4		**Memory Organization:**		
	4.1	Introduction to Memory and Memory parameters. Classifications of primary and secondary memories. Types of RAM and ROM, Allocation policies, Memory hierarchy and characteristics.	3	3
	4.2	Cache memory: Concept, architecture (L1, L2, L3), mapping techniques. Cache Coherency, Interleaved and Associative memory.	2,3	3
	4.3	Virtual Memory: Concept, Segmentation and Paging, Page replacement policies. LRU, FIFO	4,5	4

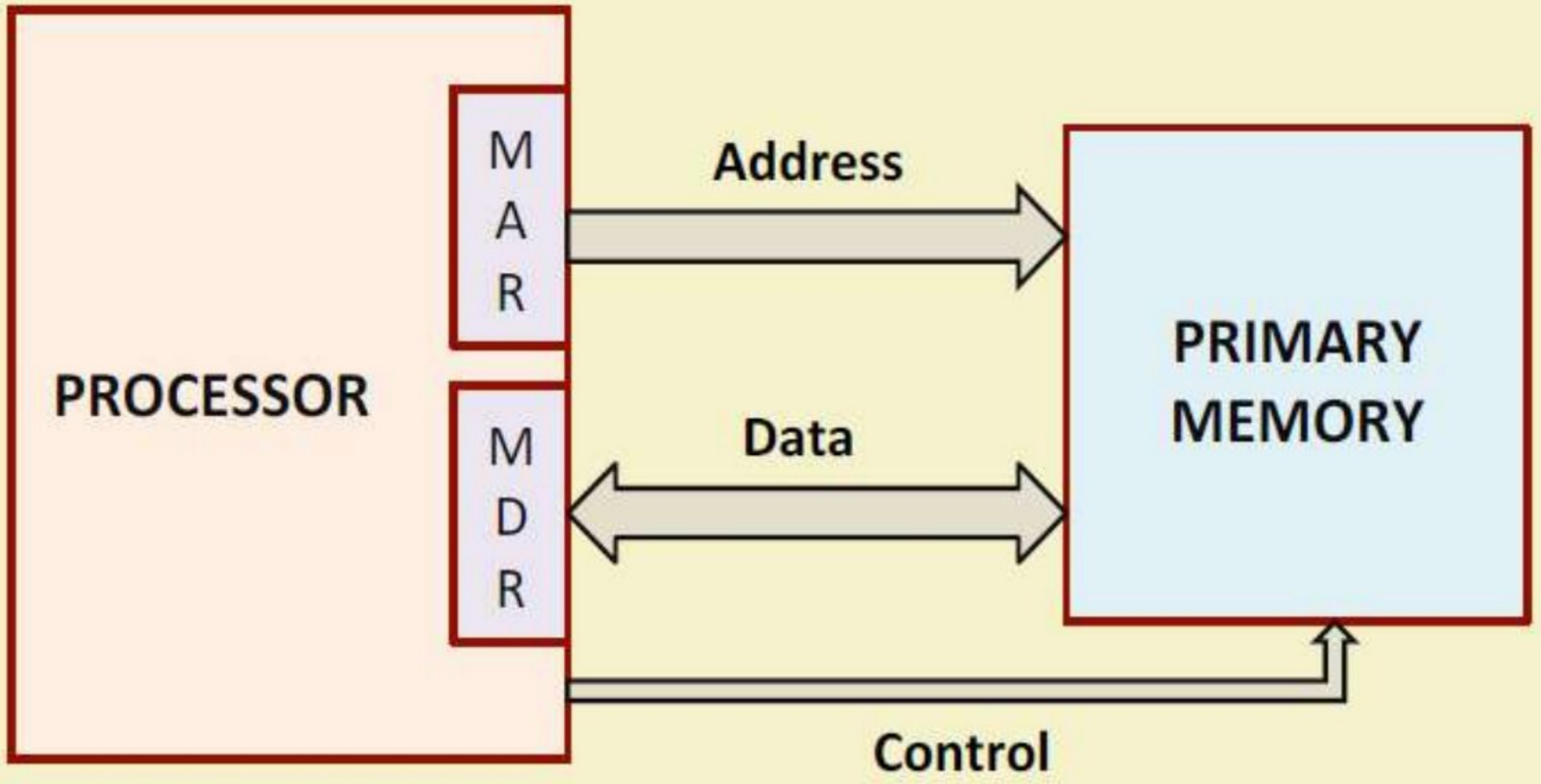
Source: NPTEL Lecture Series

Introduction

- Memory is one of the most important functional units of a computer.
  - Used to store both instructions and data.
  - Stores as bits (0's and 1's), usually organized in terms of bytes.
- How are the data stored in memory accessed?
  - Every memory location has a unique address.
  - A memory is said to be byte addressable if every byte of data has a unique address.
  - Some memory systems are word addressable also (every addressed locations consists of multiple bytes, say, 32 bits or 4 bytes).

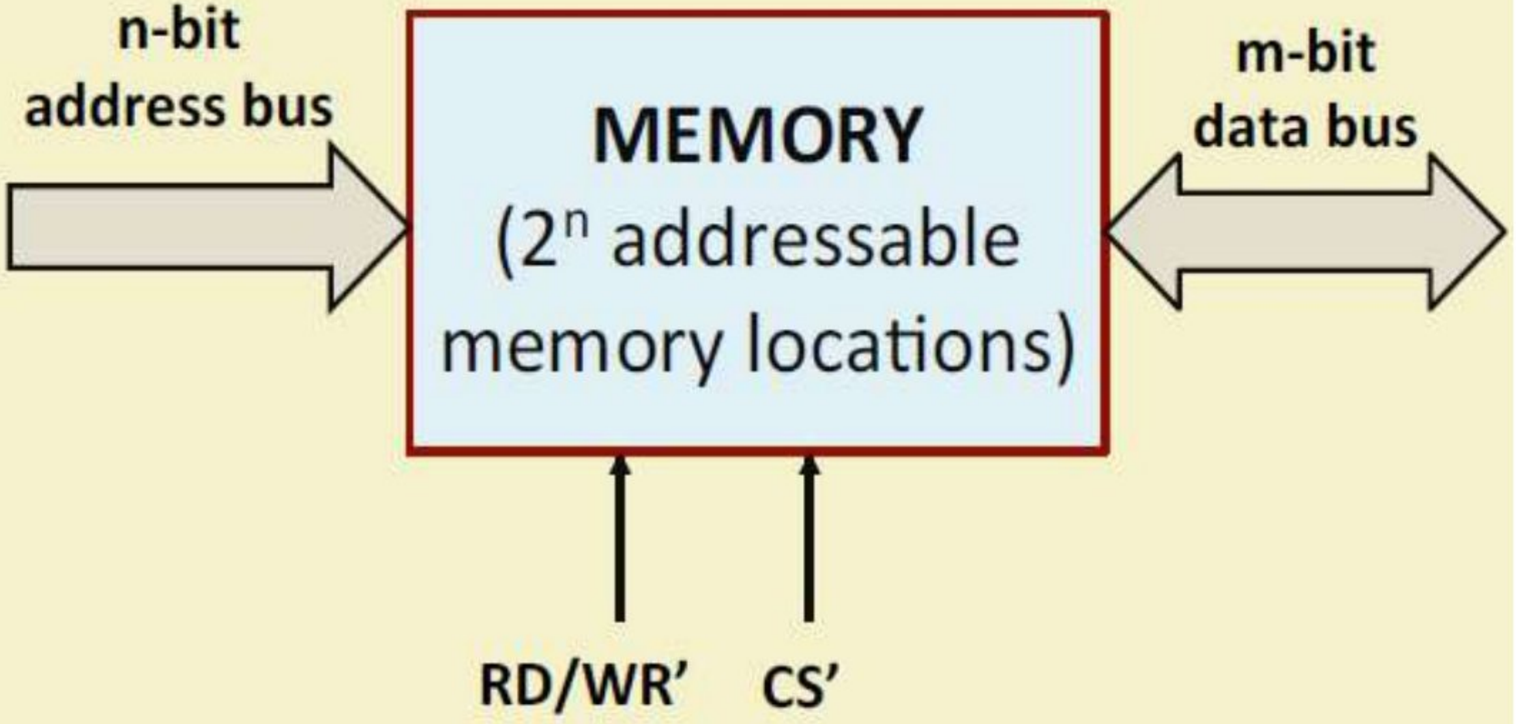
Connection between Processor and Memory

- 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

Classification of Memory Systems

a) Volatile versus Non-volatile:

- A *volatile* memory system is one where the stored data is lost when the power is switched off.
  - Examples: CMOS static memory, CMOS dynamic memory.
  - Dynamic memory in addition requires periodic refreshing.
- A *non-volatile* memory system is one where the stored data is retained even when the power is switched off.
  - Examples: Read-only memory, Magnetic disk, CDROM/DVD, Flash memory, Resistive memory.

b) Random-access versus Direct/Sequential access:

- A memory is said to be *random-access* when the read/write time is independent of the memory location being accessed.
  - Examples: CMOS memory (RAM and ROM).
- A memory is said to be *sequential access* when the stored data can only be accessed sequentially in a particular order.
  - Examples: Magnetic tape, Punched paper tape.
- A memory is said to be *direct or semi-random access* when part of the access is sequential and part is random.
  - Example: Magnetic disk.
  - We can directly go to a track after which access will be sequential.

c) Read-only versus Random-access:

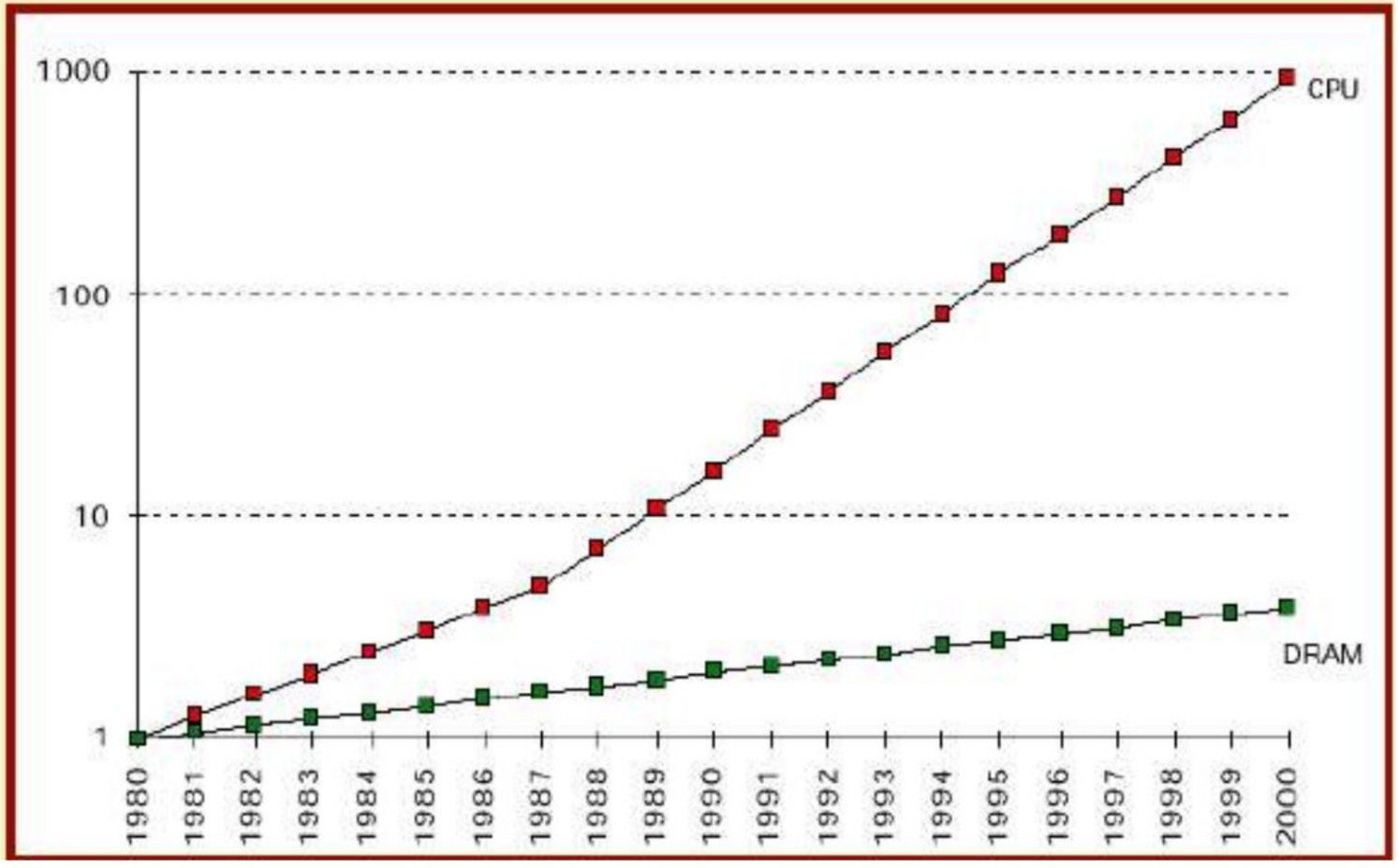
- *Read-only Memory* (ROM) is one where data once stored in permanent or semi-permanent.
  - Data written (programmed) during manufacture or in the laboratory.
  - Examples: ROM, PROM, EPROM, EEPROM.
- *Random Access Memory* (RAM) is one where data access time is the same independent of the location (address).
  - Used in main / cache memory systems.
  - Example: Static RAM (SRAM) → data once written are retained as long as power is on.
  - Example: Dynamic RAM (DRAM) → requires periodic refreshing even when power is on (data stored as charge on tiny capacitors).

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.
- In modern memory organizations, every read request reads a block of words into some high-speed registers (LATENCY), from where data are supplied to the processor one by one (ACCESS TIME).

Design Issue of Memory System

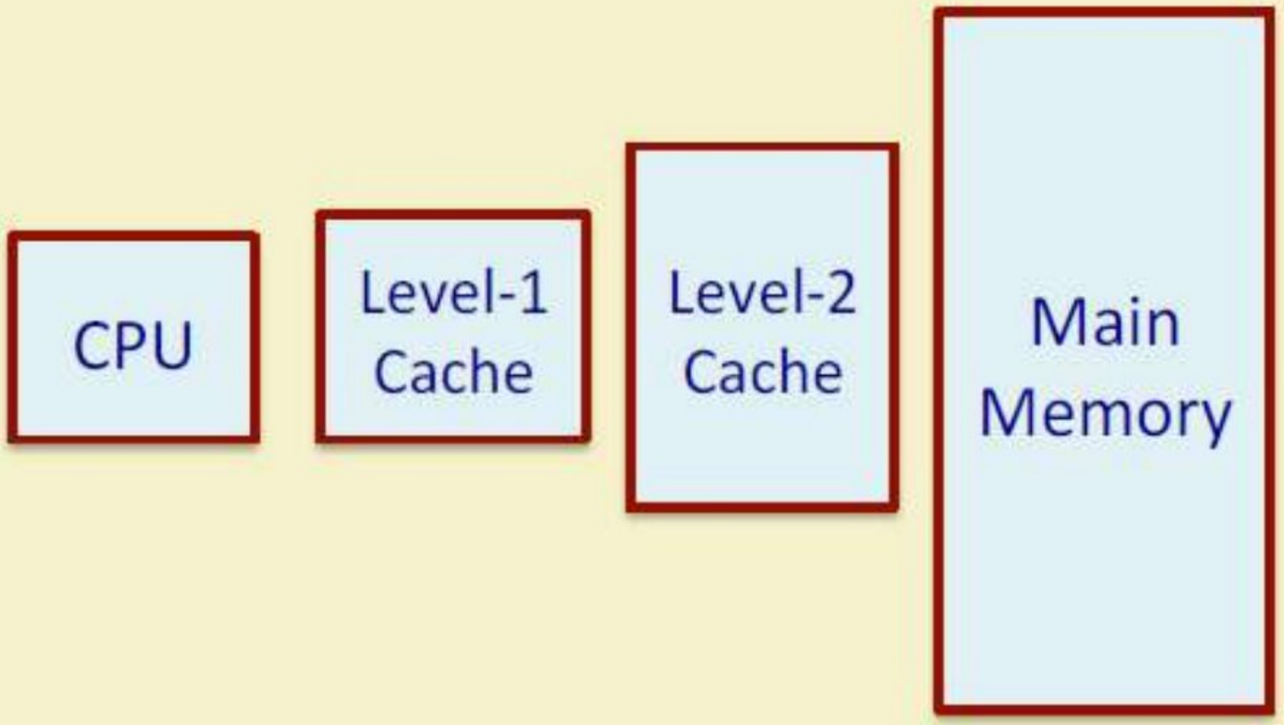
- The most important issue is to bridge the processor-memory gap that has been widening with every passing year.
  - Advancements in memory technology are unable to cope with faster advancements in processor technology.



- Some important questions?
  - How to make the memory system work faster?
  - How to increase the data transfer rate between CPU and memory?
  - How to address the ever increasing storage needs of applications?
- Some possible solutions:
  - *Cache Memory*: to increase the effective speed of the memory system.
  - *Virtual Memory*: to increase the effective size of the memory system.

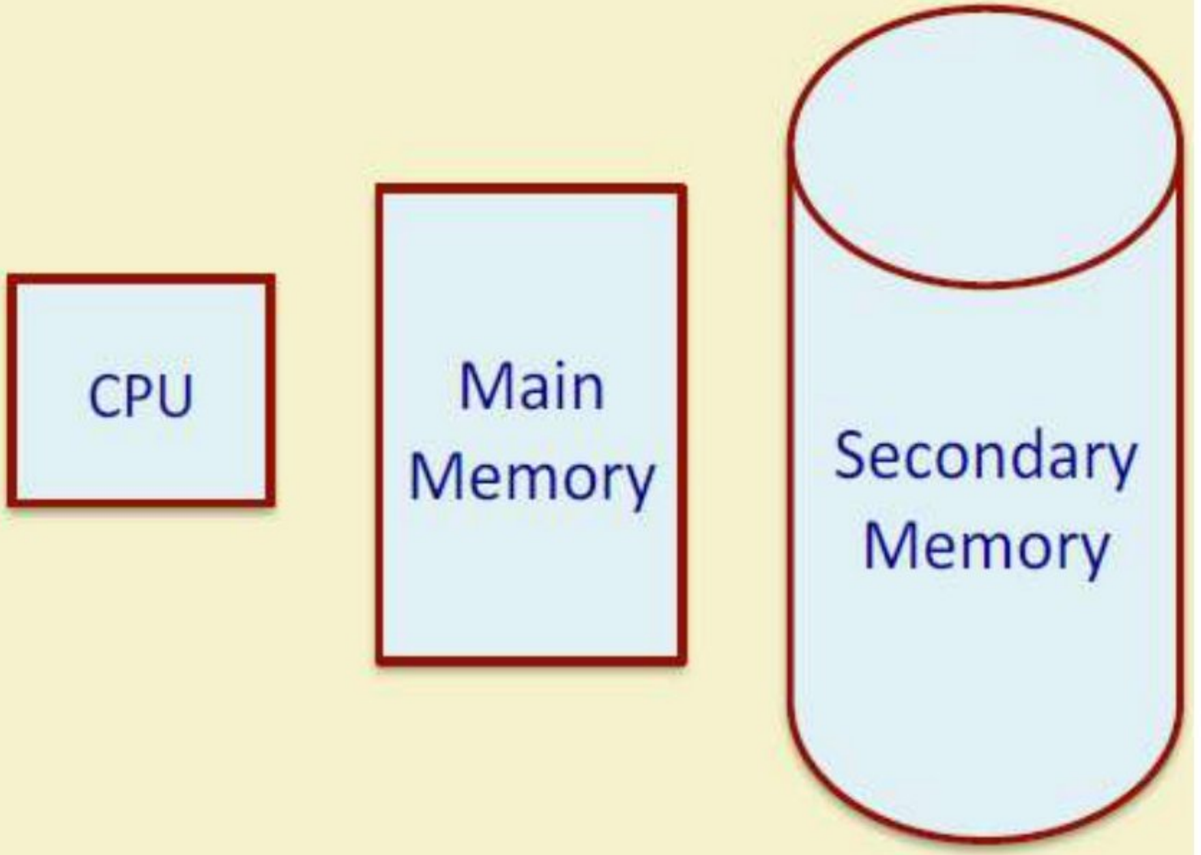
What is Cache Memory?

- A fast memory (possibly organized in several levels) that sits between processor and main memory.
- Faster than main memory and relatively small.
- Frequently accessed data and instructions are stored here.
- Cache memory makes use of the fast SRAM technology.



What is Virtual Memory?

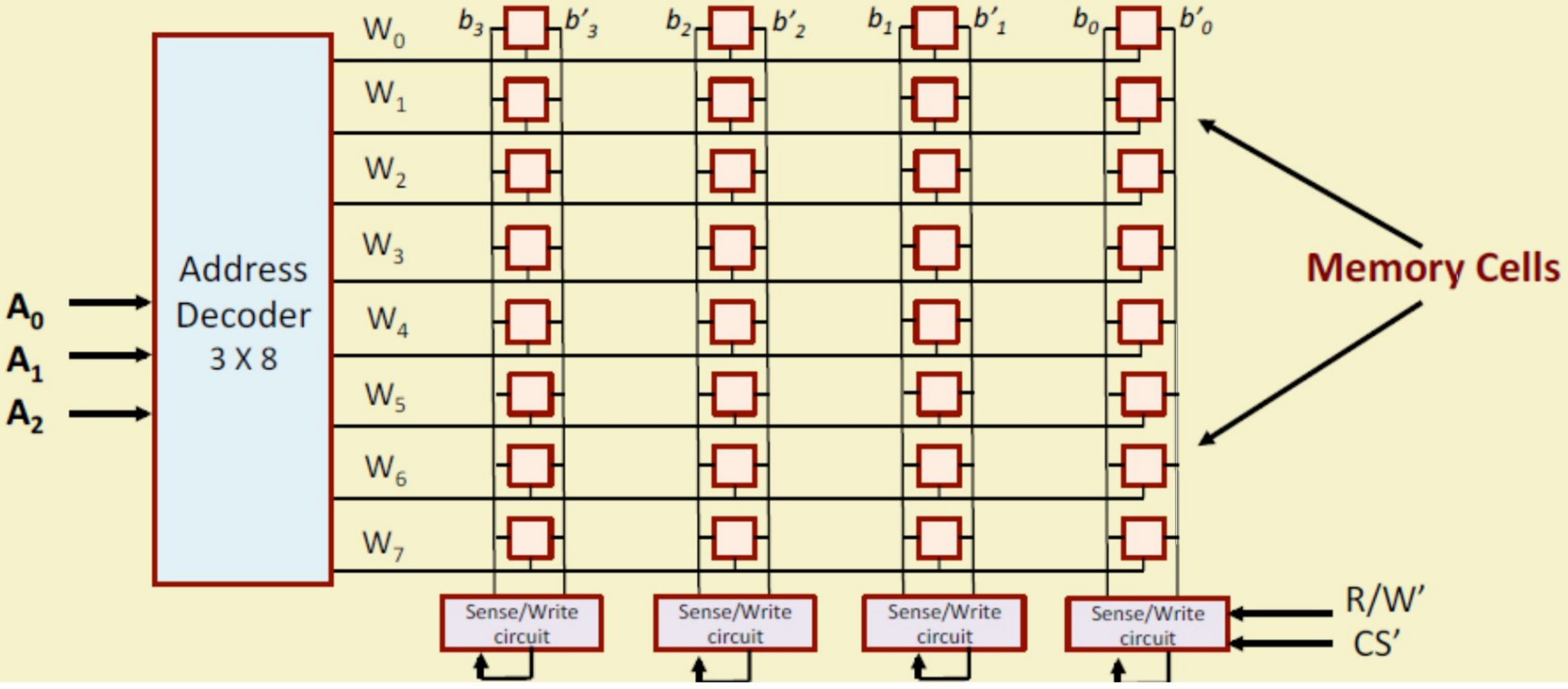
- Technique used by the operating system to provide an illusion of very large memory to the processor.
- Program and data are actually stored on secondary memory that is much larger.
- Transfer parts of program and data from secondary memory to main memory only when needed.



How a Memory Chip Looks Like?

- Memory cells are organized in the form of an array.
- Every memory cell holds one bit of data.
- Present-day VLSI technology allows one to pack billions of bits per chip.
- A memory module used in computers typically contains several such chips.

Organization of Cells in an 8x4 Memory Chip



- A 32-bit memory chip organized as 8 x 4 is shown.
- 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.

External Connection Requirements

- The 8 x 4 memory requires the following external connections:
  - Address decoder of size: 3 x 8
    - 3 external connections for address.
  - Data output : 4-bit
    - 4 external connections for data.
  - 2 external connections for R/W' and CS'.
  - 2 external connections for power supply and ground.
  - Total of 3 + 4 + 2 + 2 = 11.

What About a 256 X 16 Memory?

- Here the total number of external connections are estimated as follows.
  - Address decoder size: 8 x 256
    - 8 external connections for address.
  - Data output : 16-bit
    - 16 external connections for data.
  - 2 external connections for R/W' and CS'.
  - 2 external connections for power supply and ground.
  - Total of 8 + 16 + 2 + 2 = 28.

Introduction

- Broadly two types of semiconductor memory systems:
  a) Static Random Access Memory (SRAM)
  b) Dynamic Random Access Memory (DRAM)
     i. Asynchronous DRAM
     ii. 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.

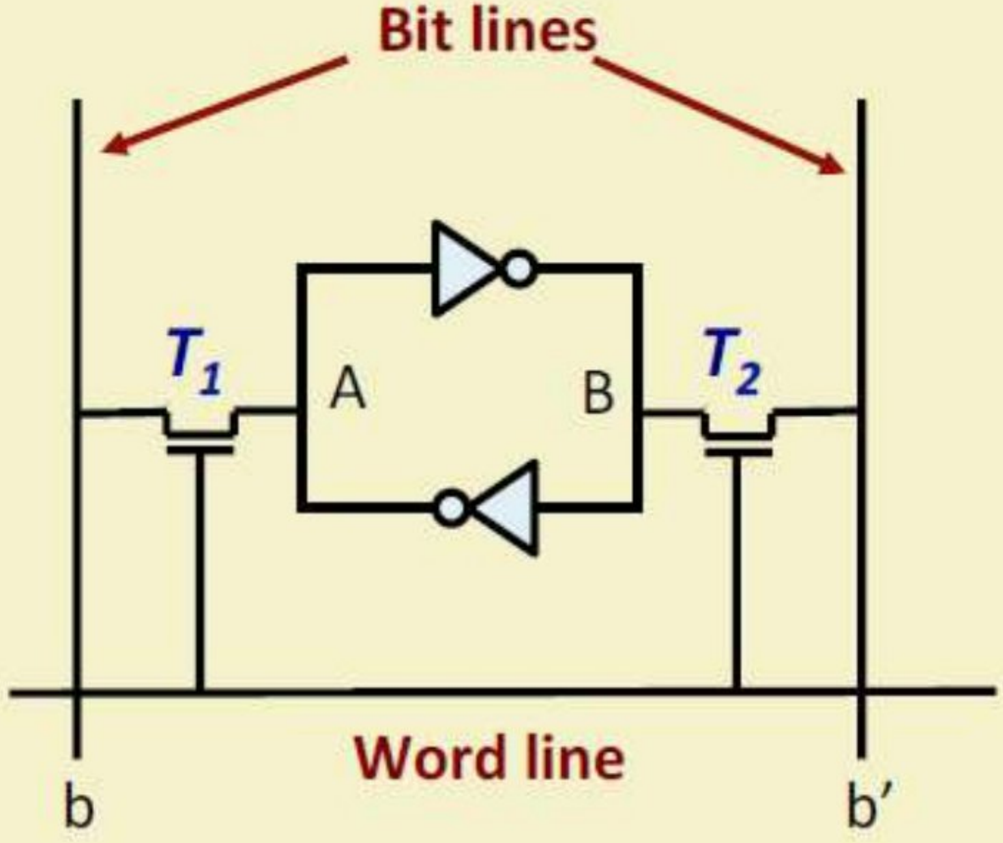
Static Random Access Memory (SRAM)

- SRAM consists of circuits which can 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 memory arrays can be arranged in rows and columns of memory cells.
  - Called *word line* and *bit line*.

- SRAM technology:
  - Can be built using 4 or 6 MOS transistors.
  - Modern SRAM chips in the market uses 6-transistor implementations for CMOS compatability.
  - Widely used in small-scale systems like microcontrollers and embedded systems.
  - Also used to implement cache memories in computer systems.
    - To be discussed later.

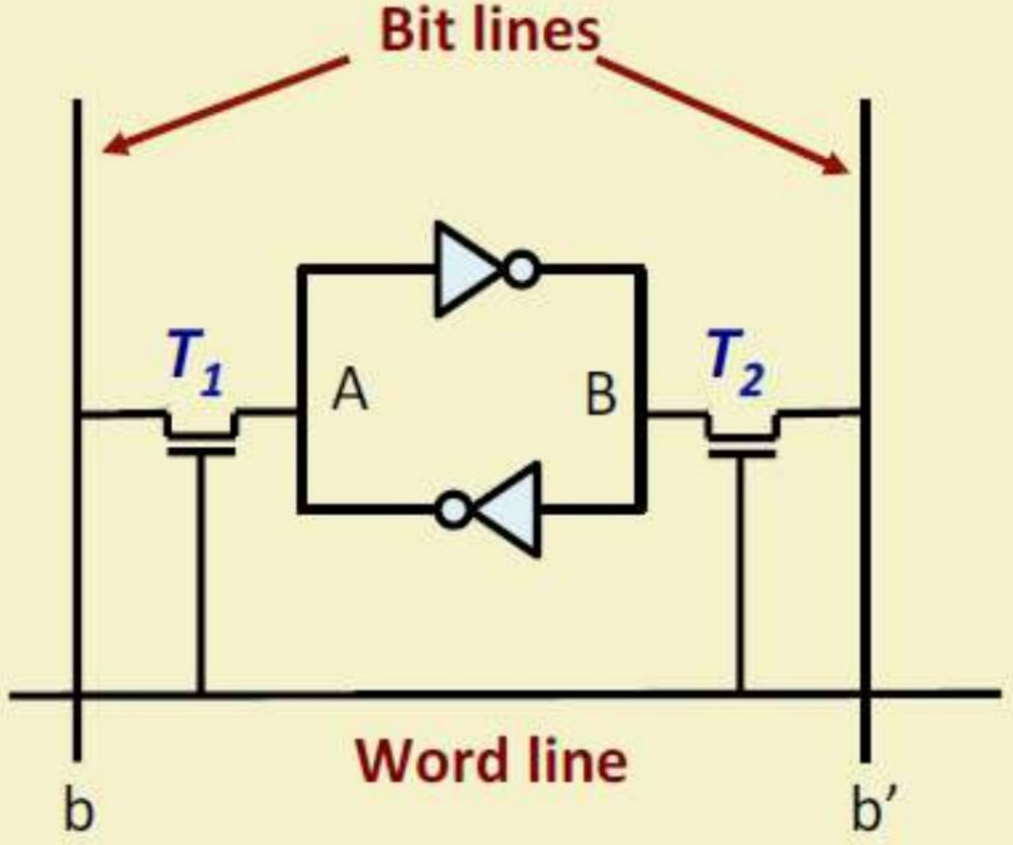
A 1-bit SRAM Cell

- 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.



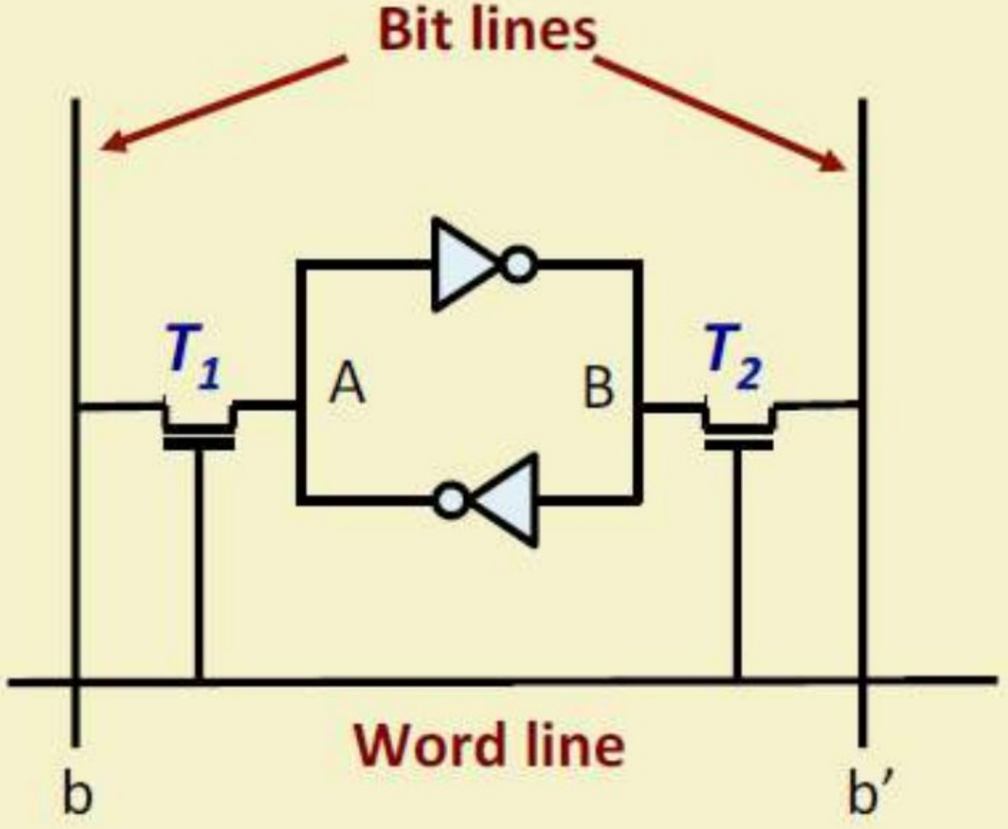
READ Operation in SRAM

- 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'*.



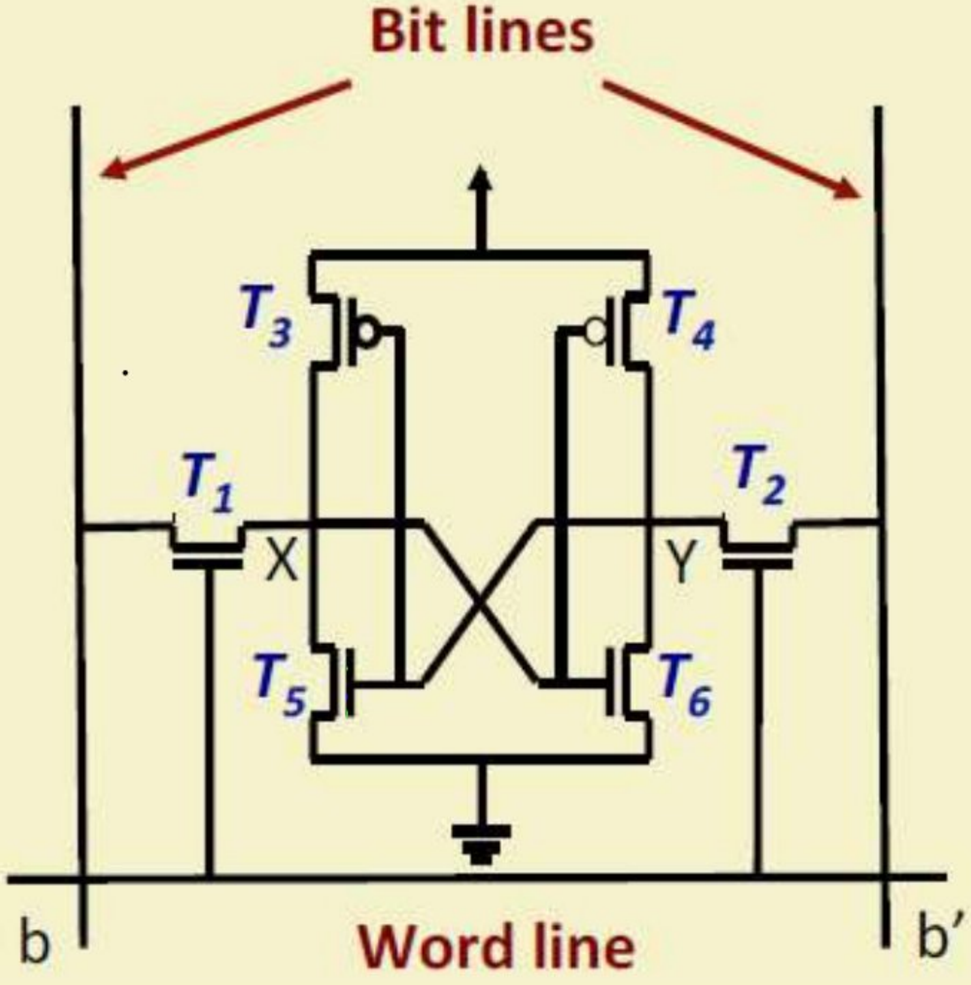
WRITE Operation in SRAM

- **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.



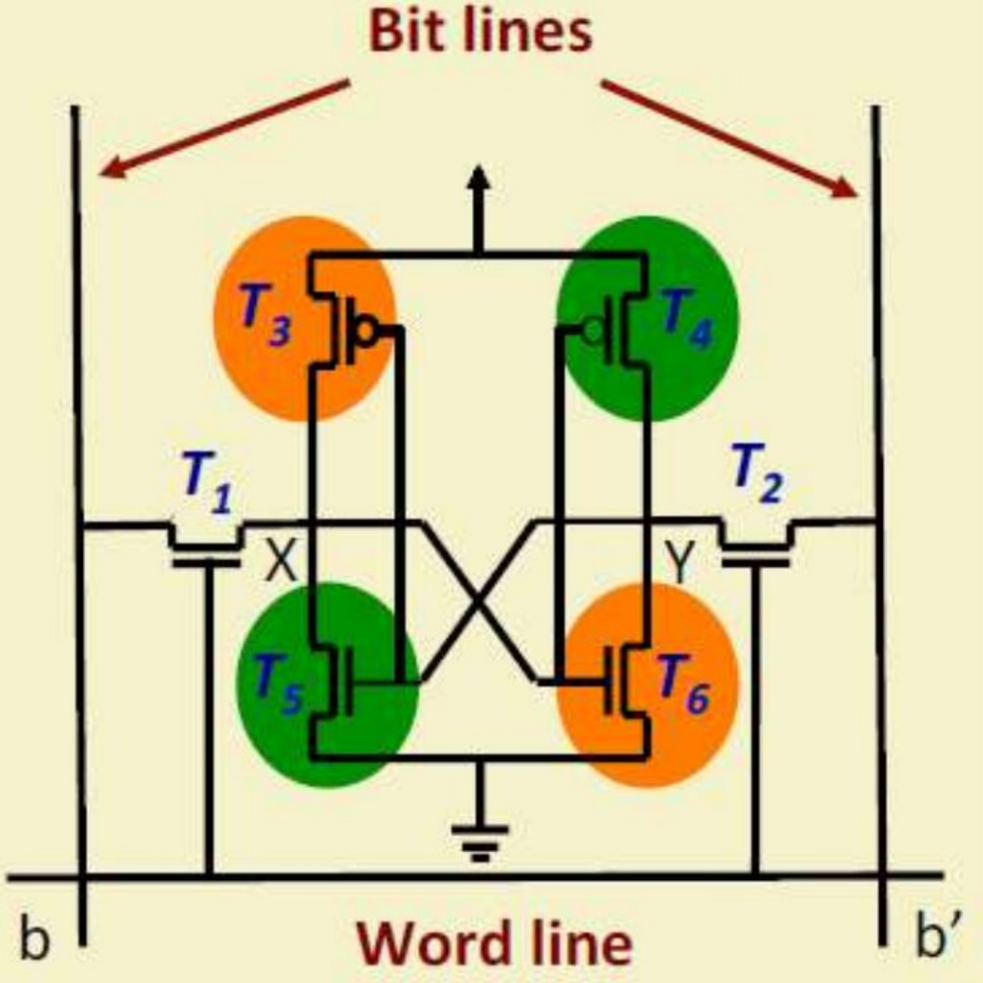
6-Transistor Static Memory cell

- 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.



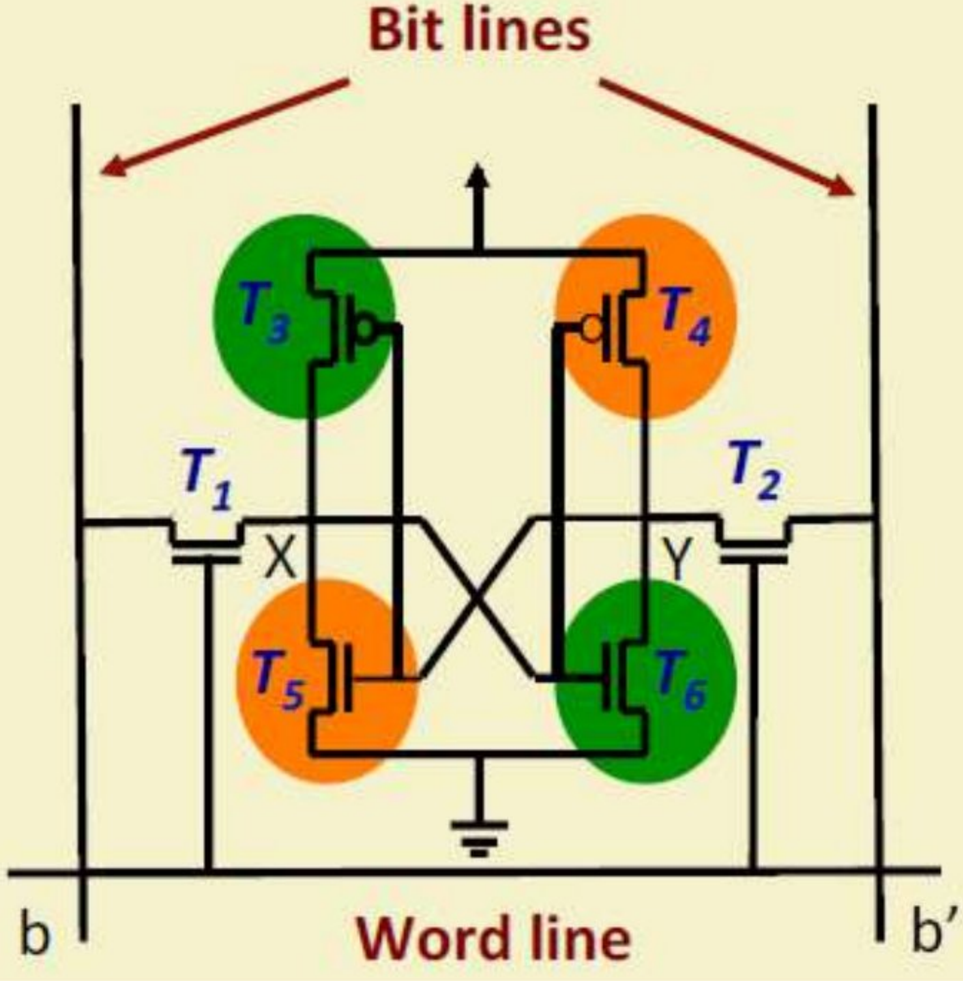
In State 0

- In state 0 the voltage at *X* is low and the voltage at *Y* is high.
- When the voltage at *X* is low, transistors (*T4* & *T5*) are on while (*T3* & *T6*) are off.
- When word line is activated, *T1* and *T2* are turned on and the bit lines *b* will have *0* and *b'* will have *1*.



In State 1

- In state 1 the voltage at *X* is high and the voltage at *Y* is low.
- When the voltage at *X* is high, transistors (*T3* & *T6*) are on while (*T4* & *T5*) are off.
- When word line is activated, *T1* and *T2* are turned on and the bit lines *b* will have *1* and *b'* will have *0*.

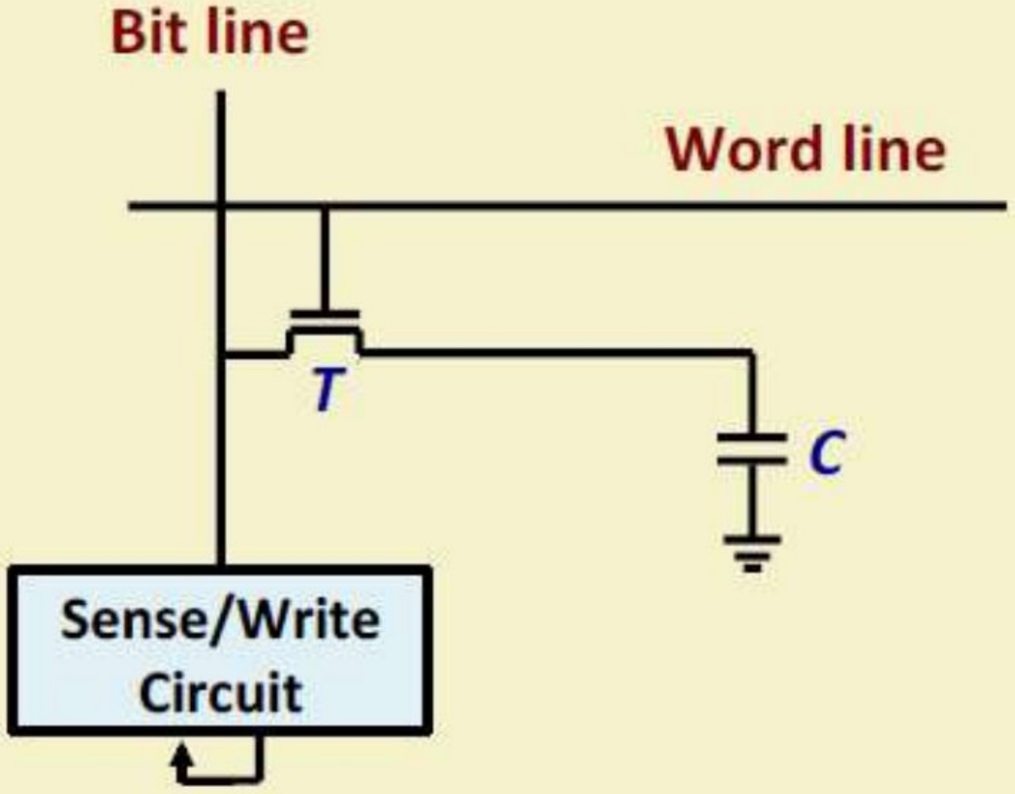


Features of SRAM

- 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.
- Simplicity – refresh circuitry is not needed.
  - Volatile :: continuous power supply is required.
- Fast operation.
  - Access time is very fast; fast memories (cache) are built using SRAM.
- High cost.
  - 6 transistors per cell.
- Limited capacity.
  - Not economical to manufacture high-capacity SRAM chips.

Dynamic Random Access Memory (DRAM)

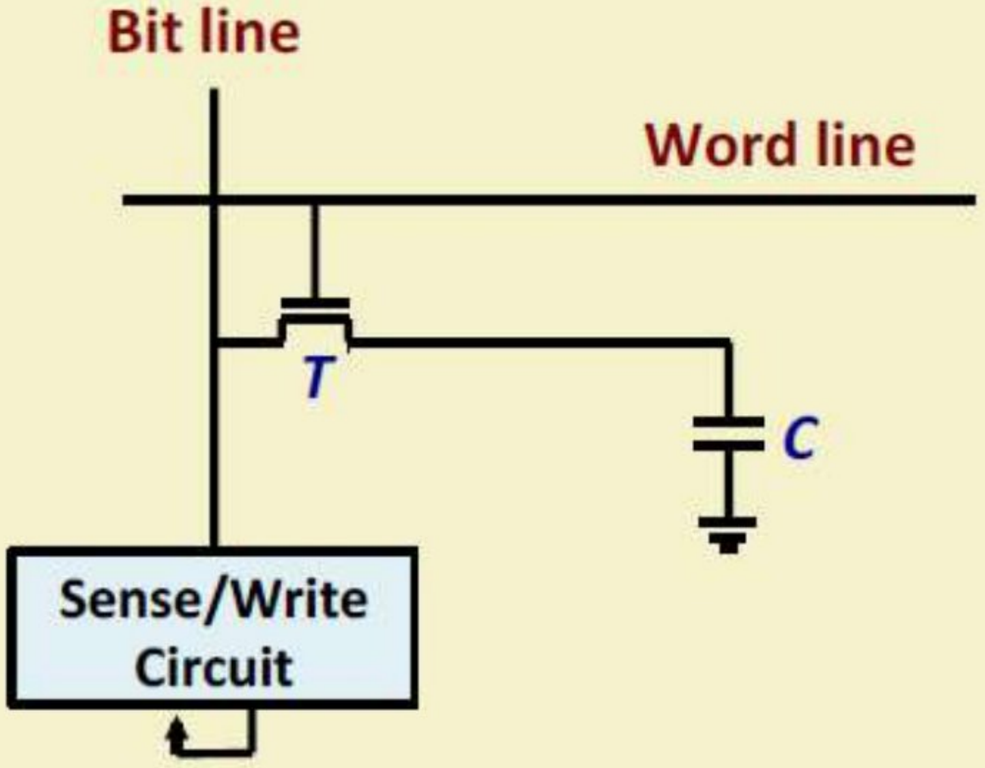
- Dynamic RAM do not retain its state even if power supply is on.
  - Data stored in the form of charge stored on a capacitor.
- Requires periodic refresh.
  - The charge stored cannot be retained over long time (due to leakage).
- Less expensive that SRAM.
  - Requires less hardware (one transistor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



1-transistor DRAM Cell

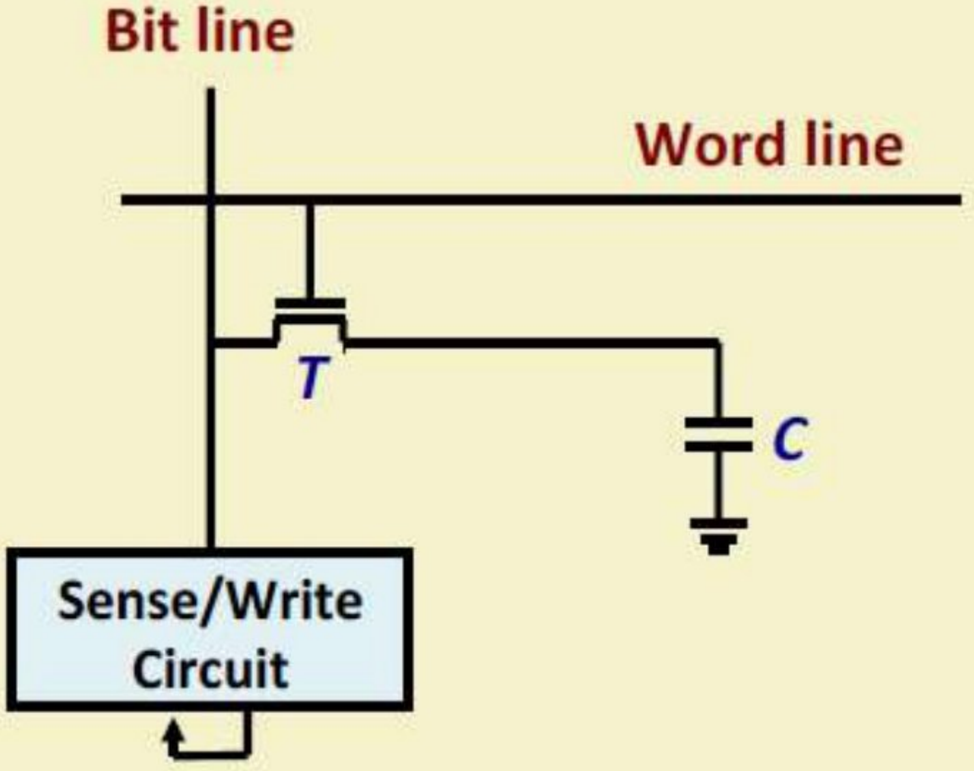
READ Operation in DRAM

- 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*.



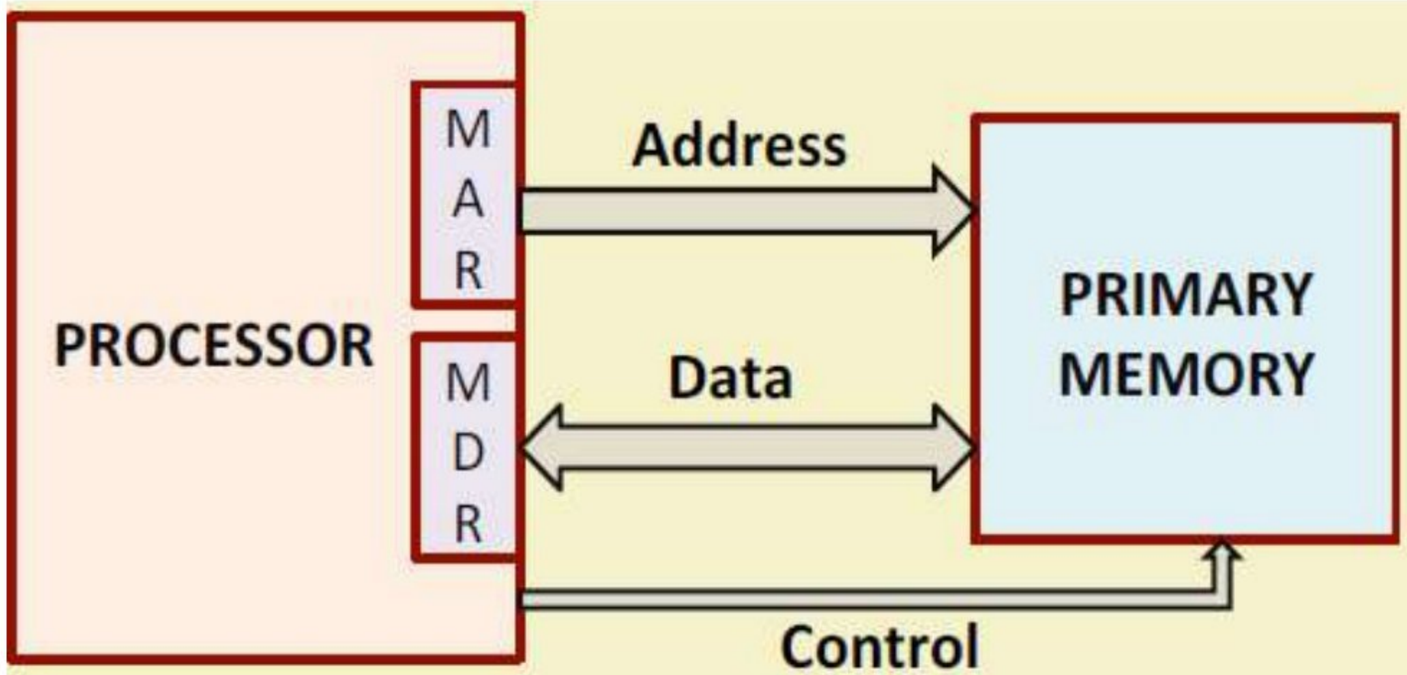
WRITE Operation in DRAM

- 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).



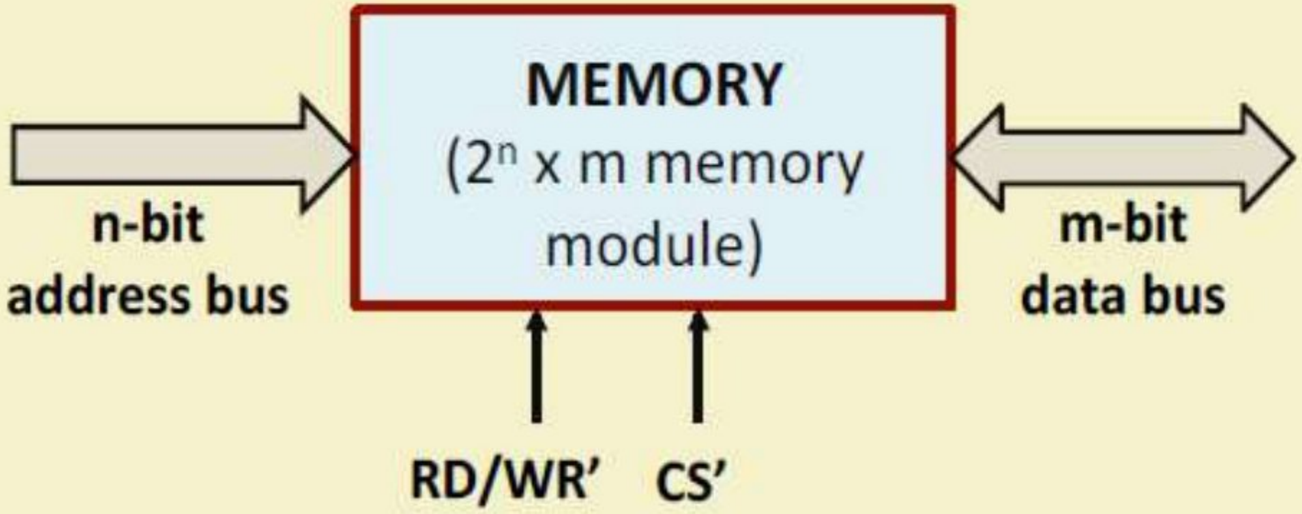
Memory Interfacing

- 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:
  - How the processor address and data lines are connected to memory modules?
  - How are the addresses decoded?
  - How are the memory addresses distributed among the memory modules?
  - How to speed up data transfer rate between processor and memory?



The processor's view of memory

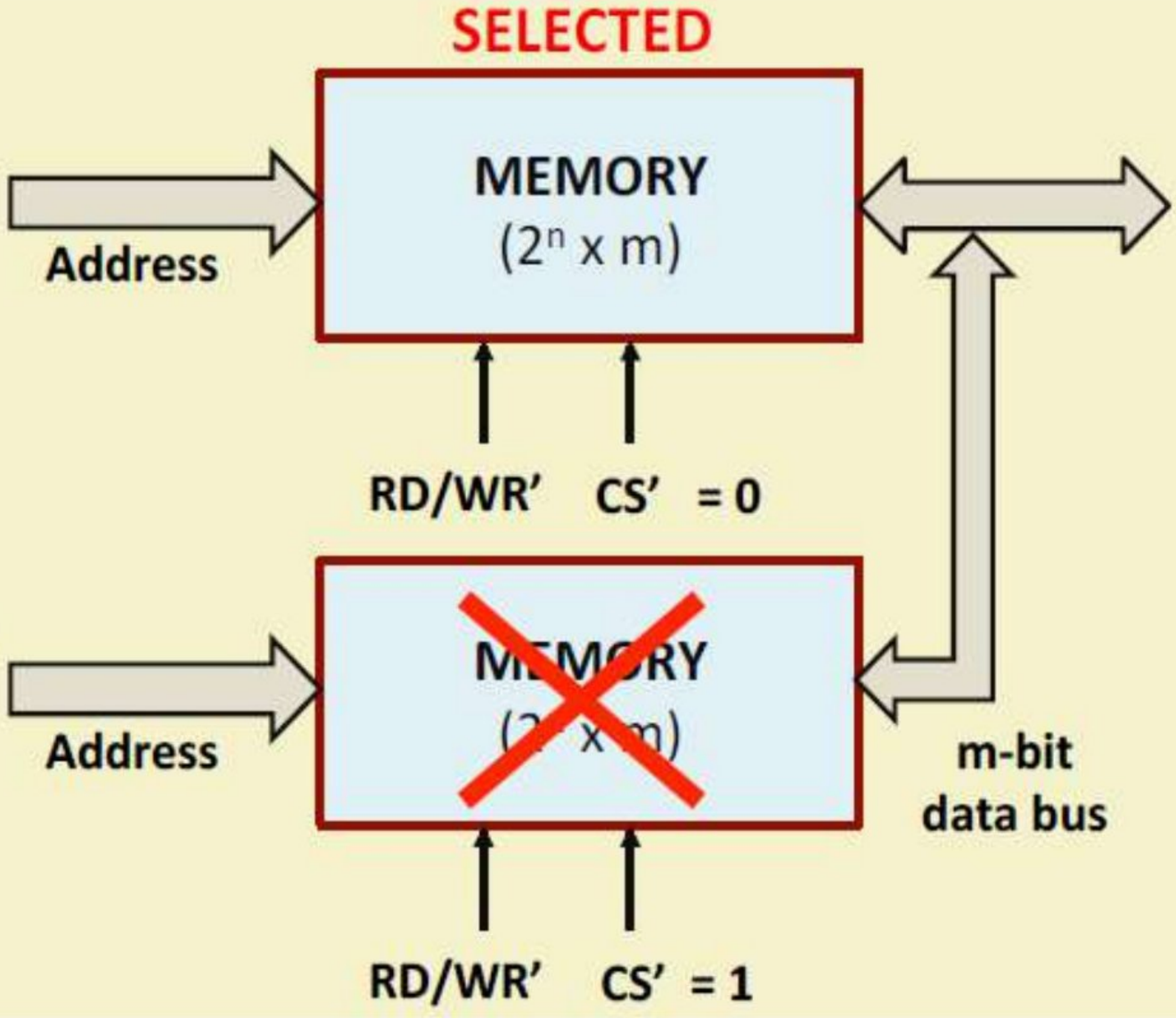
- Typical interface of a memory module.
- Real chip may contain more signal lines (e.g. DRAM).



A Note About the Memory Interface Signals

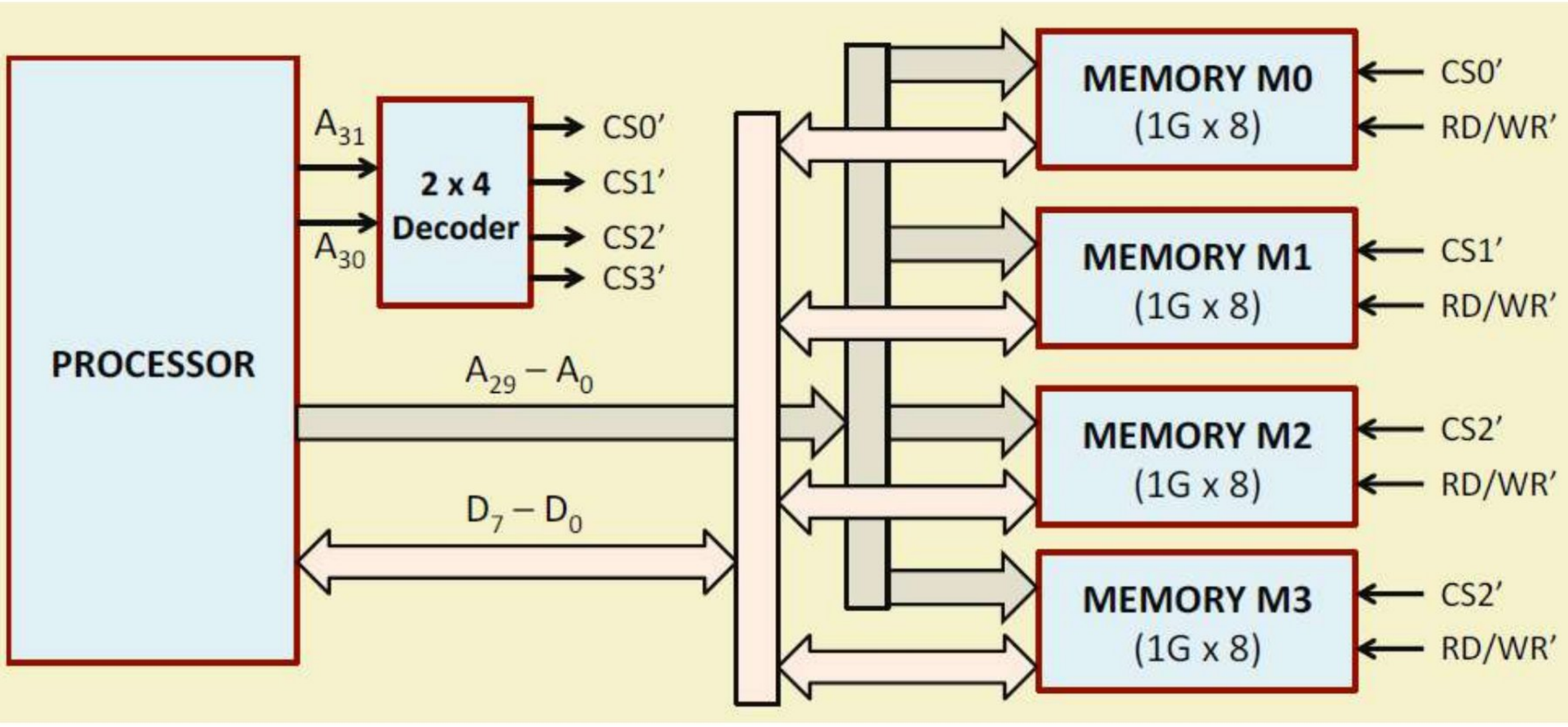
- The data signals of a memory module (RAM) are typically bidirectional.
  - Some memory chips may have separate data in and data out lines.
- For memory READ operation:
  - Address of memory location is applied to address lines.
  - RD/WR' control signal is set to 1, and CS' is set to 0.
  - Data is 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 problem.
  - 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 MIPS32 like 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.
  - Total memory of 4 Gbytes.


PROCESSOR
A31
A30
2 x 4
Decoder
CS0'
CS1'
CS2'
CS3'
A29 – A0
D7 – D0
MEMORY M0
(1G x 8)
MEMORY M1
(1G x 8)
MEMORY M2
(1G x 8)
MEMORY M3
(1G x 8)
CS0'
RD/WR'
CS1'
RD/WR'
CS2'
RD/WR'
CS2'
RD/WR'

- 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.
  - For MIPS32, we have to access 32 bits (4 bytes) of data in parallel, which requires four sequential memory accesses here.
  - We shall look at an alternate memory organization later that would make this possible.
    - Called ***memory interleaving***.

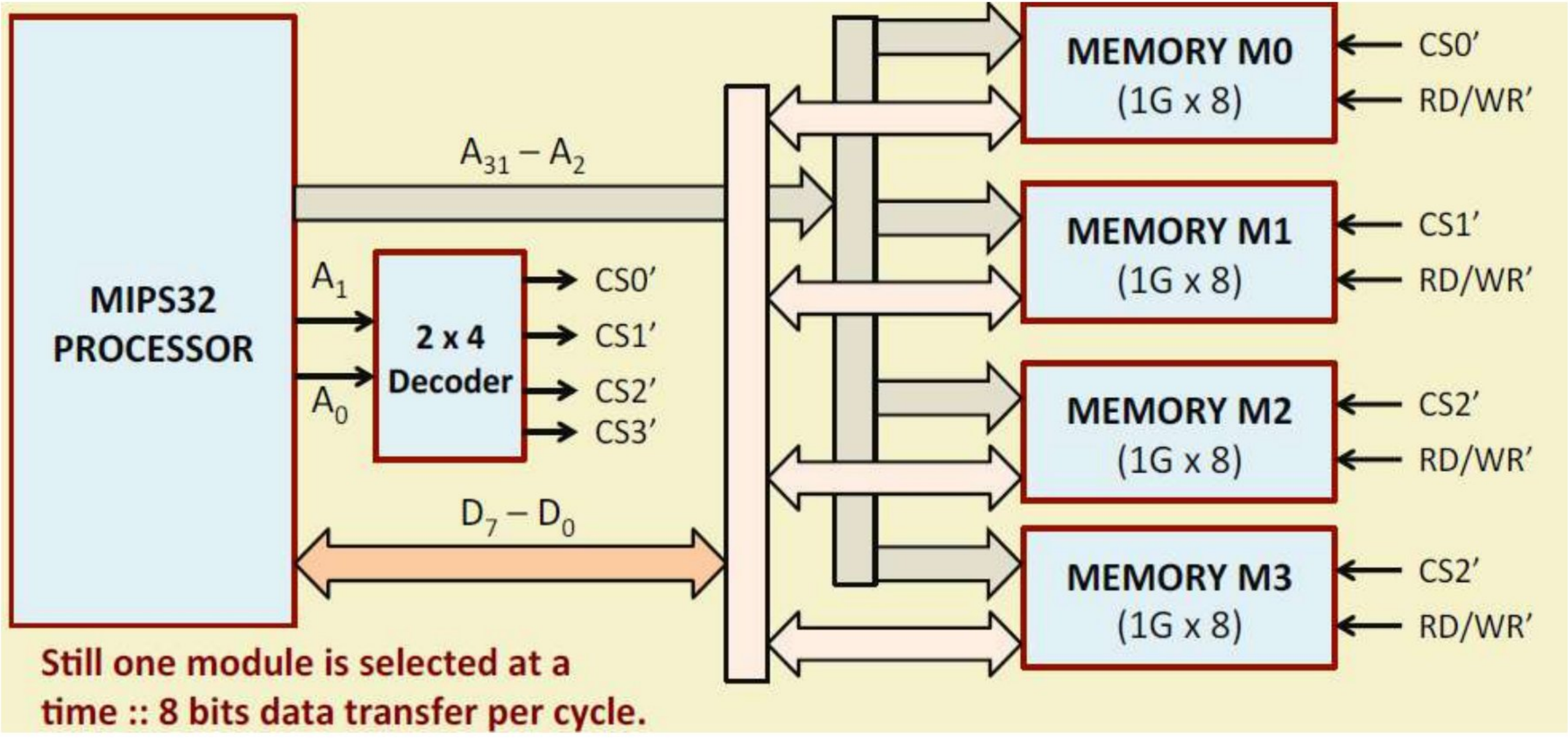
Improved Memory Interface for MIPS32

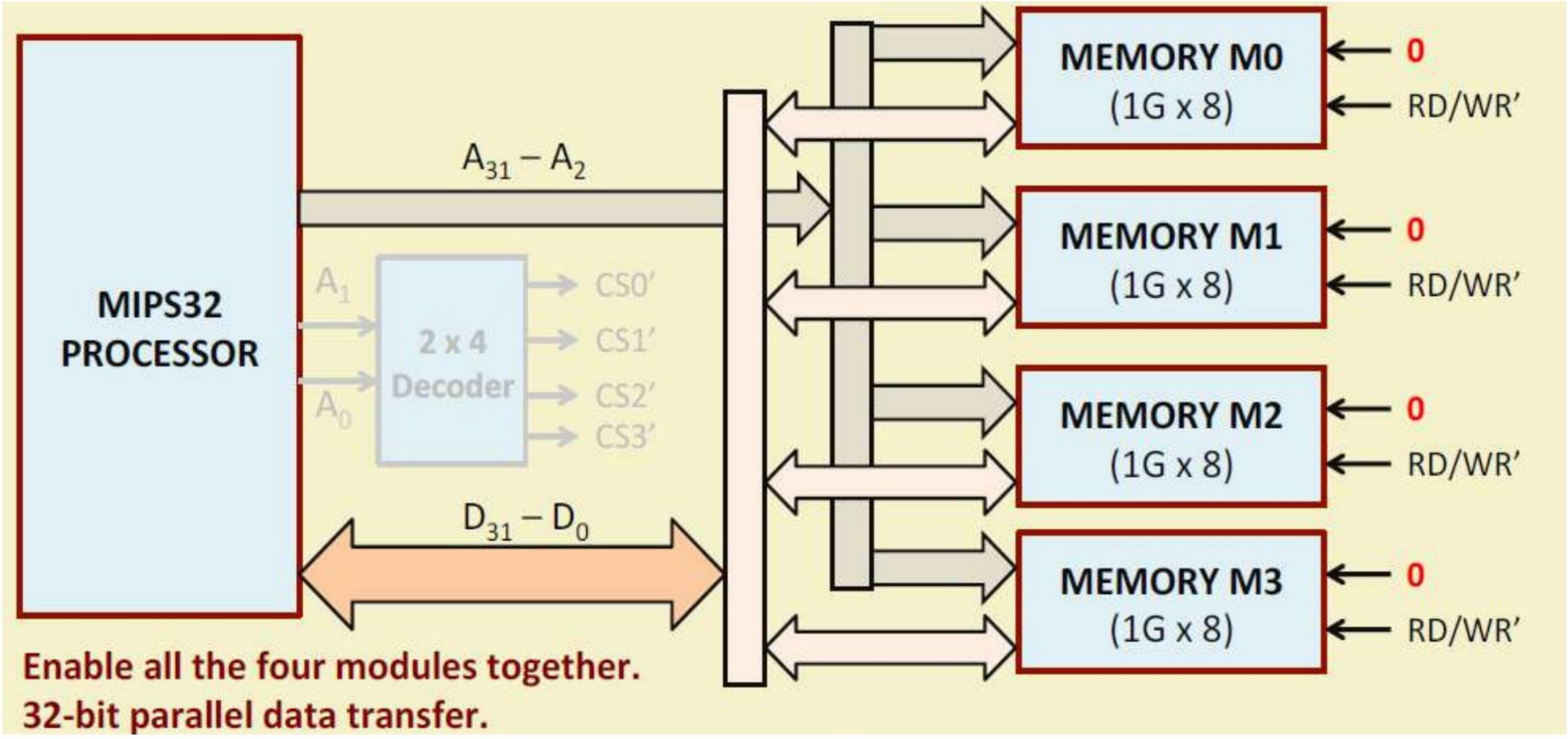
- We make small changes in the organization so that 32-bits of data can be fetched in a single memory access cycle.
  - Exploit the concept of memory interleaving.
- The main changes:
  - High order 30 address lines (A_{31}-A_2) are connected to memory modules.
  - Low order two address lines (A_1 and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

- Motivation for word alignment in MIPS32 data words.
  - 32-bit words start from a memory address that is divisible by 4.
    - Corresponding byte addresses are (0, 1, 2, 3), (4, 5, 6, 7), (8, 9, 10, 11) (12, 13, 14, 15), etc.
    - Possible to transfer all the four bytes in a single memory cycle.
  - What happens if a word is not aligned?
    - Say: (1, 2, 3, 4) or (2, 3, 4, 5) or (3, 4, 5, 6).
    - Two of the bytes will be mapped to the same memory module.
    - Hence the word cannot be transferred in a single memory cycle.

2 memory cycles required


MIPS32
PROCESSOR
$A_{31} - A_2$
A_1
A_0
2 x 4
Decoder
CS0'
CS1'
CS2'
CS3'
$D_7 - D_0$
MEMORY M0
(1G x 8)
MEMORY M1
(1G x 8)
MEMORY M2
(1G x 8)
MEMORY M3
(1G x 8)
CS0'
RD/WR'
CS1'
RD/WR'
CS2'
RD/WR'
CS2'
RD/WR'
Still one module is selected at a
time :: 8 bits data transfer per cycle.


MIPS32
PROCESSOR
$A_{31} - A_2$
A_1
A_0
2 x 4
Decoder
CS0'
CS1'
CS2'
CS3'
$D_{31} - D_0$
MEMORY M0
(1G x 8)
MEMORY M1
(1G x 8)
MEMORY M2
(1G x 8)
MEMORY M3
(1G x 8)
0
RD/WR'
0
RD/WR'
0
RD/WR'
0
RD/WR'
Enable all the four modules together.
32-bit parallel data transfer.

Memory Latency and Bandwidth

- Memory Latency:
  - The delay from the issue of a memory read request to the first byte of data becoming available.
- Memory Bandwidth:
  - The maximum number of bytes that can be transferred between the processor and the memory system per unit time.

- Example 1:

  Consider a memory system that takes 20 ns to service the access of a single 32-bit word.

  - Latency L = 20 ns per 32-bit word.
  - Bandwidth BW = 32 / (20 x 10^{-9}) = 200 Mbits per second.

- Example 2:

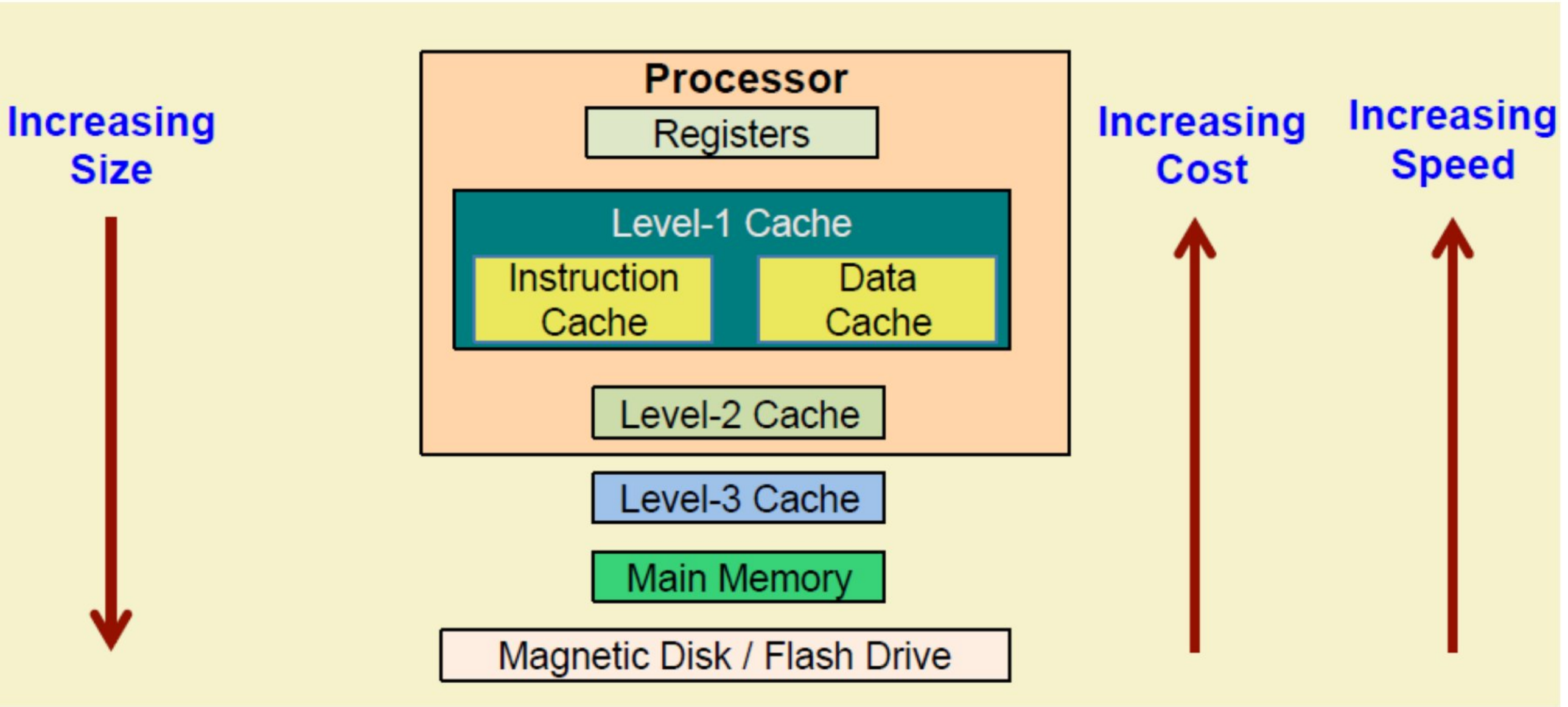
  – The memory system is modified to accept a new (still 20ns) request for a 32-bit word every 5 ns by overlapping requests.

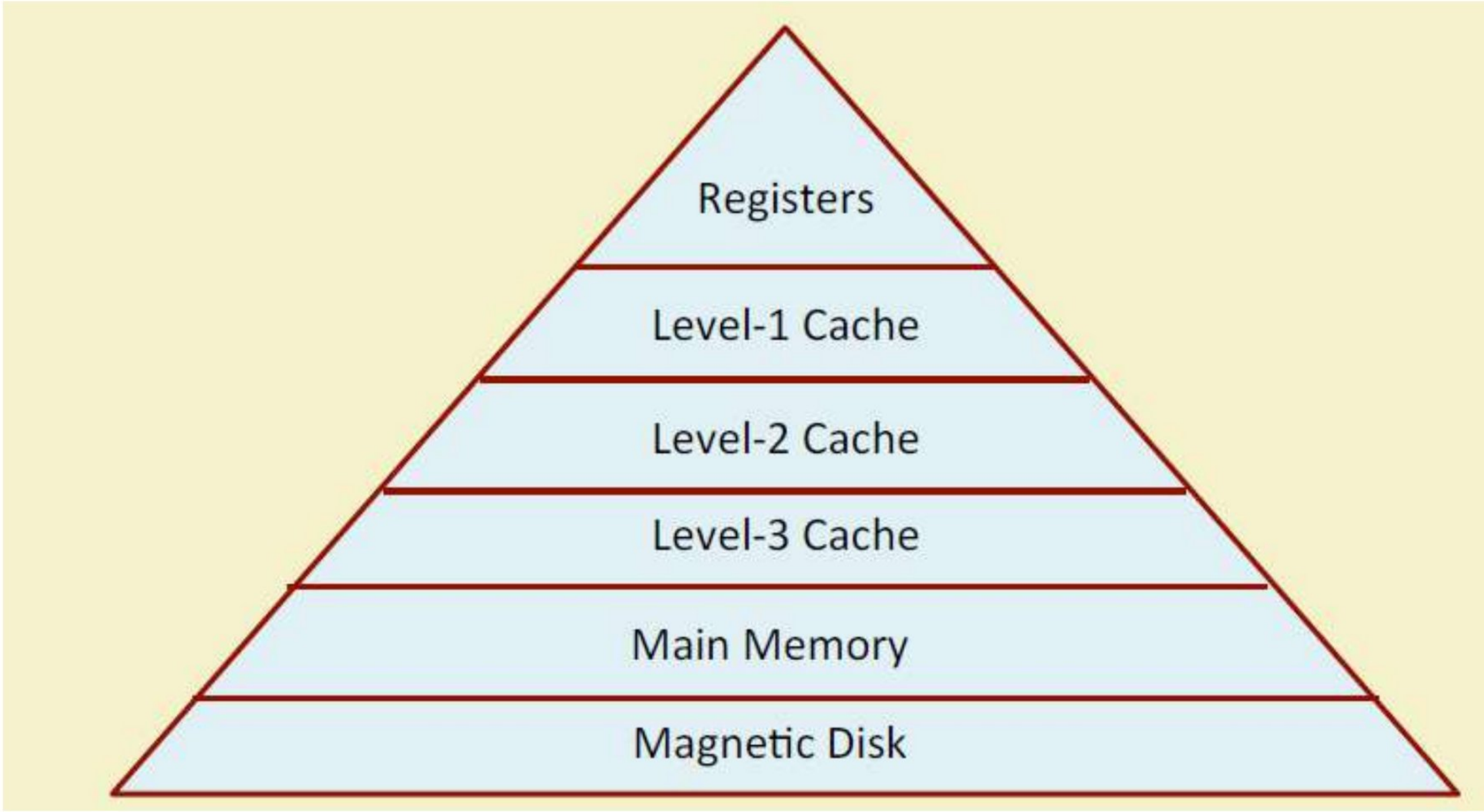
  - Latency L = 20 ns per 32-bit word (*no change*).
  - Bandwidth BW = 32 / (5 x 10^{-9}) = 800 Mbits per second.

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 (typically, magnetic disk or flash storage).
  - We incrementally add smaller (but faster) memories, each containing a subset of the data stored in the memory below it.
  - We proceed in steps towards the processor.

- Typical hierarchy (starting with closest to the processor):
  1. Processor registers
  2. Level-1 cache (typically divided into separate instruction and data cache)
  3. Level-2 cache
  4. Level-3 cache
  5. Main memory
  6. Secondary memory (magnetic disk / flash drive)
- As we move away from the processor:
  - Size increases
  - Cost decreases
  - Speed decreases


Increasing Size
Processor
Registers
Level-1 Cache
Instruction Cache
Data Cache
Level-2 Cache
Level-3 Cache
Main Memory
Magnetic Disk / Flash Drive
Increasing Cost
Increasing Speed

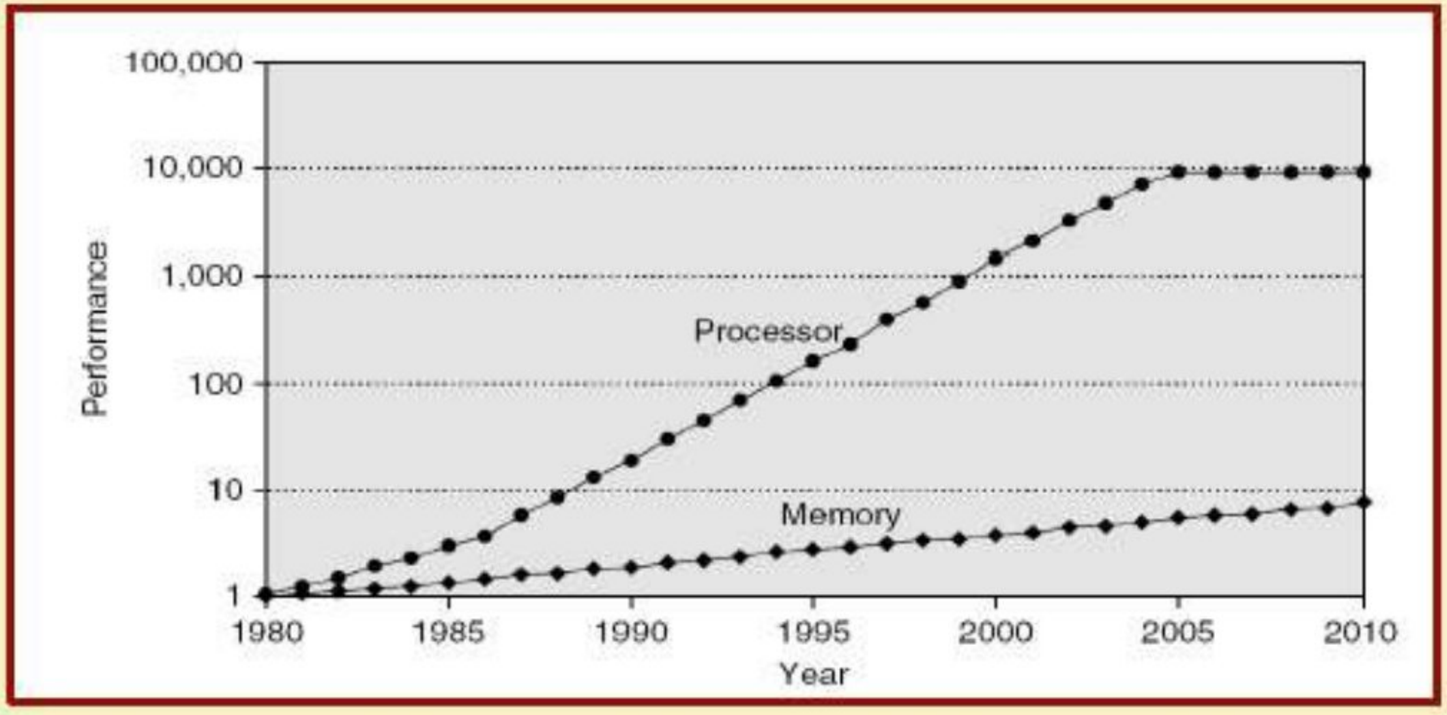

Registers
Level-1 Cache
Level-2 Cache
Level-3 Cache
Main Memory
Magnetic Disk

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	
Magnetic disk	5-50 ms	100 GB – 16 TB	

Major Obstacle in Memory System Design

- Processor is much faster than main memory.
  - Has to spend much of the time waiting while instructions and data are being fetched from main memory.
  - Memory speed cannot be increased beyond a certain point.



Impact of Processor / Memory Performance Gap

Year	CPU Clock	Clock Cycle	Memory Access	Minimum CPU Stall Cycles
1986	8 MHz	125 ns	190 ns	190 / 125 – 1 = 0.5
1989	33 MHz	30 ns	165 ns	165 / 30 – 1 = 4.5
1992	60 MHz	16.6 ns	120 ns	120 / 16.6 – 1 = 6.2
1996	200 MHz	5 ns	110 ns	110 / 5 – 1 = 21.0
1998	300 MHz	3.33 ns	100 ns	100 / 3.33 – 1 = 29.0
2000	1 GHz	1 ns	90 ns	90 / 1 – 1 = 89.0
2002	2 GHz	0.5 ns	80 ns	80 / 0.5 – 1 = 159.0
2004	3 GHz	0.33 ns	60 ns	60 / 0.33 – 1 = 179.0

Ideal memory access time = 1 CPU cycle

Real memory access time >> 1 CPU cycle

Locality of Reference

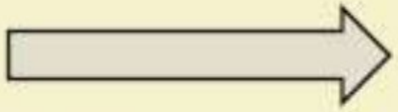
- Programs tend to reuse data and instructions they have used recently.
  - Rule of thumb: 90% of the total execution time of a program is spent in only 10% of the code (also called 90/10 rule).
  - Reason: nested loops in a program, few procedures calling each other repeatedly, arrays of data items 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;
```



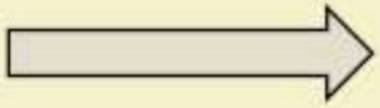
```
        ADDI    $t1,$zero,1
        ADDI    $t2,$zero,N
        ADDI    $t3,$zero,1
Loop:   MUL     $t1,$t1,$t3
        ADDI    $t3,$t3,1
        SGT     $t4,$t3,$t2
        BNEZ    $t4,Loop
```

- 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];
```



```
        SUB     $t1,$t1,$t1
        ADDI    $t2,$zero,N
        ADDI    $t3,$zero,1
        ADDI    $t5,$zero,A
Loop:   LW      $t8,0($t5)
        ADD     $t1,$t1,$t8
        ADDI    $t3,$t3,1
        SGT     $t4,$t3,$t2
        BNEZ    $t4,Loop
```

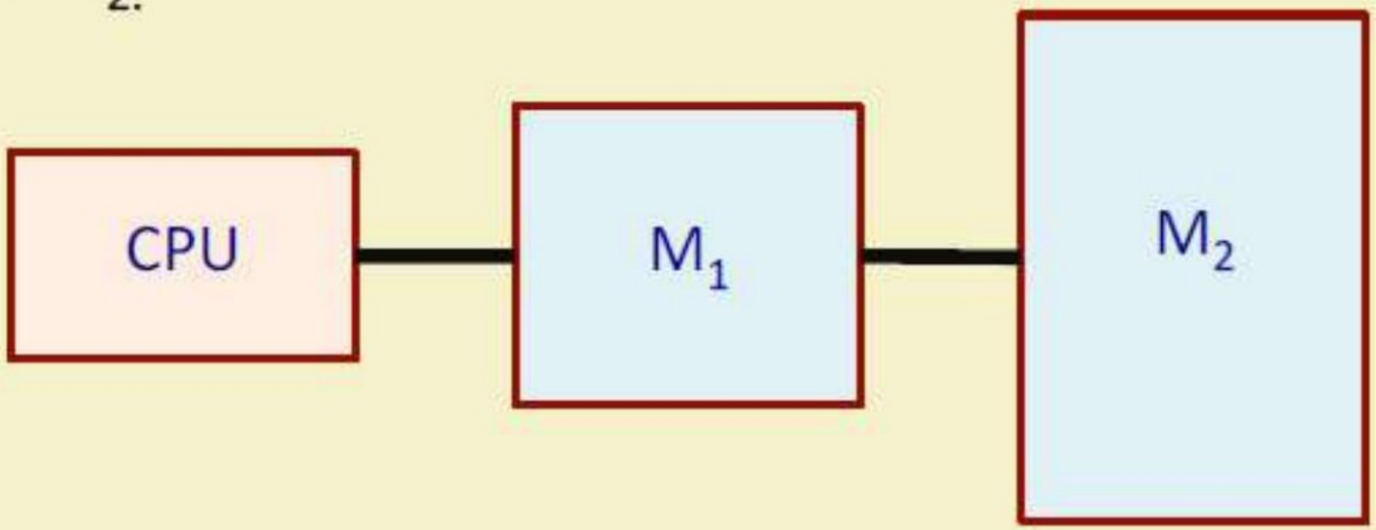
- Performance can be improved by copying the array into cache memory.

Some Common Terminologies Used

- Block: The smallest unit of information transferred between two levels.
- Hit Rate: The fraction of memory accesses found in the upper level.
- Hit Time: Time to access the upper level
  - Upper level access time + Time to determine hit/miss
- Miss: Data item needs to be retrieved from a block in the lower level.
- Miss Rate: The fraction of memory accesses not found in the upper level.
- Miss Penalty: Overhead whenever a miss occurs.
  - Time to replace a block in the upper level + Time to transfer the missed block

Performance of Memory Hierarchy

- We first consider a 2-level hierarchy consisting of two levels of memory, say, M_1 and M_2.



- 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.

- 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.

- Efficiency:
  - Let $r = t_{A2} / t_{A1}$ denote the access time ratio of the two levels of memory.
  - We define the access efficiency as $e = t_{A1} / t_A$, which is the factor by which t_A differs from its minimum possible value.

$$\text{Efficiency } e = \frac{t_{A1}}{H.t_{A1} + (1 - H).t_{A2}} = \frac{1}{H + (1 - H).r}$$

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

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

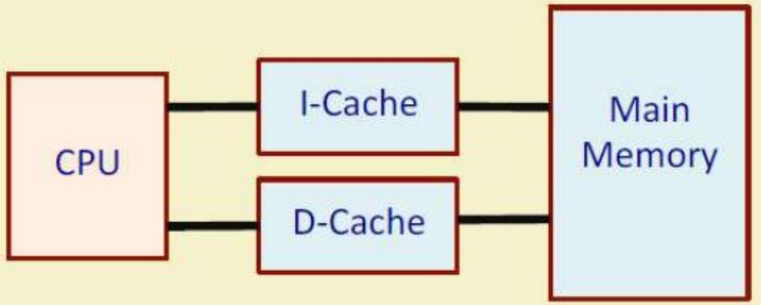
  - The same result follows from Amadahl's law.

Example 1

- Consider a 2-level memory hierarchy consisting of a cache memory M_1 and the main memory M_2. Suppose that the cache is 6 times faster than the main memory, and the cache can be used 90% of the time. How much speedup do we gain by using the cache?
  - Here, $r = 6$ and $H = 0.90$
  - Thus, $S = 1 / [H / r + (1 - H)] = 1 / (0.90 / 6 + 0.10) = 1 / 0.25 = 4$

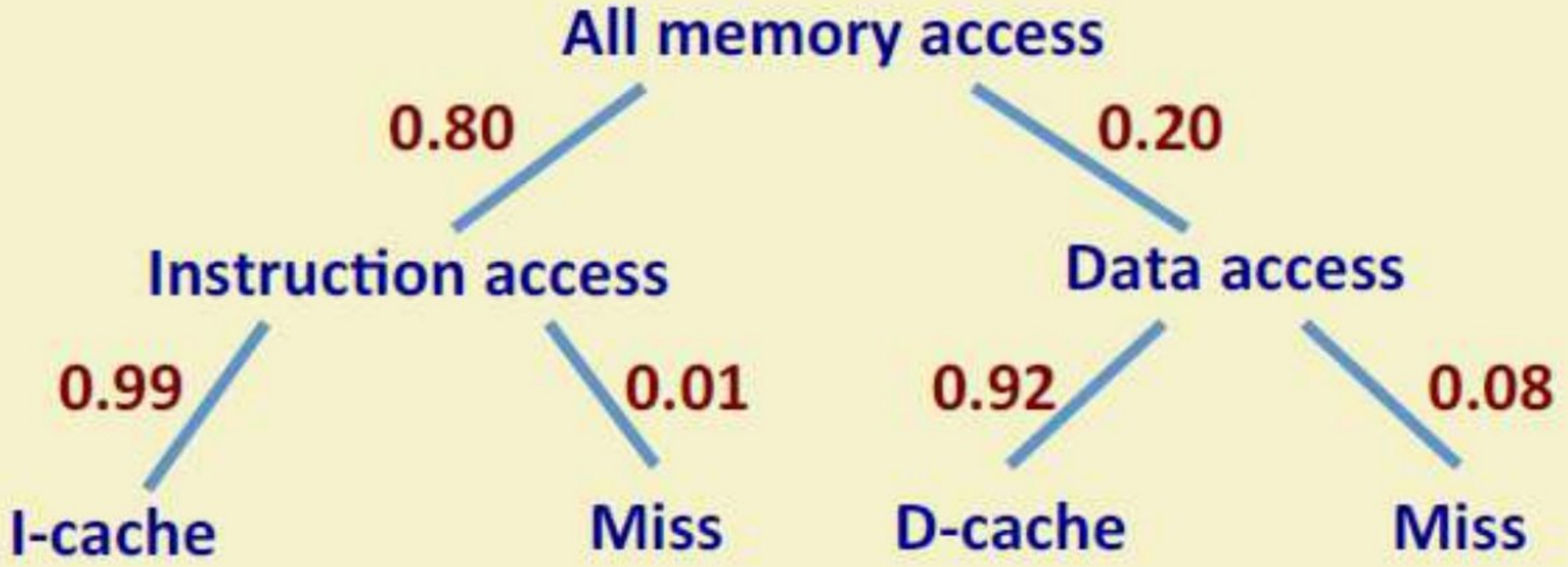
Example 2

- 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.

Find the average access time



t_{MISS} = 1 + 15 = 16 cycles

Average number of cycles per access:

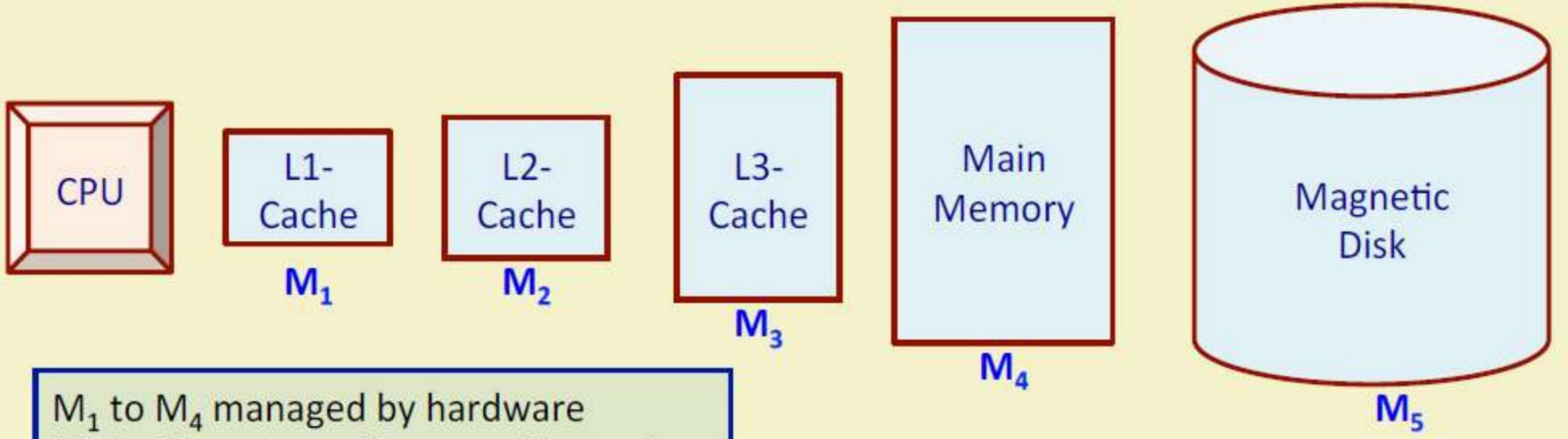
0.80 x (0.99 x 1 + 0.01 x 16) + 0.20 x (0.92 x 1 + 0.08 x 16)

= 0.92 + 0.44 = 1.36

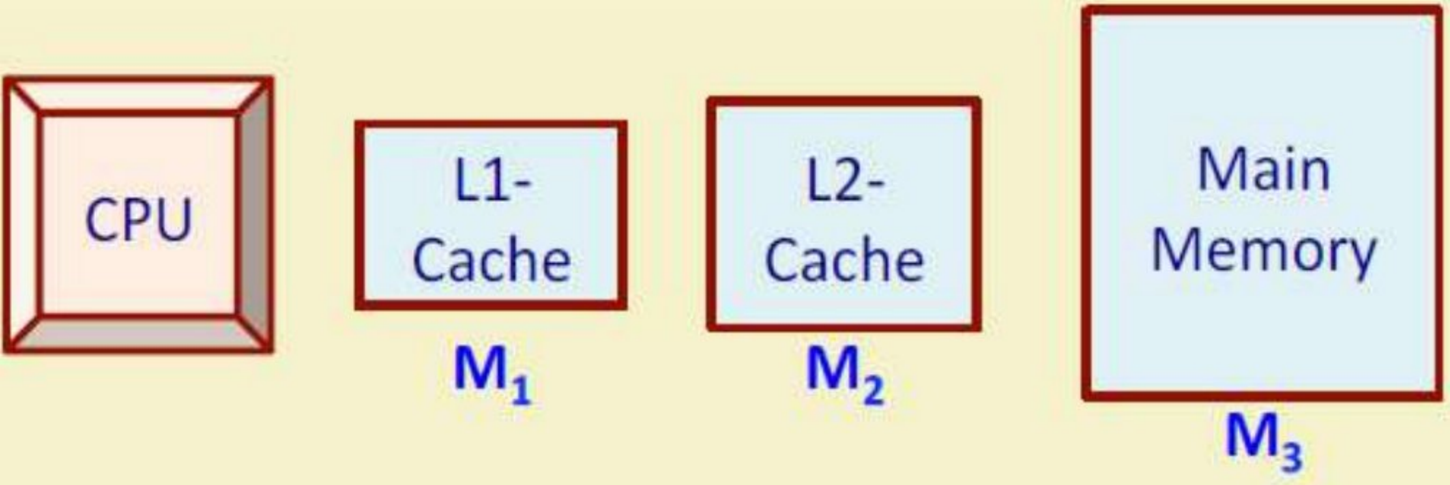
Thus, average access time t_A = 1.36 x 2 ns = 2.72 ns

Performance Calculation for Multi-Level Hierarchy

- Most of the practical memory systems use more than 2 levels of hierarchy.



M_1 to M_4 managed by hardware
M_4 to M_5 managed by operating system



t_{L1} : access time of M_1

t_{L2} : access time of M_2

H_{L1} : hit ratio of M_1

H_{L2} : hit ratio of M_2 with respect to the residual accesses that try to access M_2

- Consider a 3-level hierarchy consisting of L1-cache, L2-cache and main memory.
- Whenever there is a miss in L1, we go to L2.
- Average access time can be calculated as:

$$t_A = H_{L1}.t_{L1} + (1 - H_{L1}) . [H_{L2}.t_{L2} + (1 - H_{L2}).t_{MISS2}]$$

- Here, t_{MISS2} is the miss penalty when the requested data is found neither in M_1 nor in M_2.

Implications of a Memory Hierarchy to the CPU

- Processors designed without memory hierarchy are simpler because all memory accesses take the same amount of time.
  - Misses in a memory hierarchy implies variable memory access times as seen by the CPU.
- Some mechanism is required to determine whether or not the requested information is present in the top level of the memory hierarchy.
  - Check happens on every memory access and affects hit time.
  - Implemented in hardware to provide acceptable performance.

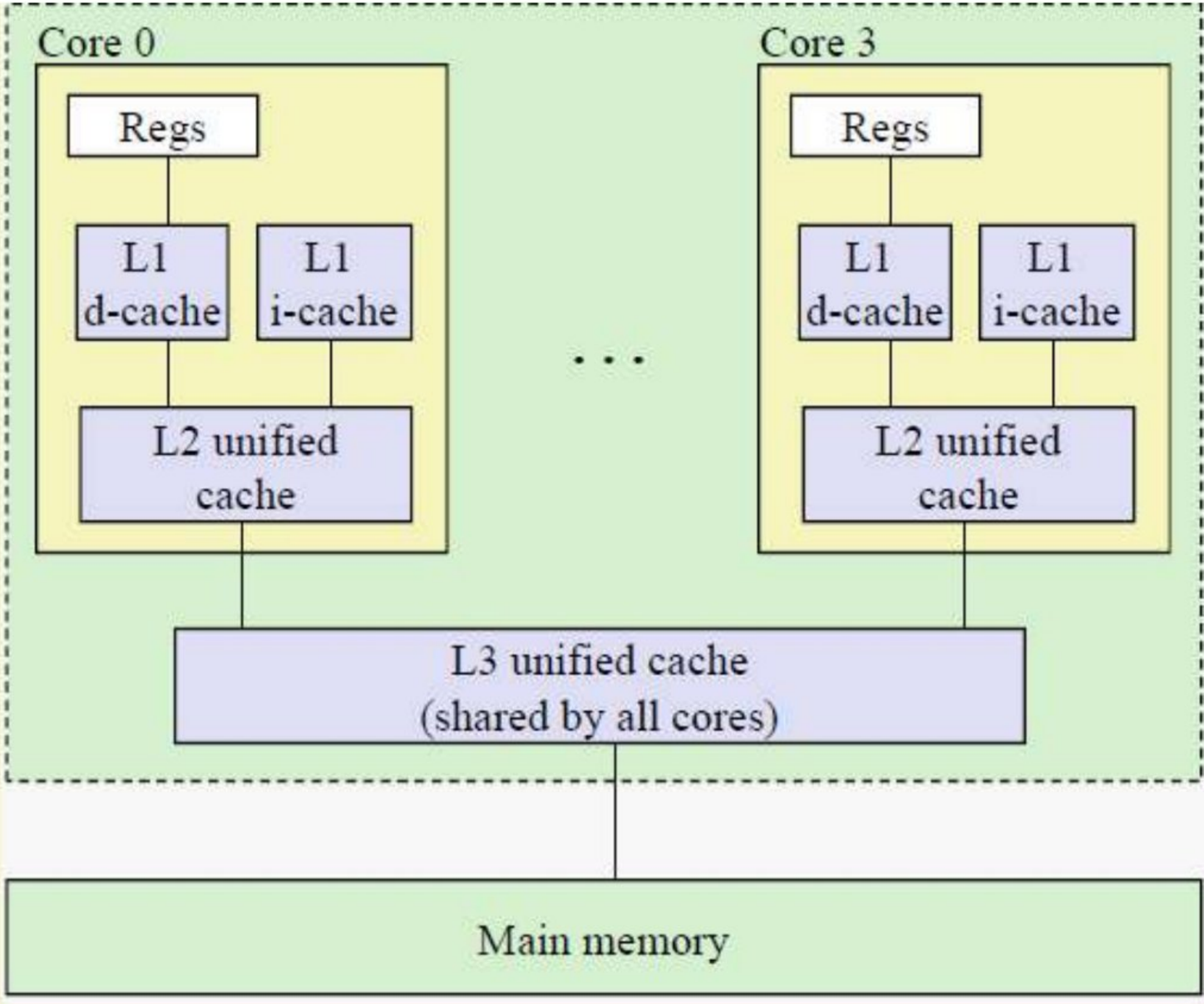
- Some mechanism is required to transfer blocks between consecutive levels.
  - If the block transfer requires 10's of clock cycles (like in cache / main memory hierarchy), it is controlled by hardware.
  - If the block transfer requires 1000's of clock cycles (like in main memory / secondary memory hierarchy), it can be controlled by software.
- Four main questions:
  1. *Block Placement*: Where to place a block in the upper level?
  2. *Block Identification*: How is a block found if present in the upper level?
  3. *Block Replacement*: Which block is to be replaced on a miss?
  4. *Write Strategy*: What happens on a write?

Common Memory Hierarchies

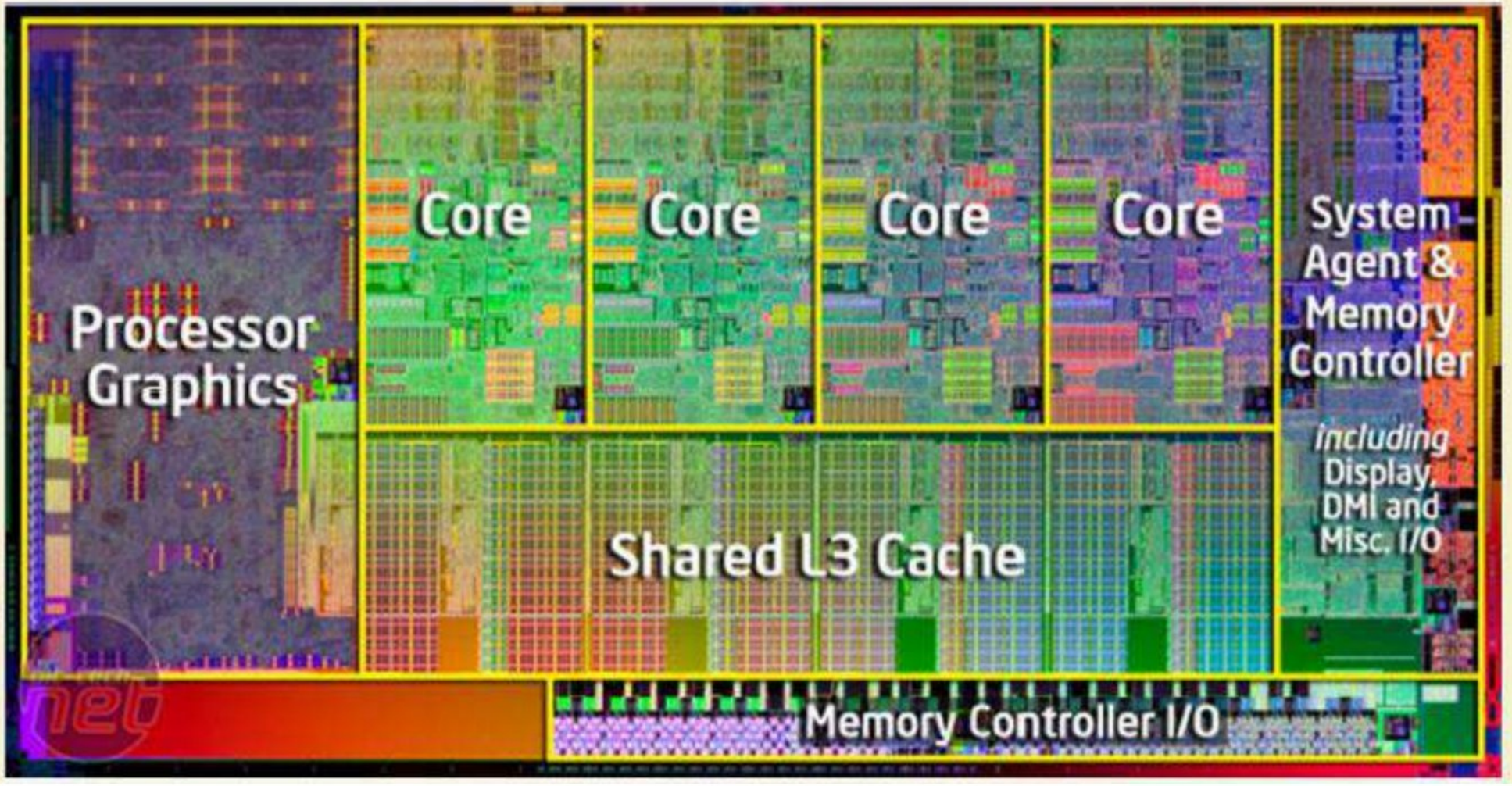
- In a typical computer system, the memory system is managed as two different hierarchies.
  - The Cache / Main Memory hierarchy, which consists of 2 to 4 levels and is managed by hardware.
    - Main objective: provide fast average memory access.
  - The Main Memory / Secondary Memory hierarchy, which consists of 2 levels and is managed by software (operating system).
    - Main objective: provide large memory space for users (virtual memory).

Intel Core-i7 Cache Hierarchy

- L1 i-cache and d-cache:
  - 32 KB, 8-way set associative
  - Access: 4 cycles
- L2 unified cache:
  - 256 KB, 8-way set associative
  - Access: 11 cycles
- L3 unified cache:
  - 8 MB, 16-way set associative
  - Access: 30-40 cycles
- Block size: 64 bytes for all caches



Core-i7 Sandybridge

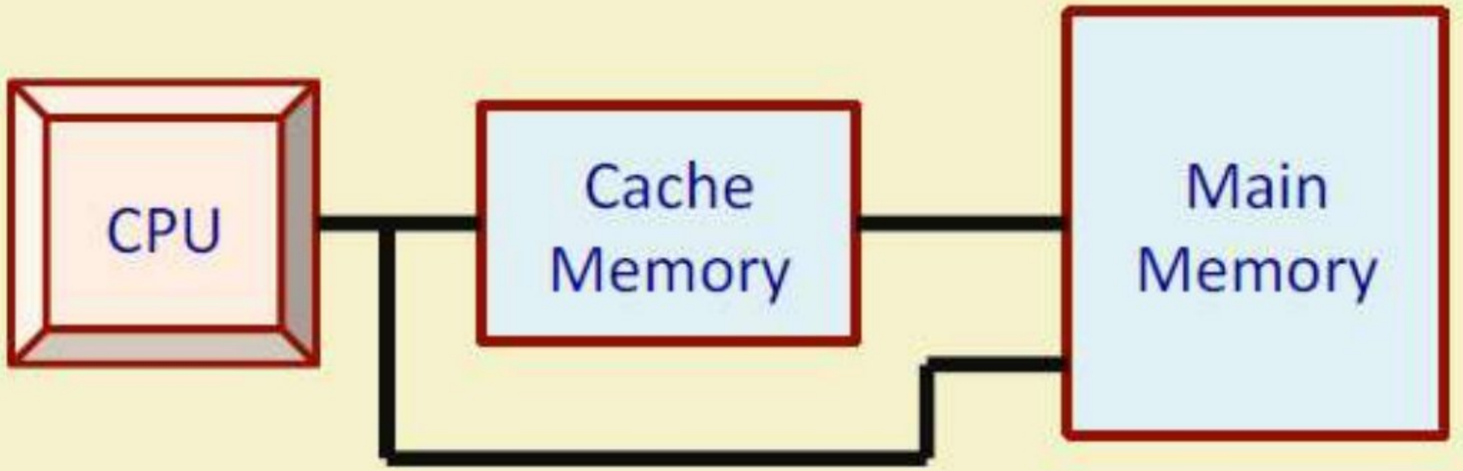


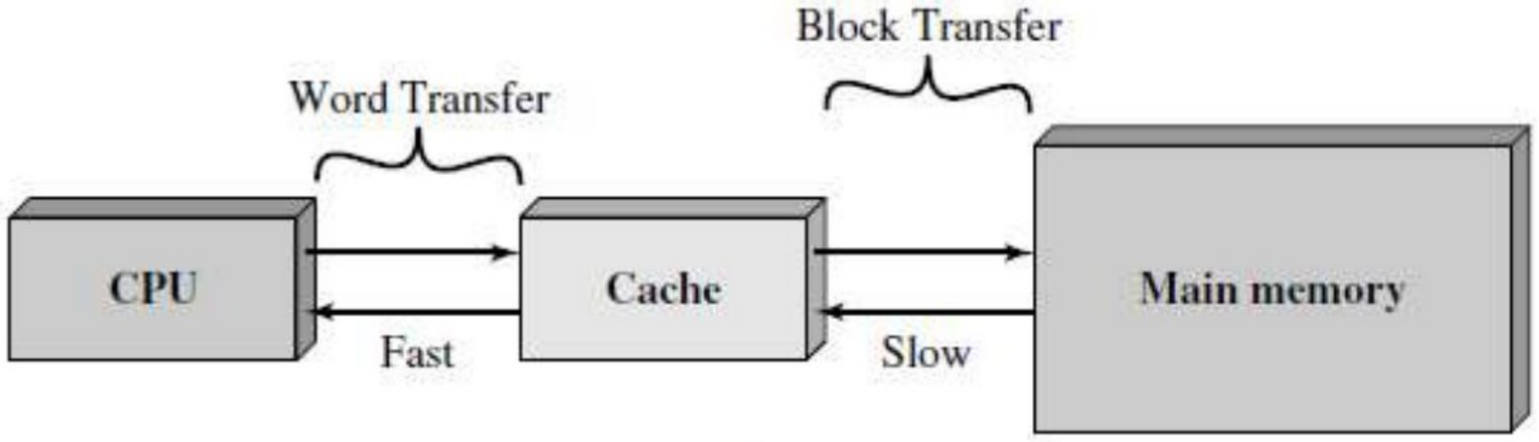
- Every core has its own L1 and L2 caches.
- The L3 cache is shared by all the cores and is also on-chip.

Cache Memory

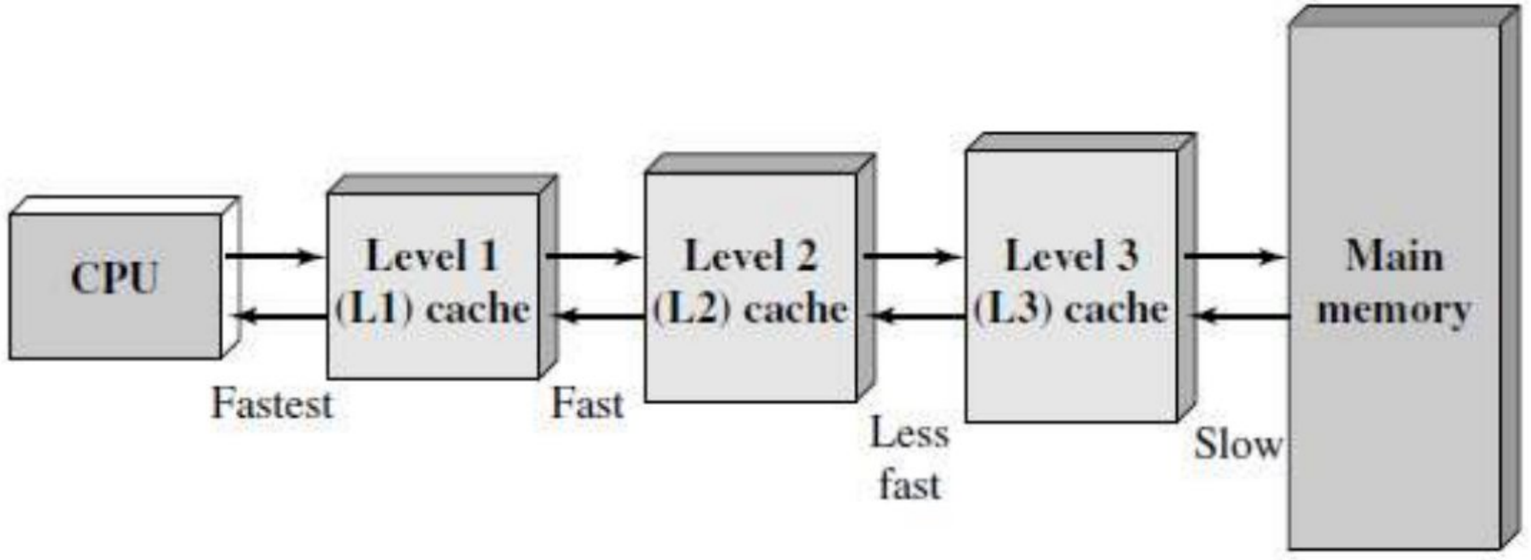
Introduction

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





(a) Single cache



(b) Three-level cache organization

Figure 4.3 Cache and Main Memory

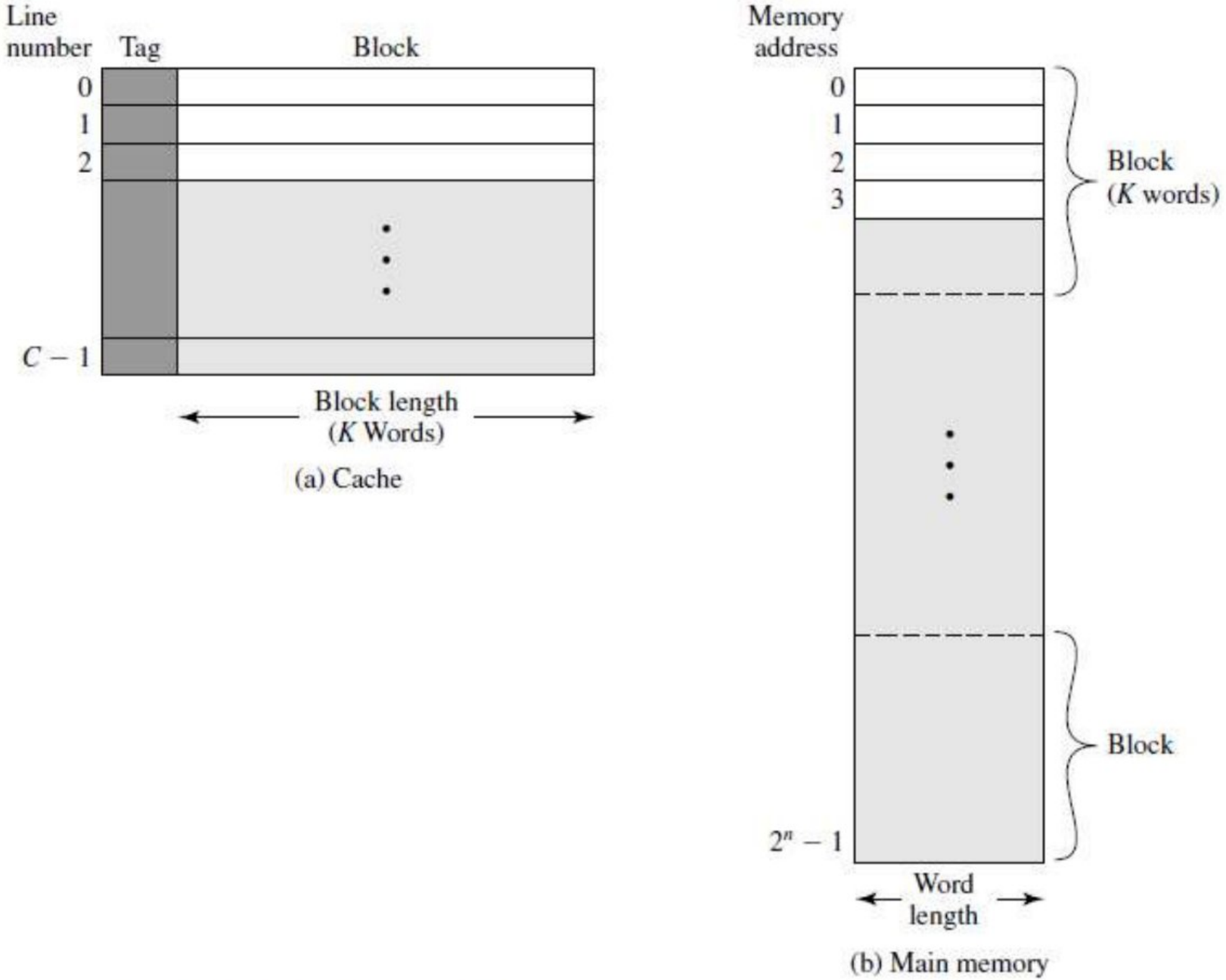


Figure 4.4 Cache/Main Memory Structure

- 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 main memory to cache.
  - For writes, sometimes the CPU can also directly write to main memory.
- 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 main memory blocks can reside in which cache memory blocks.
  - At any given time, only a small subset of the main memory blocks can be held in main memory.
- Three common block mapping techniques are used:
  a) Direct Mapping
  b) Associative Mapping
  c) (N-way) Set Associative Mapping
- The algorithms shall be explained with the help of an example.

What do mean by Line Entry?

→ present in Cache controller IC (IC no. 82385)

→ It is present is the form of Cache Directory

→ cache controller IC is interfaced with cache memory

No. of Line Entries depend on no. of lines in cache

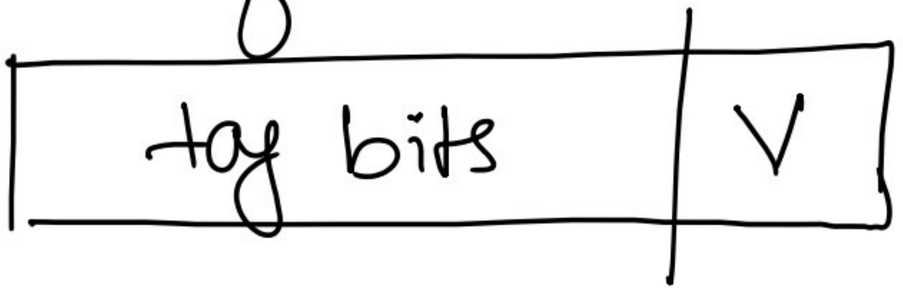
Consider cache memory = 64kB

Size of each line = 4 bytes.

No. of lines in cache memory = $\frac{64\,kB}{4}$ = 16kB

∴ No. of line Entries in = 16kB

Format of line entry.



Case 1

V = 0 invalid

No valid data is available

cache miss.

Case 2 V = 1 valid data is available

tag bit ≠ page bit

cache miss

Case 3

V = 1 valid data is available

tag bit == page bit

cache hit

Line entry is correspondiy to every line

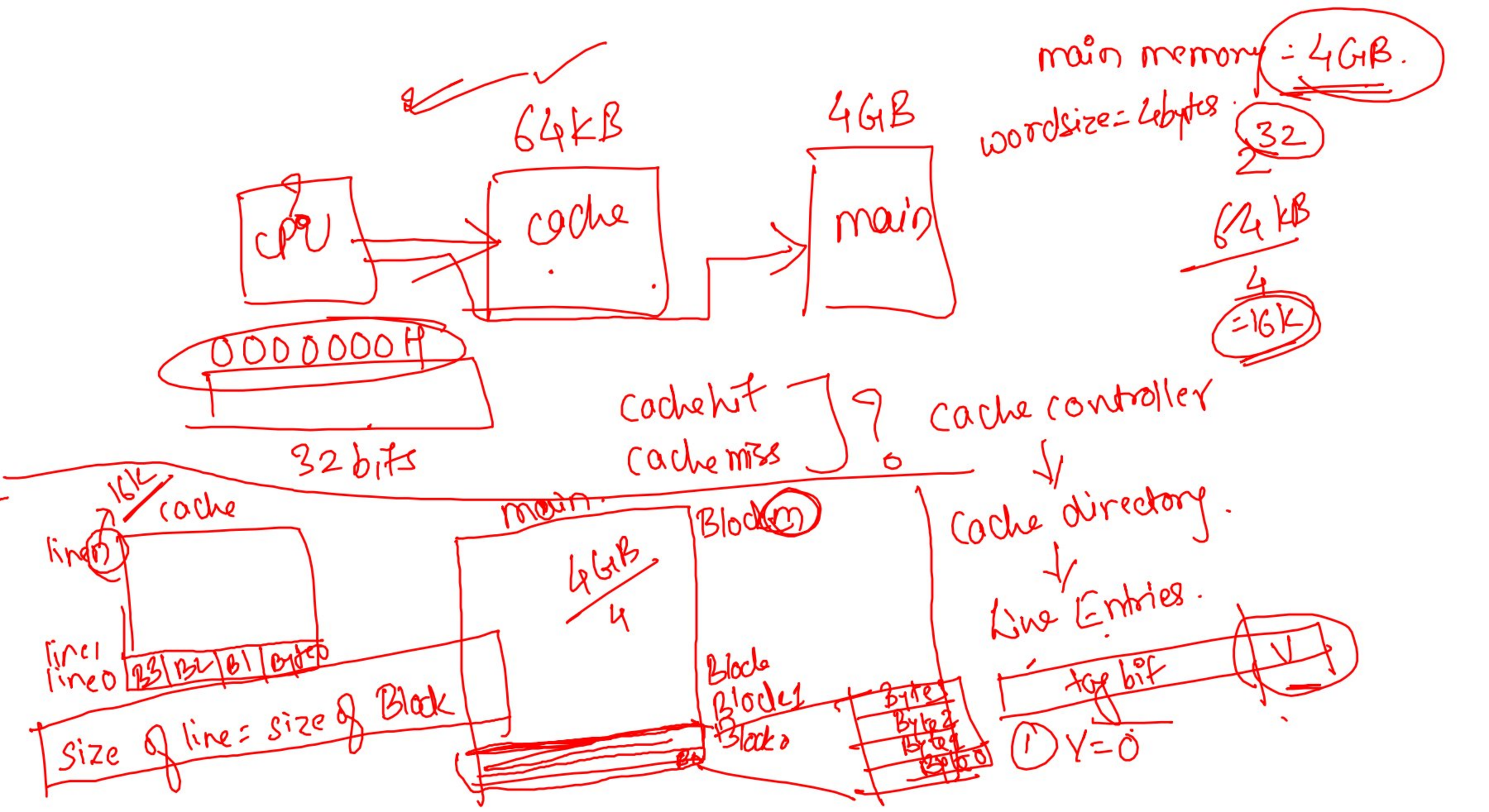
Memory required to store all the line entries =

Let say page bits = 16 bit
will same as tag bits

the size of each line entry = 16 bit + 1 bit (dirty)
= 17 bits

$17 \times 16K = 17 \times 16 \times 1024 =$

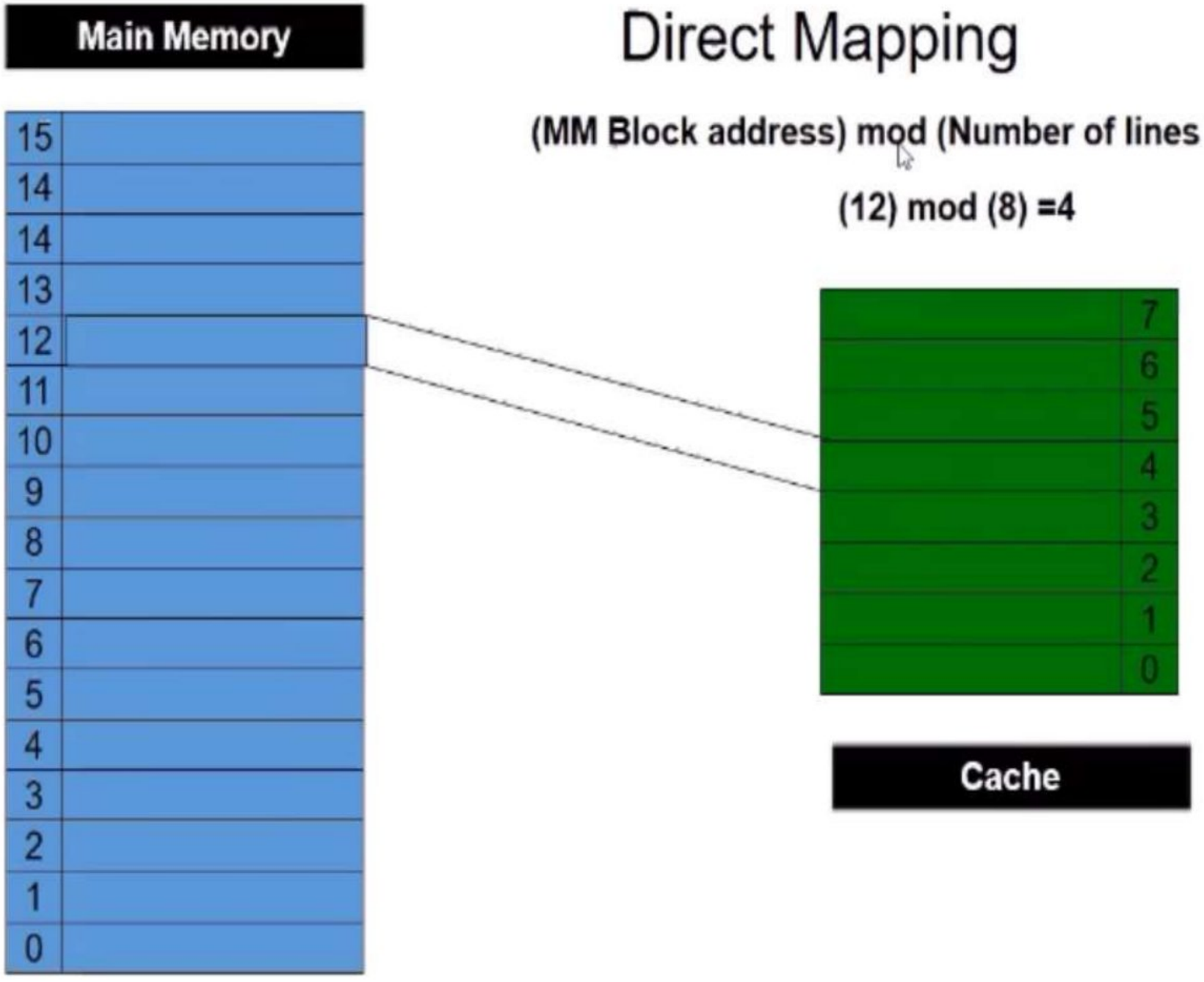
Memory taken by Line Entry


main memory = 4GiB.
wordsize = 2 bytes
32
2
64kB
4GiB
CPU
cache
main
64kB
4
=16K
00000000H
32 bits
cache hit
cache miss
0
cache controller
cache directory.
Line Entries.
16K
cache
line n
line 1
line 0
B3
B2
B1
B0
Size of line = size of Block
main.
4GiB
4
Block m
Block 2
Block 1
Block 0
Byte 3
Byte 2
Byte 1
Byte 0
tag bit
V
1
V=0

Direct Mapping

(MM Block address) mod (Number of lines in a cache)

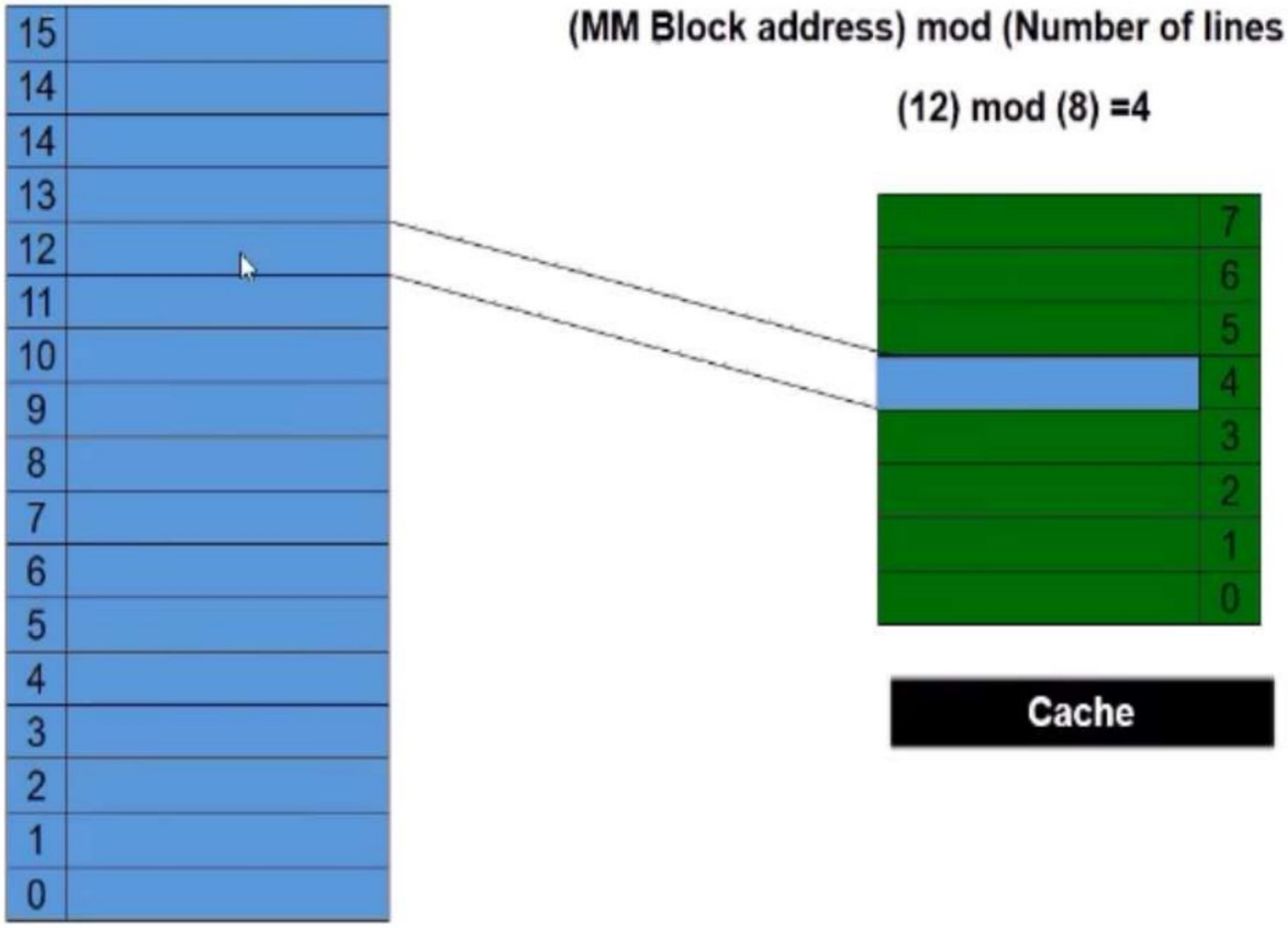
(12) mod (8) =4



Direct Mapping

(MM Block address) mod (Number of lines in a cache)

(12) mod (8) =4



(a) Direct Mapping

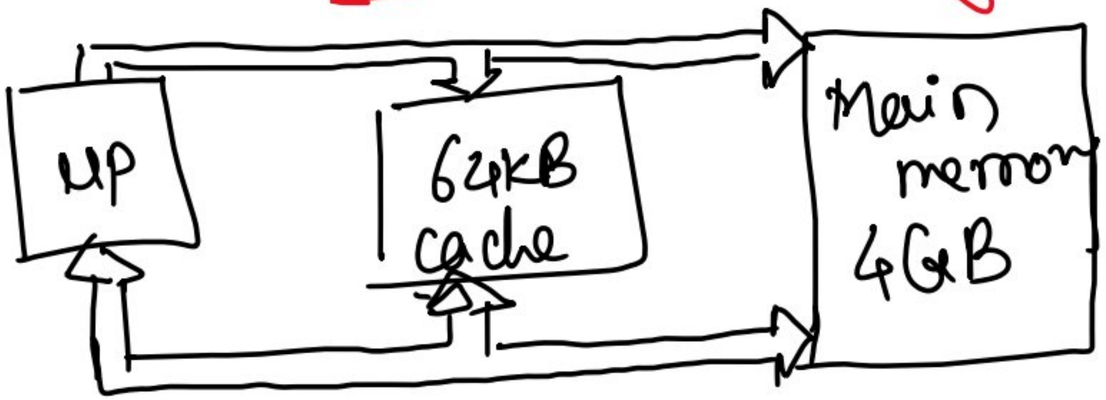
- Each main memory block can be placed in only one block in the cache.
- The mapping function is:

  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.

Direct Mapping



Given: Main memory = 4GB
cache memory = 64kB
Size of Block = 32 bits (4 bytes)

Size of line = Size of Block.

Blocks are vertically stored and lines are horizontally stored

Answer the following

(a) Give the address interpretation done by main memory

(b) Give address interpretation by cache memory

Step ① No. of address lines

$4GB = 2^x$

$x = 32$

$A_0 - A_{31}$

Step ② Find no. of Blocks in main memory

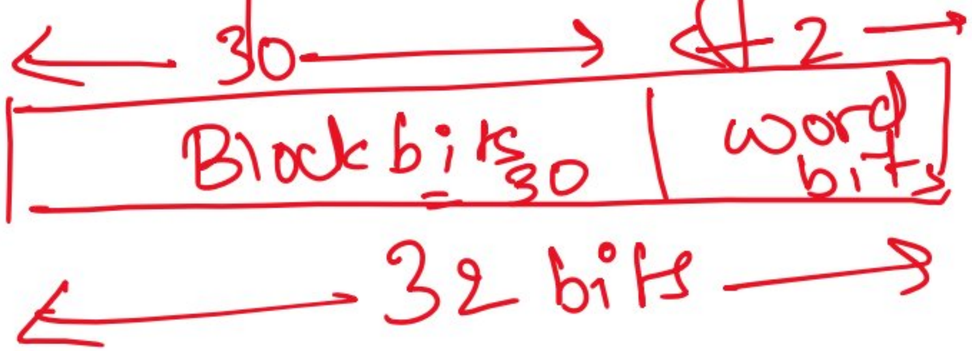
$$\frac{4\,GB}{4\,bytes} = 1\,GB.$$

Step ③ Find no. of address lines uniquely identifying each block

$1GB = 2^x$

$x = \underline{30}$

(a) Address interpretation by main memory



Size of word = 4 byte

No. of address line for identifying 4 bytes.

$4 = 2^x$

$x = 2$

Step 4: Find no. of addressline required from cache memory

$$64kB = 2^x$$

$$x = 16$$

Step 5: Find no. of lines in cache memory

$$\frac{64kB}{4} = 16kB = 16384$$

Step 6: No. of address lines to identify each line

$$16K = 2^x$$

$$x = 14$$

Step 7 Find page size in main memory using Direct mapped cache organization

page size of main memory = size of cache memory

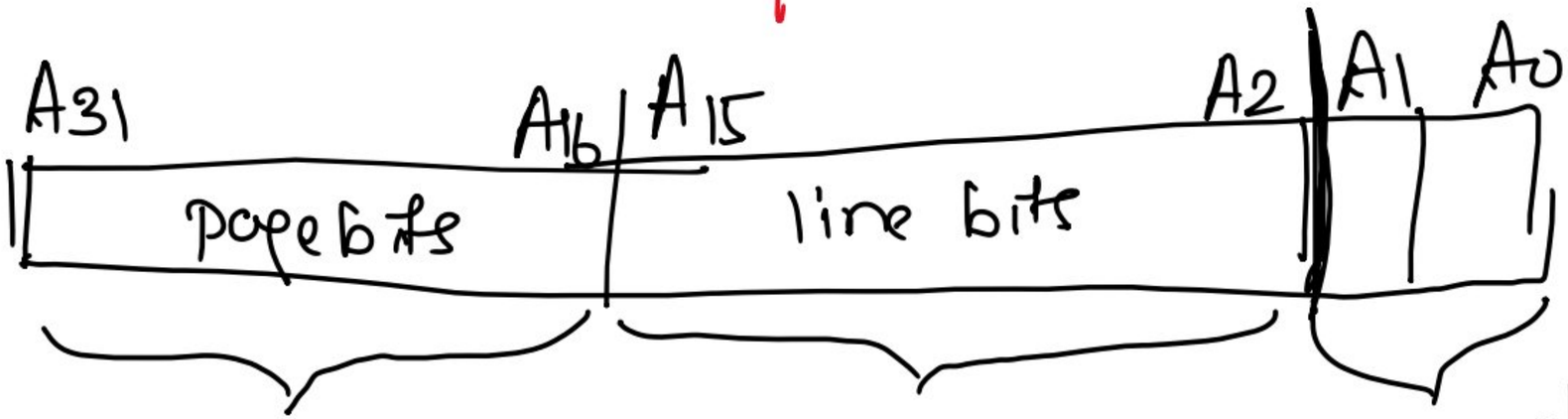
= 64 kB

Step 8 Find no. of pages in main memory = $\frac{4\,GB}{64\,kB}$

= 64k

= 65536

(b) Address interpretation by cache memory



16 bits
page number

14 bits line
Number

word bits
= 2 bits

Conceptual Diagram for Direct mapping

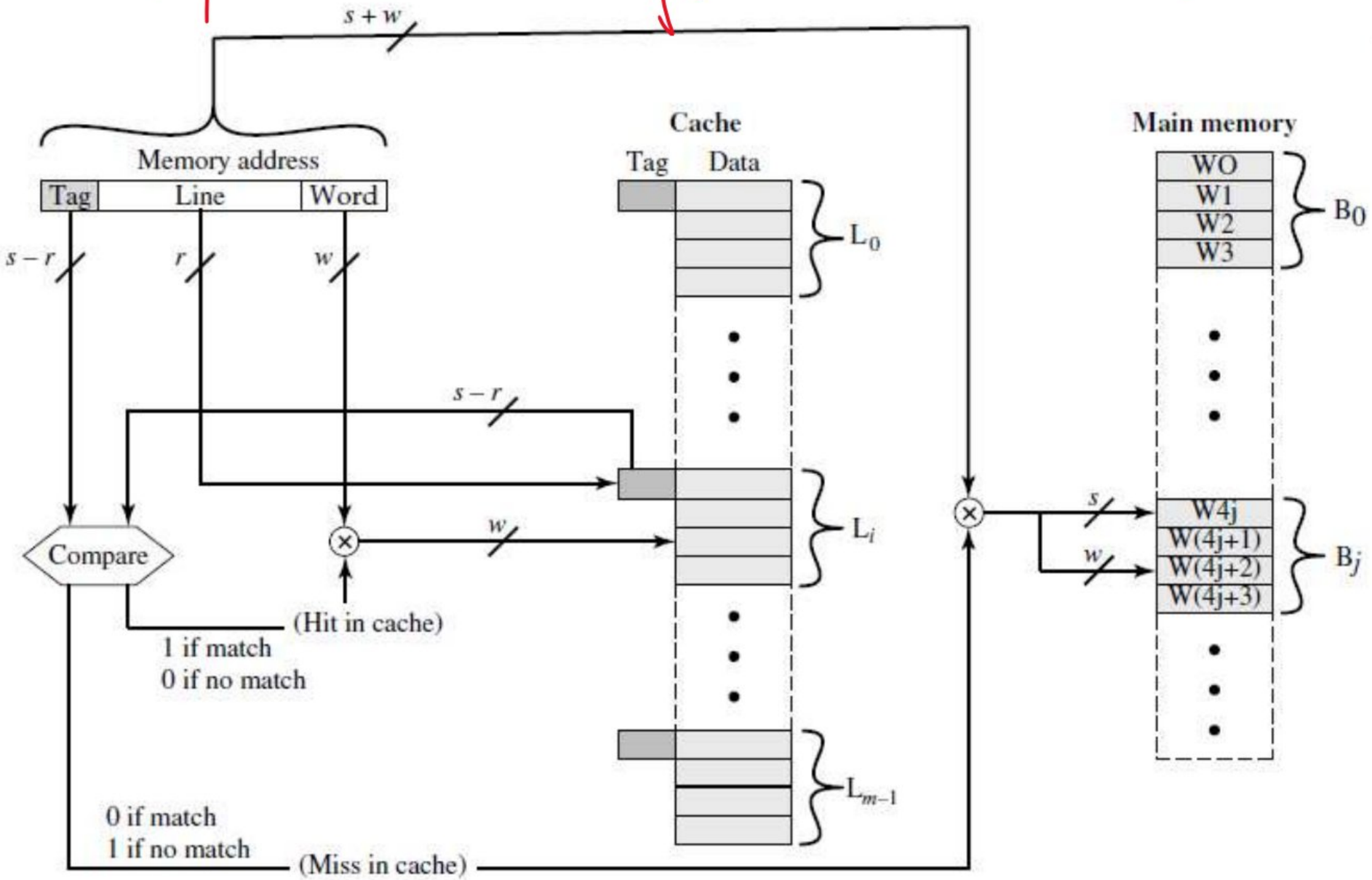
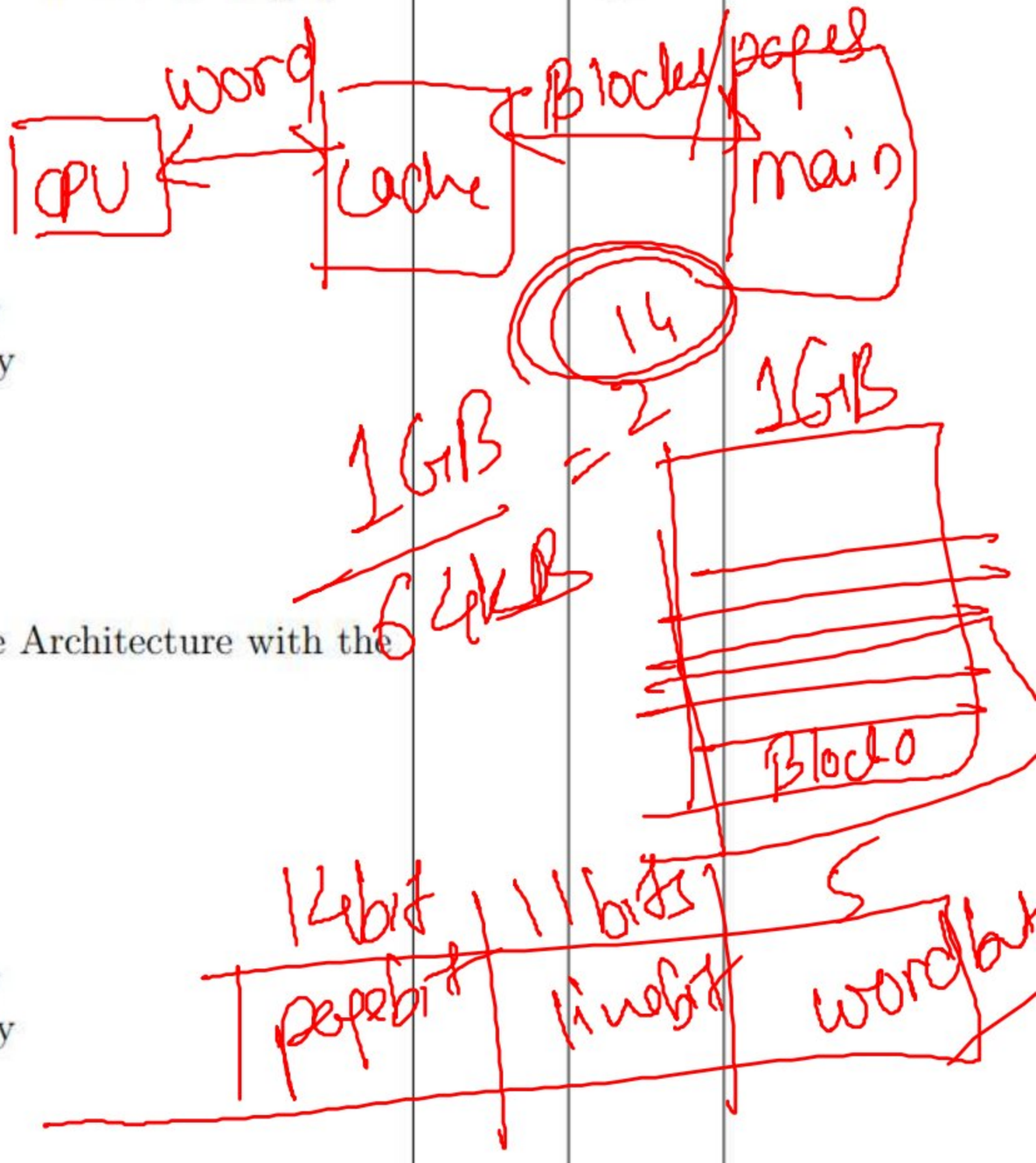


Figure 4.9 Direct-Mapping Cache Organization

Q.4 (a)	Design Direct Mapped Cache Architecture with the following specifications: a. Main Memory Size of 1GB b. Block Size of 32 Bytes c Cache Memory Size of 64 KB d. Line Size of 32 Bytes Answer the following: 1) Address Interpretation by Main Memory 2) Address Interpretation by Cache Memory 3) Design of Line Entry 4) Conceptual Diagram of the System OR Design Two way associative Mapped Cache Architecture with the following specifications: a. Main Memory Size of 1GB b. Block Size of 32 Bytes c Cache Memory Size of 64 KB d. Line Size of 32 Bytes Answer the following: 1) Address Interpretation by Main Memory 2) Address Interpretation by Cache Memory 3) Design of Line Entry 4) Conceptual Diagram of the System	08	3,4-3



(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.

Two Way Set Associative Mapping

Given Main memory = 4 GB
Cache memory = 64 kB
Block size = 4 bytes

Answer the following using 2-way set Associative mapping

(a) Address interpretation by main memory

(b) Address interpretation by cache memory

Step ① No. of address lines

$4GB = 2^x$

$x = 32$

$A_0 - A_{31}$

Step ② Find no. of Blocks in main memory

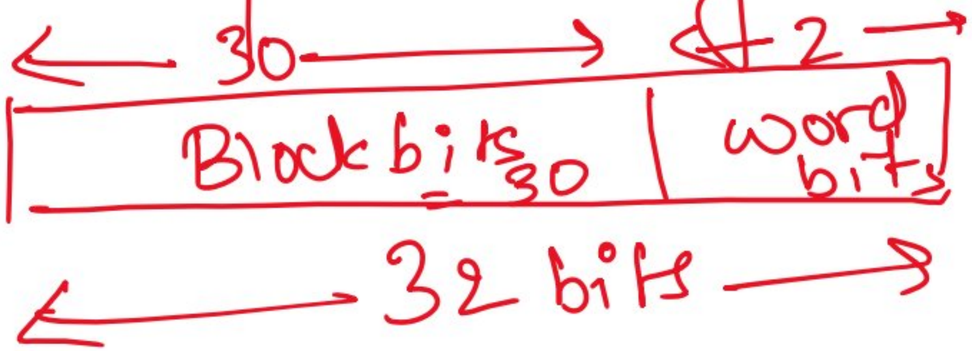
$$\frac{4\,GB}{4\,bytes} = 1\,GB.$$

Step ③ Find no. of address lines uniquely identifying each block

$1GB = 2^x$

$x = \underline{30}$

(a) Address interpretation by main memory



Size of word = 4 byte

No. of address line for identifying 4 bytes.

$4 = 2^x$

$x = 2$

Step4 : 2-way Set Associative memory

↳ 2 Banks of Cache memory

Total cache memory = 64kB

Size of each bank = 32kB

Step5 : No. of lines in cache memory bank

$$\frac{32\ kB}{4} = 8kB$$

Step6 : No. of address lines uniquely identify each line

$$8k = 2^x$$

$x = 13$ ← line bits.

Step 7 Page size in main memory = Size of cache memory bank

= 32kB

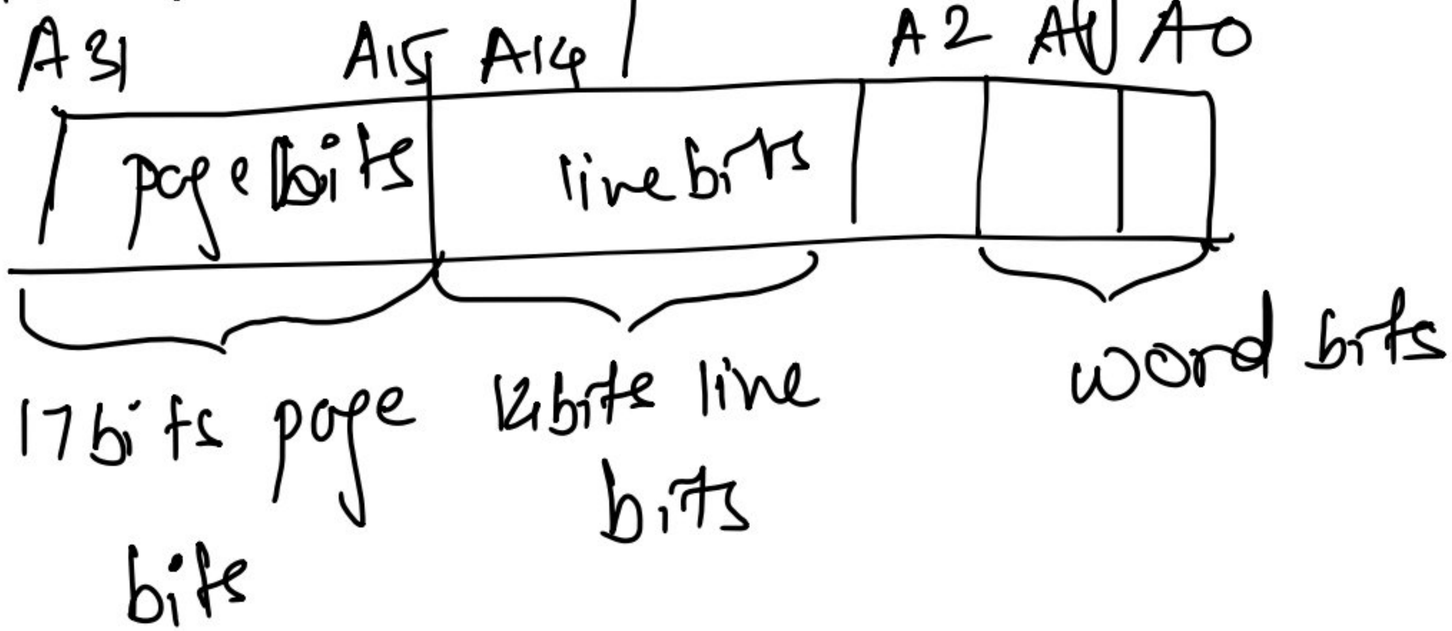
Step 8 No. of pages in main memory = $\frac{4\,GB}{32\,kB}$ = 128 k

Step 9 Address line uniquely identifying page

$128 = 2^x$

$\boxed{x = 17}$ → tag bits

(b) Address interpretation by cache memory



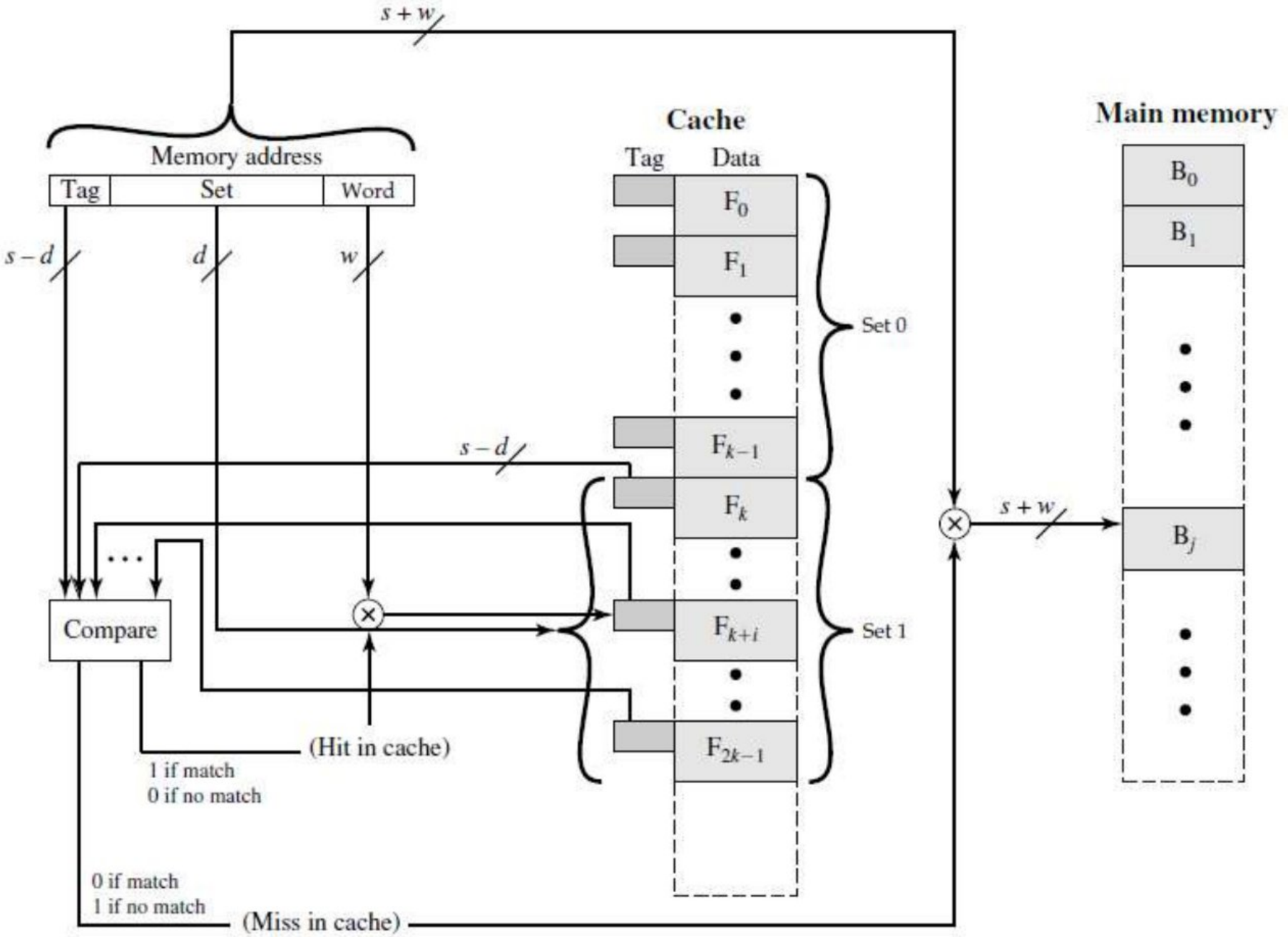
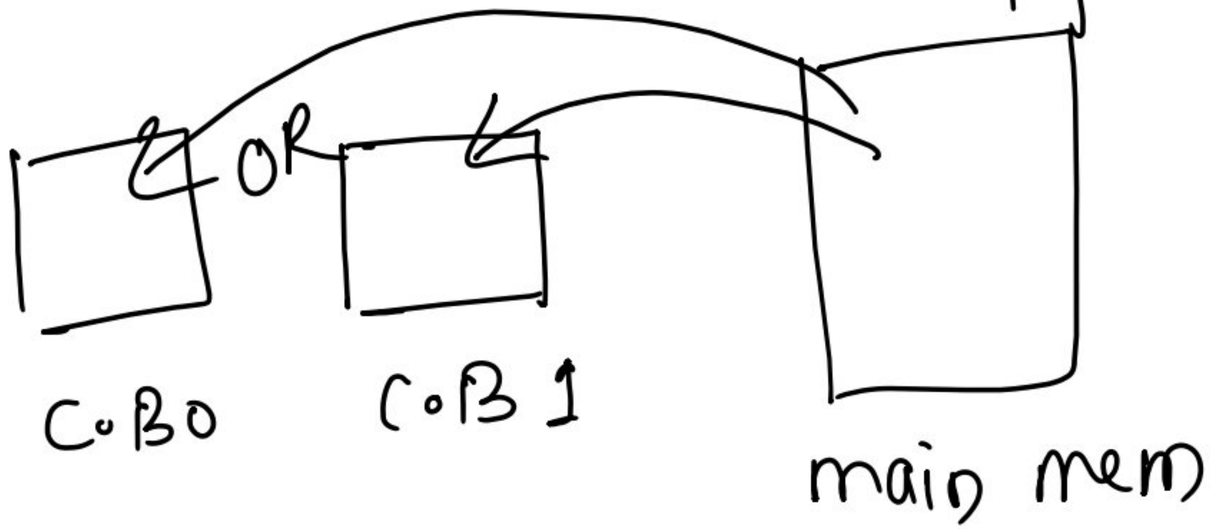


Figure 4.14 *K*-Way Set Associative Cache Organization

Limitations of 2-way Set Associative Mapping

→ When there is cache miss. page will be brought in which memory bank?



→ To solve this problem Least Recently used (LRU) is implemented in hardware.

→ The memory bank which is LRU is to be replaced.

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.

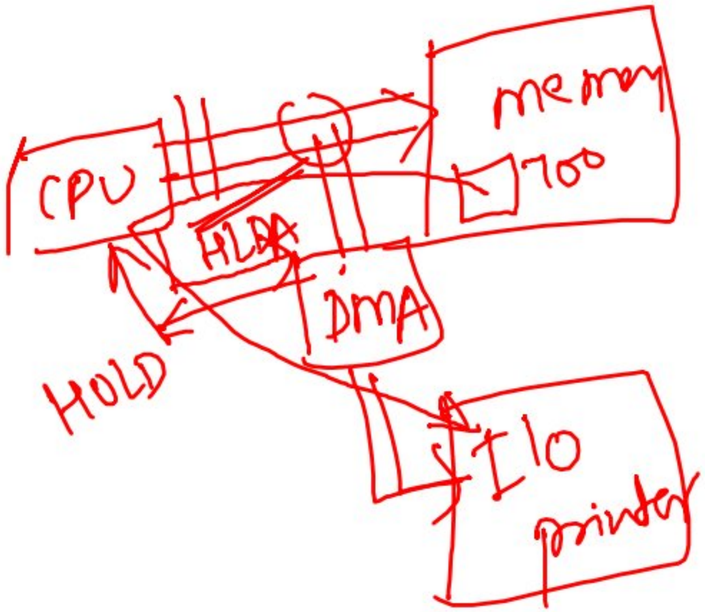
Q3. Which block should be replaced on a cache miss?

- 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.
- Example: Consider a 4-way set associative cache.
  - For tracking the LRU block within a set, we use a 2-bit counter with every block.
  - When hit occurs:
    - Counter of the referenced block is reset to 0.
    - Counters with values originally lower than the referenced one are incremented by 1, and all others remain unchanged.
  - When miss occurs:
    - If the set is not full, the counter associated with the new block loaded is set to 0, and all other counters are incremented by 1.
    - If the set is full, the block with counter value 3 is removed, the new block put in its place, and the counter set to 0. The other three counters are incremented by 1.

Updating policies of Cache

- Write Through
- Write Back
- Write Around



Write Through

- Cache copy and main memory copy updated simultaneously
- Advantage: Main memory always contain the same data as cache
- It is important during DMA transfers to ensure data in main memory is valid
- Disadvantage: slow due to memory access

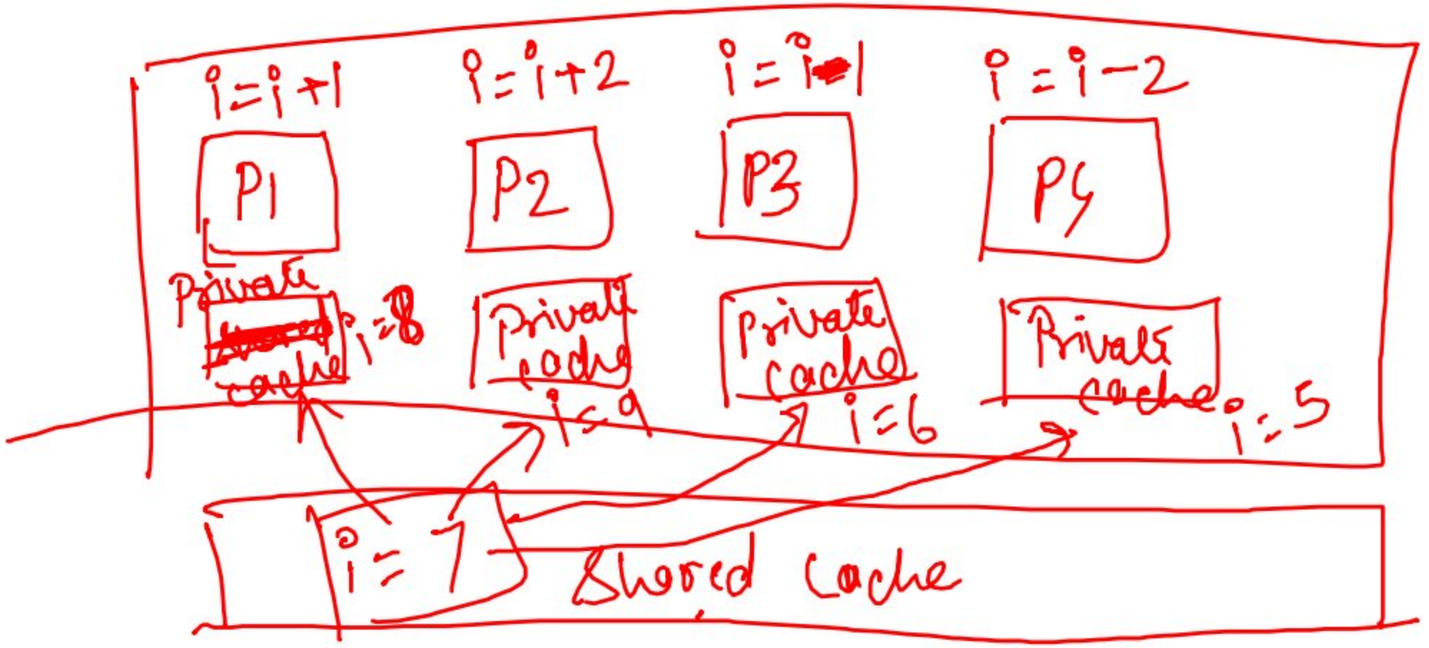
Write Back

- Only cache is updated during write and marked as a flag (“modified”)
- When word is removed from cache it is copied into main memory
- Main memory is not updated. Same item in cache and main memory may have different value.

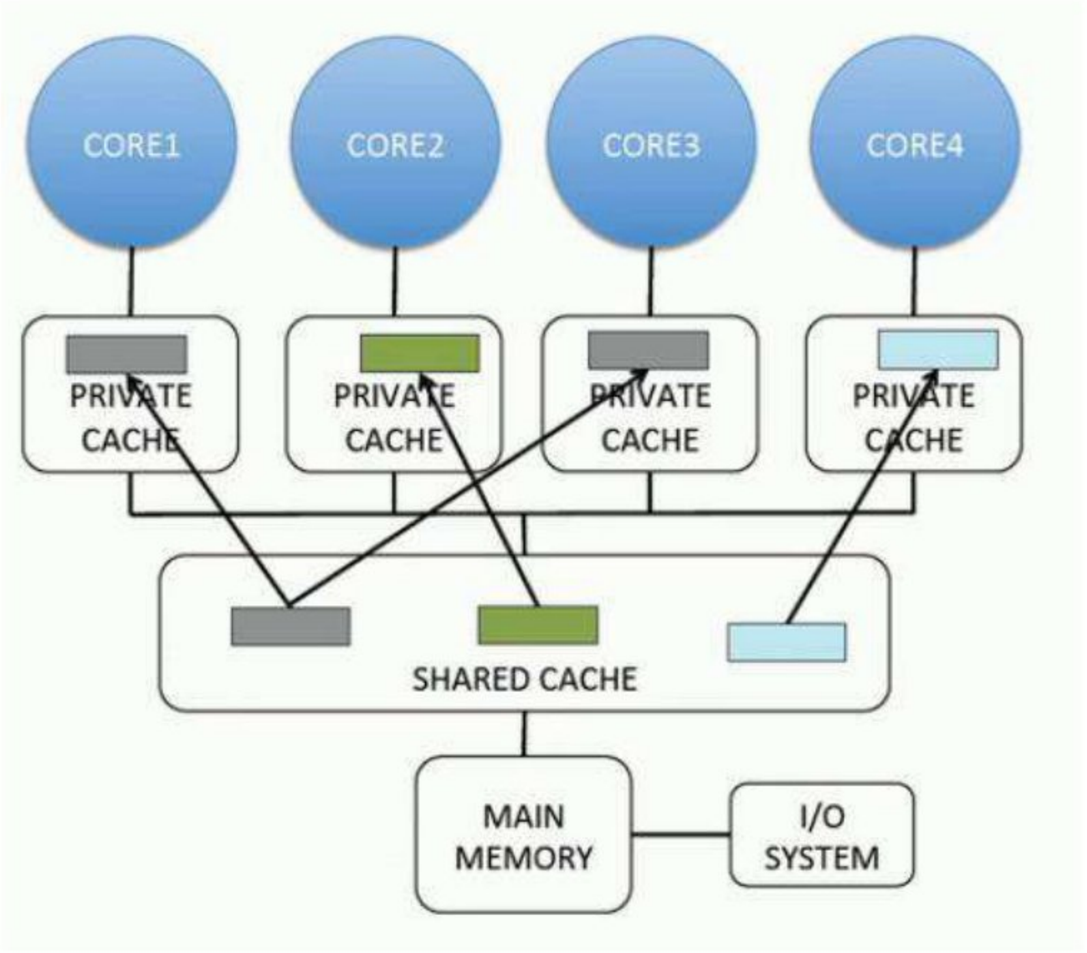
Write Around

- The data is updated in main memory only.
- Data in cache is not updated.

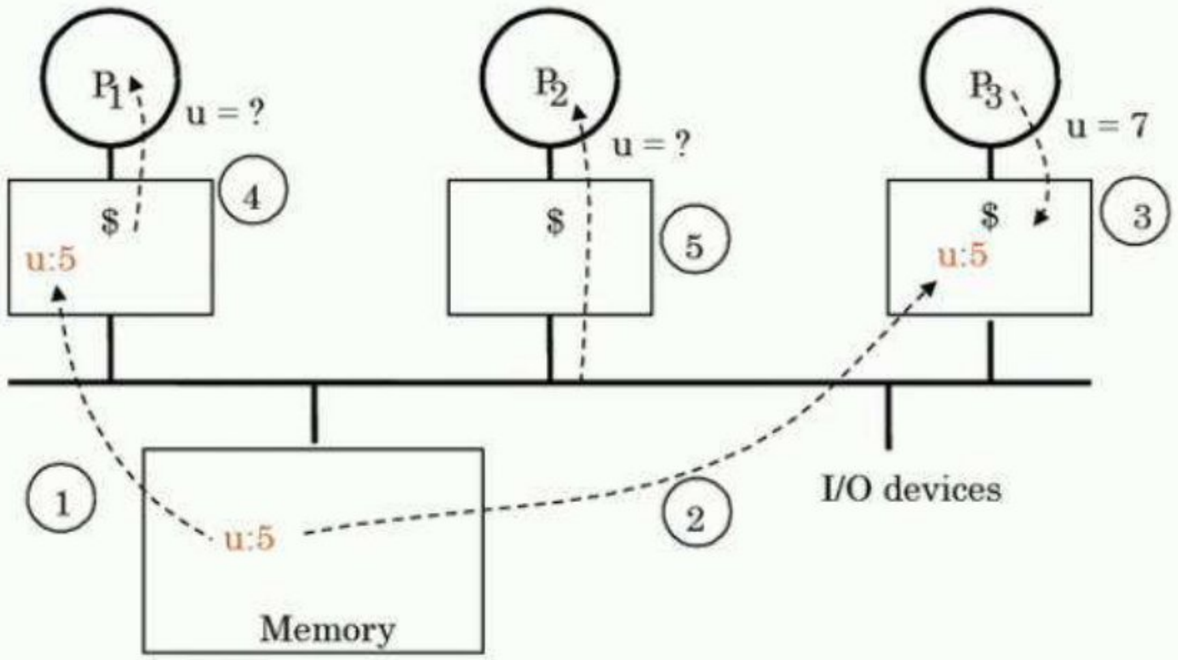
Write Back



Cache Coherency



The Cache Coherence Problem



- Processors will see different value for u after event 3
- Write-through caches: P_1 reads a stale copy
- Write-back caches: P_1 and P_2 read a stale copy

There are various Cache Coherence Protocols in multiprocessor system.

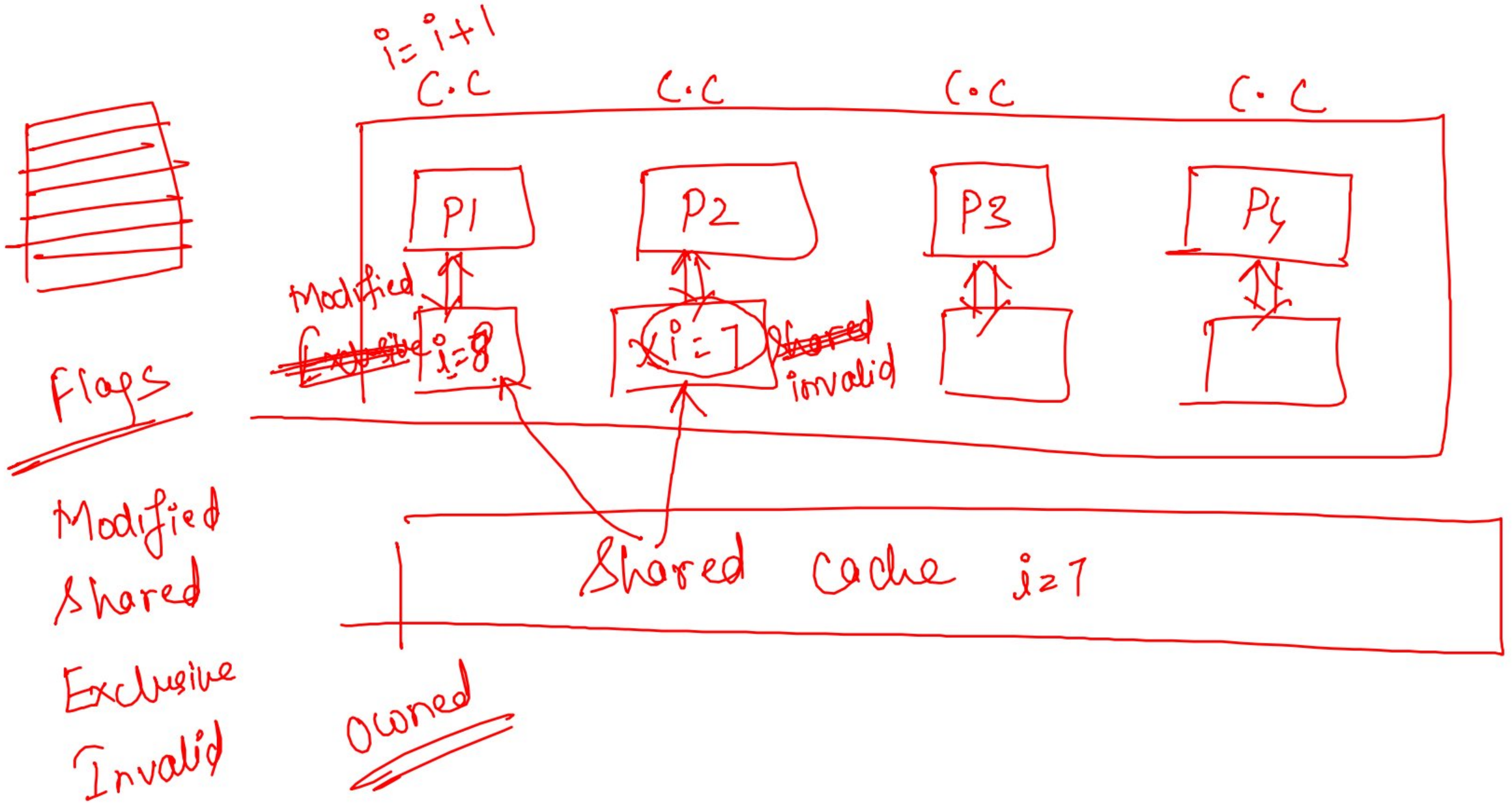
These are :-

1. MSI protocol (Modified, Shared, Invalid)
2. MOSI protocol (Modified, Owned, Shared, Invalid)
3. MESI protocol (Modified, Exclusive, Shared, Invalid)
4. MOESI protocol (Modified, Owned, Exclusive, Shared, Invalid)

cache

These important terms are discussed as follows:

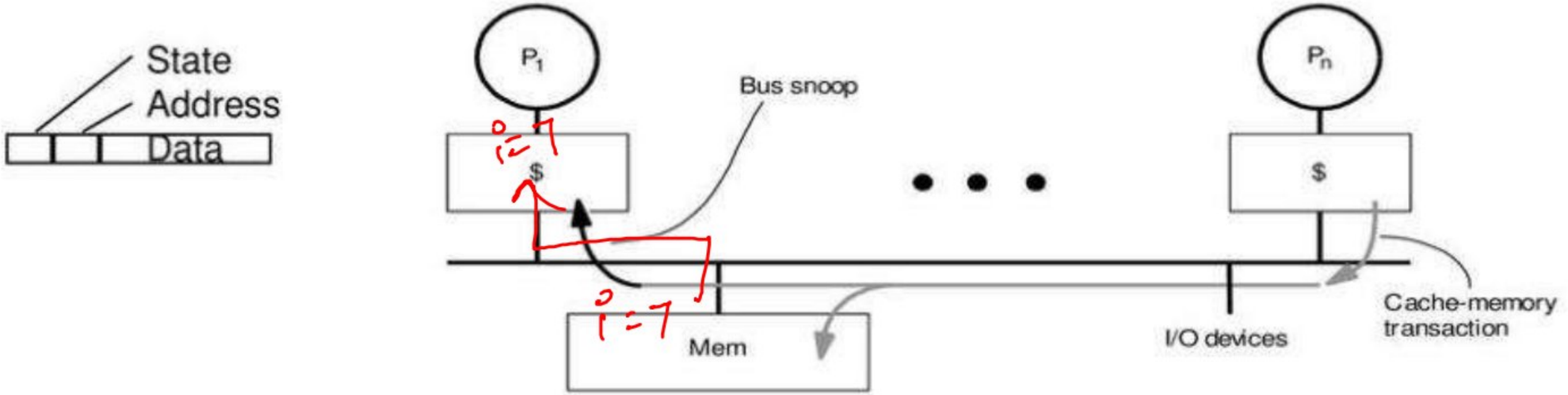
- **Modified –** It means that the value in the cache is dirty, that is the value in current cache is different from the main memory.
- **Exclusive –** It means that the value present in the cache is same as that present in the main memory, that is the value is clean.
- **Shared –** It means that the cache value holds the most recent data copy and that is what shared among all the cache and main memory as well.
- **Owned –** It means that the current cache holds the block and is now the owner of that block, that is having all rights on that particular blocks.
- **Invalid –** This states that the current cache block itself is invalid and is required to be fetched from other cache or main memory.


i = i+1
C.C
C.C
C.C
C.C
P1
P2
P3
P4
Modified
Exclusive
i=8
i=7
Shared
invalid
Flags
Modified
Shared
Exclusive
Invalid
Owned
Shared cache i=7

Two Classes of Cache Coherence Protocols

1. Directory based — Sharing status of a block of physical memory is kept in just one location, the directory
2. Snooping — Every cache with a copy of data also has a copy of sharing status of block, but no centralized state is kept
   - All caches are accessible via some broadcast medium (a bus or switch)
   - All cache controllers monitor or snoop on the medium to determine whether or not they have a copy of a block that is requested on a bus or switch access

Snoopy Cache-Coherence Protocols



- Cache Controller "snoops" all transactions on the shared medium (bus or switch)
  - take action to ensure coherence
    - » invalidate, update, or supply value
  - depends on state of the block and the protocol
- Either get exclusive access before write via write invalidate or update all copies on write

Page Replacement Policies

- When a new page is brought cache, one of the existing page must be replaced by new block.

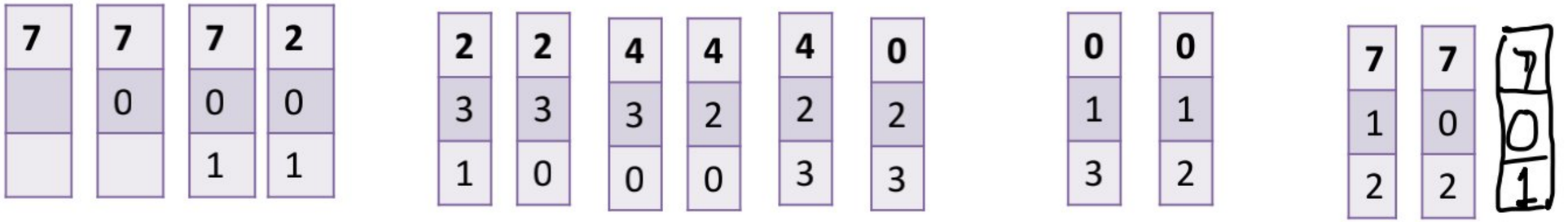
Algorithms:

- First In First Out (FIFO)
- Optimal Page Replacement
- Least Recently used (LRU)

First In First Out (FIFO)

- When a page is replaced oldest page is chosen
- Reference string (page frame)

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

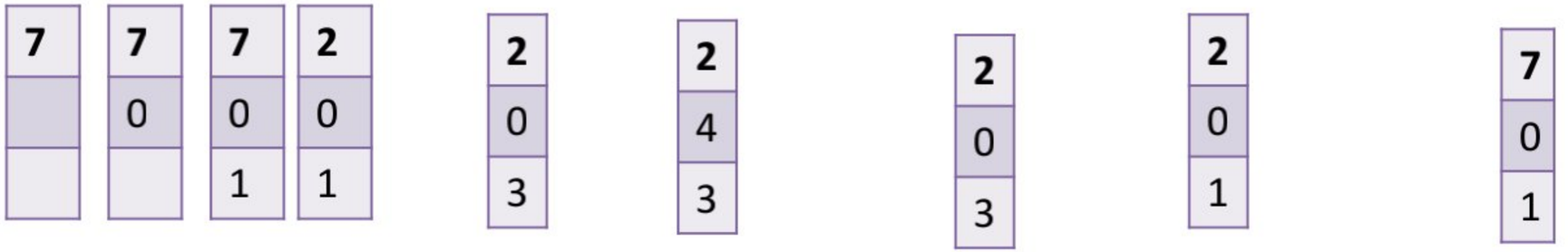


page hit Ratio = $\frac{\text{No. of page hits}}{\text{No. of pages}} = \frac{5}{20}$

Optimum Page Replacement Algorithm

- Reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

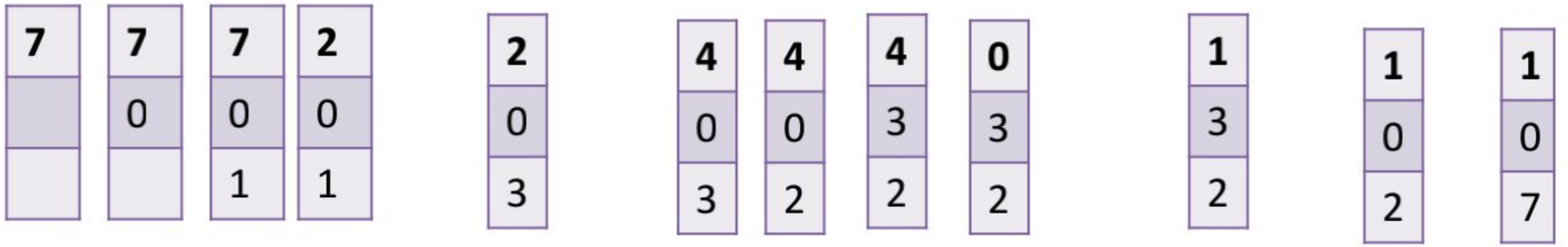


page hit ratio = $\frac{11}{20}$

Least Recently used (LRU)

- Reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

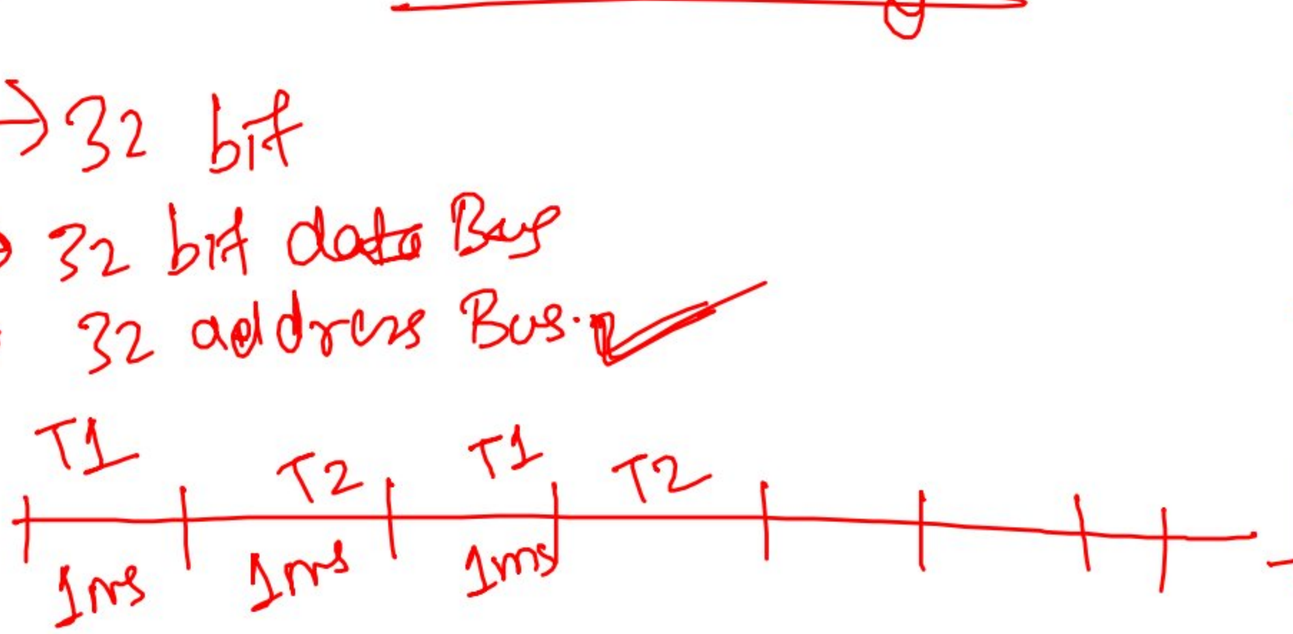


page hit ratio = $\frac{8}{20}$

80386

Multitasking

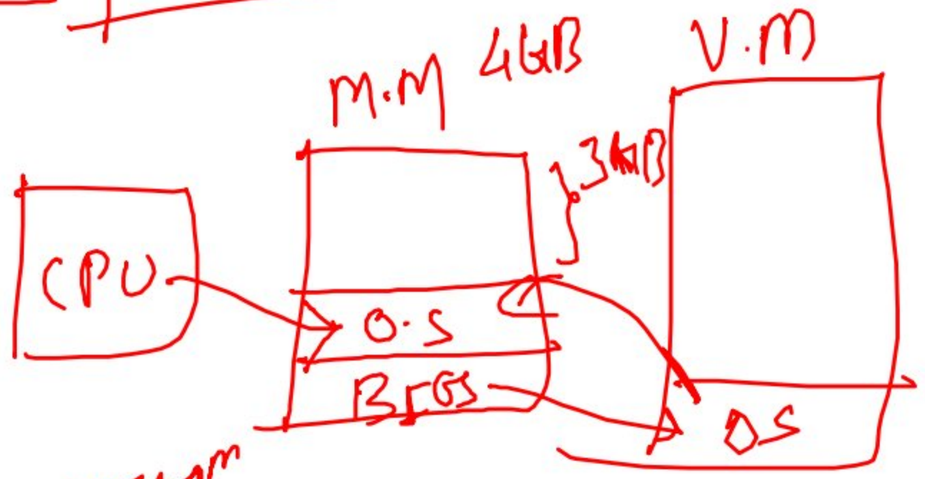
- 32 bit
- 32 bit data Bus
- 32 address Bus.



3GHz

$\phi = \frac{1}{3 \times 10^{9}}$

Time Slice = 1ms.



Task

- Global
  - BIOS
  - Linker
  - Loader
  - Debug
- Local
  - User program

80386

Segments

1. Starting address of segment
2. Size of segment
3. Type
4. protection level → privilege → Read / write
   → Non privileged → Read

Global → Code → CS
Local → Data → DS, ES, FS, GS
Stack → SS

Descriptors

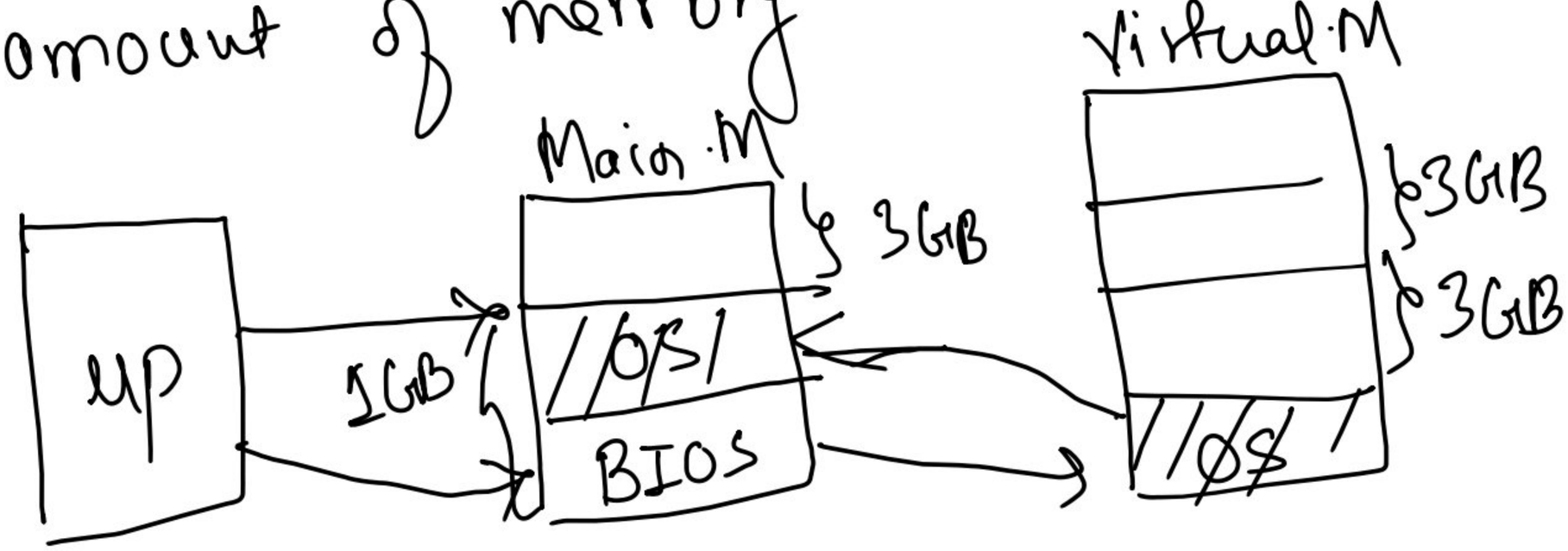
Global → G·D·T
Local → L·D·T

Global → BIOS → 100 segments → 100 descriptors

Virtual memory

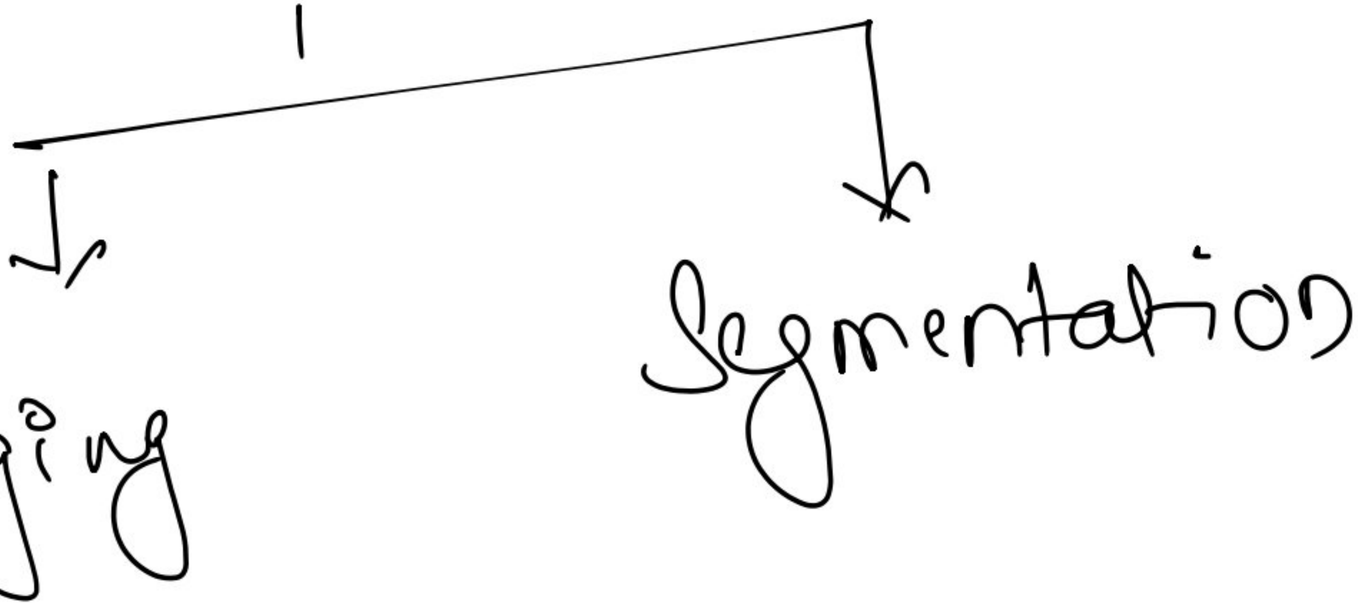
→ It is just a concept, not existing

→ Illusion created to programmer up to access large amount of memory



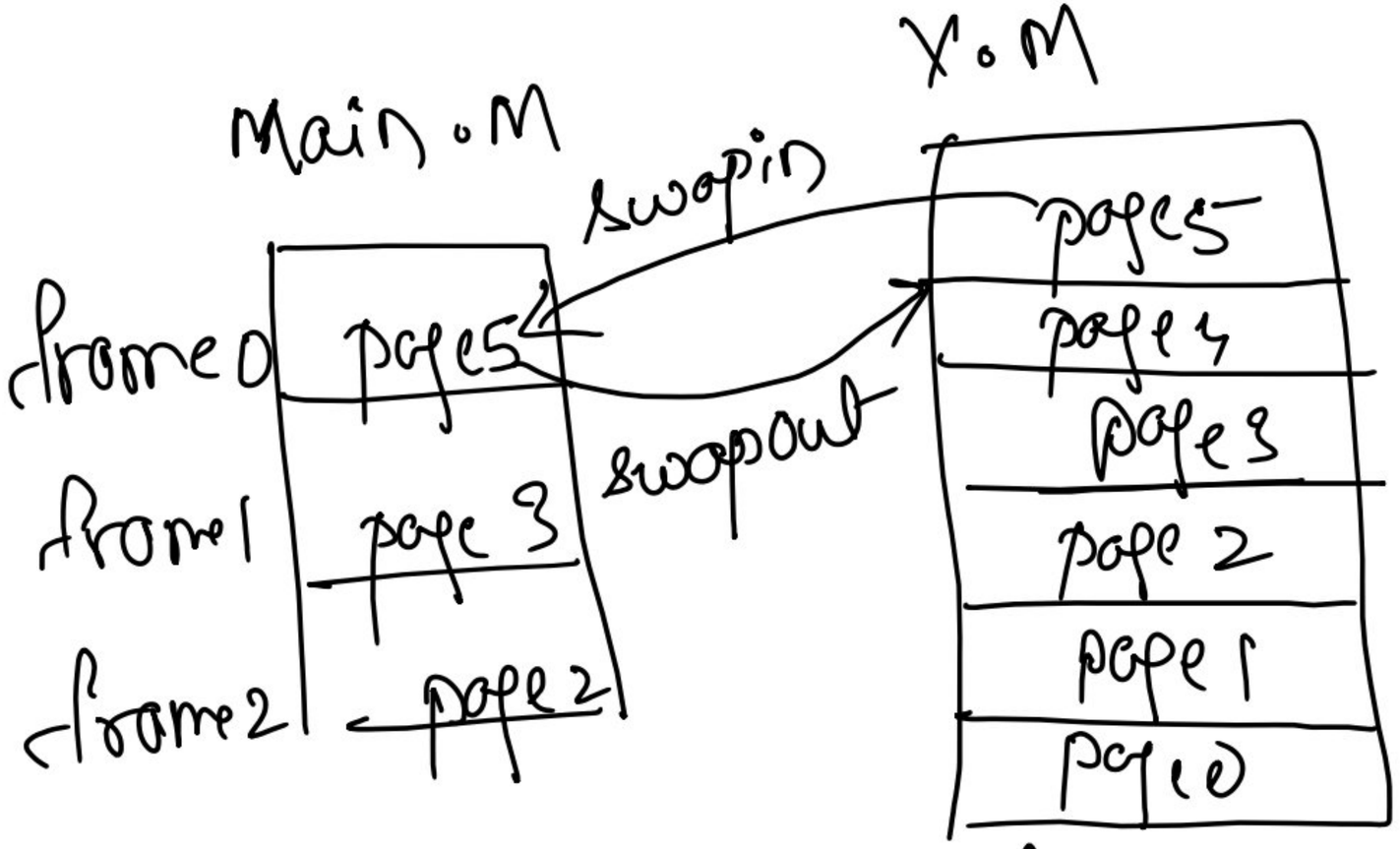
Virtual memory can be accessed

by 2 ways



Demand Paging

- To load a program from disk into memory we can either load the entire program in physical memory at execution time, or load the part of program which is needed to perform the task.
- However, when a program is entirely loaded into memory and we may initially not needed the whole code to be executed then it will be waste of memory use.
- With demand paging the page is loaded into physical memory only when they are needed.
- A demand-paging system is similar to a paging system with swapping where processes resides in secondary memory and when we want to execute a process, we swap in into memory.



O.S performes swapping operation M.M.U (memory management unit) does.

and update page table

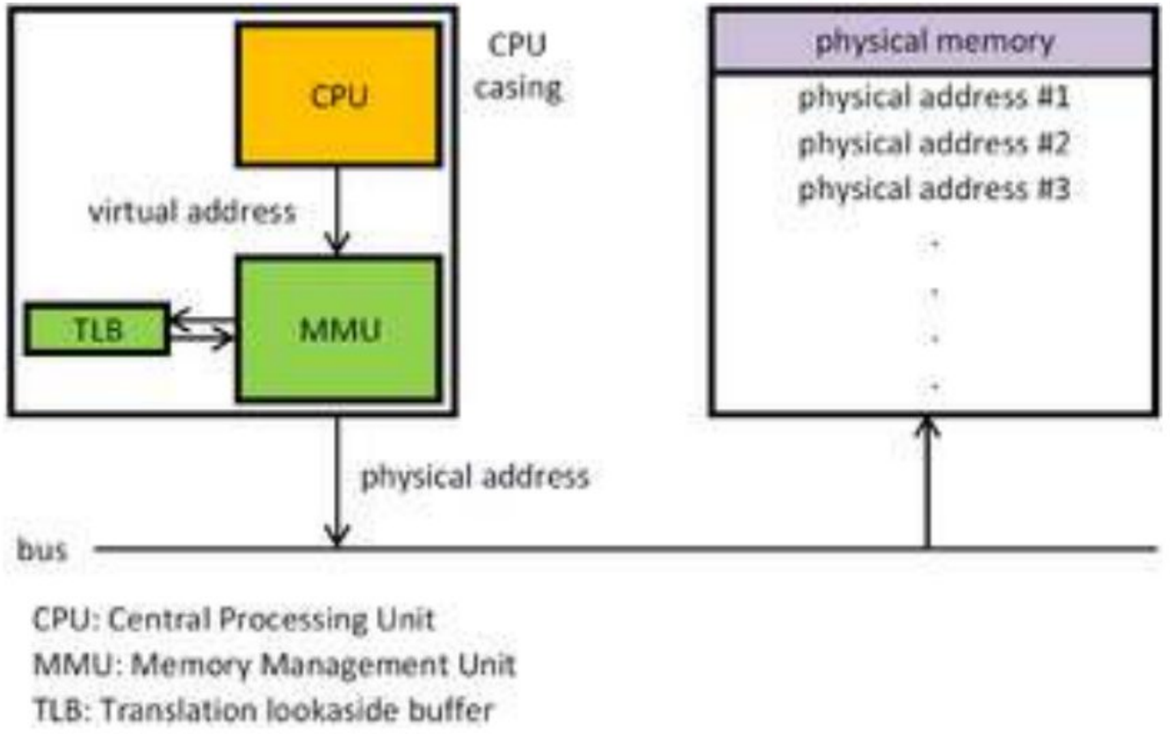
swapout is optional ?

↳ Its optional depends on dirty bit

		Dirty bit
frame1	page1	0
frame2	page2	1
frame3	page3	0

↳ Determines whether data is edited or not

Memory management unit



CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer

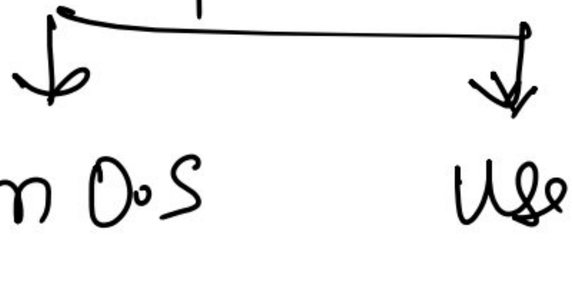
- A **memory management unit** (**MMU**), sometimes called **paged memory management unit** (**PMMU**), is a computer hardware unit having all memory references passed through itself, primarily performing the translation of virtual memory addresses to physical addresses.

Multitasking

Let say multiple task are stored in main memory

Task is a single program or can be part of program

Task can be



System OS — User

Ex: BIOS
Linker
Loader

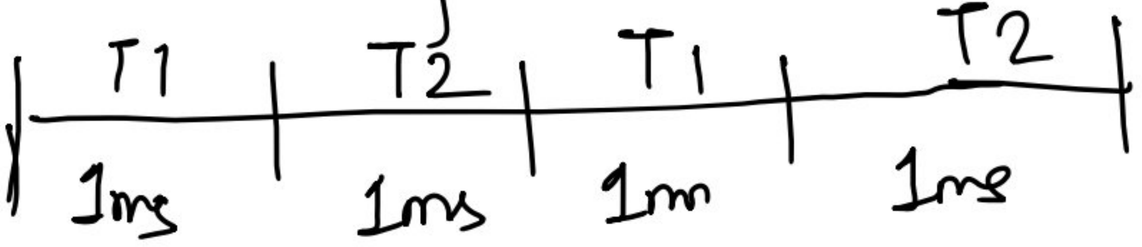
Both system and user level task contain code, data and stack.

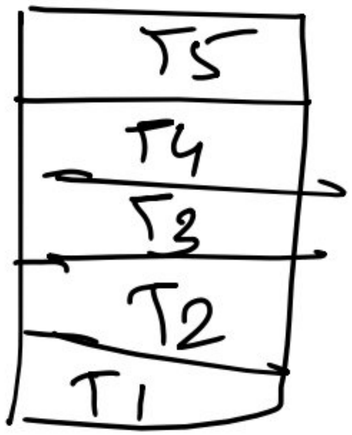
∴ So both need code, data and stack segments to store

Multitasking cont.d

Assume on single core

Time sharing





Time Slice = 1ms

Processor will execute 2 task alternately but we see that 2 task are simultaneously

Hardware Support for V.M

Ex: 80386 has a support for multitasking and segmentation

It is a 32-bit processor with 32 bit data bus and 32 bit address bus

* Segment Information (Code / Data / Stack)

Stored in Descriptors

1. Size of segment
2. Starting address of segment
3. Type of segment
4. protection level of segment

Suppose BIOS → Global Task

↓ has

100 segments

↓

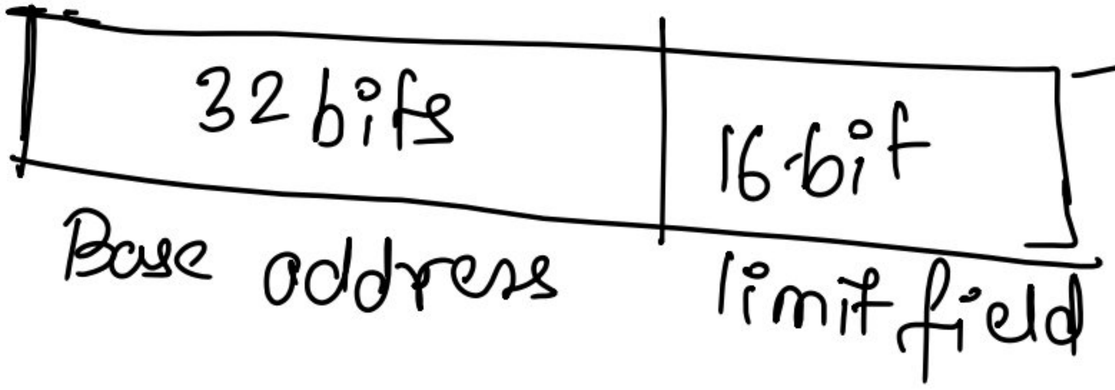
100 Global descriptors

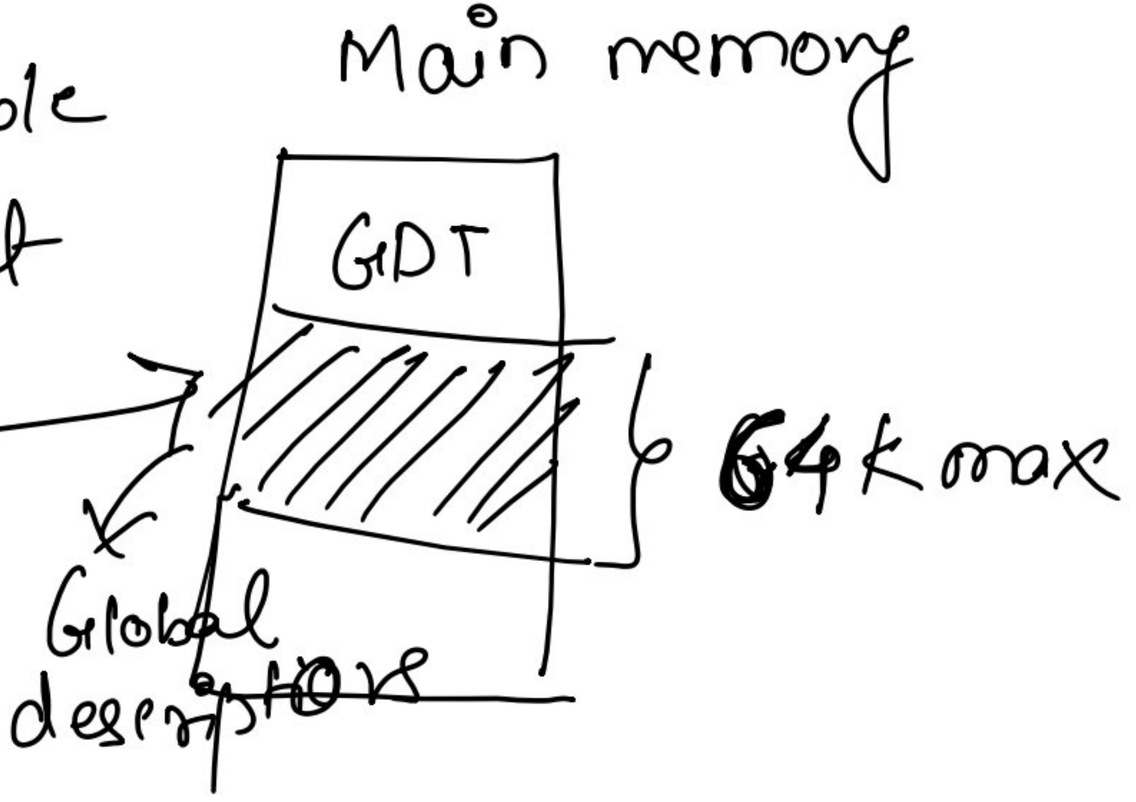
where it is stored? → Main memory

Information of descriptors → stored in Global descriptor Table (GDT)

Pointer to GDT is GDTR

GDTR (Global Descriptor table Register) format





Size of one descriptors is 8 bytes

Max. Global descriptors available in table = $\frac{64kB}{8}$

= 8k = 8192 descriptors

For Local task we have LDTR

Size of LDTR should be 48 bit some but Intel has designed as 16 bits are rename its LDTR selector

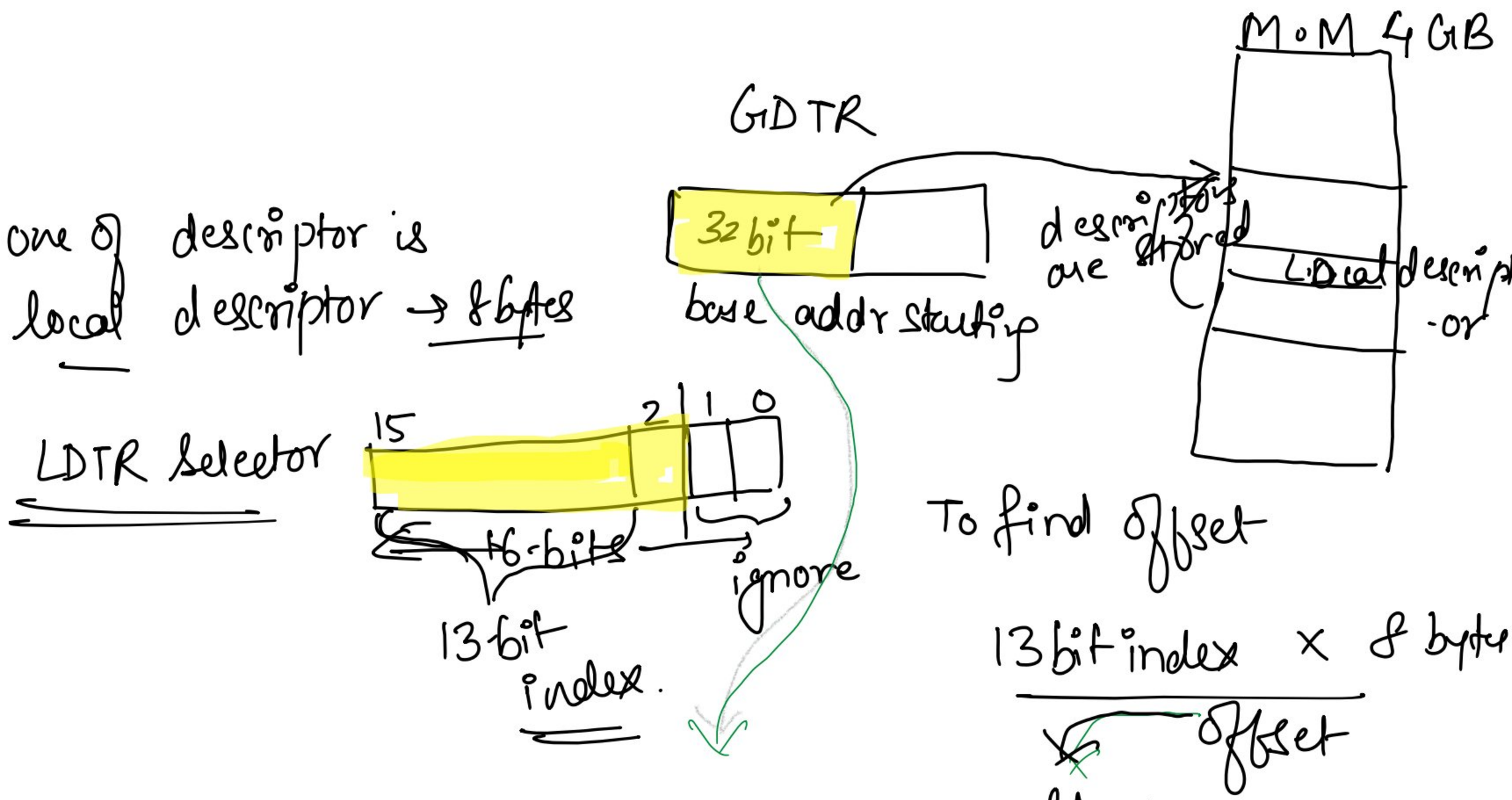
* It selects LDT descriptors
  * Size of each descriptor = 8 bytes

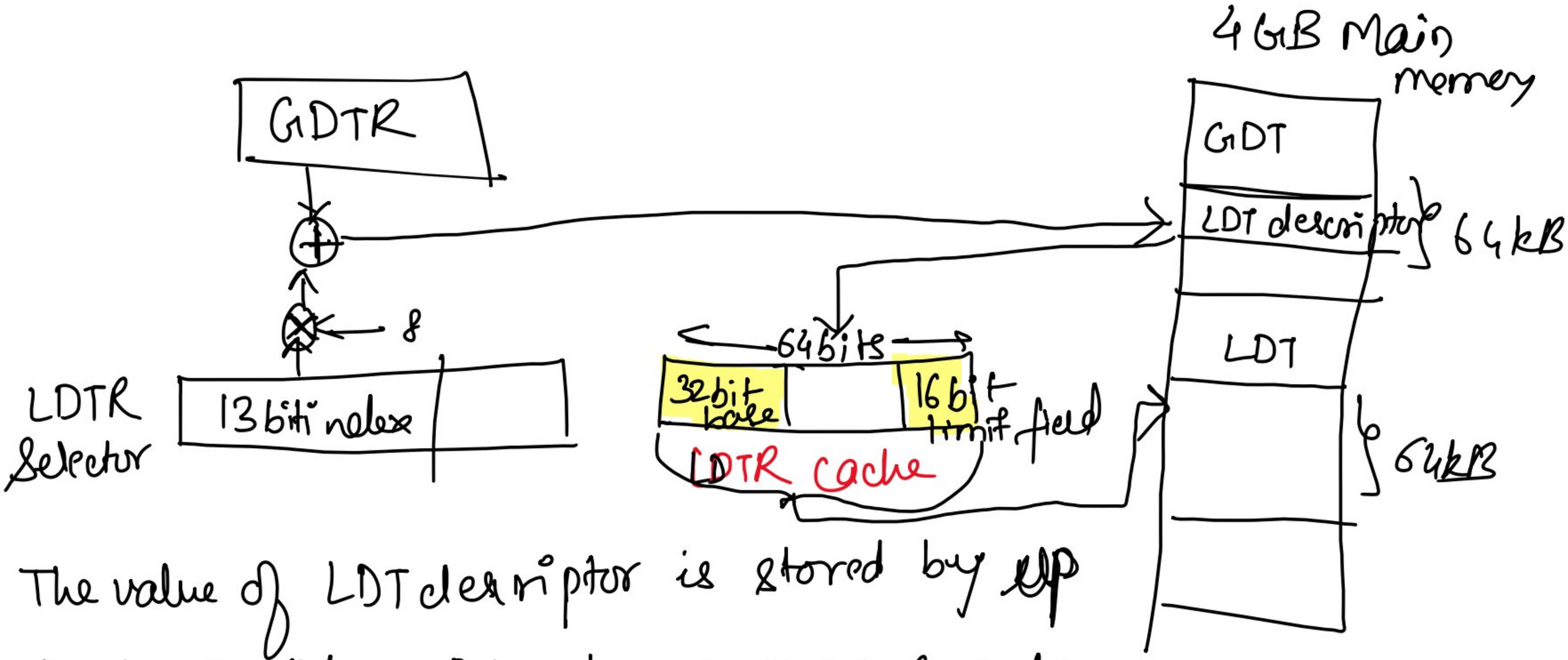
GDT contain

1. LDT descriptors
2. Global Segment descriptors

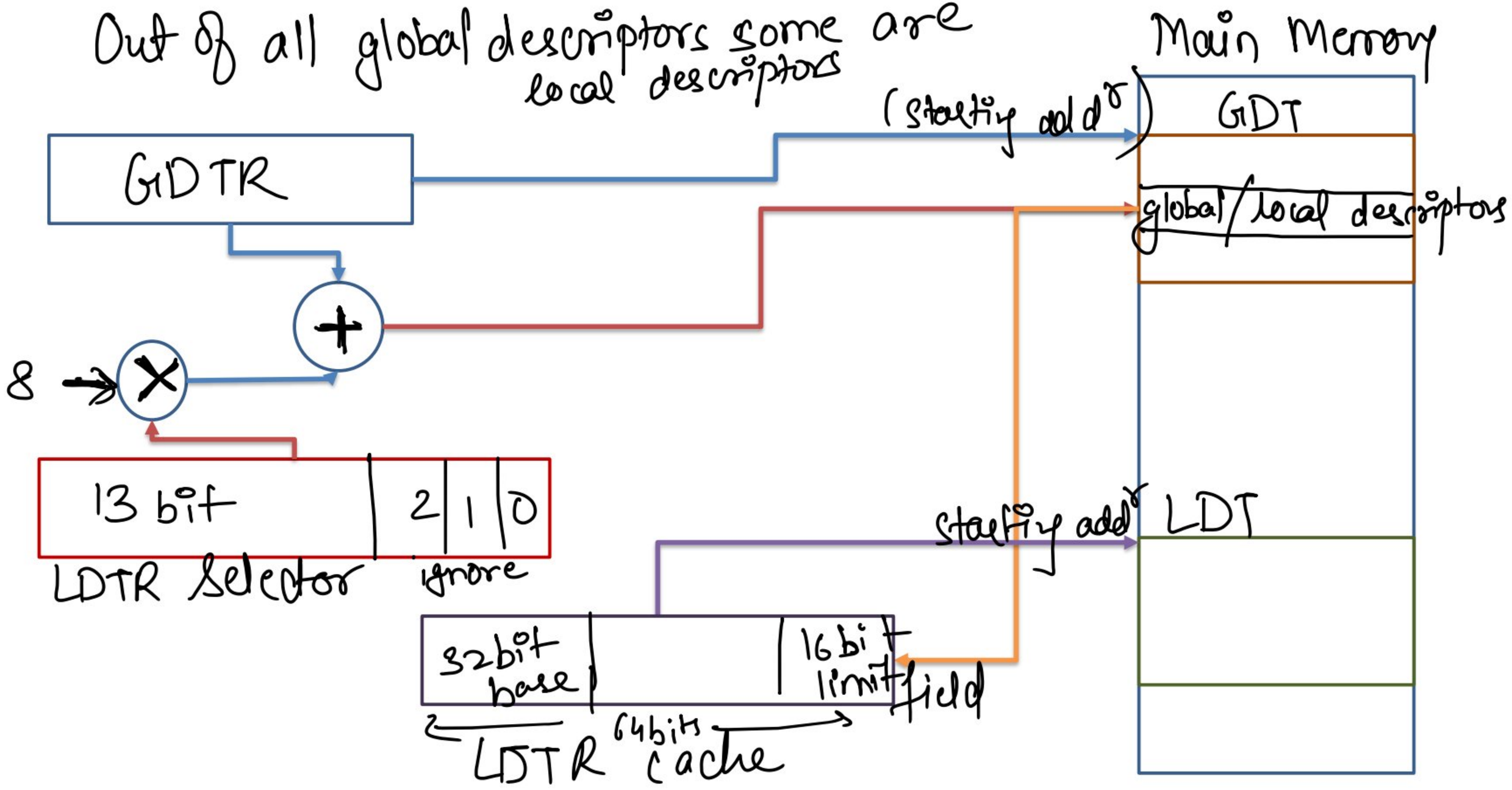
LDT contain

1. Local Segment Descriptors


MoM 4 GB
GDTR
32 bit
base addr startup
descriptors are stored
Local descrip-or
one of descriptor is local descriptor → 8 bytes
LDTR Selector
15
2
1
0
16-bits
ignore
13 bit index.
To find offset
13 bit index × 8 byte
offset
Pointer to Local descriptor = 32 bit base + offset



The value of LDT descriptor is stored by µp in the special register known as LDTR cache from LDTR cache (32bit) base address-ie start of LDT


Out of all global descriptors some are local descriptors
Main Memory
GDT
(starting addr)
GDTR
global/local descriptors
+
8
X
13 bit
2
1
0
LDTR Selector
ignore
starting addr
LDT
32bit base
16bit limit
field
64bits
LDTR cache

Because of 16-bit limit field of LDTR cache

Max. local descriptors are possible = $\frac{64kB}{8} = 8k$

64kB is size of LDT

$= 8192$

Size of each descriptor is 8 bytes

∴ Total descriptors = 8k LD + 8k GD

= 16k. descriptors

∴ Total segment possible = 16k.

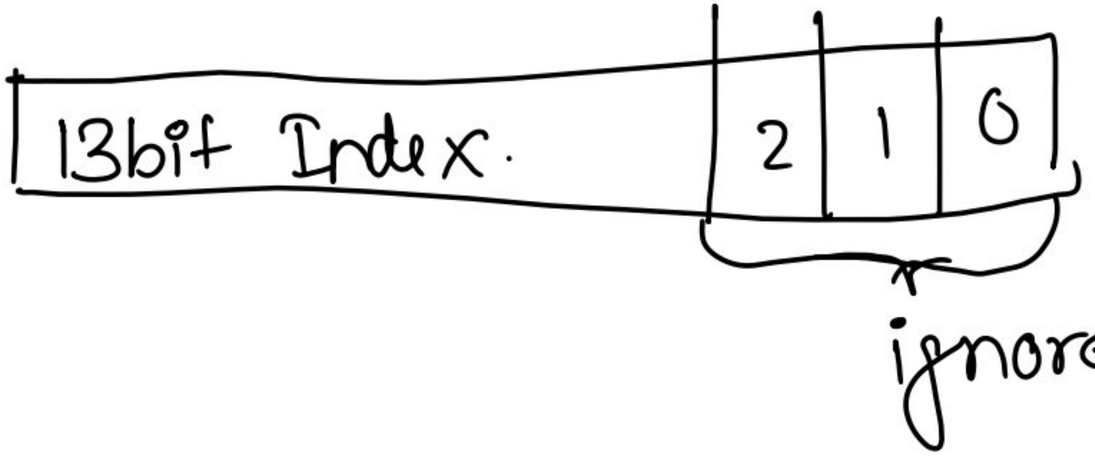
Now we will find location of segment and then data from segment

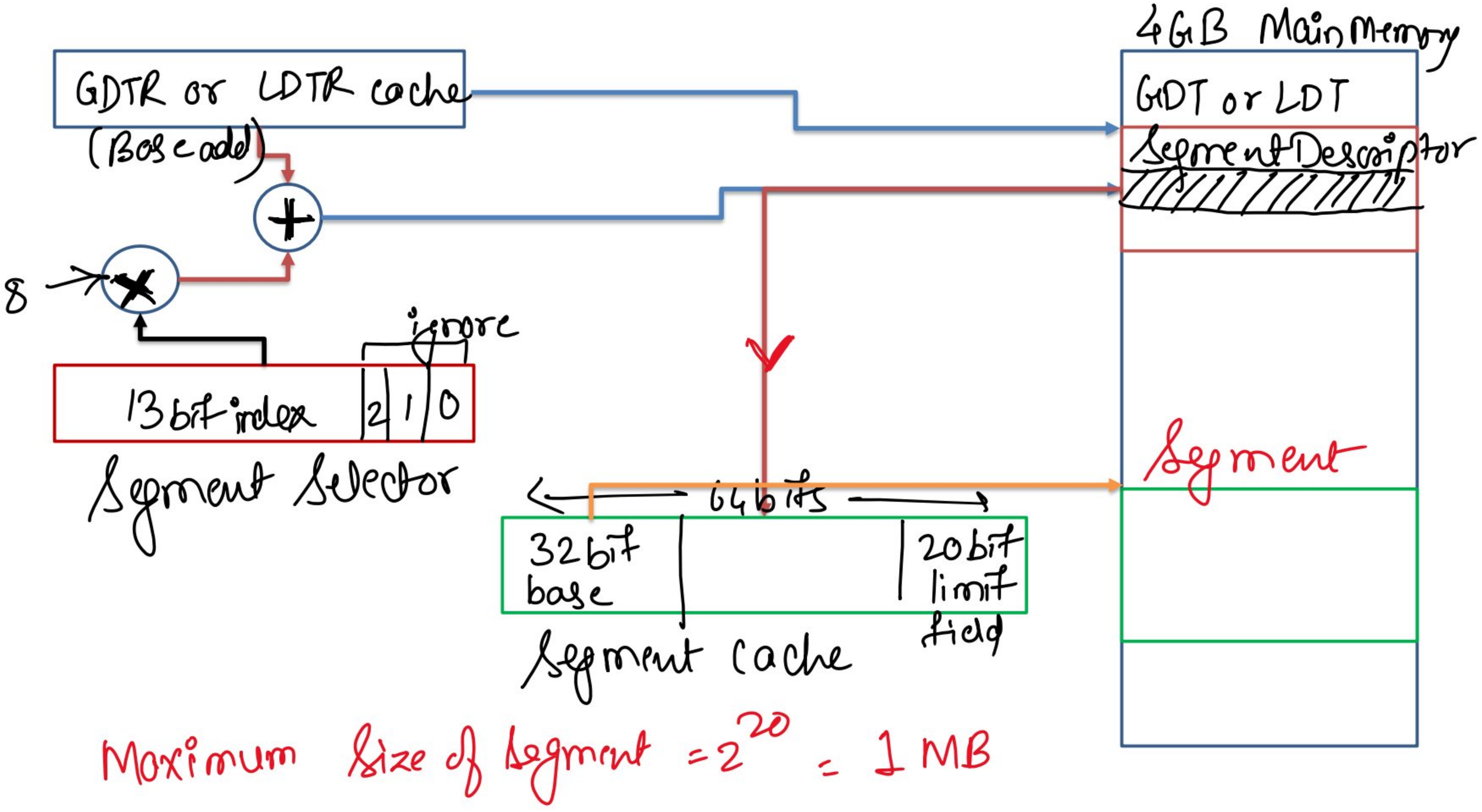
(3) Segment Register / Segment Selector

↳ It selects segment descriptors

Size = 16 bits

↳ It can be global or local




4GB Main Memory
GDTR or LDTR cache
(Base add)
GDT or LDT
Segment Descriptor
8
ignore
13 bit index
2
1
0
Segment Selector
Segment
64 bits
32 bit base
20 bit limit field
Segment cache
Maximum Size of Segment = 2^{20} = 1 MB

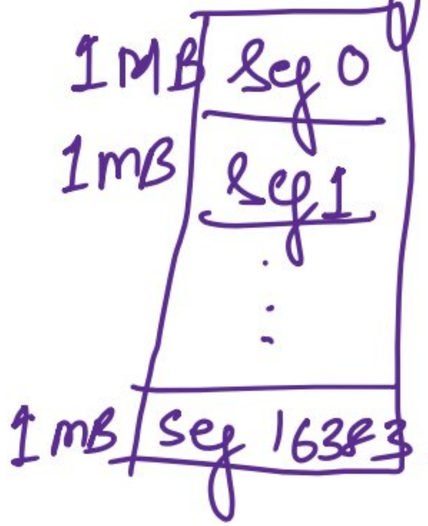
Segmentation Model

Size of One Segment = 1 MB

Total No. of descriptors = 8k GDTR + 8k LDTR

= 16k Descriptors

Total No. of Segment = 16k Segments

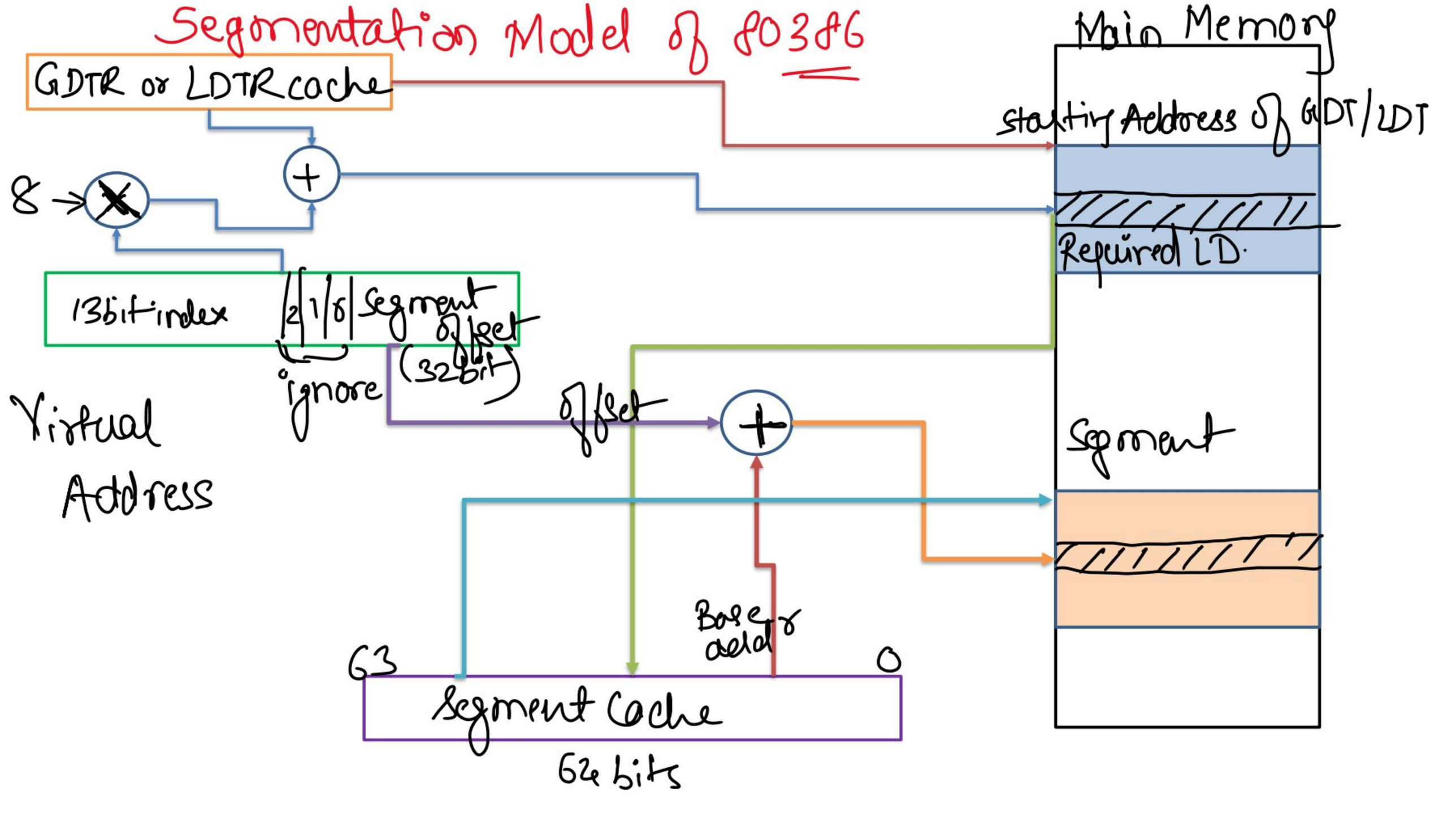


Virtual Memory Address Space (VMAS)

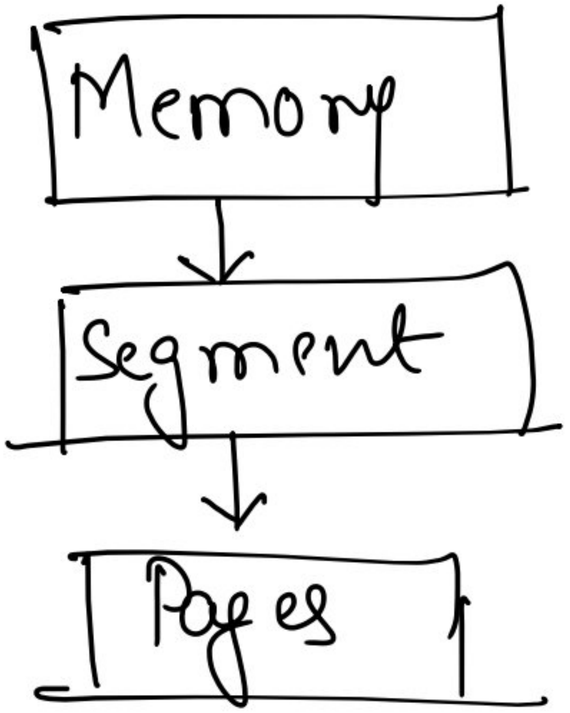
Physical Address Space (PMAS) is 4GB

∴ Total Virtual Memory = 1MB x 16k

= 16GB


Segmentation Model of 80386
Main Memory
GDTR or LDTR cache
Starting Address of GDT/LDT
8
13bit index
2
1
0
Segment offset (32bit)
Required LD.
ignore
Virtual Address
offset
Segment
Base address
63
0
Segment Cache
64 bits

Paged Model



From Segmentation model maximum size of segment = 2^{20}

= 1 MB → Now this unit is not in Byte but in pages

= 1 M pages

∴ Page size (given by Intel) = 4 kB

∴ Maximum size of segment = 4k × 1M = 4 GB

So in paged model Virtual memory address space

(VMAS) = 4GB × $\frac{16k}{}$

↖ Total no. of segment

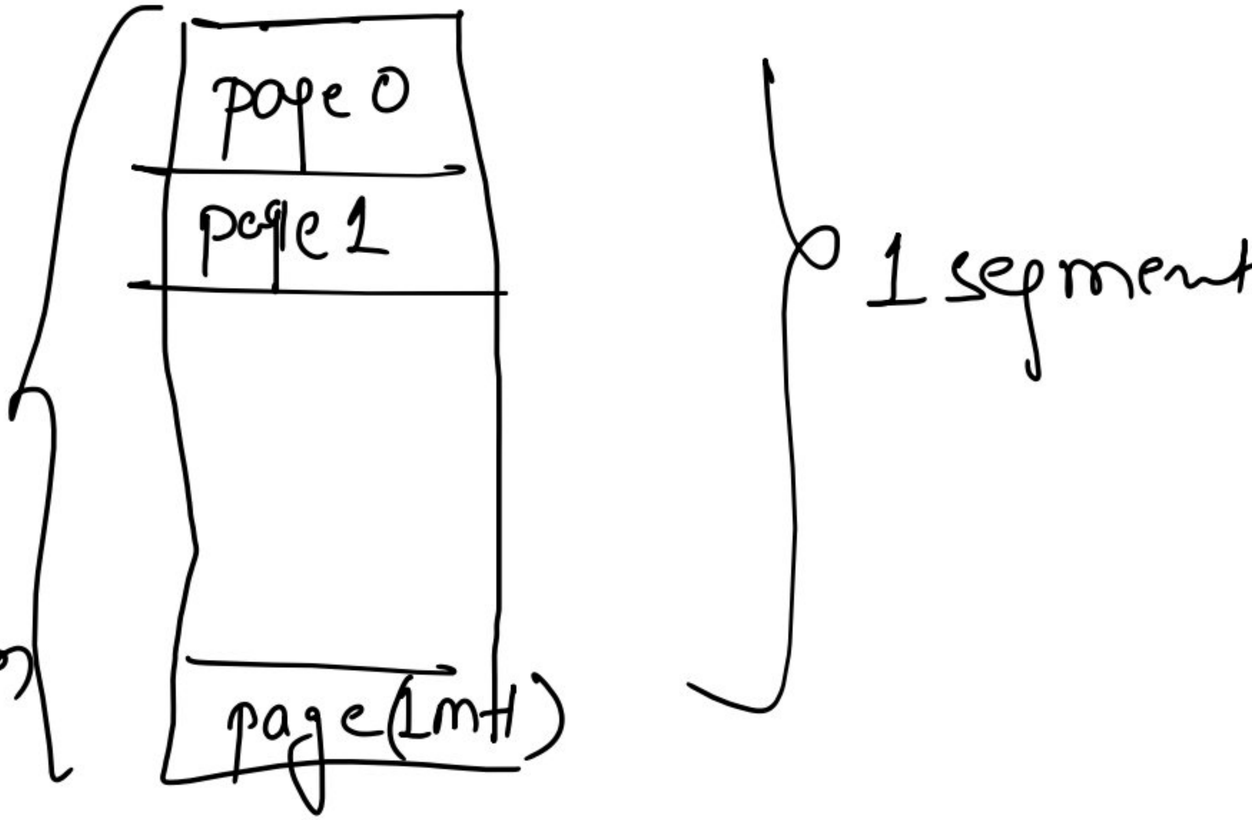
= 64TB

In case of paged model

Size of page is fixed 4kB

but segment size is variable

(Difference bet^n segmentation model)



- ***PAGING OPERATION***: Paging is one of the memory management techniques used for virtual memory multitasking operating system.
- The segmentation scheme may divide the physical memory into a variable size segments but the paging divides the memory into a fixed size pages.
- The segments are supposed to be the logical segments of the program, but the pages do not have any logical relation with the program.
- The pages are just fixed size portions of the program module or data.

Page Translation Mechanism

Stage (A) → Virtual Address (48 bits)

↓

Linear Address (32 bits)

Stage (B) → Linear Address (32 bits)

↓

Physical Address (32 bits)

* In main memory create a page directory (4kB)
* pointer to page directory is PDBR (page directory base Register)
  ↳ size = 20bits

* These 20bits get added with 12 bits offset from up (internally added)

∴ Size of page directory = 4kB

* Page Directory contain Page Directory Entries (PDE)

Size of one PDE = 4 bytes = 32 bits

∴ No of PDE = $\frac{4kB}{4}$ = 1024

* These PDE will point to Page table

* Directory field (10bit) x 4 + PDBR → Desired PDE from Linear Address

* Mp copy desired PDE into TLBI register
↓
(Translation look aside Buffer)

* TLB1 point starting address of page table

TLB1 (32bit) + (page field of L.A x 4)
(10 bits

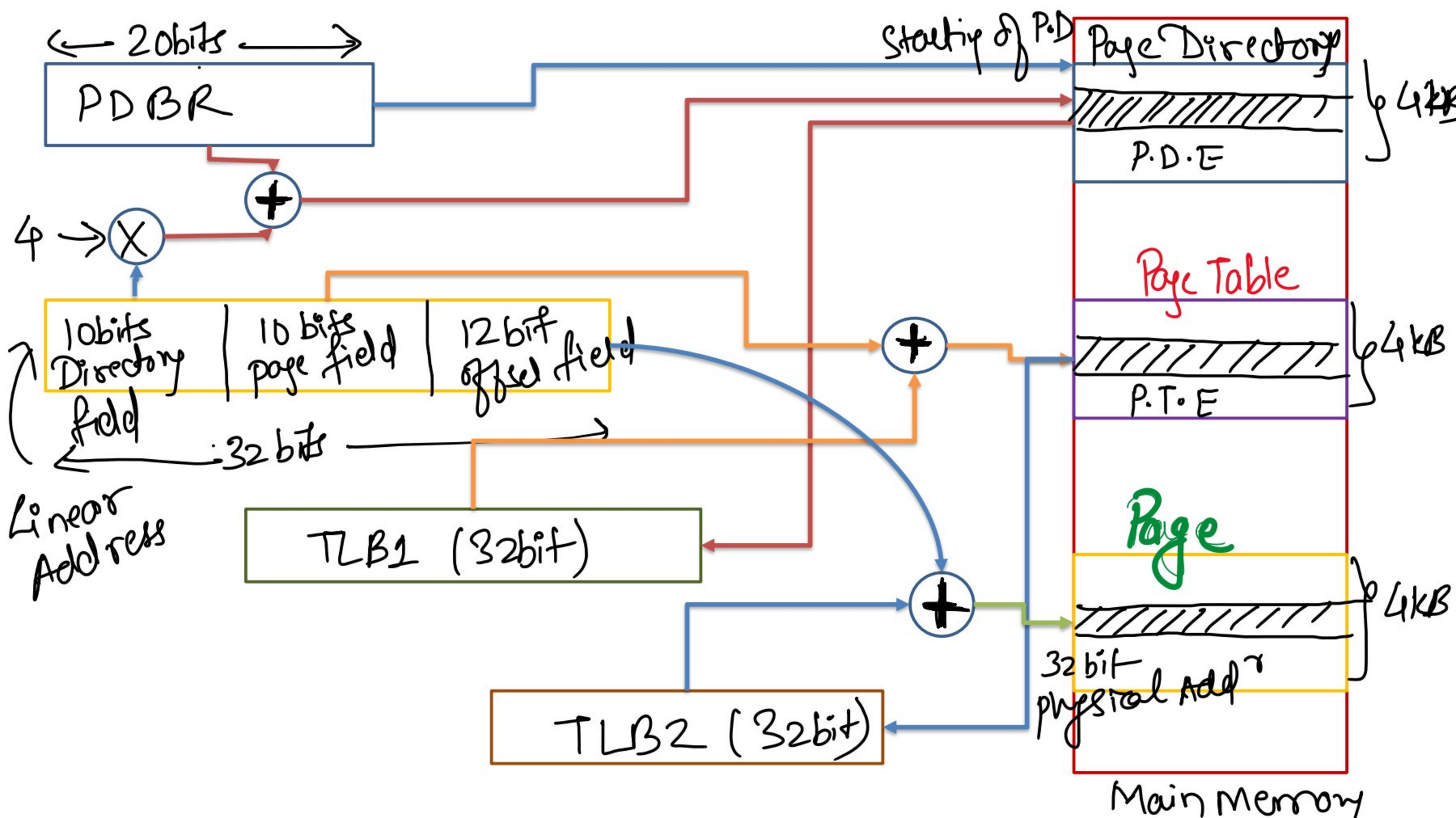
⇓

Starting address of required page table Entry (P.T.E)

* P.T.E get stored in TLB2 (Translational lookaside buffer2)
* TLB2 point to starting address of Actual page
* TLB2 (32bits) + (12bit offset field of L.A)

⇓

32 bit physical Address of Actual page


20bits
PDBR
Starting of P.D
Page Directory
4KB
P.D.E
4
X
Page Table
10bits Directory field
10 bits page field
12bit offset field
32 bits
4KB
P.T.E
Linear Address
TLB1 (32bit)
Page
4KB
32 bit physical Addr
TLB2 (32bit)
Main Memory

Paging.

- The D bit (Dirty bit) is set before a write operation to the page is carried out. The D-bit is undefined for page director entries.
- The OS reserved bits are defined by the operating system software.
- The User / Supervisor (U/S) bit and read/write bit are used to provide protection. These bits are decoded to provide protection under the 4 level protection model.
- The level 0 is supposed to have the highest privilege, while the level 3 is supposed to have the least privilege.
- This protection provide by the paging unit is transparent to the segmentation unit.