# Contents

# Main Memory ( 1 )

- Main Memory - part of computer where program and data are stored during execution.
- It consists of a number of cells (or locations), each of which can store a piece of information (data, instruction, character or number).
- The size of the cell can be single byte or several successive bytes (word) - ***byte-addressable*** or ***word-addressable*** computer.

# Main Memory ( 2 )

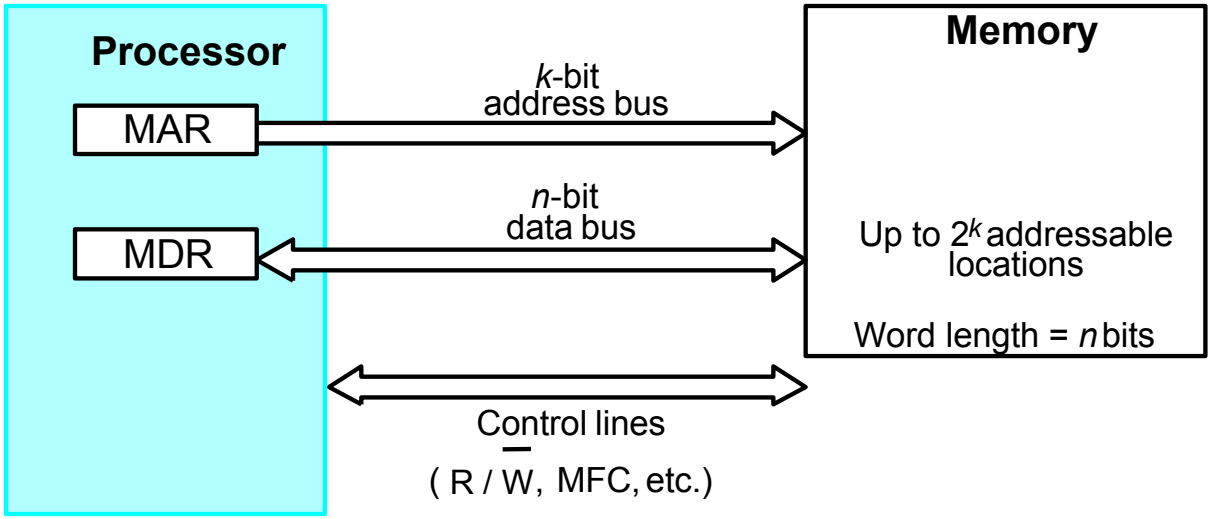
- Each cell has a reference number, called address, by which program can refer to it.
- If a memory address has *k* bits, the maximum number of cell directly addressable is $2^k$.
  - Example: For 16-bit addressing, the maximum number of cells will be:

$$2^{16} = 65536 \text{ memory cells}$$

- The maximum size of address references available in main memory for a computer system is called the size of the main memory.

# Main Memory ( 3 )

- The basic unit of memory is the binary digit "***bit***". A bit contains a "0" or "1".
- The most common definition of the ***word*** length of a computer is the number of bits actually stored or retrieved in one memory access.
- Word length and address length are independent.




Connection of the memory to the processor.

# Main Memory ( 4 )

- **Memory Access Time**
  - The time that elapses between the initiation and the completion of a memory access operation.
    - e.g., the time between the READ and MFC (Memory Function Complete) signals
- **Memory Cycle Time**
  - The minimum time delay required between two successive memory access operations.
  - The cycle time is usually slighter longer than the access time.

# MEMORY ORGANISATION

# C Programming Preview

## Sizes of C Objects (in Bytes)

| C Data Type | Typical 32-bit | Intel IA32 | x86-64 |
|---|---|---|---|
| char | 1 | 1 | 1 |
| short | 2 | 2 | 2 |
| int | 4 | 4 | 4 |
| long | 4 | 4 | 8 |
| long long | 8 | 8 | 8 |
| float | 4 | 4 | 4 |
| double | 8 | 8 | 8 |
| long double | 8 | 10/12 | 10/16 |
| char * | 4 | 4 | 8 |

- Or any other pointer

Memory is always organized Byte-Oriented

**BUT**

Machine has “Word Size”

# Machine Words

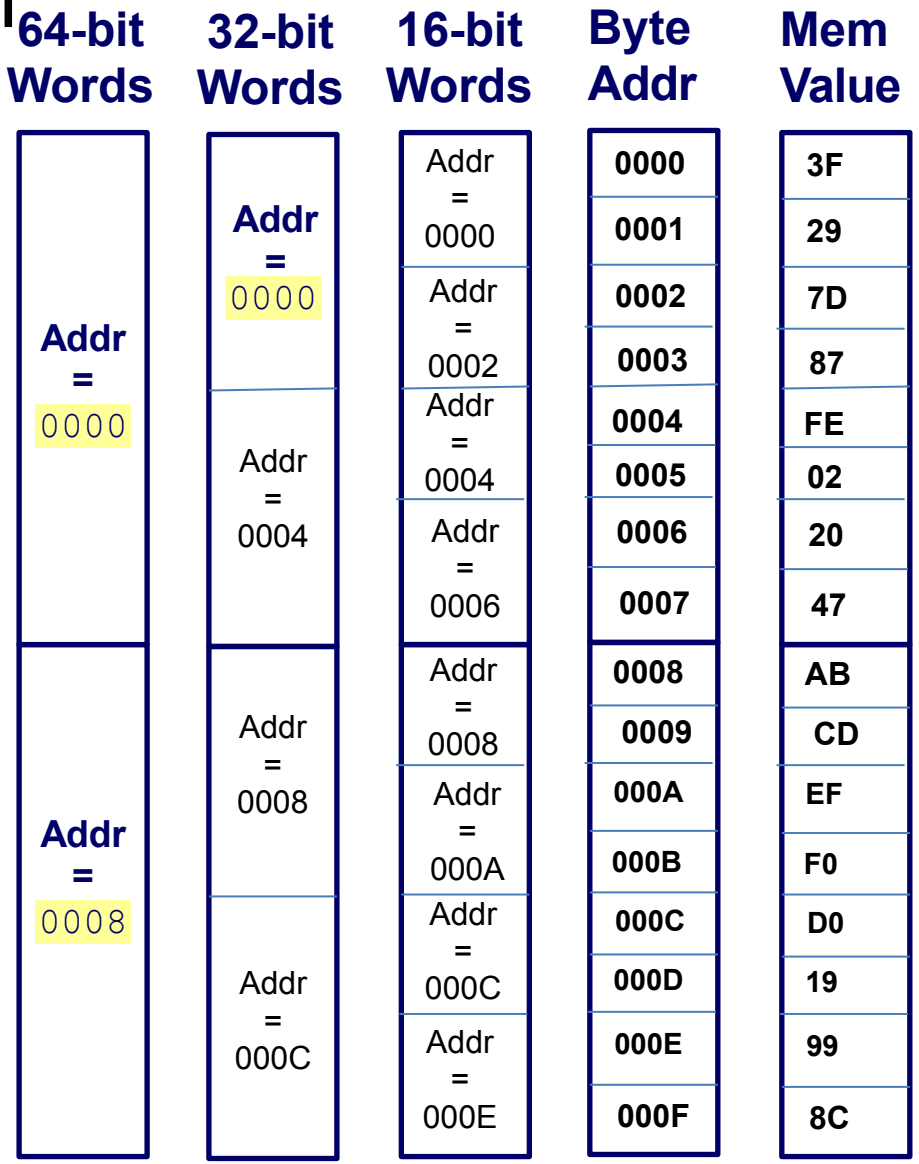
## Machine Has "Word Size"

- Nominal size of integer-valued data
  - Including addresses
- Most current machines use 32 bits (4 bytes) words
  - Limits addresses to 4GB
  - Becoming too small for memory-intensive applications
- High-end systems use 64 bits (8 bytes) words
  - Potential address space ≈ 1.8 X $10^{19}$ bytes
  - x86-64 machines support 48-bit addresses: 256 Terabytes
- Machines support multiple data formats
  - Fractions or multiples of word size
  - Always integral number of bytes

# Byte & Word Oriented Memory Organization

## Addresses Specify Byte Locations

- Addresses of successive words differ by 2 (16-bit), 4 (32-bit) or 8 (64-bit)



# Byte Ordering

**How should bytes within multi-byte word be ordered in memory?**

**Conventions**

- **Big Endian: Sun, PPC Mac, Internet**
  - **Least significant byte has highest address**
- **Little Endian: x86**
  - **Least significant byte has lowest address**

# Byte Ordering Example

## Big Endian

- Least significant byte has highest address

## Little Endian

- Least significant byte has lowest address

## Example

- Variable `x` has 4-byte representation `0x01234567`
- Address given by `&x` is `0x100`

**Big Endian**

| 0x100 | 0x101 | 0x102 | 0x103 |
|---|---|---|---|
| 01 | 23 | 45 | 67 |

**Little Endian**

| 0x100 | 0x101 | 0x102 | 0x103 |
|---|---|---|---|
| 67 | 45 | 23 | 01 |

# Types of Memory Unit ( 1 )

- **Random-Access Memory (RAM)**
  - Any location can be accessed for a Read or Write operation in some constant amount of time that is independent of the memory location.
  - Static RAM (SRAM)
    - Memories that consist of circuits capable of retaining their state as long as power is applied.
    - SRAMs are fast (a few nanoseconds access time) but their cost is high.
  - Dynamic RAM (DRAM)
    - These memory units are capable of storing information for only tens of milliseconds, thus require periodical refresh to maintain the contents.

# Types of Memory Unit ( 2 )

- **Read-Only Memory (ROM)**
  - Nonvolatile memory
  - Data are written into a ROM when it is manufactured. Normal operation involves only reading of stored data.
  - ROM are useful as control store component in a micro-programmed CPU (micro-coded CPU).
  - ROM is also commonly used for storing the *bootstrap loader*, a program whose function is to load the boot program from the disk into the memory when the power is turn on.

# Types of Memory Unit ( 3 )

- PROM (Programmable ROM)
  - data is allowed to be loaded by user but this process is irreversible
  - provide a faster and less expensive approach when only a small number of data are required
- EPROM (Erasable, Programmable ROM)
  - stored data can be erased by exposing the chip to ultraviolet light and new data to be loaded
- EEPROM
  - stored data can be erased electrically and selectively
  - different voltages are needed for erasing, writing, and reading the stored data

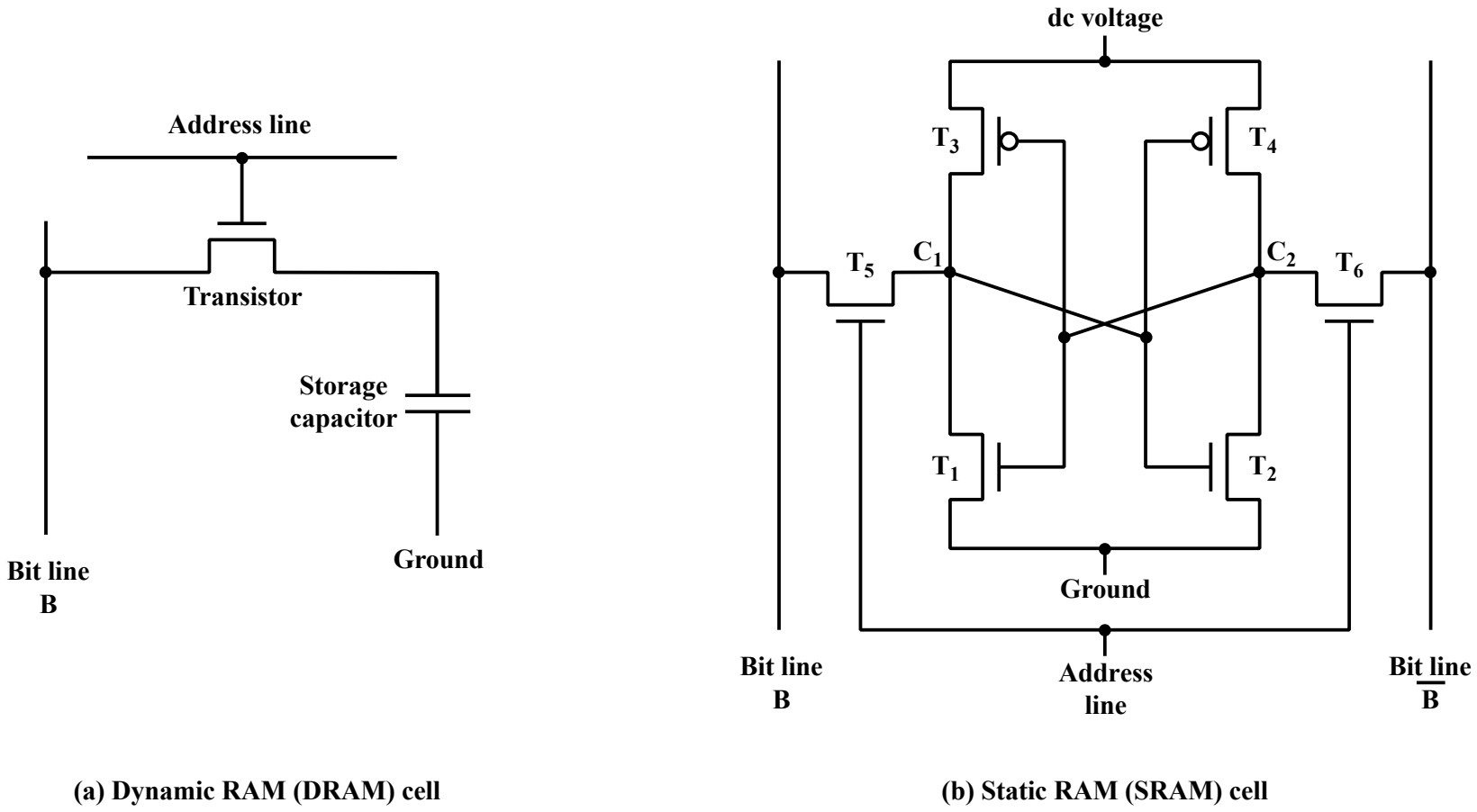
# Types of Memory Unit ( 4 )

- Flash memory
  - similar to EEPROM technology
  - it is possible to read the contents of a single cell, but it is only possible to write an entire block of cells
  - greater density, higher capacity, lower cost per bit, low power consumption
  - typical applications: hand-held computers, digital cameras, MP3 music players
  - large memory modules implementation: flash cards and flash drives

# Types of Memory Unit ( 5 )

| Type | Category | Erasure | Byte alterable | Volatile | Typical use |
|---|---|---|---|---|---|
| SRAM | Read/write | Electrical | Yes | Yes | Level 2 cache |
| DRAM | Read/write | Electrical | Yes | Yes | Main memory |
| ROM | Read-only | Not possible | No | No | Large volume appliances |
| PROM | Read-only | Not possible | No | No | Small volume equipment |
| EPROM | Read-mostly | UV light | No | No | Device prototyping |
| EEPROM | Read-mostly | Electrical | Yes | No | Device prototyping |
| Flash | Read/write | Electrical | No | No | Film for digital camera |

A comparison of various memory types.



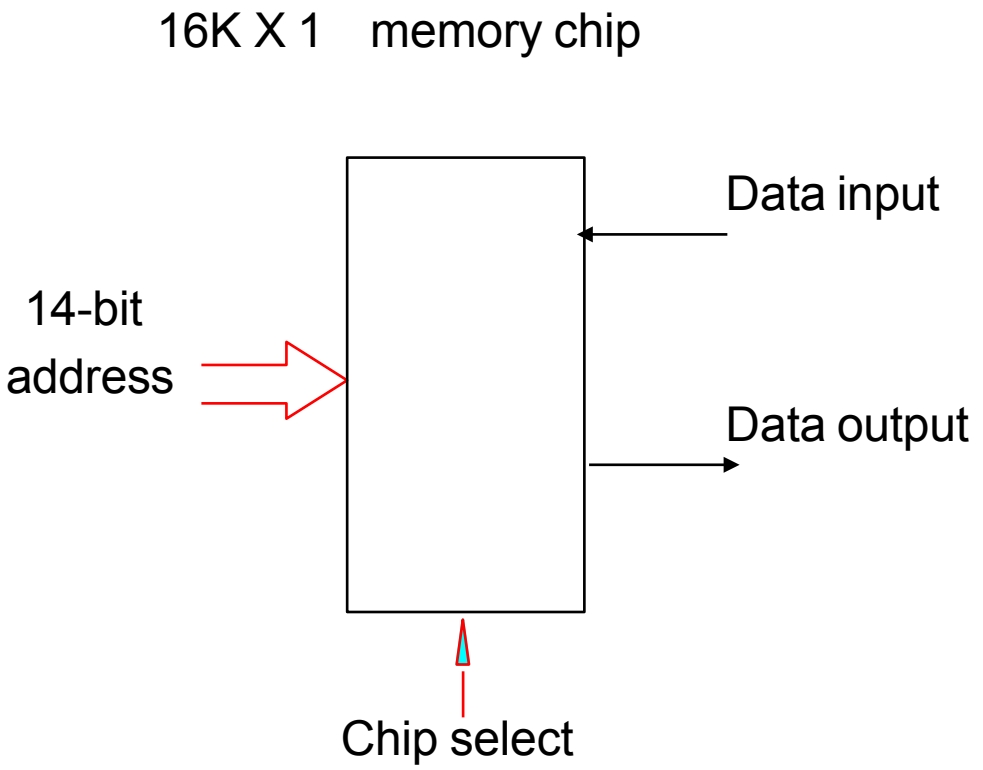

(a) Dynamic RAM (DRAM) cell

(b) Static RAM (SRAM) cell

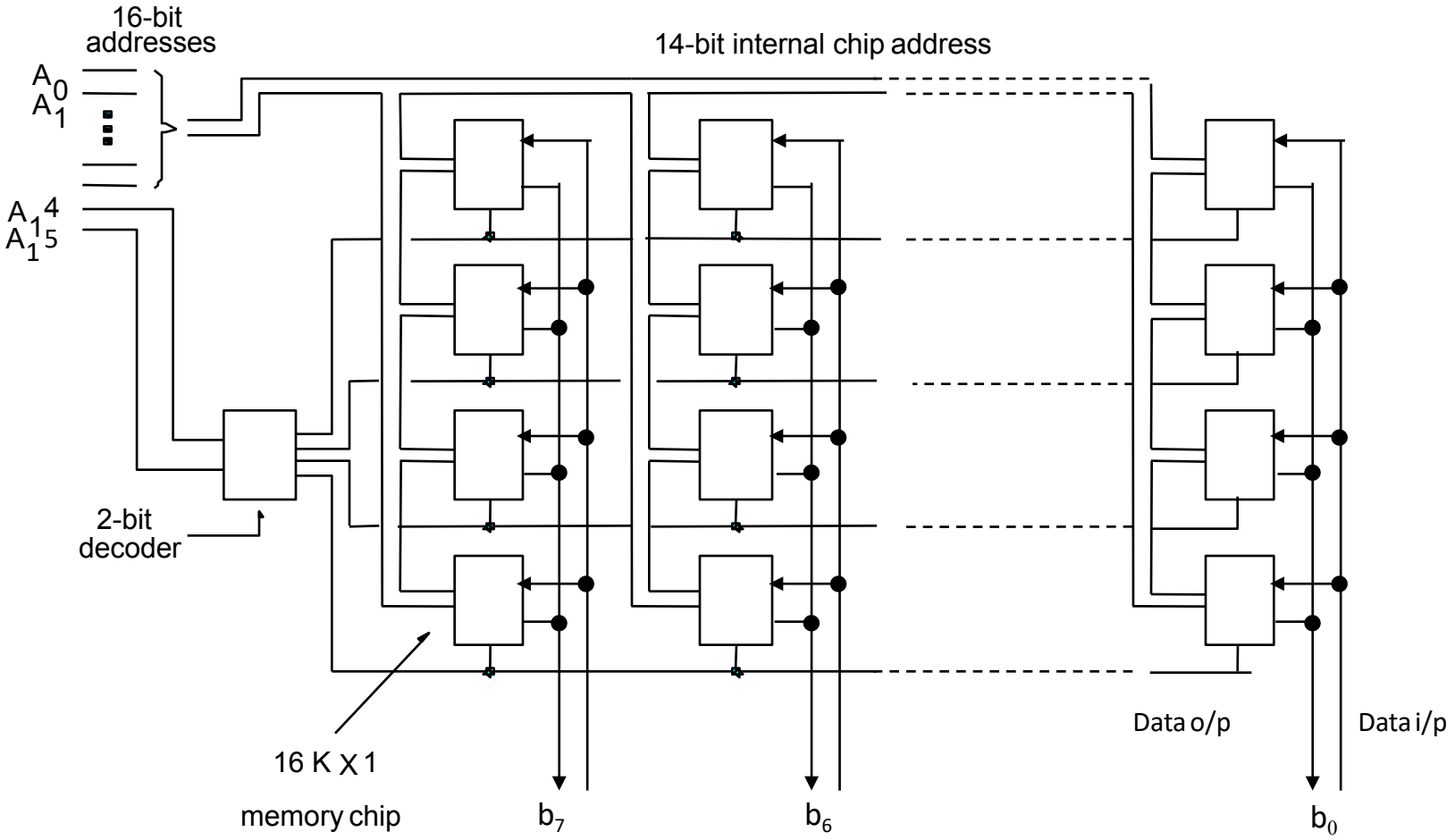
**Figure 5.2 Typical Memory Cell Structures**

# Memmory Systems (1)

- Implement a 64K X 8 memory using 16K X 1 memory chips
- Number of rows = 64K/16K = 4
- Number of columns = 8/1 = 8.
- 64K = 16 bit address.
- Each small chip needs 14 bit (16K) address
- 16-14 = 2 bits for selecting one of the four rows.

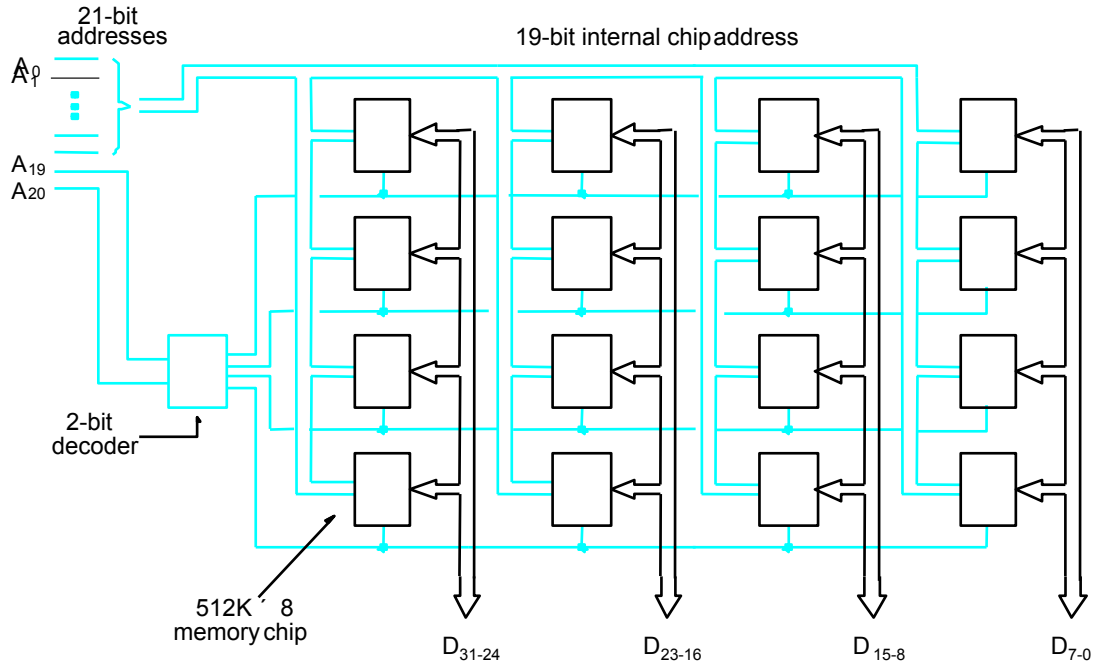


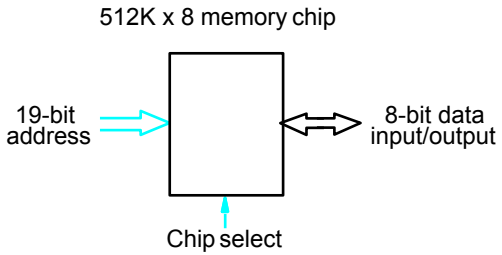
# Memory Systems (2)



# Memory Systems (3)

A memory system with 2M words (2Mx32) formed by (512K x 8) memory chips.



# Memory Systems ( 4 )

- Each chip has a control input called Chip Select (CS) used to enable the chip.
- 21 address bits are needed to select a 32-bit word.
  - The high-order 2 bits are decoded to determined which of the 4 CS control signals are activated.
  - The remaining 19 bits are used to access specific byte locations inside each chip of the selected row.

# Memory Systems ( 5 )

- Single In-line Memory Modules (SIMMs) and Dual In-line Memory Modules (DIMMs)
  - An assembly of several memory chips on a separate small board that plugs vertically into a single socket on the motherboard.
  - Occupy a smaller amount of space.
  - Allow easy expansion.

# Memory Interleaving ( 1 )

- Main memory is structured as a number of physical modules (chip).
- Each memory module has its own *Address Buffer Register* (ABR) and *Data Buffer Register* (DBR).
- Memory access may proceed in more than one module simultaneously → the aggregate rate of transmission of words to and from the main memory can be increased.
- How individual addresses are distributed over the modules is critical in determining the average number of modules that can be kept busy.

# Memory Interleaving ( 2 )

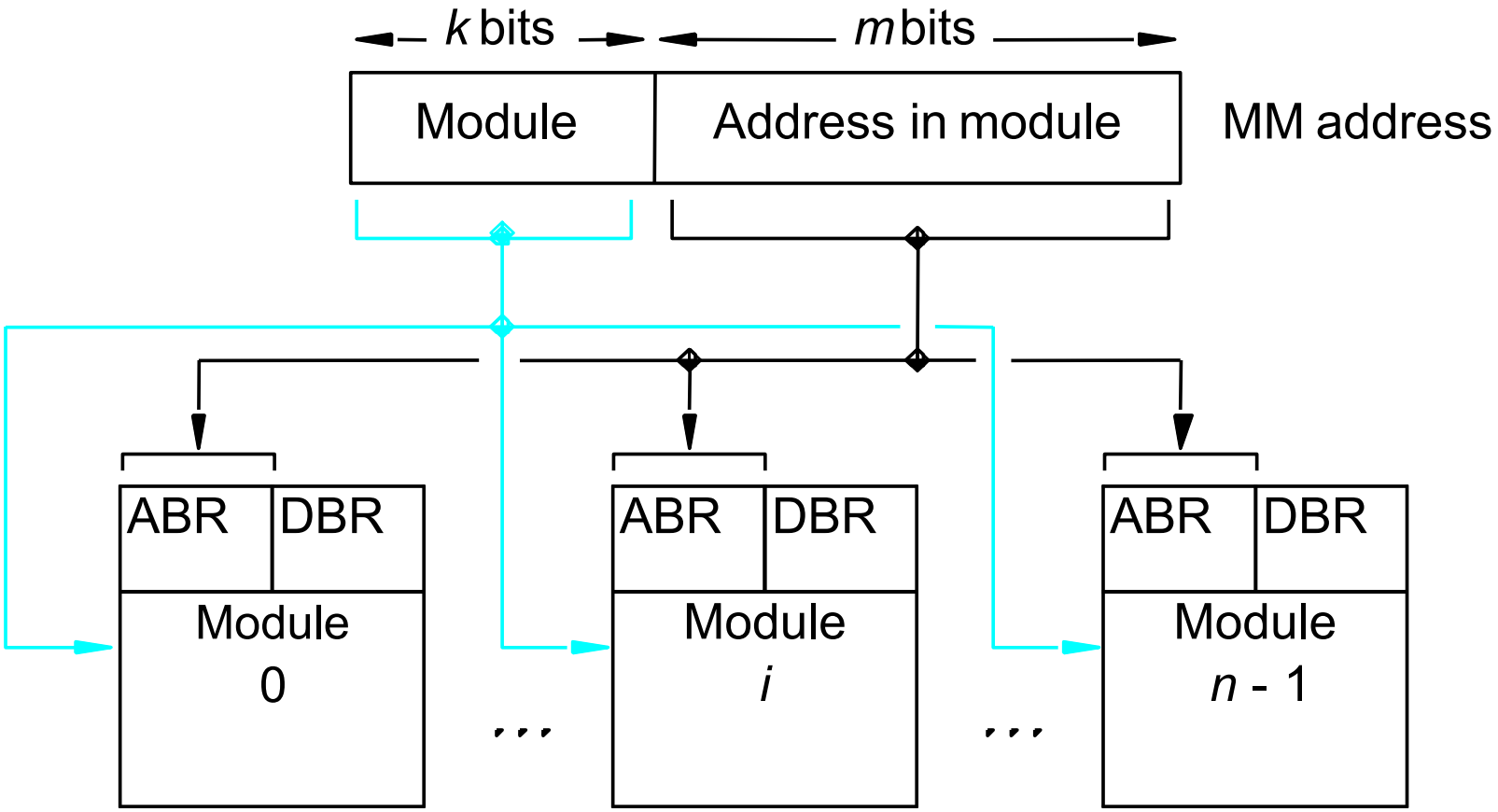
- There are two memory address layouts :

  (a) Consecutive words in a module

  – The address consists of :

    - (1) high-order *k* bits identify a single module (0 to *n*-1)
    - (2) low-order *m* bits point to a particular word in that module
    - (3) Accessing consecutive addresses will keep ___ module busy.

# Memory Interleaving ( 3 )



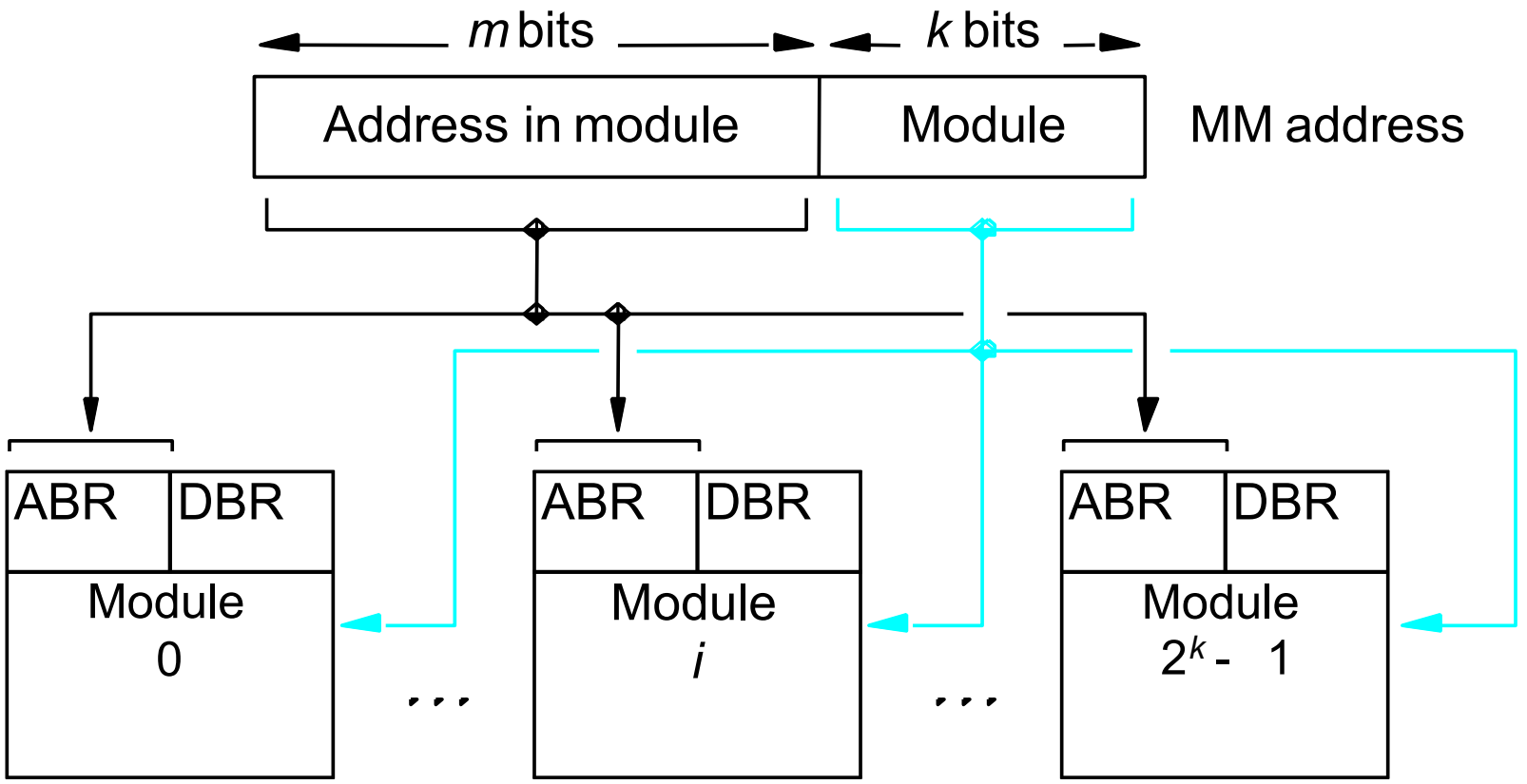

(a) Consecutive words in a module

# Memory Interleaving ( 4 )

(b) Consecutive words in consecutive modules

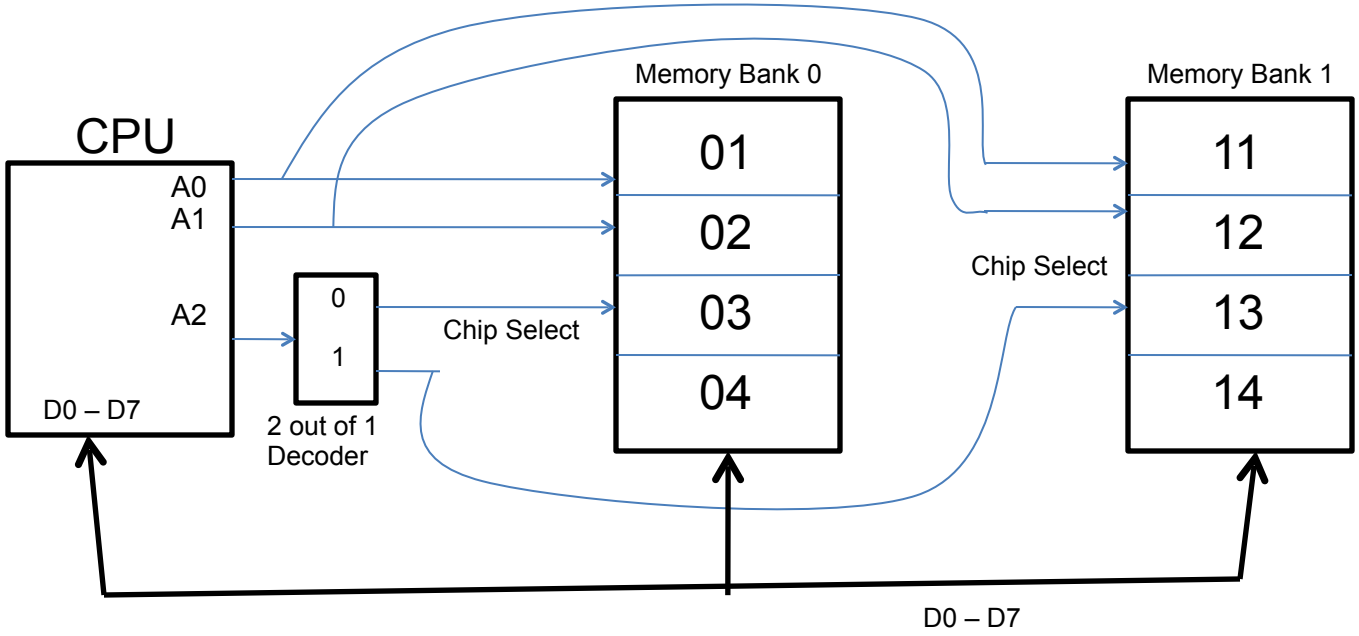
- The address consists of :
  - (1) low-order *k* bits determine a module
  - (2) high-order *m* bits name a location within that module
  - (3) Accessing consecutive addresses will keep several modules busy at any one time
- It is called ***Memory Interleaving***.
- Faster access to a block of data.
- Higher average utilization of the memory system.

# Memory Interleaving ( 5 )




(b) Consecutive words in consecutive modules

# Memory Interleaving Example*



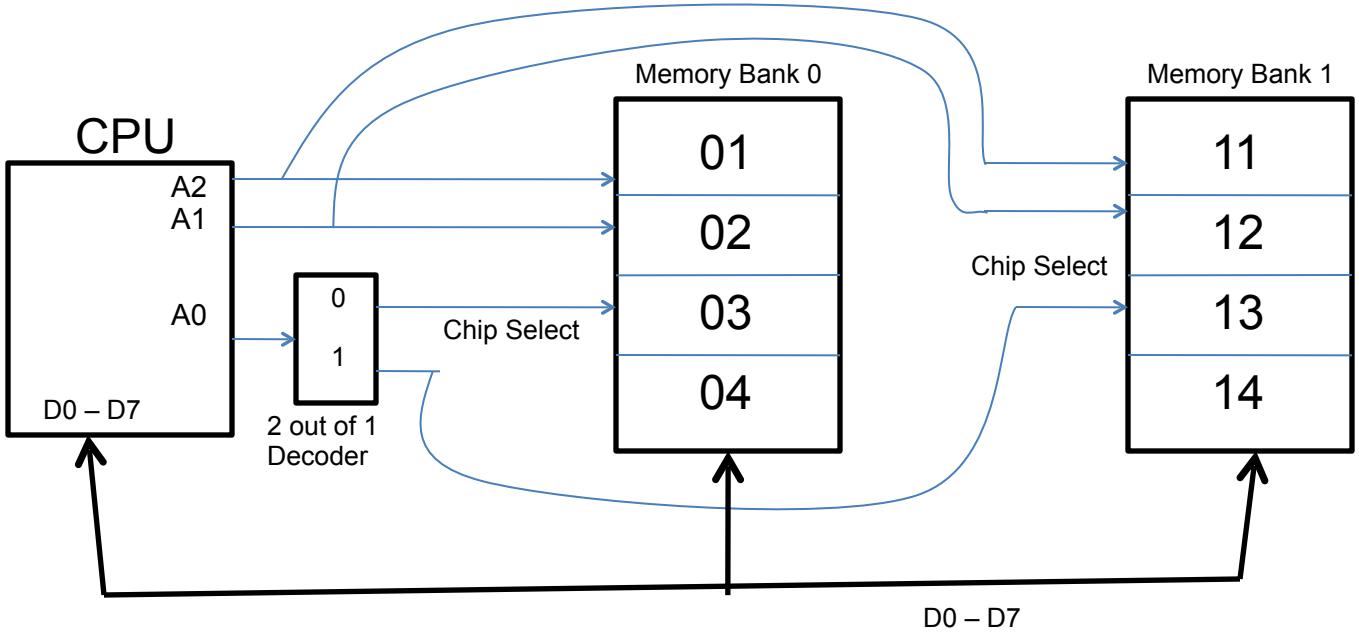

| Addr (bits) A2 A1 A0 | Data (Byte) | Read Time (Clock Cycle) |
|---|---|---|
| 000 | 01 | 2 |
| 001 | 02 | 3 |
| 010 | 03 | 3 |
| 011 | 04 | 3 |
| 100 | 11 | 2 |
| 101 | 12 | 3 |
| 110 | 13 | 3 |
| 111 | 14 | 3 |

Consecutive Bytes in the same Memory Bank



*lots of assumptions to elaborate the Basic Concept

# Memory Interleaving Example*




| Addr (bits) A2 A1 A0 | Data (Byte) | Read Time (Clock Cycle) |
|---|---|---|
| 000 | 01 | 2 |
| 001 | 11 | 2 |
| 010 | 02 | 2 |
| 011 | 12 | 2 |
| 100 | 03 | 2 |
| 101 | 13 | 2 |
| 110 | 04 | 2 |
| 111 | 14 | 2 |

Consecutive Bytes in the Consecutive Memory Bank



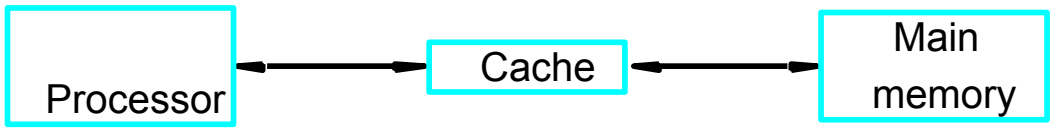
*lots of assumptions to elaborate the Basic Concept

# Concept of Cache

# Cache Memory ( 1 )

- During the execution of a typical program it is often occurred in a few localized areas of the program (in memory) at any given interval of time –
- **Locality of Reference**
  - ***temporal***: a recently executed instructions is likely to be executed again very soon, e.g., loop, stack.
  - ***spatial***: instructions in close proximity to a recently executed instruction are also likely to be executed soon.

# Cache Memory ( 2 )




- *Cache memory* used to store the active segments of the program will then reduce the average memory access time, resulting faster execution for the program.
- It is usually implemented using SRAM, which are very fast memory (a few ns access time) but expensive.

# Cache Memory ( 3 )

- In a read operation, the *block* containing the location specified is transferred into the cache from the main memory, if it is not in the cache (a *miss*). Otherwise (a *hit*), the block can be read from the cache directly.
- The performance of cache memory is frequently measured in terms of hit ratio. High hit ratio verifies the validity of the local reference property.

$$\text{hit ratio} = \frac{\text{Number of hits}}{\text{Total number of memory references}}$$

# Cache Memory ( 4 )

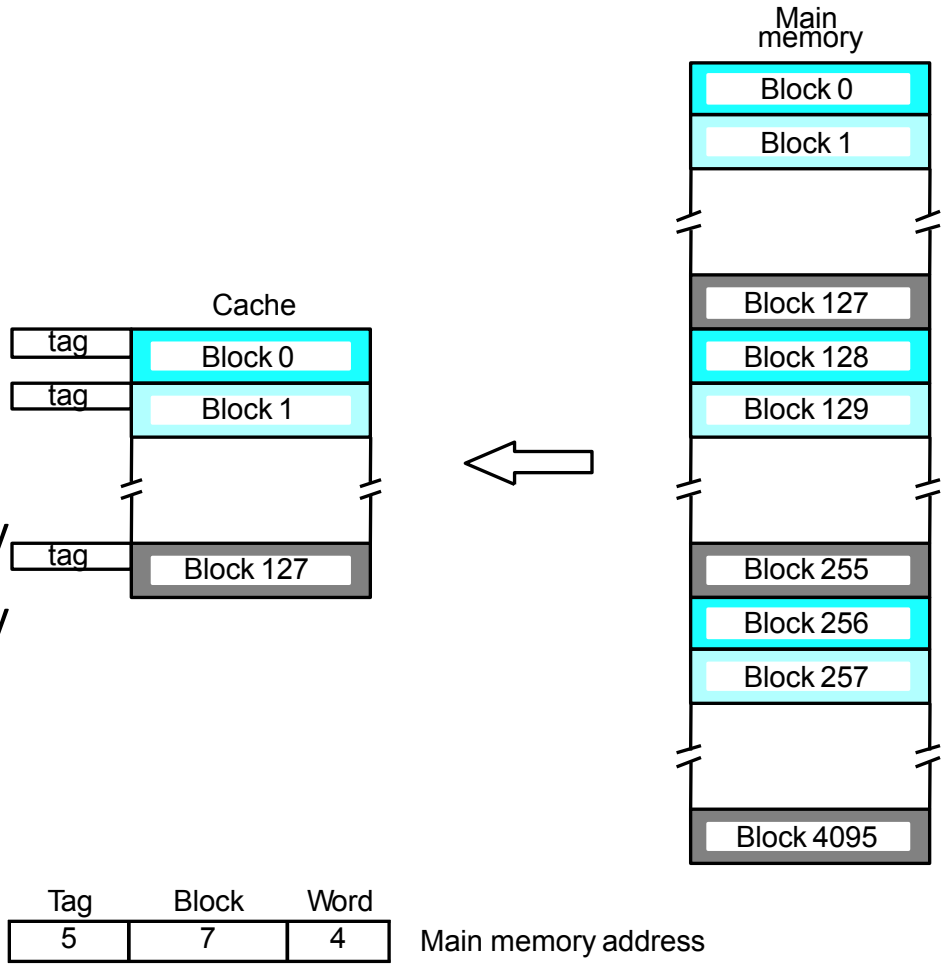
- Two different ways of write access for system with cache memory :
  - (1) ***Write-through*** method – the cache and the main memory locations are updated simultaneously.
  - (2) ***Write-back*** method - cache location updated during a write operation is marked with a dirty or modified bit. The main memory location is updated later when the block is to be removed from the cache.

# Cache Memory ( 5 )

- The correspondence between the main memory blocks and those in the cache is specified by a mapping function.
  - Direct Mapping
  - Associative Mapping
  - Set-associative Mapping
- To explain the mapping procedures, we consider
  - a 2K cache consisting of 128 blocks of 16 words each, and
  - a 64K main memory addressable by a 16-bit address, 4096 blocks of 16 words each. [Assumed Memory is Word Organized]

# Direct Mapping ( 1 )

- Block *j* of the main memory maps onto block *j* modulo 128 of the cache.
- The 7-bit cache block field determines the cache position.
- The high-order 5 tag bits identify which of the 32 blocks is currently resident in the cache.

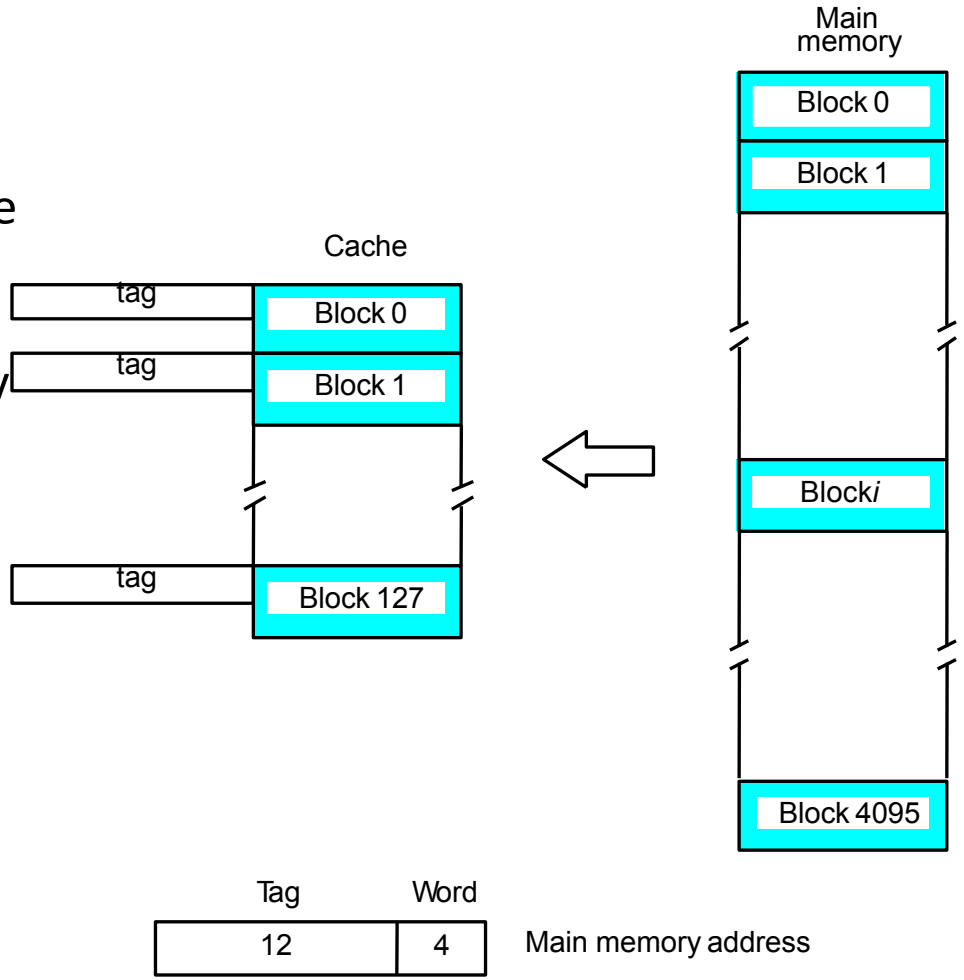


# Direct Mapping ( 2 )

- Since more than one memory block is mapped onto a given cache block position, contention may arise for that position even when the cache is not full.
- This technique is easy to implement, but it is not flexible.

# Associative Mapping ( 1 )

- A main memory block can be placed into any cache block position ⇒ the space in the cache can be used more efficiently.
- The 12 tag bits identify a memory block residing in the cache.
- The lower-order 4 bits select one of 16 words in a block.



# Associative Mapping ( 2 )

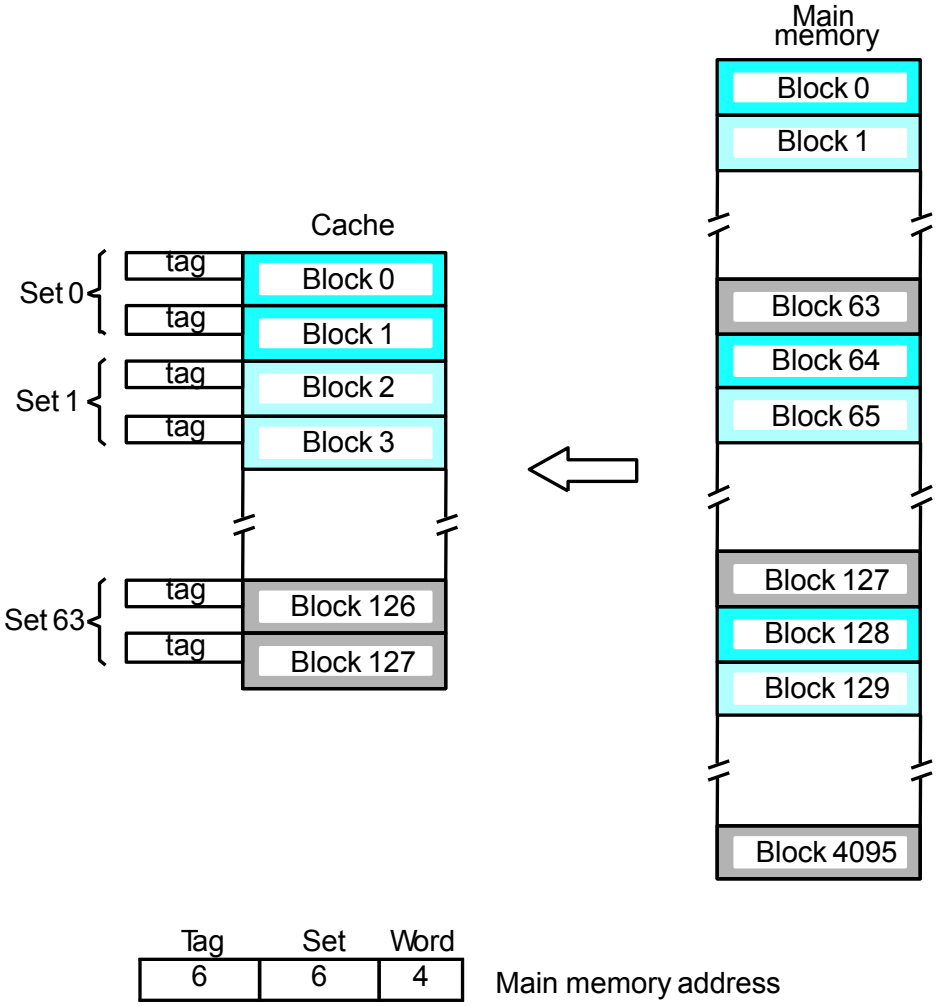
- The cost of an associative cache is relatively high because of the need to search all 128 tags to determine whether a given block is in the cache.
- For performance reasons, ***associative search*** must be done in parallel.

# Set-Associative Mapping ( 1 )

- Blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set.
- A cache that has *k* blocks per set is referred to as a *k*-way set-associative cache.
- The contention problem of the direct method is eased.
- The hardware cost of the associative method is reduced.

# Set-Associative Mapping ( 2 )

- The 6-bit set field determines which set of the cache might contain the desired block.
- The tag field is associatively compared to the tags of the two blocks of the set to check if the desired block is present.



# Replacement Algorithms

- Difficult to determine which blocks to kick out
- Least Recently Used (LRU) block
- The cache controller tracks references to all blocks as computation proceeds.
- Increase / clear track counters when a hit/miss occurs

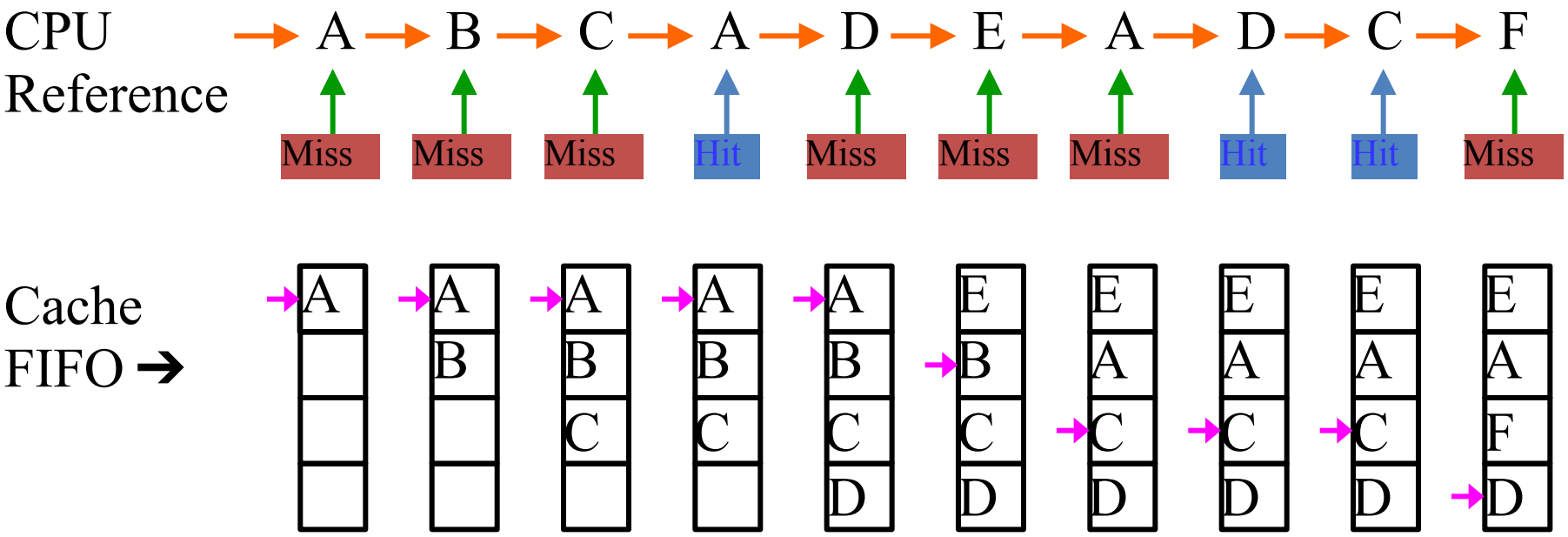
# Replacement Algorithms

- For Associative & Set-Associative Cache

  Which location should be emptied when the cache is full and a miss occurs?

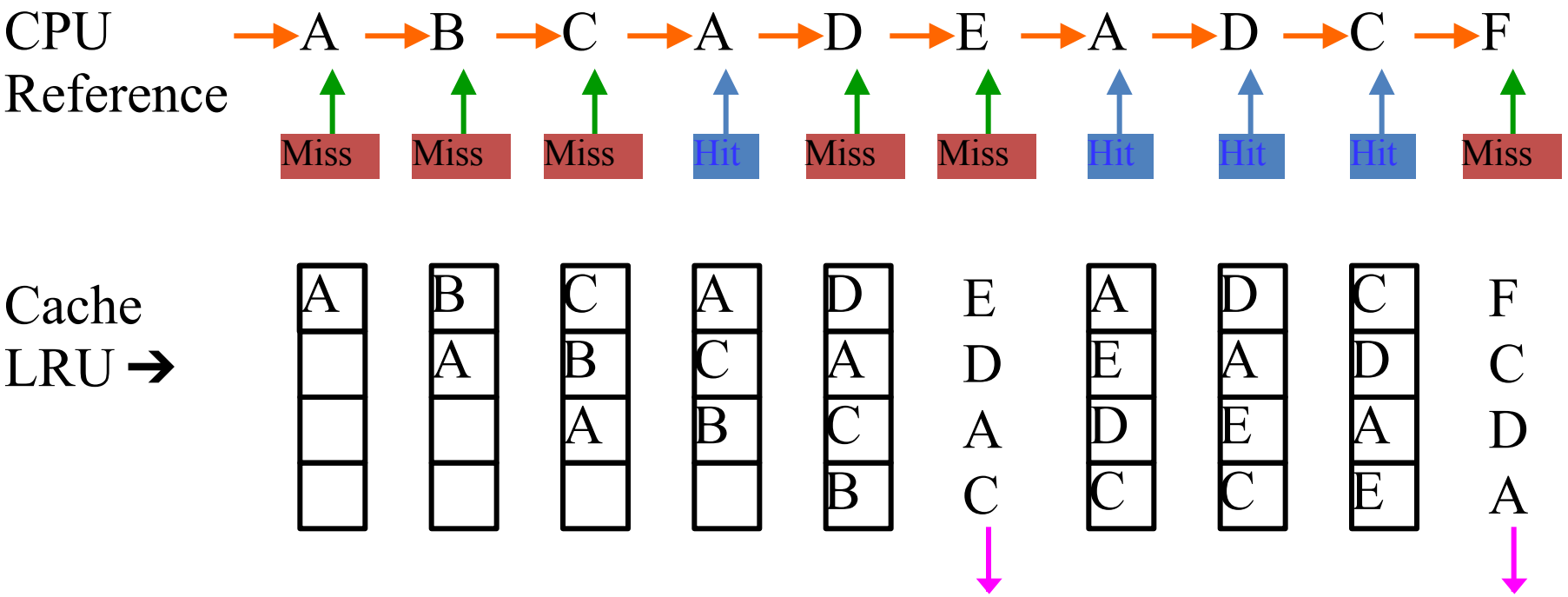
  - First In First Out (FIFO)
  - Least Recently Used (LRU)
- Distinguish an *Empty* location from a *Full* one
  - Valid Bit

# Replacement Algorithms




Hit Ratio = 3 / 10 = 0.3

# Replacement Algorithms




Hit Ratio = 4 / 10 = 0.4

Watch a YouTube video on cache