6.4. CACHE MEMORY

Cache is a very small and fast memory that is inserted between the larger, slower main memory and the CPU. This memory holds the currently active segments of a program and its data.

(*i*) 'Cache' refers to a fast intermediate memory within a larger memory system: It serves as a buffer between CPU and main memory. (Fig. 6.27)

(*ii*) It provides the CPU with fast single cycle access to its external memory and an efficient way to place a small portion of the memory on the same chip as a micro-processor.

Here memory words are stored in a cache data memory and are grouped into small pages, called cache blocks or lines. Each block is marked with a block address called the tag. The collection of tag addresses is stored in a special memory called cache tag memory. (Fig. 6.28)

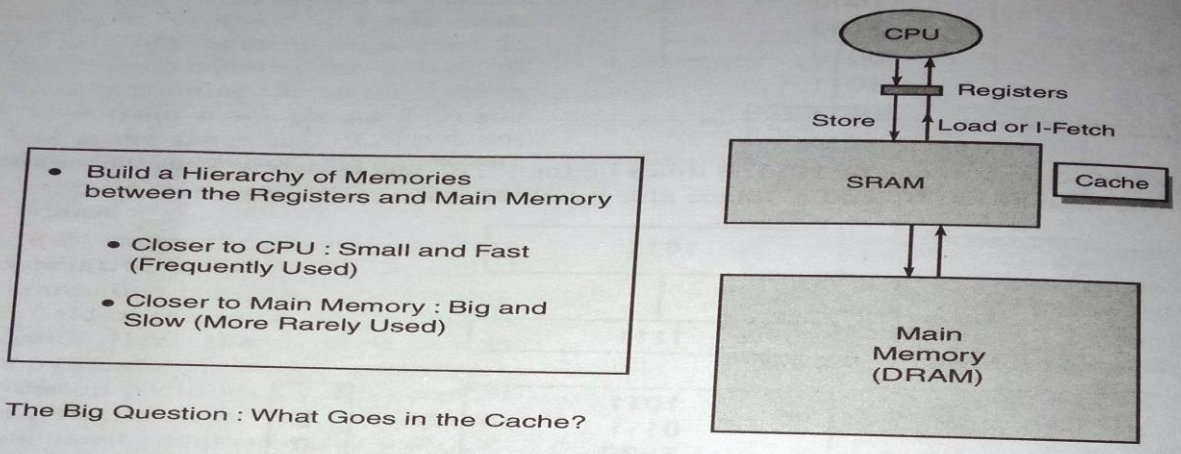


Fig. 6.27. Cache Memory-SRAM

6.5. CACHE MEMORY ORGANIZATION

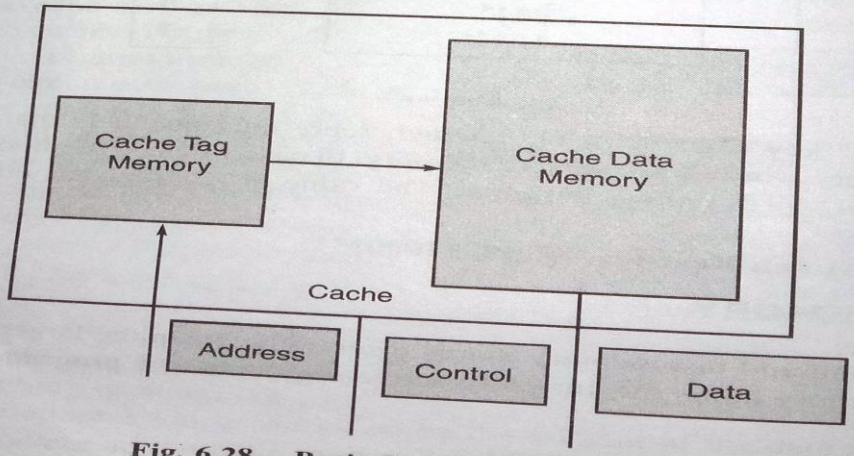


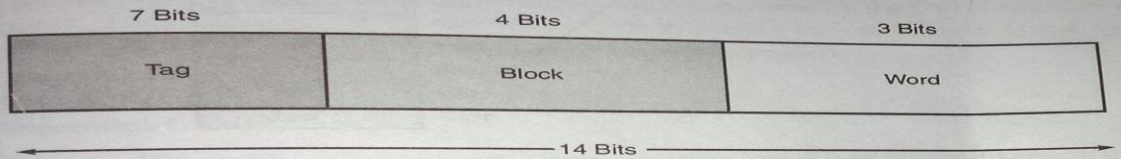
Fig. 6.28. Basic Structure of a Cache

The fields into which a memory address is divided provide a many-to-one mapping between larger main memory and the smaller cache memory. Many blocks of main memory map to a single block of cache. A *tag* field in the cache block distinguishes one cached memory block from another. The size of each field into which a memory address is divided depends on the size of the cache. Suppose our memory consists of 2^{14} words, cache has $16 = 2^4$ blocks, and each block holds 8 words.

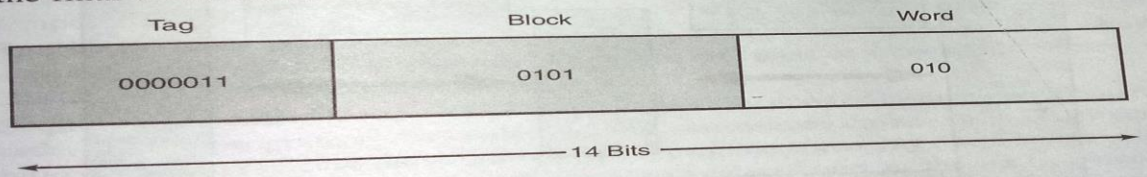
DO YOU KNOW

The "content" that is addressed in content addressable cache memory is a subset of the bits of a main memory address called a field.

Thus memory is divided into $2^{14} / 2^3 = 2^7$ blocks. For our field sizes, we know we need 4 bits for the block, 3 bits for the word, and the tag is what's left over :



As an example, suppose a program generates the address **1AA**. In 14-bit binary, this number is: **00000110101010**.The first 7 bits of this address go in the tag field, the next 4 bits go in the block field, and the final 3 bits indicate the word within the block.



Cache Working Concept

To access a particular piece of data, the CPU first sends a request to its nearest memory, usually cache. If the data is not in cache, then main memory is queried. If the data is not in main memory, then the request goes to disk. Once the data is located, then the data, and a number of its nearby data elements are fetched into cache memory.

This leads us to some definitions.

1. A *hit* is when data is found at a given memory level.
2. A *miss* is when it is not found.
3. The *hit rate* is the percentage of time data is found at a given memory level.
4. The *miss rate* is the percentage of time it is not.
5. Miss rate = 1 – hit rate.
6. The *hit time* is the time required to access data at a given memory level.
7. The *miss penalty* is the time required to process a miss, including the time that it takes to replace a block of memory plus the time it takes to deliver the data to the processor.

An entire blocks of data is copied after a hit because the principle of locality tells us that one byte is accessed, it is likely that a nearby data element will be needed soon.

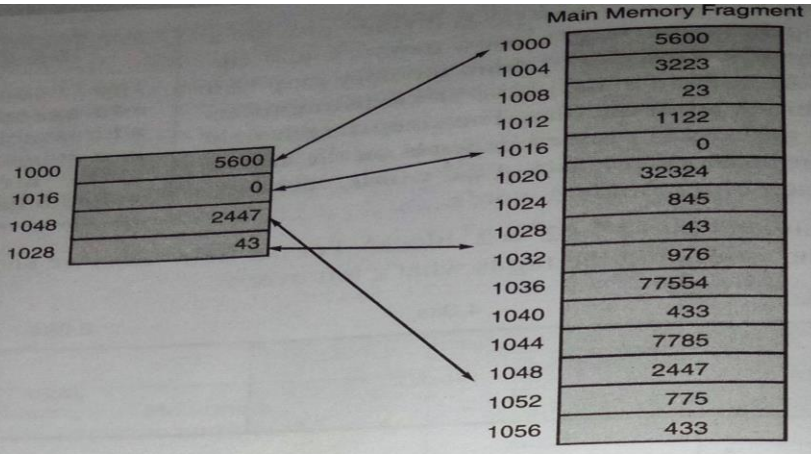


Fig. 6.29. Cache And Main Memory Data

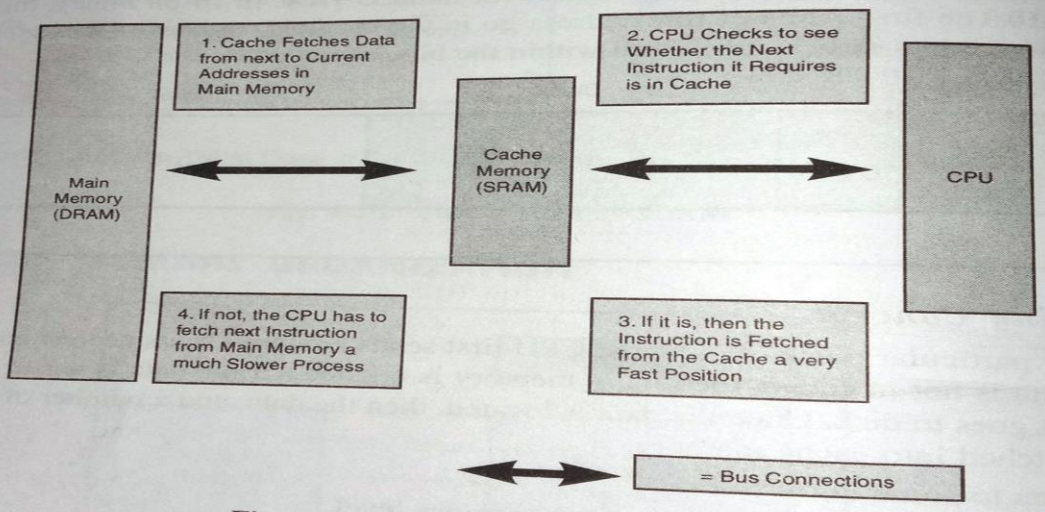


Fig. 6.30. Cache Memory Operation

e purpose of cache memory is to speed up accesses by storing recently used data closer
stead of storing it in main memory. Although cache is much smaller than main memo
me is a fraction of that of main memory. Unlike main memory, which is accessed by ad
typically accessed by content; hence, it is often called *content addressable memory*. B
single large cache memory isn't always desirable— it takes longer to search.

PRINCIPLE OF LOCALITY OF REFERENCE

memory works on the same basic principles of **locality of references,** by copying frec
into the cache rather than requiring an access to main memory to retrieve th

9**

every time. The computer really has no way to know, *a priori*, what data is most likely to be accessed, so it uses the locality principle and transfers an entire block from main memory into cache whenever it has to make a main memory access. If the probability of using something else in that block is high, then transferring the entire block saves on access time. The cache location for this new block depends on two things:

(*i*) The cache mapping policy
(*ii*) The cache size

Which affects whether there is room for the new block or no? : According to the property, over a short interval of time the addresses generated by a typical program refer to a few localized areas of memory repeatedly. The memory hierarchy was developed based on a program behavior known as locality of reference, memory references are generated by the CPU for either instruction or data access. These accesses tend to be clustered in certain regions in time, space and ordering.

These are three dimensions of the locality of reference property:

(*a*) Temporal Locality
(*b*) Spatial Locality
(*c*) Sequential Locality

(*a*) **Temporal Locality :** Recently referenced items (Inst. Or data) are likely to be referenced again in the near future. This is often caused by special program constructs such as iterative loops, process stacks, and temporary variable or sub routines. Once a loop is entered or a subroutine is called, a small code segment will be referenced repeatedly many times, thus temporal locality tends to cluster the access in the recently used areas.

(*b*) **Spatial Locality :** This refers to the tendency for a process to access items whose addresses are near one another. For example operations on tables or array involve access of a certain clustered area in the address space.

(*c*) **Sequential Locality :** In a typical program the execution of instructions follows a sequential order (or the program order) unless branch instructions create out-of-order execution. The ratio of in-order execution to out of order execution is roughly 5 to 1 in ordinary programs. The sequentially in program behavior also contributes to the spatial locality because sequentially coded instructions and array elements are often stored in adjacent locations each type of locality affect the design of memory hierarchy. The spatial locality assists us in determining the size of unit data transfers between adjacent memory levels. The temporal locality also helps to determine the size of memory at successive levels. The principal of locality will guide us the design of cache, main memory and even virtual memory organization.

Principle of Locality

Programs tend to use data and instructions with addressed near or equal to those they have used recently. Temporal Locality items are likely to be referenced again in the near future. Spatial locality items with nearby addressed tend to be referenced close together in time.

Locality Example

Data : Reference array element in succession (stride-1 reference pattern) : Spatial locality reference sum each iteration temporal locality. Reference instructions in sequence. Spatial locality cycle through loop repeatedly : temporal locality


```
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

There are three forms of locality to summarize;

(*i*) *Temporal locality* - Recently-accessed data elements tend to be accessed again.
(*ii*) *Spatial locality* - Accesses tend to cluster.
(*iii*) *Sequential locality* - Instructions tend to be accessed sequentially.