \overline{A_j \oplus F_{ij}}$

$X_{ij} = \overline{K_j} + A_j \odot F_{ij}$

$X_{ij} = \overline{K_j} + A_j \cdot F_{ij} + \overline{A_j} \cdot \overline{F_{ij}}$

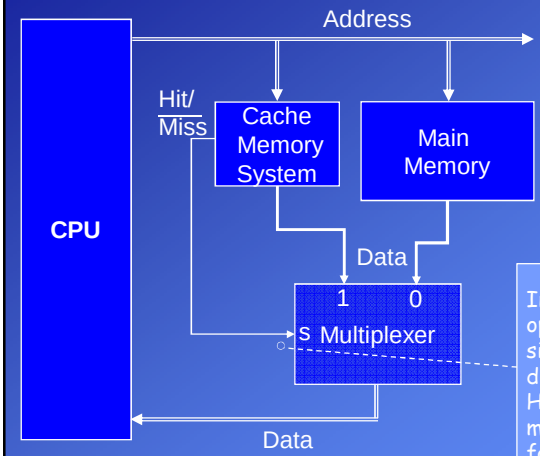
$M_i = V_i \cdot \prod_{j=1}^{n} x_{ij}$

8.5 Cache Memory Principles

The goal is to store the copies of frequently accessed data in the fast cache memory to reduce the average access time.

Hit: The accessed word is found in the cache memory.

Miss: A **miss** occurs if the accessed data is not found in the cache memory.



Example:
Only read operations:
Cache memory access time: 20 ns.
Main memory access time: 100 ns.
Hit ratio: H=0.9 (90% hit)
Average memory access time:
t_a= 0.9*20 + 0.1*100 = 28 ns.

In systems with asynchronous bus operations like MC68000 the DTACK signal can be sent to the CPU in different times according to hit or miss. Hence, if data is received from cache memory, the bus cycle can be completed faster.

8.5 Cache Memory Principles (cont'd)

Block transfer:

Upon a cache miss (the requested element is not found in the cache), a *block* of elements that contains the requested element is brought (copied) from the main memory to the cache memory.

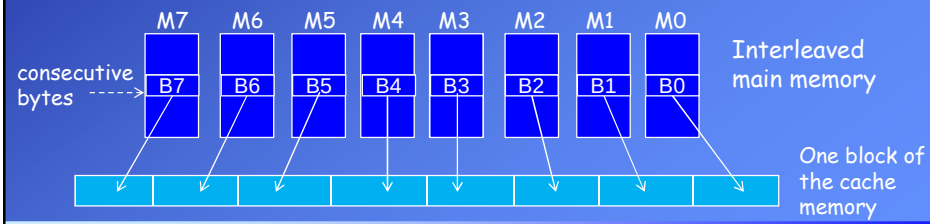
Reason: spatial locality.

It is expected that the next requested element will be residing in the neighboring locality of the current requested element.

The disadvantage of block transfer can be the long duration.

To decrease the block transfer time between main and cache memories, *memory interleaving technique* is used.

Data are stored in different memory modules, that is, consecutive memory addresses are stored in successive memory modules (Similar to RAID).



8.5 Cache Memory Principles (cont'd)

Replacement Techniques:

The capacity of the cache memory is lower than the main memory.
At any moment, only a part of the data in the main memory can be kept in the cache memory.

When the cache memory is full, a *replacement algorithm must be applied to select the block* in the cache which is to be replaced by the new block from the main memory.

There are different replacement techniques; the most common techniques:

- **FIFO** (*First In First Out*): The block that has been in the cache the longest is replaced.
- **LRU** (*Least Recently Used*): The block that has been used the least while residing in the cache is replaced.

  The history of block usage is taken into consideration.

  Aging counters are necessary to keep track of references to blocks in cache.

Cache operations are performed by a **hardware unit** called *Cache Memory Controller* or *Cache Memory Management Unit*.

8.6 Cache Memory Mapping Techniques

- Which contents of the main memory are currently in the cache memory?
- In which location of the cache memory are the referenced data?

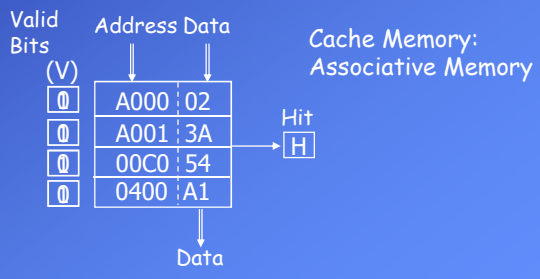
1. Full Associative Mapping

a) Without blocks:

Actually all mapping techniques are applied on data blocks.
Firstly, just for simplicity, I will show the technique without blocks.

Method: The most referenced addresses and data are kept in a an associative memory.

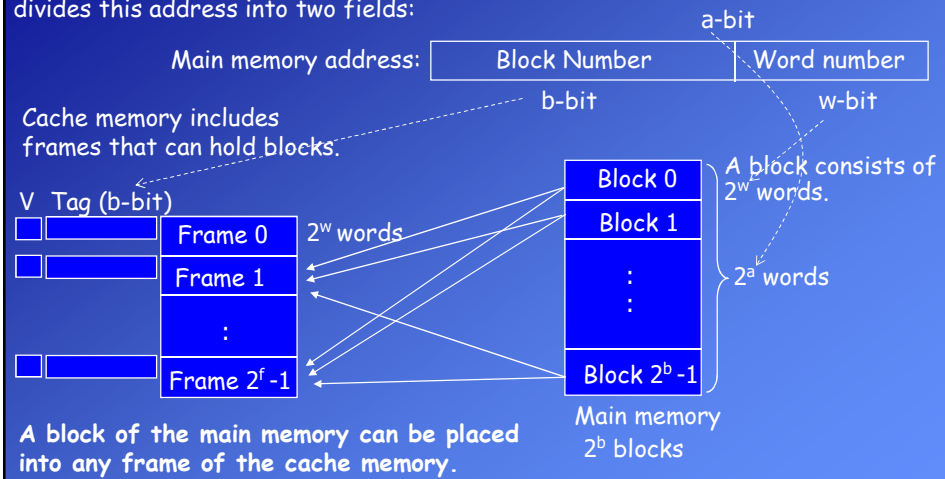
The address generated by the CPU is searched in the cache (content addressable memory).
If there is a hit, data is read from cache.
If a miss occurs, data is read from the main memory and data in the cache is replaced.



Without blocks the technique only benefits from the temporal locality but not from the spatial locality.
Therefore, in practice data blocks are moved between main and cache memories.

b) Full Associative Mapping (with blocks)

The CPU generates a main memory address and the cache controller unit (CCU) divides this address into two fields:

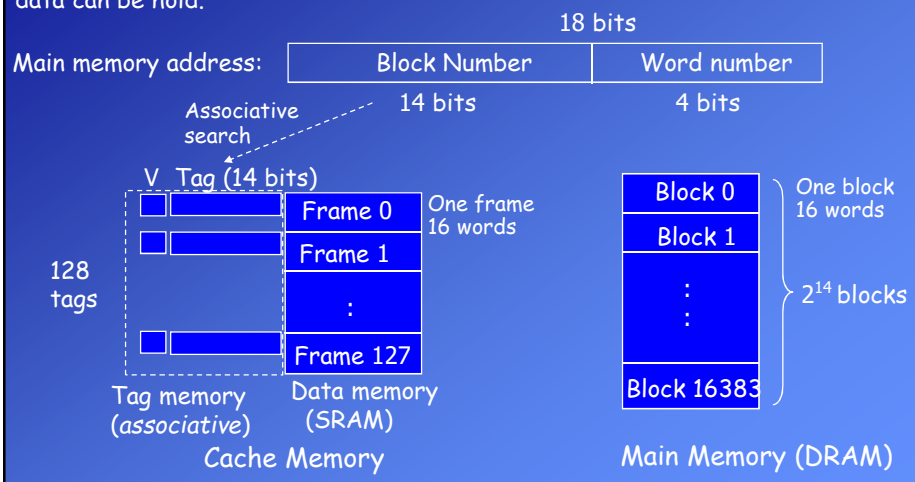


A block of the main memory can be placed into any frame of the cache memory.

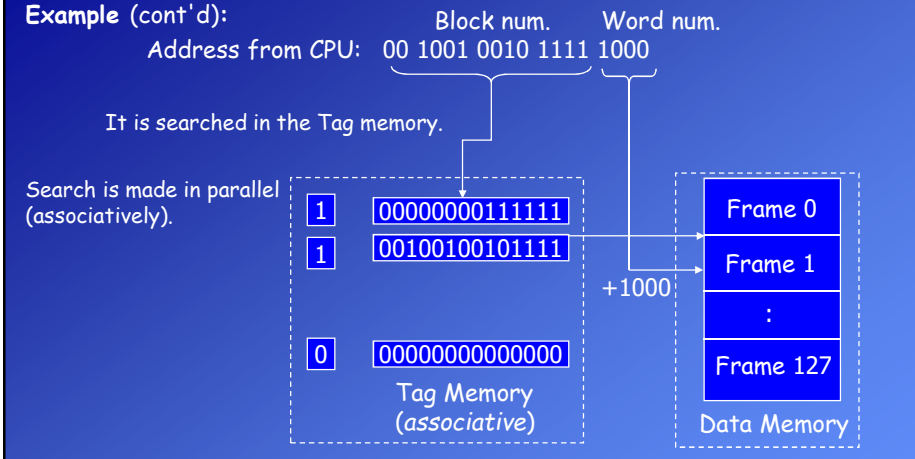
To determine which block is currently residing in a frame the tag is used.
In this technique tag is the block number in the address.

Example: *Full Associative Mapping*

Main Memory: 256K x words Address: a = 18 bits
Block size: 16 words w = 4 bits Main memory includes 2^{14} blocks. b = 14
Cache Memory: 2K x words Cache memory includes 2^{7} =128 frames. f = 7
data can be hold.



Example (cont'd):



If the replacement technique is LRU (*Least Recently Used*) then tag memory also includes the *aging counters*. Each frame has its own counter.

When a frame is not referenced its counter is incremented.

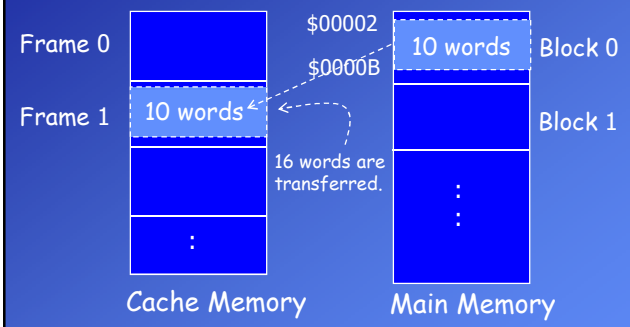
When a replacement is necessary the frame that has the highest counter value ("oldest") is replaced.

Example: Array in a cache memory system

Main Memory: 256K x words Cache Memory: 2K x words Block size: 16 words

Case A) A program in this system accesses an array with the starting address $00002 and size of 10 words.
Assume that, as the CPU accesses the array, the least recently used frame is F #1.

Starting address of the array: 00 0000 0000 0000 0010 ($00002)
End address of the array: 00 0000 0000 0000 1011 ($0000B)



When the CPU accesses the first word of the array ($00002) a miss occurs.

Although the array has 10 words, the cache management system transfers the entire Block 0 (16 words) to the Frame 1.

When the CPU accesses the next 9 elements of the array, hits will occur.

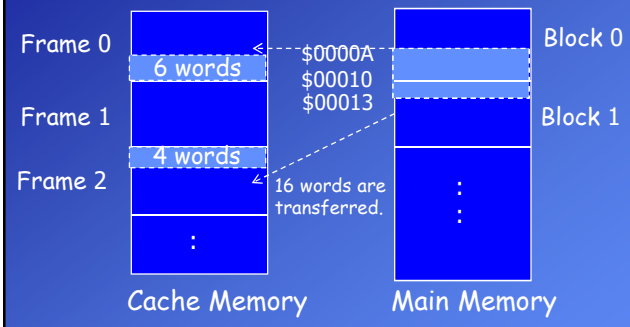
In total; 1 miss, 9 hits.

Example: Array in a cache memory system (cont'd)

Main Memory: 256K x words Cache Memory: 2K x words Block size: 16 words

Case B) A program in this system accesses an array with the starting address $0000A and size of 10 words.
Assume that, the least recently used frames are Frame #0 and #2.

Starting address of the array: 00 0000 0000 0000 1010 ($0000A)
End address of the array: 00 0000 0000 0001 0011 ($00013)



Each block is processed separately.

Different blocks have different tag values.

In this example two block replacements are necessary.

Although the size of the array is smaller than the block (frame) size, it occupies two frames in the cache memory.

Example: Array in a cache memory system (cont'd)

- When the CPU accesses the first word of the array ($0000A), a miss occurs.
- The cache management system transfers the entire Block 0 (16 words) to the Frame 0.
- Next 5 accesses to the array generate hits.
- When the CPU accesses the 7th word of the array ($00010), a miss occurs, because this word is in another block and its address has a new tag value.
- The cache management system transfers the entire Block 1 (16 words) to the Frame 2 (because of LRU).
- Next 3 accesses to the array generate hits.
- In total; 2 misses, 8 hits

If the starting address of the array was selected properly (for example $00000), the array would occupy only one block and one frame in the cache memory as in case A.

Therefore placement policies for arrays in the main memory effect the performance of the cache systems.

2. Direct Mapping

An incoming main memory block is always placed into a specific fixed cache frame location.

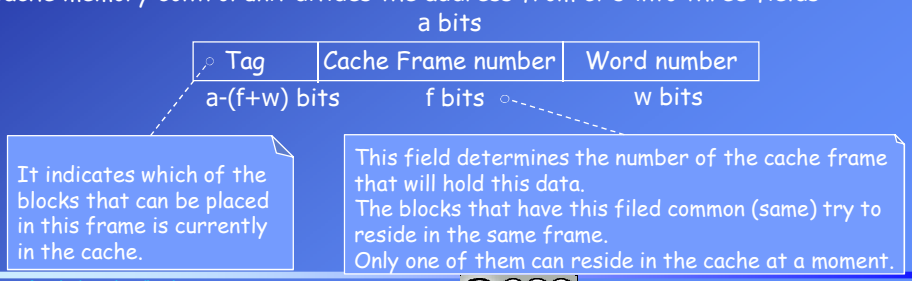
It is not necessary to search the location of a block in the cache, because it is predetermined and fixed.

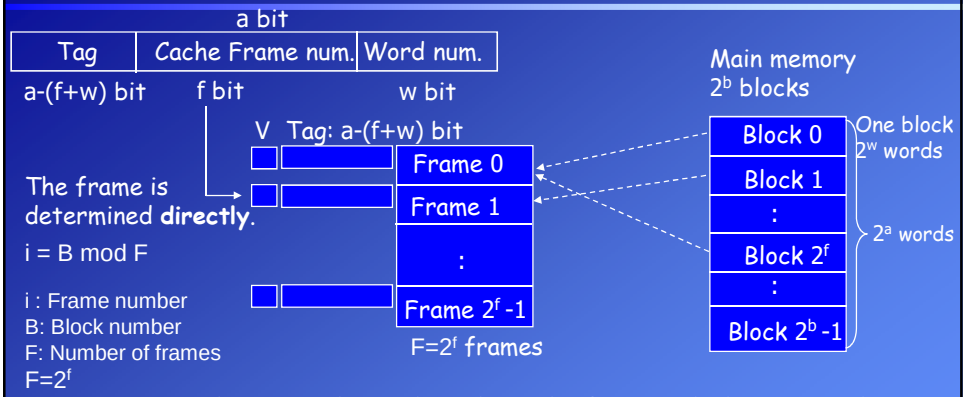
Therefore associative memory is not necessary.

As the size of the main memory is greater than the cache, several blocks of the main memory try to reside in the same cache frame.

It is necessary to determine which main memory block is currently residing in a frame.

Cache memory control unit divides the address from CPU into three fields:





A main memory block can only reside in the cache frame with the number that is determined by the "Cache Frame number" filed of the address.

Even if there are unused frames in the cache, two blocks with the same "Cache Frame Number" field cannot be in the cache at the same time.

During replacement a decision algorithm is not necessary.
Because the frame of the incoming block is fixed, this frame will be replaced.

Associative memory is not necessary, because there is no need to search in the cache.

Example (*Direct Mapping*):

Main Memory: 256K x word
Block size: 16 words
Cache Memory: 2K x word data capacity

Address: a = 18 bits
w = 4 bits Main memory contains 2^{14} blocks. b = 14
Cache memory contains 2^7 =128 frames. f = 7

Main memory address:

18 bits		
Tag	Frame number	Word number
7 bits	7 bits	4 bits

In this system, the data in the following two different addresses try to reside in the same cache frame.

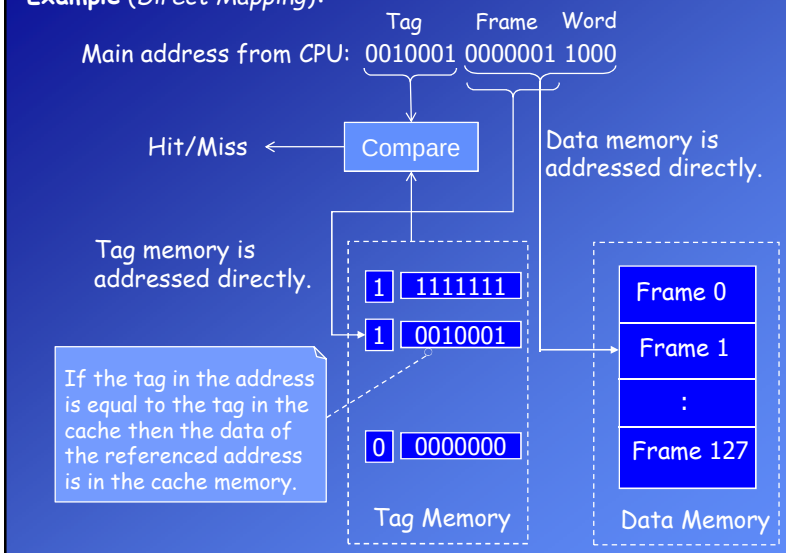
Tag	Frame num.	Word num.
0000000	0000000	XXXX
0000001	0000000	XXXX

The "Frame number" fields of both addresses are same: 0000000. They will be placed in Frame 0.

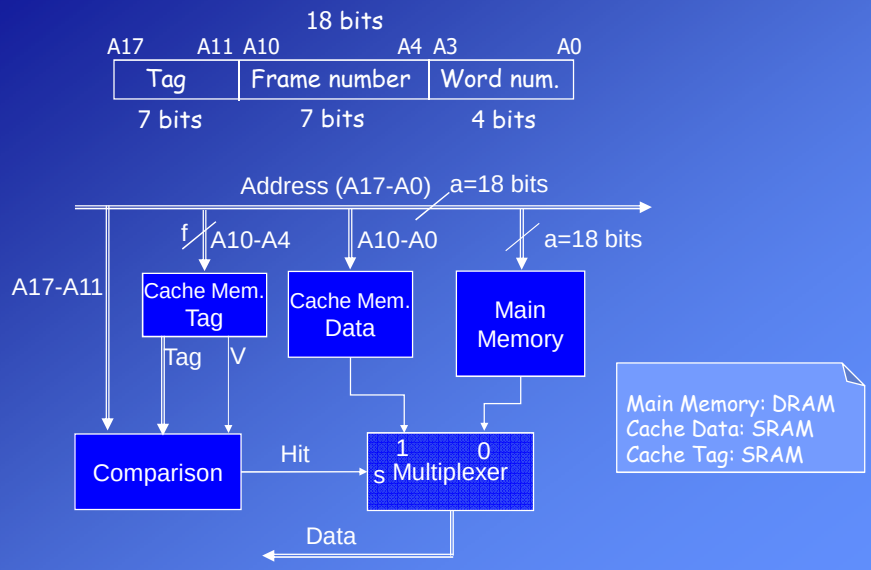
At a specific point in time only one of these data can be in the cache memory.

To determine which data is currently in the Frame 0 of the cache memory, the tag value of the address is compared with the tag value stored in cache memory.

Example (*Direct Mapping*):



Block diagram of the exemplary system:



3. Set Associative Mapping

This method is between direct mapping and full associative mapping.

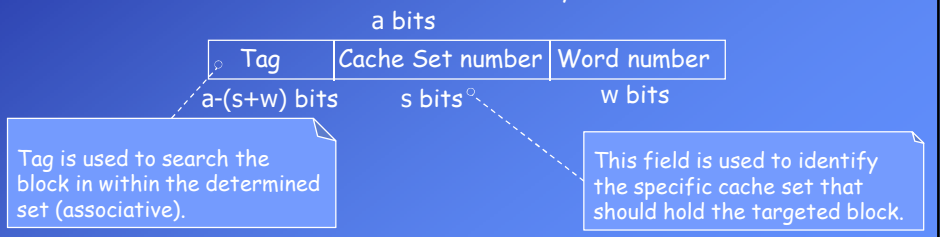
The cache is divided into a number of **sets** and each set consists of a number of frames.

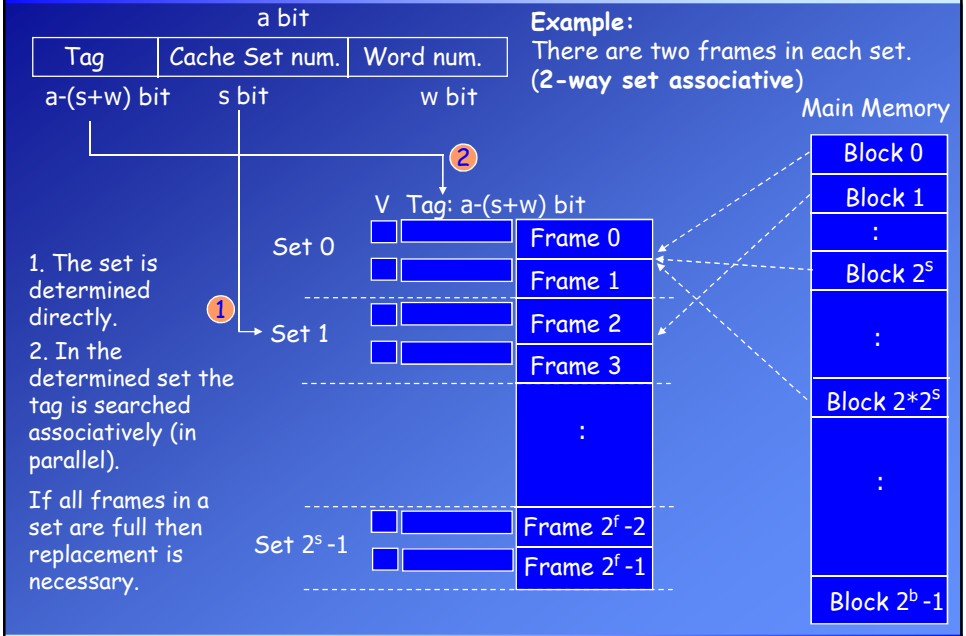
A given main memory block maps to a specific, fixed cache set (Snum) based on the equation Snum= b mod S, where S is the number of sets in the cache, b is the main memory block number, and Snum is the specific cache set to which block b maps.

However, an incoming block maps to any frame in the assigned cache set.

- Direct mapping to determine the set (fixed).
- Associative mapping to determine the frame in the set (flexible).

The cache controller divides the address issued by the CPU into three fields:

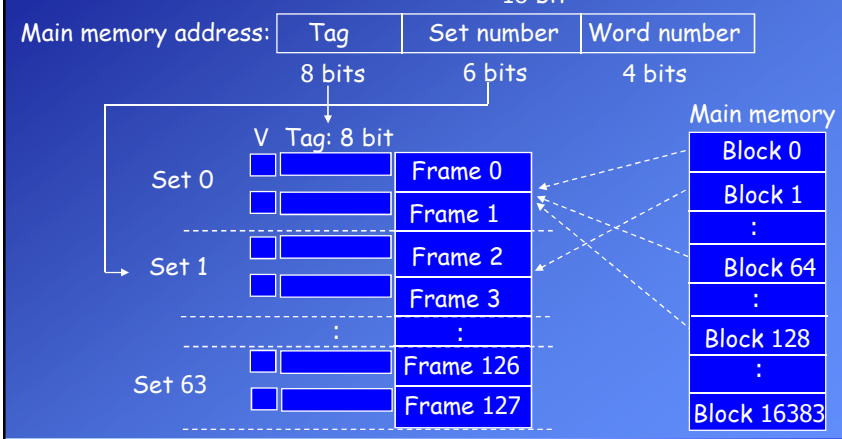




Example (*Set Associative*) :

Main Memory: 256K x words Address: a = 18 bits
Block size: 16 words w = 4 bits Main memory contains 2^{14} blocks. b = 14
Cache Memory: 2K x words Cache memory contains 2^7 = 128 frames. f = 7
data capacity.

Given: Each set contain 2 frames. Hence the cache memory contains 64 sets.



Summary of mapping techniques:

- **Associative mapping** is the most *flexible* and *efficient* technique because an incoming main memory block can reside in any frame of the cache.
  The disadvantage of this technique is the high *hardware cost* due to the associative memory.
- In **direct mapping** the cache frame for each memory block is fixed.
  The main disadvantage is the *inefficient* use of the cache, because a number of main memory blocks may compete for a cache frame even if there exist other empty frames.
  This disadvantage decreases the hit ratio.
  The main advantage of the this technique is its simplicity; no search is needed.
  It is also simple in terms of the replacement mechanism (No need for decision).
- The cache utilization efficiency of the **set associative mapping** technique is expected to be *moderate;* namely between fully associative technique and the direct mapping technique.
  However, the technique inherits the simplicity of the direct mapping technique in terms of determining the target set.
  By changing the number of frames in a set we can make it close to one of the other techniques.

8.7 Cache Memory – Main Memory Interactions

→ **Read (Hit):** Data is read from the cache memory.

→ **Read (Miss):**

a) **Read Through (RT) :** While the data (block) is being brought from the main memory to the cache, it is also read by the CPU simultaneously.
Cache memory and the main memory are accessed in parallel.

b) **No Read Through (NRT):** Data are first brought from the main memory to the cache memory and then the CPU reads data from the cache.

→ **Write (Hit):**

a) **Write Through (WT):** In each write operation data is written to the cache and also to the main memory.
The write-through policy increases the access time but provides coherence between the cache frames and their counterparts in the main memory.

b) **Write Back (WB):** All writes are made only to the cache.
A block is written back to the main memory only when a replacement is needed.
There are two types of write-back policy: *Simple write back* and *flagged write back*.

→ **Write (Hit)** (cont'd):

b) **Write Back (WB)** (cont'd):

- **Simple Write Back (SWB):**
  The replaced frame is always written back to the main memory.
  It is not checked whether the frame was changed or not.
- **Flagged Write Back (FWB):**
  Every cache frame is assigned a bit, called the *dirty bit*, to indicate that at least one write operation has been made to the block while residing in the cache.
  At replacement time, the dirty bit is checked; if it is set, then the block is written back to the main memory, otherwise, it is simply overwritten by the incoming block.
  Dirty bit takes place in the tag memory of the cache.

→ **Write (Miss):**

a) **Write Allocate (WA):** The main memory block is updated and brought to the cache.

b) **No Write Allocate (NWA):** The missed main memory block is updated in the main memory and not brought to the cache.

If an attempt is made to read this block later, a miss will occur and data will be brought to the cache.

The write-through (WT) policy can be used together with write-allocate (WA) or no-write-allocate (NWA) methods. WTWA, WTNWA

In write-back (WB) policy to maintain coherence between cache and main memory the write-allocate (WA) method is used. WBWA

Information hold in Tag memory:

In addition to Valid (V) and tag bits, dependent on the used method, following data must be also kept in the tag memory:

If LRU is used *aging counters*, if flagged Write-Back (FWB) method is used "dirty" bit (D).

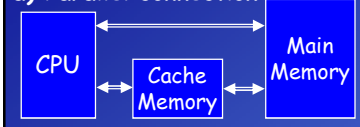
One line of a tag memory:

V	D	Counter	Tag

8.8 CPU - Cache Memory – Main Memory Interconnection

CPU - cache memory – main memory interconnection can be implemented in to ways:

a) Parallel connection



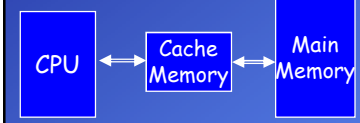
• **Read Through (RT):** While a block is brought from the main memory to the cache it is also read by the CPU simultaneously (in parallel).

• **Load Through (LT):** A data is written to the cache and to the main memory simultaneously.

Parallel connection is suitable for the *Write-Through (WT)* method.

It can also be used with the *Write-Back (WB)* method.

b) Serial connection



• **No Read Through (NRT):** The block is first brought from the main memory to the cache and then the CPU reads the data from the cache.

• **No Load Through (NLT):** First the data in the cache is updated, then the frame is written to the main memory.

Serial connection is suitable for the *Write-Back (WB)* method.

8.9 Access Time:

t_a : Average Memory Access Time
w : Write ratio (number of write accesses / total number of all accesses)
h : Hit ratio
t_{cache} : Cache memory access time
t_{main} : Main memory access time
t_{trans}: Time to transfer a block between main memory and the cache
w_d: The probability that a block in a cache is updated

	WT, RT/LT (Write-through , Parallel read/write)		WB,WA, NRT/NLT (Write-back, Serial read/write)	
Probability	NWA	WA	SWB	FWB
	Access Time		Access Time	
Read Hit $(1-w)h$	t_{cache}	t_{cache}	t_{cache}	t_{cache}
Read Miss $(1-w)(1-h)$	t_{trans}	t_{trans}	$2t_{trans}+t_{cache}$	$w_d(2t_{trans}+t_{cache}) + (1-w_d)(t_{trans}+t_{cache})$
Write Hit wh	t_{main}	t_{main}	t_{cache}	t_{cache}
Write Miss $w(1-h)$	t_{main}	$t_{main}+t_{trans}$	$2t_{trans}+t_{cache}$	$w_d(2t_{trans}+t_{cache}) + (1-w_d)(t_{trans}+t_{cache})$

Access Time Calculation:

- **Write Through with Write Allocate, Read/Load Through (WTWA, RT/LT):**

Read Hit + Read Miss + Write Hit + Write Miss

$t_a = (1-w)h\ t_{cache} + (1-w)(1-h)t_{trans} + w \cdot h \cdot t_{main} + w(1-h)(t_{main} + t_{trans})$

$t_a = (1-w)h\ t_{cache} + (1-h)t_{trans} + w \cdot t_{main}$

- **Write Through with No Write Allocate, Read/Load Through (WTNWA,RT/LT)**

$t_a = (1-w)h\ t_{cache} + (1-w)(1-h)t_{trans} + w \cdot h \cdot t_{main} + w(1-h)t_{main}$

$t_a = (1-w)h\ t_{cache} + (1-w)(1-h)t_{trans} + w \cdot t_{main}$

- **Simple Write Back with Write Allocate,No Read Through (SWBWA,NRT/NLT)**

Read Hit + Read Miss +Write Hit + Write Miss

$t_a = (1-w)h\ t_{cache} + (1-w)(1-h)(2t_{trans} + t_{cache}) + w \cdot h \cdot t_{cache} + w(1-h)(2t_{trans} + t_{cache})$

$t_a = t_{cache} + (1-h) \cdot 2 \cdot t_{trans}$

One t_{trans} is necessary to transfer a frame from the cache to the main memory and the second one to bring the new block from the main memory to the cache.

- **Flagged Write Back,Write Allocate, No Read Through (FWBWA,NRT/NLT):**

$t_a = t_{cache} + (1-h)t_{trans} + w_d \cdot (1-h)t_{trans}$

$t_a = t_{cache} + (1-h)(1+w_d)t_{trans}$

Exemplary processors with cache memories:

- **Intel386™:** Cache memory is outside of the CPU chip. SRAM memory.
- **Intel486™ (1989)**
  8-KByte on-chip (L1)
- **Intel® Pentium® (1993)**
  L1 on-chip: 8 KB instruction, 8 KB data cache (Harvard architecture)
- **Intel P6 Family: (1995-1999)**
  - **Intel Pentium Pro:**
    L1 on-chip: 8 KB instruction, 8 KB data cache (Harvard architecture)
    First L2 cache memory in the CPU chip.
    L2 on-chip: 256 KB. Different interconnections between L1, L2 and the CPU.
  - **Intel Pentium II:**
    L1 on-chip: 16 KB instruction, 16 KB data cache (Harvard architecture)
    L2 on-chip: 256 KB, 512 KB, 1 MB
- **Intel® Pentium® M (2003)**
  L1 on-chip: 32 KB instruction, 32 KB data cache
  L2 on-chip: up to 2 MByte
- **Intel® Core™ i7-980X Processor Extreme Edition (2010)**
  Multicore: 6 cores. Private caches (L1) and shared caches (L2)
  12 MB smartcache: All cores share this cache.