Computer Architecture and Operating System

Memory Systems

Prof. Indranil Sengupta

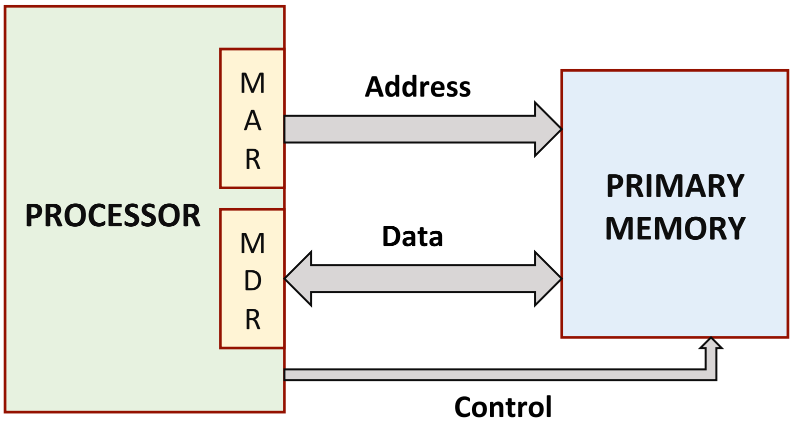
Department of Computer Science and Engineering

IIT Kharagpur

PROCESSOR MEMORY INTERACTION

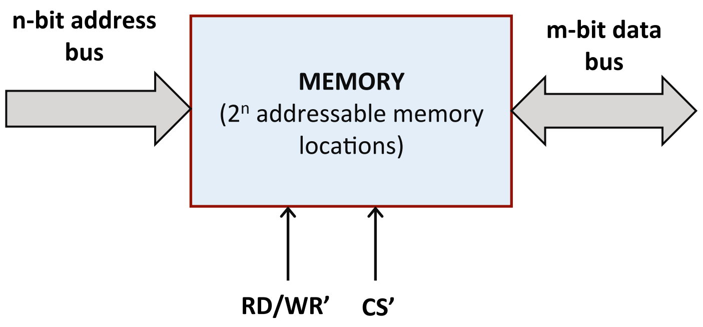
Connection between Processor and Memory

- Address bus provides the address of the memory location to be accessed.
- Data bus transfers the data read from memory, or data to be written into memory.
  - Bidirectional.
- Control bus provides various signals like READ, WRITE, etc.



An Example Memory Module

- *n address lines* :: The maximum number of memory locations that can be accessed is 2^n.
- *m data lines* :: The number of bits stored in every addressable location is *m*.
- The RD/WR’ control line selects the memory for reading or writing (1: read, 0: write).
- The chip select line (CS’) when active (=0) will enable the chip; otherwise, the data bus is in the ***high impedance state***.

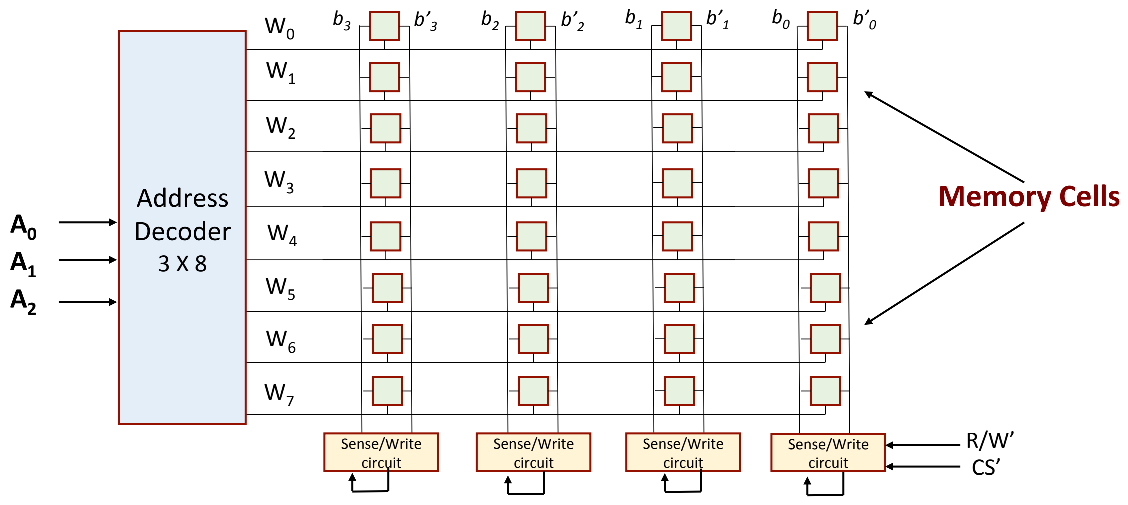


The memory size is specified as 2^n x m

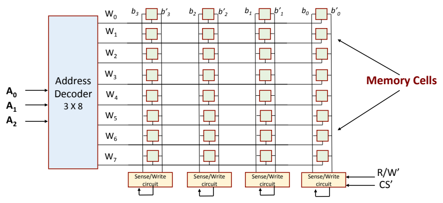
Access Time, Latency and Bandwidth

- Terminologies used to measure speed of the memory system.
  - a) **Memory Access Time**: Time between initiation of an operation (Read or Write) and completion of that operation.
  - b) **Latency**: Initial delay from the initiation of an operation to the time the first data is available.
  - c) **Bandwidth**: Maximum speed of data transfer in bytes per second.

Organization of Cells in an 8 x 4 Memory Chip



- Every row of the cell array constitutes a *memory word*.
- A 3 x 8 *decoder* is required to access any one of the 8 rows.
- The rows of the cells are connected to the *word lines*.
- Individual cells are connected to two *bit lines*.
  - Bit *b* and its complement *b'*.
  - Required for reading and writing.



- Cells in each column are connected to a *sense/write circuit* by the two bit lines.
- Other than address and data lines, there are two *control lines*: R/W' and CS' (Chip Select).
  - CS is required to select one single chip in a multi-chip memory system.

STATIC AND DYNAMIC RAM

Introduction

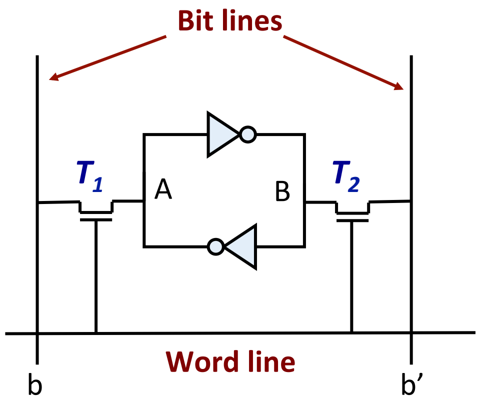
- Broadly two types of semiconductor memory systems:
  - a) Static Random Access Memory (SRAM)
  - b) Dynamic Random Access Memory (DRAM)
    - Asynchronous DRAM
    - Synchronous DRAM
- Vary in terms of speed, density, volatility properties, and cost.
  - Present-day main memory systems are built using DRAM.
  - Cache memory systems are built using SRAM.

Static Random Access Memory (SRAM)

- SRAM consists of circuits that store the data as long as power is applied.
  - It is a type of semiconductor memory that uses bistable latching circuitry (flip-flop) to store each bit.
- SRAM technology:
  - Can be built using 4 or 6 MOS transistors.
  - Modern SRAM chips in the market uses 6-transistor implementations for CMOS compatibility.
  - Also used to implement cache memories in computer systems.
    - To be discussed later.

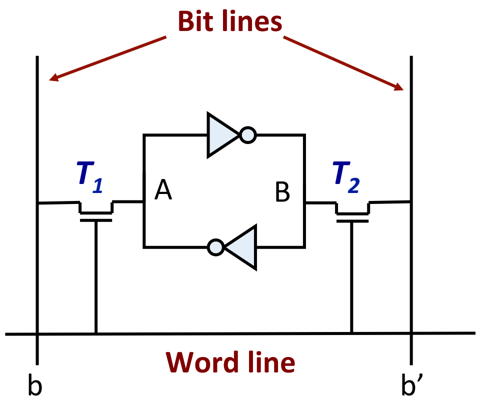
A 1-bit SRAM Cell

- Two inverters are cross connected to form a *latch*.
- The latch is connected to two bit lines with transistors *T1* and *T2*.
- Transistors behave like switches that can be opened (OFF) or closed (ON) under the control of the word line.
- To retain the state of the latch, the word line can be grounded which makes the transistors off.



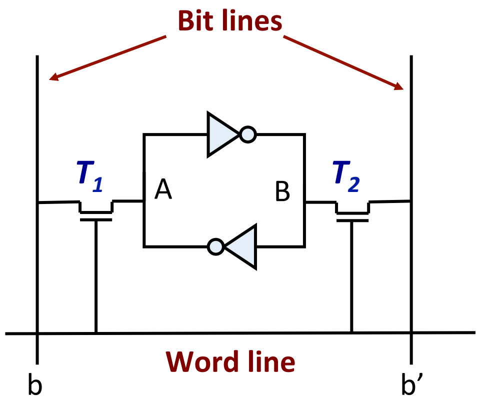
(a) READ Operation in SRAM

- To read the content of the cell, the word line is activated (= *1*) to make the transistors *T1* and *T2* on.
- The value stored in latch is available on bit line *b* and its complement on *b’*.
- Sense/write circuits connected to the bit lines monitor the states of *b* and *b’*.



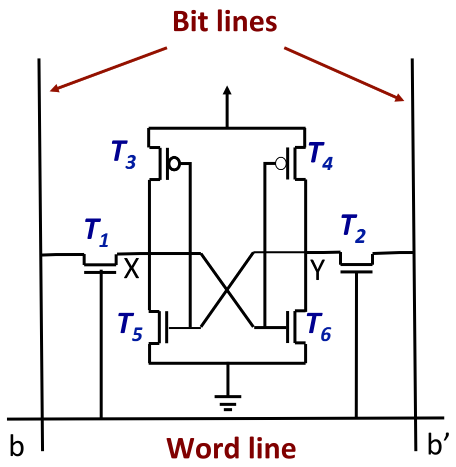
(b) WRITE Operation in SRAM

- **To write 1**: The bit line *b* is set with *1* and bit line *b'* is set with *0*. Then the word line is activated and the data is written to the latch.
- **To write 0**: The bit line *b* is set with *0* and bit line *b'* is set with *1*. Then the word line is activated and the data is written to the latch.
- The required signals (either *1* or *0*) are generated by the sense/write circuit.



6-Transistor Static Memory cell

- 1-bit SRAM cell with 6-transistors are used in modern-day SRAM implementations.
- Transistors (*T3* & *T5*) and (*T4* & *T6*) form the CMOS inverters in the latch.
- The data can be read or written in the same way as explained.

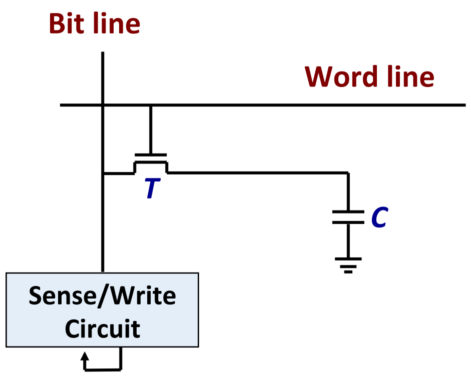


Features of SRAM

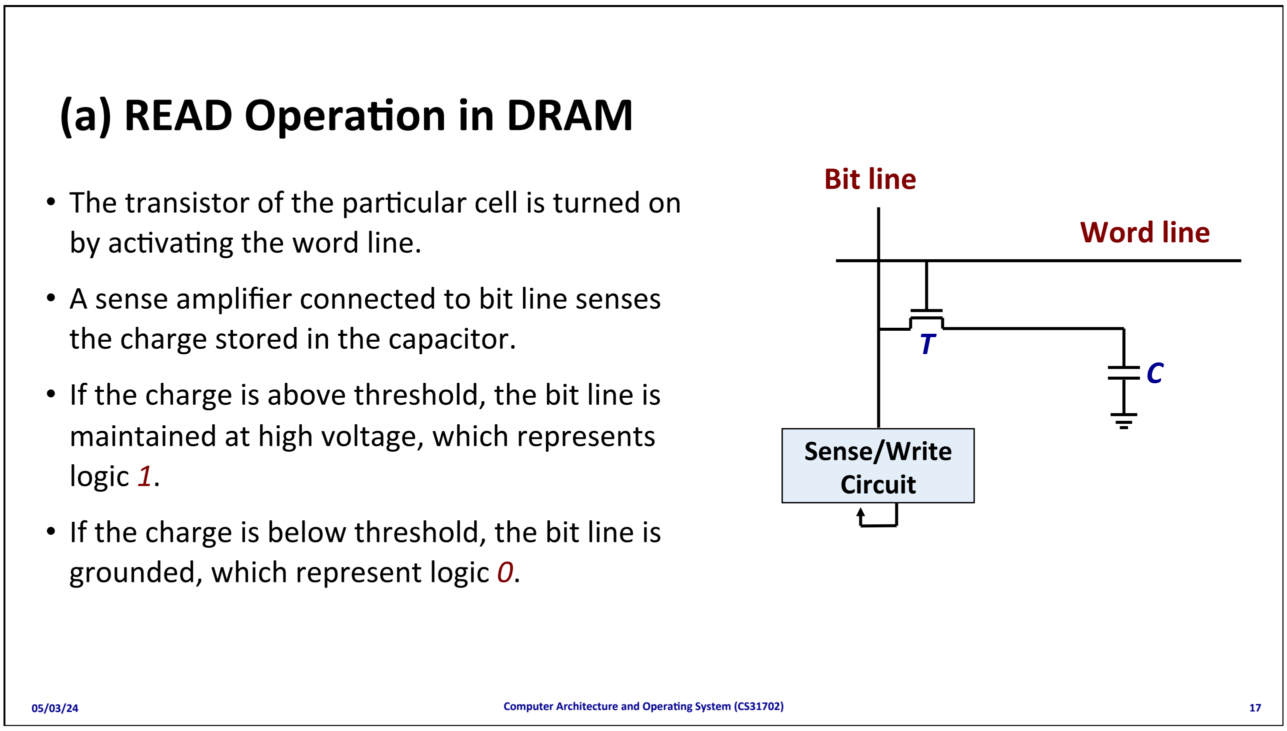
- Moderate / High power consumption.
  - Current flows in the cells only when the cell is accessed.
  - Because of latch operation, power consumption is higher than DRAM.
- Volatile :: continuous power supply is required.
- Access time is very fast; fast memories (cache) are built using SRAM.
- High cost.
  - 6 transistors per cell.

Dynamic Random Access Memory (DRAM)

- Dynamic RAM do not retain its state even if power supply is on.
  - Data stored in the form of charge stored on a capacitor.
- Requires *periodic refresh*.
  - The charge stored cannot be retained over long time (due to leakage).
- Less expensive that SRAM.
  - Requires less hardware (one transistor and one capacitor per cell).
- Address lines are multiplexed.

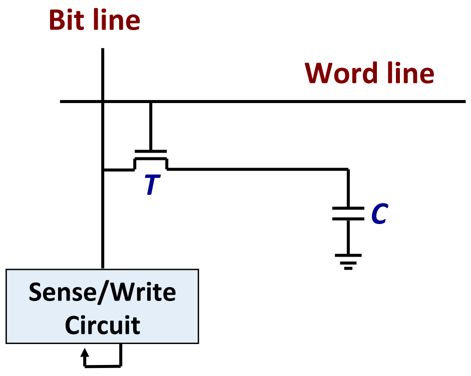


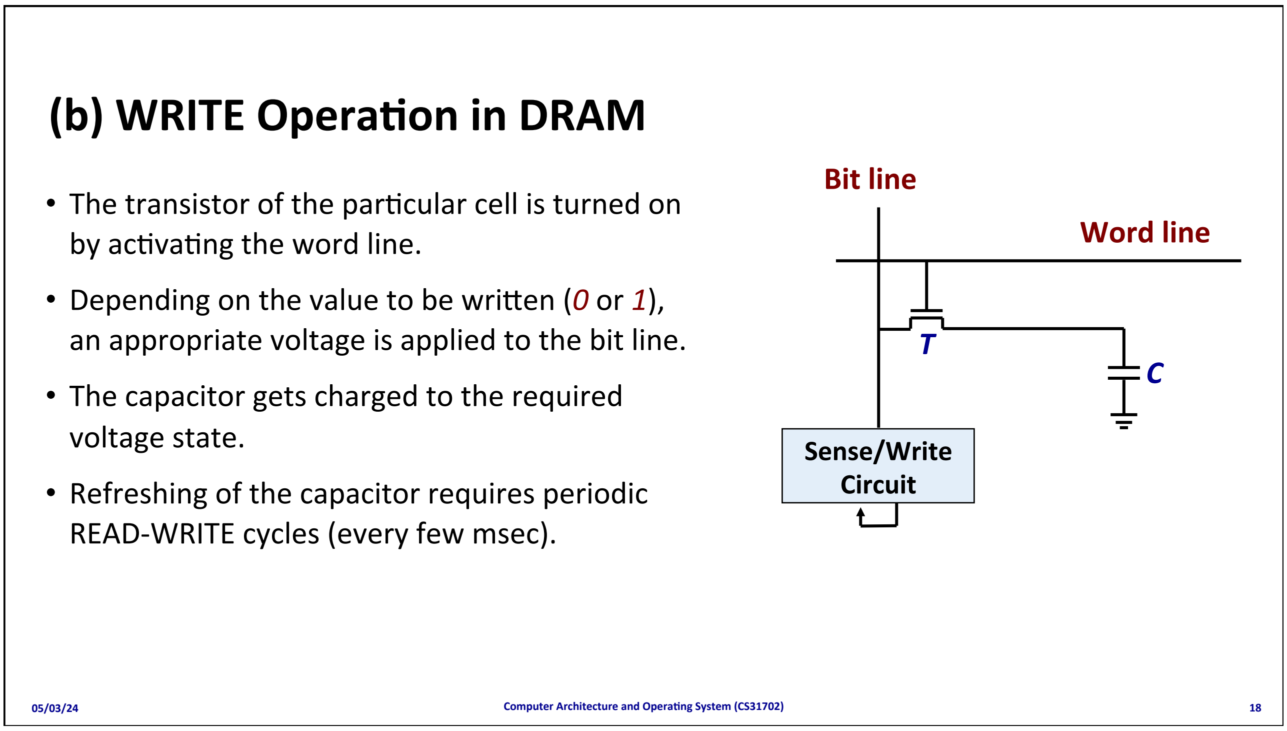
1-transistor DRAM Cell



(a) READ Operation in DRAM

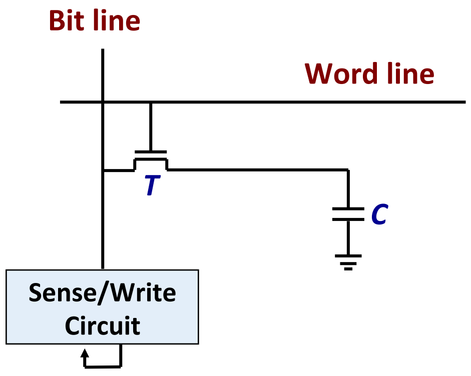
- The transistor of the particular cell is turned on by activating the word line.
- A sense amplifier connected to bit line senses the charge stored in the capacitor.
- If the charge is above threshold, the bit line is maintained at high voltage, which represents logic *1*.
- If the charge is below threshold, the bit line is grounded, which represent logic *0*.





(b) WRITE Operation in DRAM

- The transistor of the particular cell is turned on by activating the word line.
- Depending on the value to be written (*0* or *1*), an appropriate voltage is applied to the bit line.
- The capacitor gets charged to the required voltage state.
- Refreshing of the capacitor requires periodic READ-WRITE cycles (every few msec).

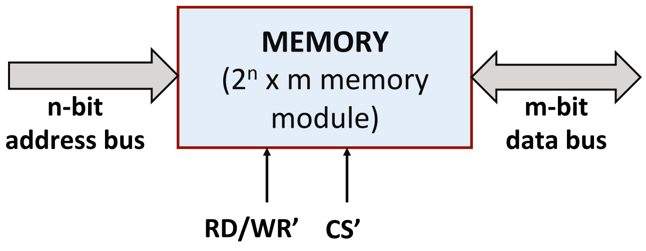


Memory Interfacing and Addressing

Memory Interfacing

- Basic problem:
  - Interfacing one of more memory modules to the processor.
  - We assume a single level memory at present (i.e. no cache memory).
- Questions to be answered:
  a) How the processor address and data lines are connected to memory modules?
  b) How are the addresses decoded?
  c) How are the memory addresses distributed among the memory modules?
  d) How to speed up data transfer rate between processor and memory?

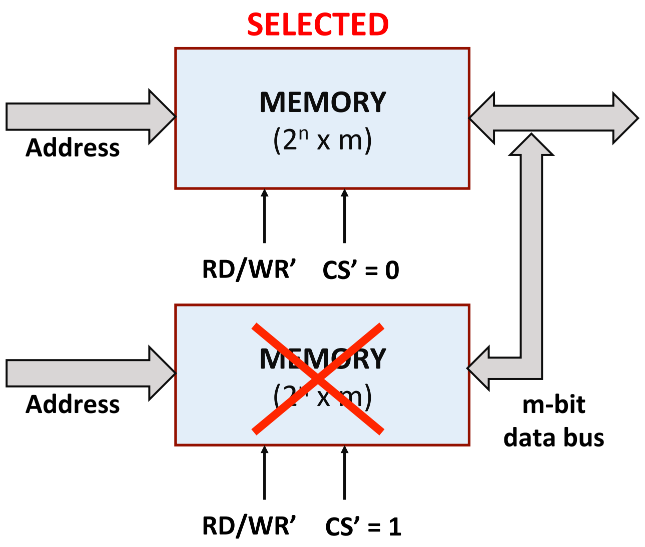
- Typical interface of a memory module.
- Real chip may contain more signal lines (e.g. DRAM).



A Note About the Memory Interface Signals

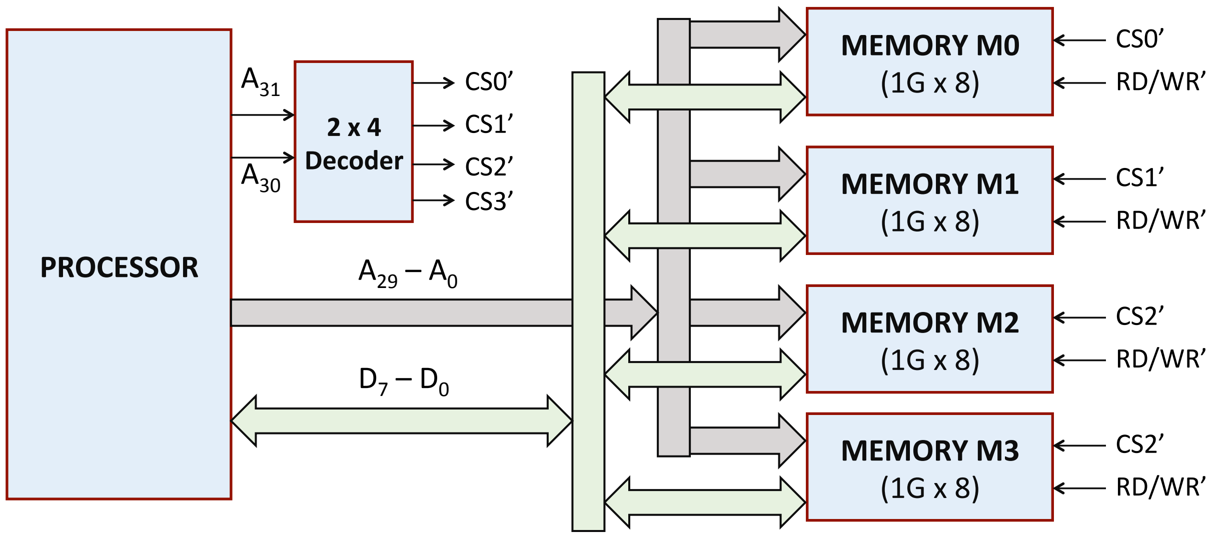
- The data signals of a memory module (RAM) are typically bidirectional.
- For memory READ operation:
  - Address of memory location is applied to **address** lines.
  - **RD/WR'** control signal is set to 1, and **CS'** is set to 0.
  - Data is read out through the **data** lines after memory access time delay.
- For memory WRITE operation:
  - Address of memory location is applied to **address** lines, and the data to be written to **data** lines.
  - **RD/WR'** control signal is set to 0, and **CS'** is set to 0.

- Why is **CS’** signal required?
  - To handle multiple memory modules interfacing.
  - We typically select only one out of several memory modules at a time.
- What happens when **CS’ = 1**?
  - When a memory module is not selected, the data lines are set to the high impedance state (i.e. electrically disconnected).
  - An example scenario is shown.



An Example Memory Interfacing Problem

- Consider a processor with a 32-bit address.
  - Maximum memory that can be connected is 2^{32} = 4 Gbytes.
  - Assume that the processor data lines are 8 bits.
- Assume that memory chips (RAM) are available with size 1 Gbyte.
  - 30 address lines and 8 data lines.
  - Low-order 30 address lines (A_{29}-A_0) are connected to the memory modules.
- We want to interface 4 such chips to the processor.
  - 4 x 1 GB = 4 GB



- High order address lines (A_{31} and A_{30}) select one of the memory modules.
- When is **M0** selected?
  - Address is: **0 0** x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
  - Range of addresses is: 0x00000000 to 0x3FFFFFFF
- When is **M1** selected?
  - Address is: **0 1** x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
  - Range of addresses is: 0x40000000 to 0x7FFFFFFF
- When is **M2** selected?
  - Address is: **1 0** x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
  - Range of addresses is: 0x80000000 to 0xBFFFFFFF
- When is **M3** selected?
  - Address is: **1 1** x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
  - Range of addresses is: 0xC0000000 to 0xFFFFFFFF

- An observation:
  - Consecutive block of bytes are mapped to the same memory module.
  - To access 32 bits (4 bytes) of data in parallel, we require four sequential memory accesses.
  - We shall look at an alternate memory organization later where we can do this using a single memory access.
    - Called *memory interleaving*.

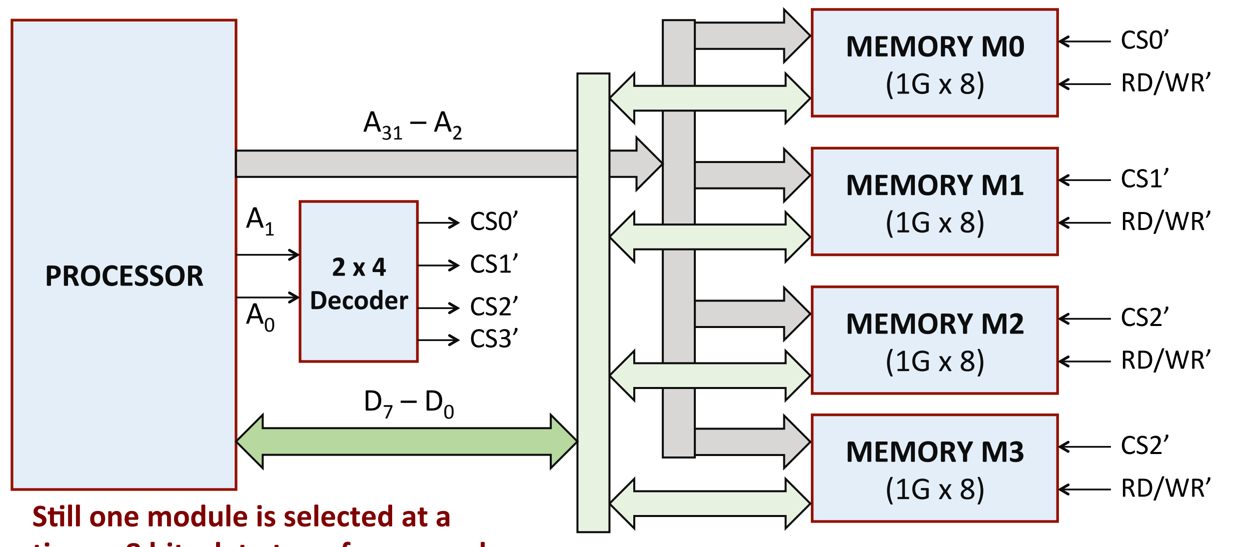
Practice Problems

1. Design a 64K x 8 memory system using 16K x 8 memory modules as basic building blocks. Hence show how the addresses are mapped to the different modules.
2. Design a 16M x 8 memory system using six 2M x 8 memory modules and four 1M x 8 memory modules as basic building blocks. Hence show how the addresses are mapped to the different modules.

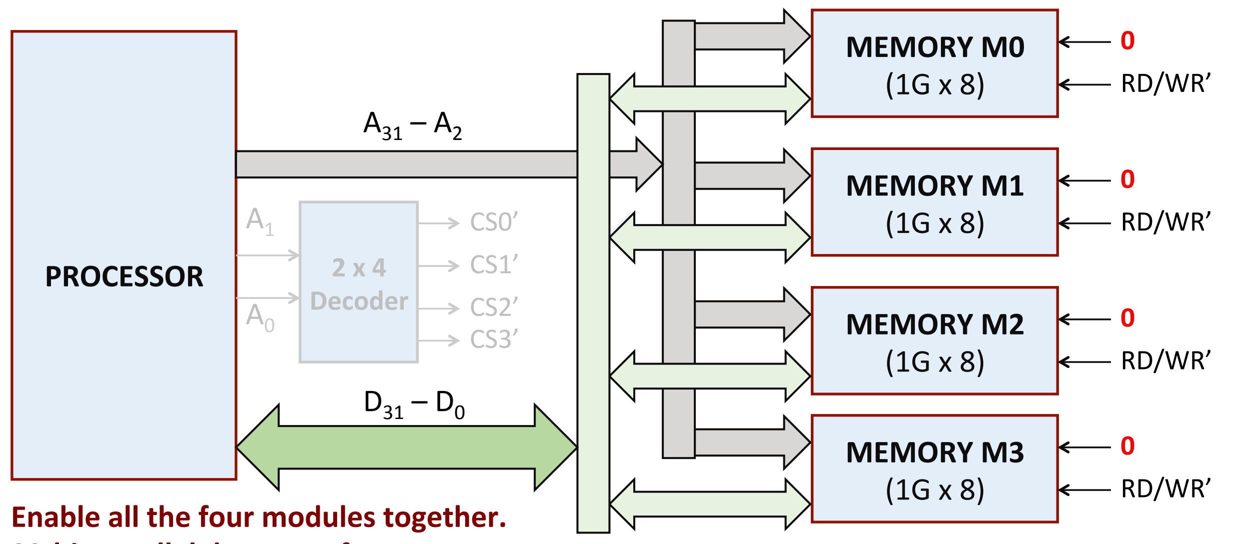
Memory Interleaving

- We make small changes in the organization so that 32-bits of data can be fetched in a single memory access cycle.
  - Consecutive bytes are mapped to different memory modules.
- The main changes:
  - High order 30 address lines (A_{31}-A_2) are connected to memory modules.
  - Low order two address lines (A_1 and A_0) are used to select one of the modules.

- How are the addresses mapped to memory modules?
  - Module M0: 0, 4, 8, 12, 16, 20, 24, …
  - Module M1: 1, 5, 9, 13, 17, 21, 25, …
  - Module M2: 2, 6, 10, 14, 18, 22, 26, …
  - Module M3: 3, 7, 11, 15, 19, 23, 27, …
- Memory addresses are *interleaved* across memory modules.
- What we can gain from this mapping?
  - Consecutive addresses are mapped to consecutive modules.
  - Possible to access four consecutive words in the same cycle, if all four modules are enabled simultaneously.



Still one module is selected at a time :: 8 bits data transfer per cycle.

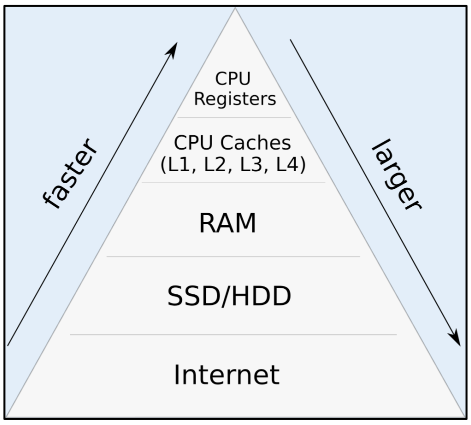


Enable all the four modules together. 32-bit parallel data transfer.

Memory Hierarchy Design

Memory Hierarchy

- The memory system is organized in several levels, using progressively faster technologies as we move towards the processor.
  - The entire addressable memory space is available in the largest (but slowest) memory.
  - Objective is to provide faster access times on an average at a low cost.



A Comparison

Level	Typical Access Time	Typical Capacity	Other Features
Register	300-500 ps	500-1000 B	On-chip
Level-1 cache	1-2 ns	16-64 KB	On-chip
Level-2 cache	5-20 ns	256 KB – 2 MB	On-chip
Level-3 cache	20-50 ns	1-32 MB	On or off chip
Main memory	50-100 ns	1-16 GB	Off-chip
Magnetic disk	5-50 ms	100 GB – 16 TB	

Locality of Reference

- Programs tend to reuse data and instructions they have used recently.
  - **Rule of thumb**: 90% of the total execution time of a program is spent in only 10% of the code (also called *90/10 rule*).
  - **Reason**: nested loops in a program, few procedures calling each other repeatedly, arrays of data items being accessed sequentially, etc.
- Basic idea to exploit this rule:
  - Based on a program's recent past, we can predict with a reasonable accuracy what instructions and data will be accessed in the near future.

- The 90/10 rule has two dimensions:
  - **a) Temporal Locality** (locality in time)
    - If an item is referenced in memory, it will tend to be referenced again soon.
  - **b) Spatial locality** (locality in space)
    - If an item is referenced in memory, nearby items will tend to be referenced soon.

(a) Temporal Locality

- Recently executed instructions are likely to be executed again very soon.
- Example: computing factorial of a number.

```
fact = 1;
for  k = 1 to N
  fact = fact * k;
```

```
       MOV   R1,#1      // R1 = 1
       LOAD  R2,N       // R2 = N
       MOV   R3,1       // R3 = 1
Loop:  MUL   R1,R1,R3   // R1 = R1 * R3
       ADD   R3,R3,#1   // R3 = R3 + 1
       CMP   R3,R2      // R3 - R2
       BGT   Loop       // Branch if R3 > R2
```

- The four instructions in the loop are executed more frequently than the others.

(b) Spatial Locality

- Instructions residing close to a recently executing instruction are likely to be executed soon.
- Example: accessing elements of an array.

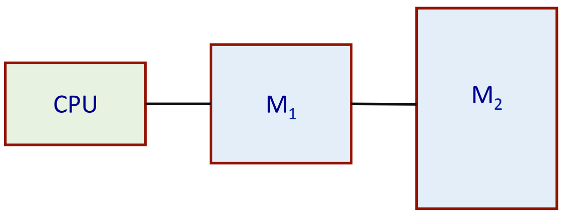
```
sum = 0;
for  k = 1 to N
  sum = sum + A[k];
```

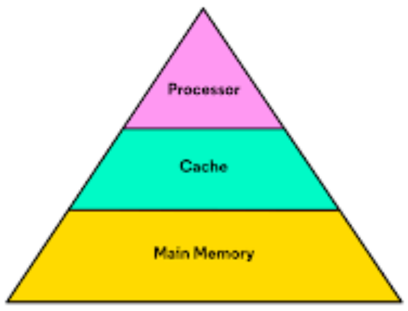
```
       MOV    R1,#0       // R1 = 0
       LOAD   R2,N        // R2 = N
       MOV    R3,1        // R3 = 1
       MOV    R5,A        // R5 = A (addr)
Loop:  LOAD   R8,0(R5)    // R8 = Mem[R5+0]
       ADD    R1,R8       // R1 = R1 + R8
       ADD    R3,#1       // R3 = R3 + 1
       CMP    R3,R2       // R3 - R2
       BGT    Loop        // Branch if R3 > R2
```

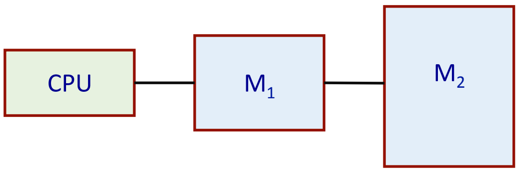
- Performance can be improved by copying the array into cache memory.

Performance of Memory Hierarchy

- We first consider a 2-level hierarchy consisting of two levels of memory, say, M_1 and M_2.
  - M_1 can be *cache memory*, and M_2 *main memory*.





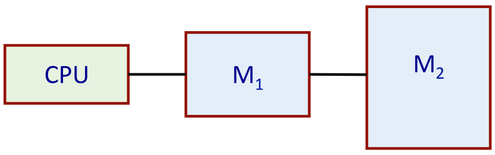


a) Cost:

- Let c_i denote the cost per bit of memory M_i, and S_i denote the storage capacity of M_i in bits.
- The *average cost per bit* of the memory hierarchy is given by:

$$c = \frac{c_1S_1 + c_2S_2}{S_1 + S_2}$$

- In order to have $c \approx c_2$, we must ensure that $S_1 << S_2$.

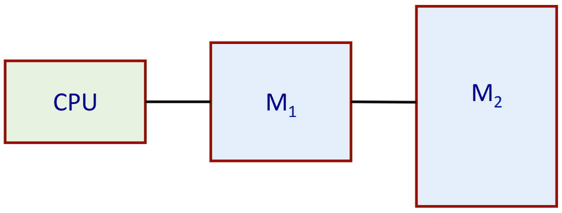


b) Hit Ratio / Hit Rate:

- The hit ratio H is defined as the probability that a logical address generated by the CPU refers to information stored in M_1.
- We can determine H *experimentally* as follows:
  - A set of representative programs is executed or simulated.
  - The number of references to M_1 and M_2, denoted by N_1 and N_2 respectively, are recorded.

$$H = \frac{N_1}{N_1 + N_2}$$

- The quantity ($1 - H$) is called the *miss ratio*.

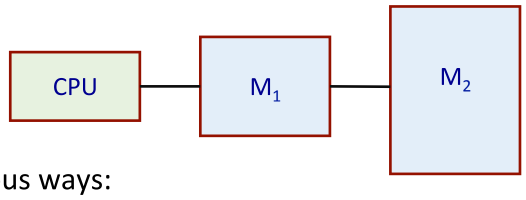


c) Access Time:

- Let t_{A1} and t_{A2} denote the access times of M_1 and M_2 respectively, relative to the CPU.
- The average time required by the CPU to access a word in memory can be expressed as:

$$t_A = H.t_{A1} + (1 - H).t_{MISS}$$

where t_{MISS} denotes the time required to handle the miss, called *miss penalty*.



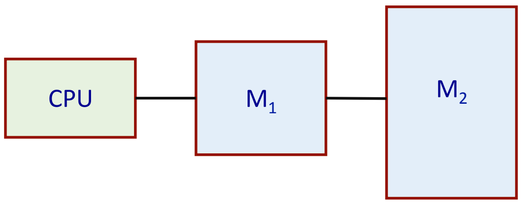
- The miss penalty t_{MISS} can be estimated in various ways:
  a) The simplest approach is to set $T_{MISS} = t_{A2}$, that is, when there is a miss the data is accessed directly from M_2.
  b) A request for a word not in M_1 typically causes a block containing the requested word to be transferred from M_2 to M_1. After completion of the transfer, the word is accessed in M_1.

     If t_B denotes the *block transfer time*, we can write

     $t_{MISS} = t_B + t_{A1}$ [since $t_B >> t_{A1}$, $t_{MISS} \approx t_B$]

     Thus, $t_A = H.t_{A1} + (1 - H).(t_B + t_{A1})$
  c) If t_{HIT} denotes the time required to check whether there is a hit, we can write

     $t_{MISS} = t_{HIT} + t_B + t_{A1}$



d) Speedup:

- The speedup gained by using the memory hierarchy is defined as $S = t_{A2} / t_A$.
- We can write:

$$S = \frac{t_{A2}}{H.t_{A1} + (1 - H).t_{A2}}$$

Example 1

The access times of cache memory and main memory are 20 nanoseconds and 100 nanoseconds respectively. If the requested word is not found in cache, the miss penalty is calculated as reading the block from main memory (assume same as the access time of main memory) and accessing it from cache. If the cache hit ratio is 0.98, what will be the average access time of memory?

Here, t_{A1} = 20 nsec, t_{A2} = 100 nsec, t_{MISS} = 100 + 20 = 120 nsec

Also, H = 0.98

Thus, average access time

$$t_A = H.t_{A1} + (I - H)\ t_{MISS}$$
$$= 0.98 \times 20 + 0.02 \times 120 = 22 \text{ nsec}$$

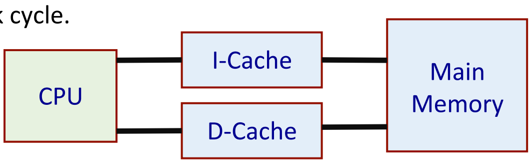
Example 2

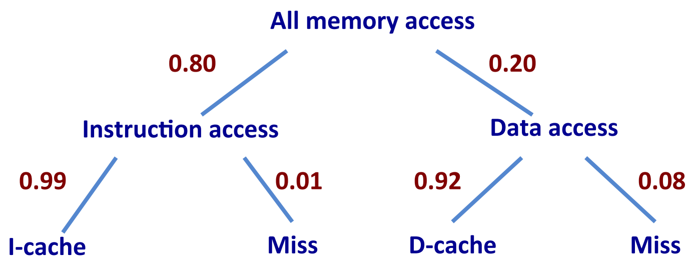
Consider a 2-level memory hierarchy with separate instruction and data caches in level 1, and main memory in level 2.

The following parameters are given:

- The clock cycle time is 2 ns.
- The miss penalty is 15 clock cycles (for both read and write).
- 1 % of instructions are not found in I-cache.
- 8 % of data references are not found in D-cache.
- 20 % of the total memory accesses are for data.
- Cache access time (including hit detection) is 1 clock cycle.

Determine the overall average access time.





t_{MISS} = 1 + 15 = 16 cycles

Average number of cycles per access:

0.80 x (0.99 x 1 + 0.01 x 16) + 0.20 x (0.92 x 1 + 0.08 x 16)

= 0.92 + 0.44 = 1.36

Thus, average access time t_A = 1.36 x 2 ns = 2.72 ns

Common Memory Hierarchies

- In a typical computer system, the memory system is managed as two different hierarchies.

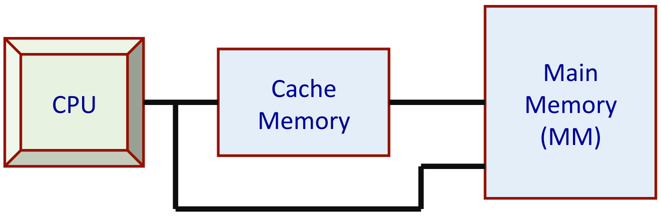
  a) The *Cache / Main Memory hierarchy*, which consists of 2 to 4 levels and is managed by hardware.
     - Main objective: provide fast average memory access.

  b) The *Main Memory / Secondary Memory hierarchy*, which consists of 2 levels and is managed by software (operating system).
     - Main objective: provide large memory space for users (virtual memory).

Cache Memory

Introduction

- Let us consider a single-level cache, and that part of the memory hierarchy consisting of *cache memory* and *main memory*.



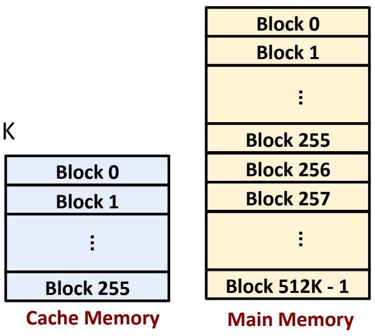
- Cache memory is logically divided into *blocks* or *lines*, where every block (line) typically contains 8 to 256 bytes.
- When the CPU wants to access a word in memory, a special hardware first checks whether it is present in cache memory.
  - If so (called *cache hit*), the word is directly accessed from the cache memory.
  - If not, the block containing the requested word is brought from MM to cache.
  - For writes, sometimes the CPU can also directly write to MM.
- Objective is to keep the commonly used blocks in the cache memory.
  - Will result in significantly improved performance due to the property of *locality of reference*.

Q1. Where can a block be placed in the cache?

- This is determined by some *mapping algorithms*.
  - Specifies which MM blocks can reside in which cache memory blocks.
  - At any given time, only a small subset of the MM blocks can be in the cache.
- Three common block mapping techniques are used:
  - **a) Direct Mapping**
  - **b) Associative Mapping**
  - **c) (N-way) Set Associative Mapping**

An Example Scenario: A 2-level memory hierarchy

- Consider a 2-level cache memory / main memory hierarchy.
  - The cache memory consists of 256 blocks (lines) of 32 words each.
    - Total cache size is 8192 (8K) words.
  - Main memory is accessed using a 24-bit address.
    - Main memory is word addressable – word size = 32 bits.
    - Total size of the main memory is 2^{24} = 16 M words.
    - Number of 32-word blocks in main memory = 16 M / 32 = 512K
    - Number of blocks in cache memory = 256.



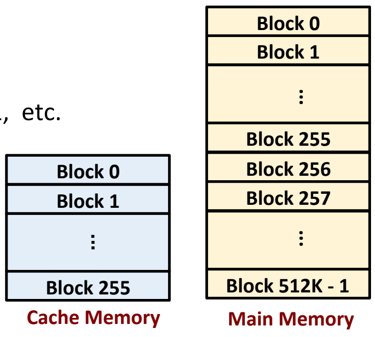
(a) Direct Mapping

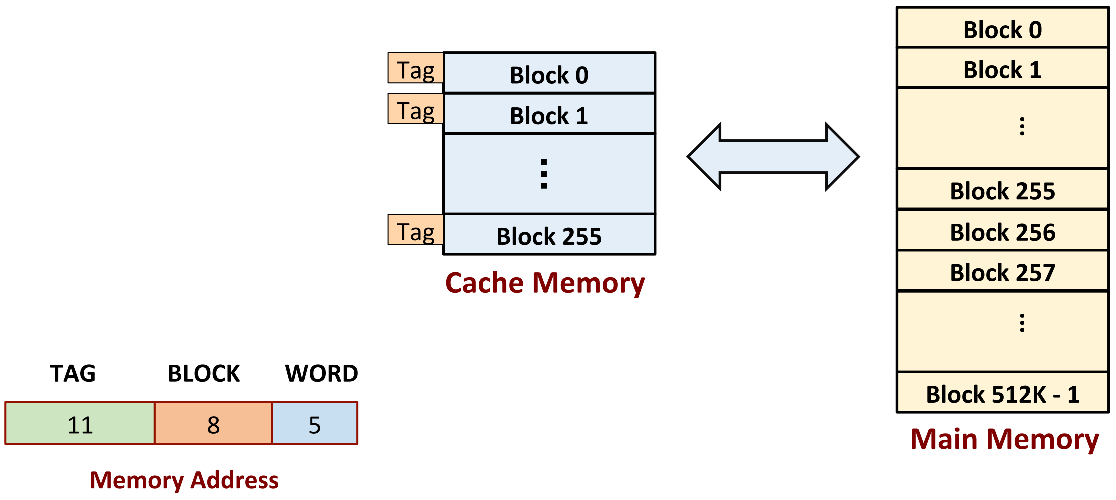
- Each main memory block can be placed in only one block in the cache.
- The mapping function is:

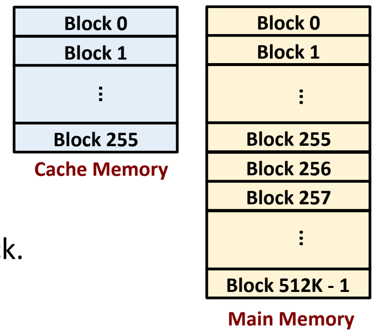
  Cache Block = (Main Memory Block) % (Number of cache blocks)
- For the example,

  Cache Block = (Main Memory Block) % 256
  - Some example mappings:

    0 → 0, 1 → 1, 255 → 255, 256 → 0, 257 → 1, 512 → 0, 512 → 1, etc.



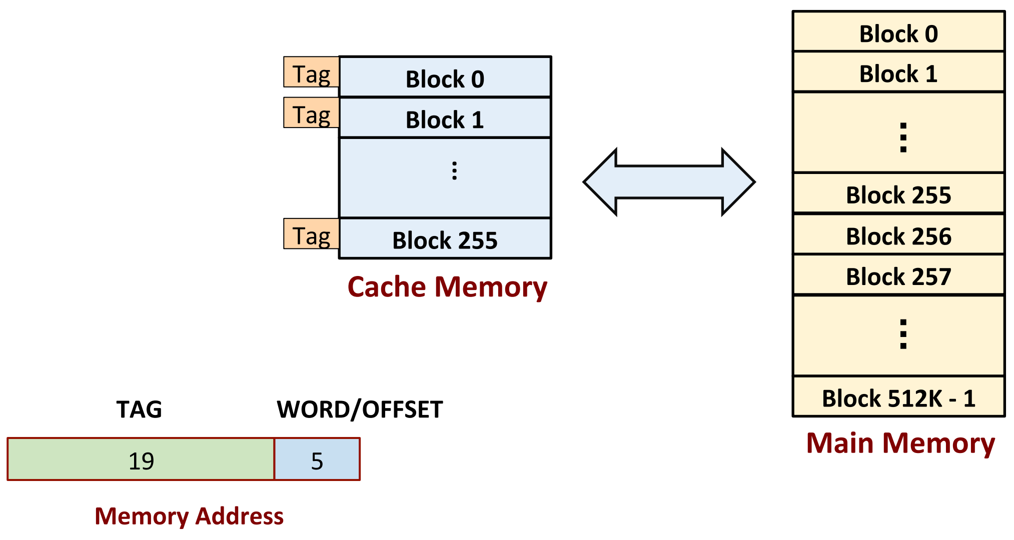




- Block replacement algorithm is trivial, as there is no choice.
  - No special hardware is required.
- More than one MM block is mapped onto the same cache block.
  - New block will replace the old block.
  - May lead to poor performance if both the blocks are frequently used.
- The MM address is divided into three fields: *TAG*, *BLOCK* and *WORD*.
  - When a new block is loaded into the cache, the 8-bit *BLOCK* field determines the cache block where it is to be stored.
  - The high-order 11 bits are stored in a *TAG* register associated with the cache block.
  - When accessing a memory word, the corresponding *TAG* field is compared.
    - Match implies *HIT*; mismatch implies *MISS*.

(b) Associative Mapping

- Here, a MM block can potentially reside in *any cache block position*.
- The memory address is divided into two fields: *TAG* and *WORD*.
  - When a block is loaded into the cache from MM, the higher order 19 bits of the address are stored into the *TAG* register corresponding to the cache block.
  - When accessing memory, the 19-bit *TAG* field of the address is compared with all the TAG registers corresponding to all the cache blocks.
- Requires associative memory for storing the *TAG* values.
  - Associative memory supports parallel search.
  - High cost / lack of scalability.
- Because of complete freedom in block positioning, a wide range of replacement algorithms is possible.

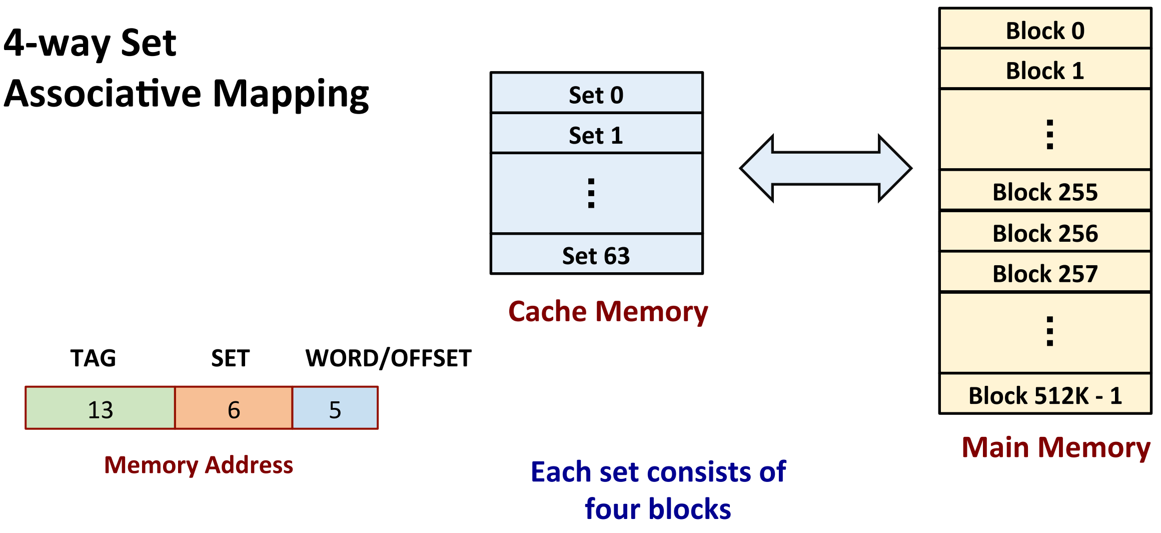


(c) N-way Set Associative Mapping

- A group of **N** consecutive blocks in the cache is called a *set*.
- This algorithm is a balance of direct mapping and associative mapping.
  - Like direct mapping, a MM block is mapped to a set.

    Set Number = (MM Block Number) % (Number of Sets in Cache)
  - The block can be placed anywhere within the set (there are **N** choices).
- The value of **N** is a design parameter:
  - **N = 1** :: same as *direct mapping*.
  - **N** = number of cache blocks :: same as *associative mapping*.
  - Typical values of **N** used in practice are: 2, 4 or 8.

4-way Set Associative Mapping



- Illustration for N = 4:
  - Number of sets in cache memory = 64.
  - Memory blocks are mapped to a set using modulo-64 operation.
  - Example: MM blocks 0, 64, 128, etc. all map to set 0, where they can occupy any of the four available positions.
- MM address is divided into three fields: *TAG*, *SET* and *WORD*.
  - The *TAG* field of the address must be associatively compared to the *TAG* fields of the 4 blocks of the selected set.
  - This instead of requiring a single large associative memory, we need a number of very small associative memories only one of which will be used at a time.
  - Each associative memory has four rows.

Example

A 64 Kbyte four-way set-associative cache is byte-addressable and contains 32-byte blocks. Memory addresses are 32 bits wide.

a) How wide are the tags in this cache?

b) Which main memory addresses are mapped to set number 5?

Solution:

The number of sets in the cache = 64KB / (4 x 32B) = 512.

a) Address (32 b) = 5 b byte offset + 9 b set index + 18 b tag

b) Addresses that have their 9-bit set index equal to 5. These are of the general form $2^{14}a + 2^{5}\times 5 + b$; e.g., 160-191, 16 554-16 575, . . .

32-bit address

TAG	SET	WORD
18 bits	**9 bits**	**5 bits**

Tag width = 32 – 5 – 9 = 18

Set size = 4 × 32 B = 128 B

Number of sets = $2^{16}/2^{7} = 2^{9}$

Block size = 32 B = 2^{5} B

Q2. How is a block found if present in cache?

- Caches include a *TAG* associated with each cache block.
  - The TAG of every cache block where the block being requested may be present needs to be compared with the TAG field of the MM address.
  - All the possible tags are compared in parallel, as speed is important.
- Mapping Algorithms?
  - Direct mapping requires a *single comparison*.
  - Associative mapping requires a *full associative search* over all the TAGs corresponding to all cache blocks.
  - Set associative mapping requires a *limited associated search* over the TAGs of only the selected set.

- Use of *valid bit*:
  - There must be a way to know whether a cache block contains valid or garbage information.
  - A valid bit can be added to the *TAG*, which indicates whether the block contains valid data.
  - If the valid bit is not set, there is no need to match the corresponding *TAG*.

Q3. Which block should be replaced on a cache miss?

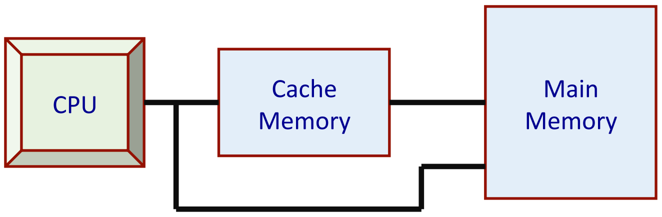
- With fully associative or set associative mapping, there can be several blocks to choose from for replacement when a miss occurs.
- Two primary strategies are used:
  a) *Random*: The candidate block is selected randomly for replacement. This simple strategy tends to spread allocation uniformly.
  b) *Least Recently Used* (LRU): The block replaced is the one that has not been used for the longest period of time.
     - Makes use of a corollary of temporal locality:

"If recently used blocks are likely to be used again, then the best candidate for replacement is the least recently used block"

- To implement the LRU algorithm, the cache controller must track the *LRU block* as the computation proceeds.
- Incurs significant hardware overhead.
  - Need to keep track of the time when each cache block was last accessed.
  - Not being discussed here.

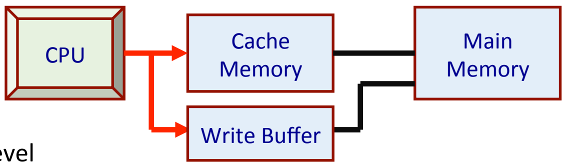
Q4. What happens on a write?

- Cache designs can be classified based on the write and memory update strategy being used.
  1. *Write Through / Store Through*
  2. *Write Back / Copy Back*



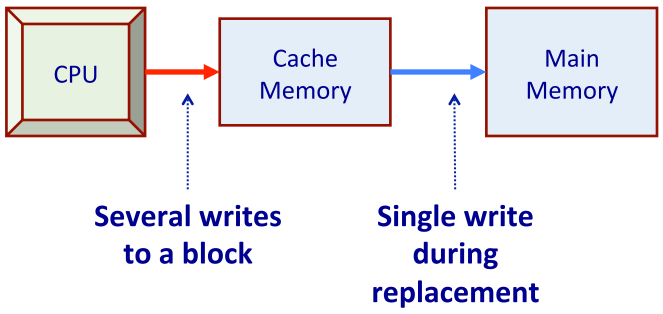
(a) Write Through Strategy

- Information is written to both the cache block and the main memory block.
- Features:
  - Easier to implement.
  - Read misses do not result in writes to the lower level (i.e. MM).
  - The lower level (i.e. MM) has the most updated version of the data – important for I/O operations and multiprocessor systems.
  - A write buffer is often used to reduce CPU write stall time while data is written to main memory.



(b) Write Back Strategy

- Information is written only to the cache block.
- A modified cache block is written to MM only when it is replaced.
- Features:
  - Writes occur at the speed of cache memory.
  - Multiple writes to a cache block requires only one write to MM.
  - Uses less memory bandwidth, makes it attractive to multiprocessors.
- Write-back cache blocks can be *clean* or *dirty*.
  - A status bit called *dirty bit* or *modified bit* is associated with each cache block, which indicates whether the block was modified in the cache (0: clean, 1: dirty).
  - If the status is clean, the block is not written back to MM while being replaced.


CPU
Cache Memory
Main Memory
Several writes to a block
Single write during replacement