# Computer Organization and Embedded System

## Computer Arithmetic and Memory System (ACtE0402)

Ishwar Kumar Singh
Computer Engineer
Government of Nepal

- Prepared by Er. Ishwar Kumar Singh

# 4.2 Computer Arithmetic and Memory System (ACtE0402):

- Arithmetic and Logical operation,
- The Memory Hierarchy, Internal and External memory,
- Memory Write Ability,
- Storage Permanence,
- Composing Memory
- Cache memory principles,
- Elements of Cache design –
  - Cache size,
  - Mapping function,
  - Replacement algorithm,
  - Write policy,
  - Number of caches,

# Cache Memory

- CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory
- A special very-high-speed memory called cache memory is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate
- The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations
- Cache Memory is placed in between the CPU and the main memory.

# Cache Memory

- When the processor attempts to read a word of memory,
    - Cache is checked first for all memory references.
    - If found, the word is delivered from cache to the processor
    - If not found, the entire block in which that reference resides in main memory is stored in a cache slot (called as a line) and then the word is delivered to the processor.
- Each **line** includes a **tag** (usually a portion of the main memory address) which identifies which particular block is being stored.
- Locality of reference implies that future references will likely come from this block of memory, so that cache line will probably be utilized repeatedly.
- The proportion of memory references, which are found already stored in cache, is called the hit ratio.

# Cache Design

The basic design elements that serve to classify and differentiate cache architectures are as follows:

- Cache Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Number of Caches.

# Cache Size

- The size of the cache should be **small enough** so that the overall average **cost per bit** is close to that of main memory alone
- And the size of the cache should be **large enough** so that the overall **average access time** is close to that of the cache alone.
- The larger the cache, the larger the number of **gates** involved in addressing the cache. The result is that large caches tend to be slightly slower than small ones .
- The available **chip and board area** also limits cache size.
- Since the performance of the cache is very sensitive to the nature of the workload, it is impossible to arrive at a single optimum cache size.

# Replacement Algorithm

- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced.
- **For Direct mapping:** There is only one possible line for any particular block due to which no choice is possible.
- **For Associative and Set-Associative Mapping**: A replacement algorithm is needed which must be implemented in hardware to achieve high speed.
    - **First-In-First-Out (FIFO):** Replace that block in the set that has been in the cache longest.
        - Implemented as a round-robin or circular buffer technique.

# Replacement Algorithm

- **Least Frequently Used (LFU):** Replace that block in the set that has experienced the fewest references.
- **Least Recently Used (LRU):** Replace that block in the set that has been in the cache longest with no reference to it.
- **Random:** Replace a random block in the set.

# Write Policy

- When a line is to be replaced, the original copy of the line in main memory must be updated if any addressable unit in the line has been changed.
- If a block has been altered in cache, it is necessary to write it back out to main memory before replacing it with another block.
- Must not overwrite a cache block unless main memory is up-to date.
- The two major write Polices are:
  - **Write Through**
  - **Write Back**

# Write Policy

- There are two problems to contend with the write polices:
  - More than **one device** may have access to main memory. For example, an **I/O module** may be able to read-write directly to memory. If a word has been altered only in the cache, then the corresponding memory word is invalid. Further, if the I/O device has altered main memory, then the cache word is invalid.
  - A more complex problem occurs when **multiple processors** are attached to the same bus and each processor has its own **local cache**. Then, if a word is altered in one cache, it could conceivably in-validate a word in other caches.

# Write Through

- Using this technique, all write operations are made to main memory as well as to the cache, ensuring that main memory is always valid.
- Any other **processor–cache module** can monitor traffic to main memory to maintain consistency within its own cache.
- The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck.

# Write Back

- Using this technique, updates are made only in the cache.
- When a block is replaced, it is written back to main memory if and only if the dirty bit is set.
- The problem with write back is that portions of main memory are invalid, and hence accesses by I/O modules can be allowed only through the cache.
- This makes for complex circuitry and a potential bottleneck.
- Multiple caches still can become invalidated, unless some **cache coherency system** is used.

# Write Back

Cache Coherency Systems include:

- **Bus Watching with Write Through**: Other caches monitor memory writes using write through and invalidates their own cache line if found the occurrence of memory write operation.
- **Hardware Transparency**: Additional hardware is used to ensure that all updates to main memory via cache are reflected in all caches.
- **Non-Cacheable Memory**: Only a portion of main memory is shared by more than one processor and it is known as non-cacheable
    - In such a system, the shared memory is never copied into the cache.

# Number of Cache

- When caches were originally introduced, the typical system had a single cache.
- More recently, the use of multiple caches has become an important aspect. There are two design issues surrounding number of caches.
- **MULTILEVEL CACHES:**
    - Most contemporary designs include both on-chip and external caches.
    - The simplest such organization is known as a **two-level cache**, with the **internal** cache designated as Level 1 (**L1**) and the **external** cache designated as Level 2 (**L2**).
    - This helps in reducing main memory accesses.

# Number of Cache

- **UNIFIED VERSUS SPLIT CACHES:**
  - Earlier on-chip cache designs consisted of a **single** cache, used to store references to both **data and instructions**, which was considered as the unified approach.
  - More recently, it has become common to **split** the cache into two: one dedicated to instructions and one dedicated to data.
  - These two caches both exist at the **same level**. This is the split cache.

# Mapping Function

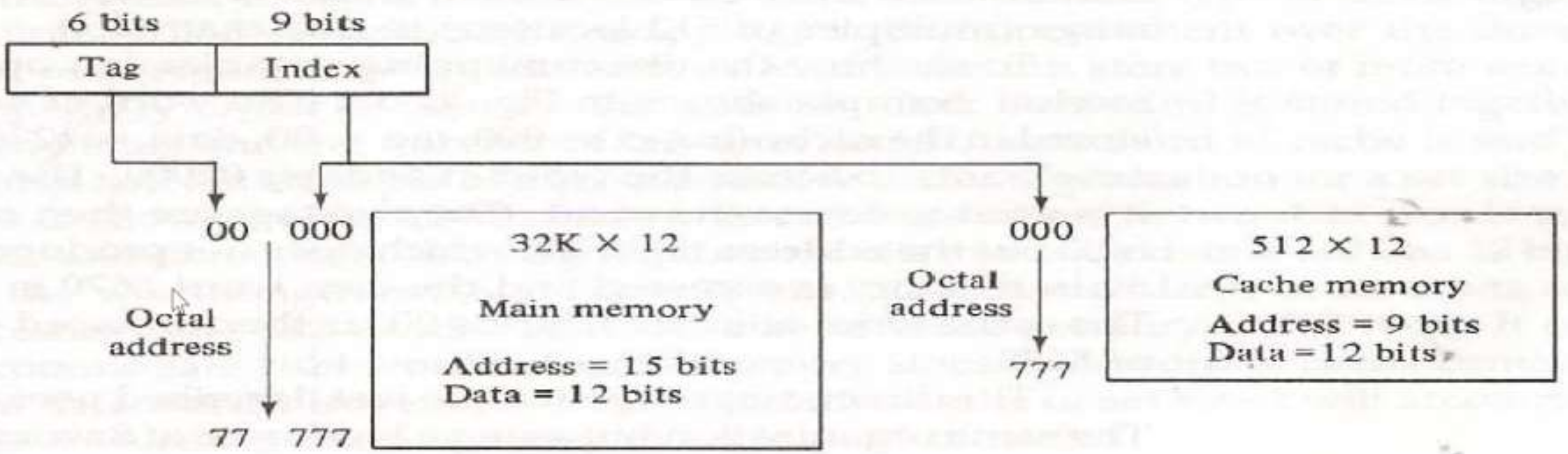
- An algorithm is needed for mapping main memory blocks into cache lines.
- A means is needed for determining which **main memory block** currently occupies a **cache line**.
- The choice of the mapping function dictates how the cache is organized.
- There are three techniques which can be used for mapping
  - Direct Mapping
  - Associative Mapping
  - Set-Associative Mapping

# Direct Mapping

- Direct Mapping is the simplest technique which maps **each block of main memory** into **only one possible cache line**.
- The mapping is expressed as, **i = j modulo m** ;
- where, i = cache line number
  j = main memory block number
  m = number of lines in the cache
- The mapping function is easily implemented using the main memory address.
- Example: The CPU address of 15 bits is divided into two fields: the nine least significant bits constitute the index field and remaining six bits form the tag field.

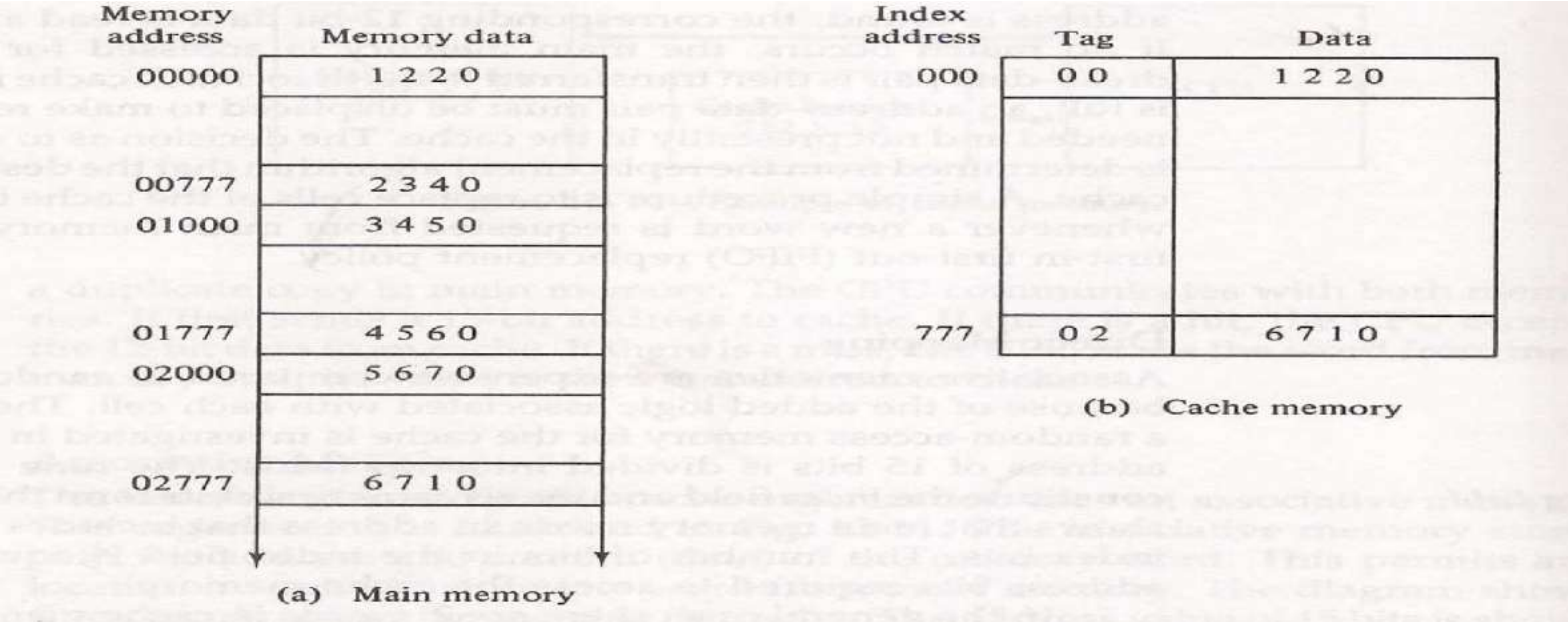
# Direct Mapping

Figure 12-12 Addressing relationships between main and cache memories.




- The main memory needs an address that includes both the tag and the index bits.
- The number of bits in the index field is equal to the number of address bits required to access the cache memory line.

# Direct Mapping



# Direct Mapping

- The direct mapping cache organization uses the n- bit address to access the main memory and the k-bit index to access the cache.
- Each word in cache consists of the data word and associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits.
- When the CPU generates a memory request, the index field is used for the address to access the cache.
- The tag field of the CPU address is compared with the tag in the word read from the cache.
- If the two tags match, there is a hit and the desired data word is in cache.
- If there is no match, there is a miss and the required word is read from main memory.

# Direct Mapping

- **Advantages:** Simple and Inexpensive to Implement.
- **Disadvantage:**
  - Since there is a fixed cache location for any given block.
  - Thus, if a program happens to reference words repeatedly from two different blocks that map into the same line,
  - then the blocks will be continually swapped in the cache, and the hit ratio will be low (a phenomenon known as ***thrashing***).

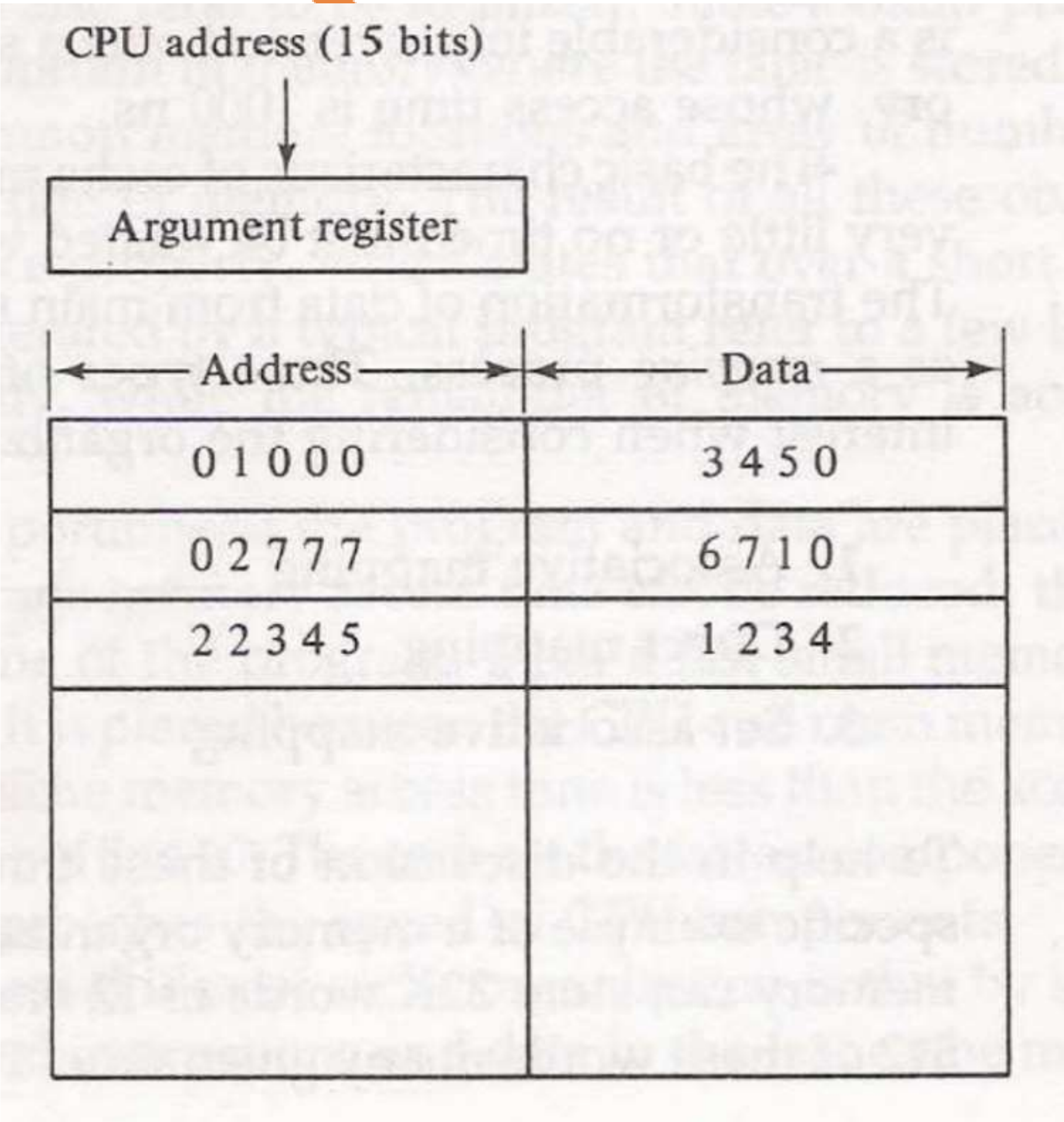
# Associative Mapping

- It overcomes the disadvantage of direct mapping by **permitting each main memory block** to be loaded into **any line of the cache**.
- In this case, the cache control logic interprets a memory address simply as a Tag and a Word field.
- The Tag field uniquely identifies a block of main memory.

  i.e. The associative memory stores both address and content(data) of the memory word.
- To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's tag for a match.

# Associative Mapping

- A CPU address of 15-bits is placed in the argument register and the associative memory is searched for a matching address.
- If address is found, the corresponding 12-bit data is read and sent to the CPU.
- If no match occurs, the main memory is accessed for the word.



# Associative Mapping

- **Advantages:**
  - Flexibility to replace block when a new block is read into the cache.
  - Replacement Algorithm can be used to maximize the hit ratio.

- **Disadvantage:** Complex Circuitry is required to examine the tags of all cache lines in parallel.

# Set-Associative Mapping

- In this method, blocks of cache are grouped into sets, and the mapping allows a block of main memory to reside in any block of a specific set.

- From the flexibility point of view, it is in between to the other two methods.

- In this case, the cache is divided into ‘v’ sets, each of which consists of ‘k’ lines.

- The relationships are **m = v * k** & **i = j modulo v** where,
  m = number of lines in the cache, v = number of sets,
  k = number of lines in each set
  i = cache set number, j = main memory block number,

- Prepared by Er. Ishwar Kumar Singh

# Set-Associative Mapping

| Index | Tag | Data | Tag | Data |
|---|---|---|---|---|
| 000 | 01 | 3450 | 02 | 5670 |
| 777 | 02 | 6710 | 00 | 2340 |

# Set-Associative Mapping

- When the CPU generates a memory request,
  - the index values of the address is used to access the cache
  - the tag field of the CPU address is then compared with both tags in the cache
  - the comparison logic is defined by an associative search of the tags in the set.

# Thank You.