Memory Unit

Some Basic Concepts

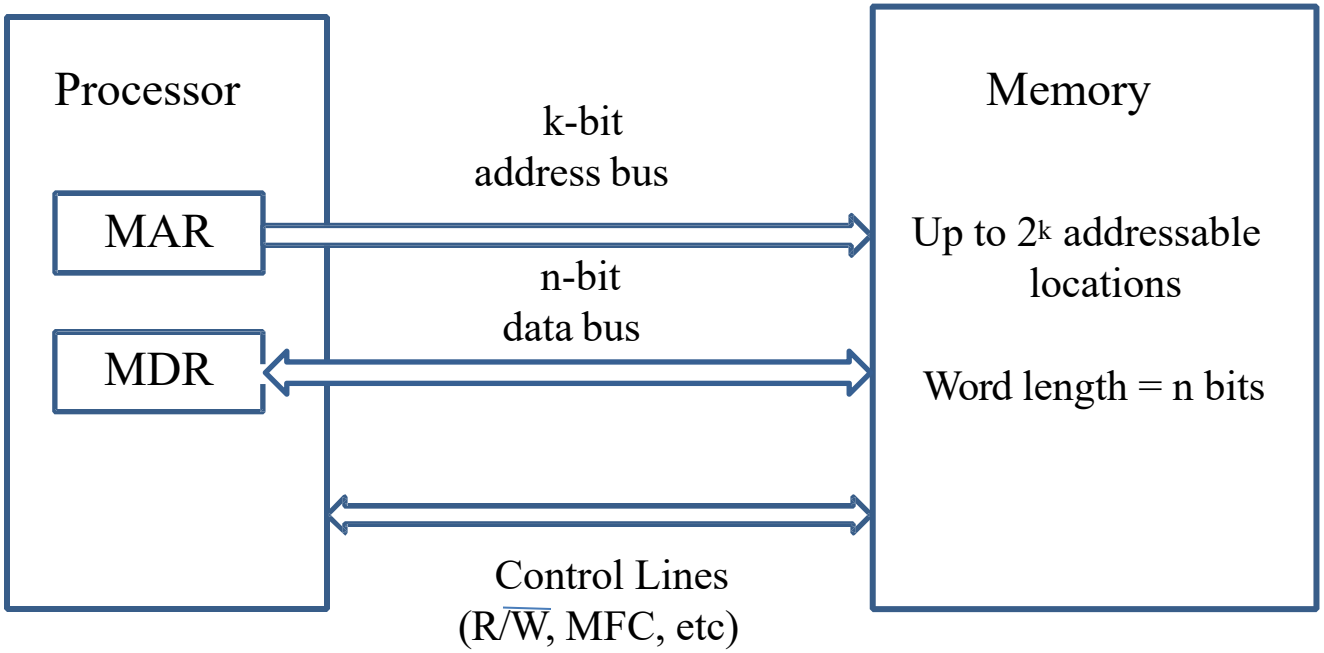


Fig : Connection of memory to the processor

Some Basic concepts

Meaning of Random Access Memory

Byte Addressable memory

Word Addressable memory

Memory Capacity

Some Basic concepts

Measures for the speed of a memory:

- **memory access time.**
- **memory cycle time.**

The time gap between the initiation of an memory operation(read/write) and the completion of that operation(MFC) is called as the memory access time.

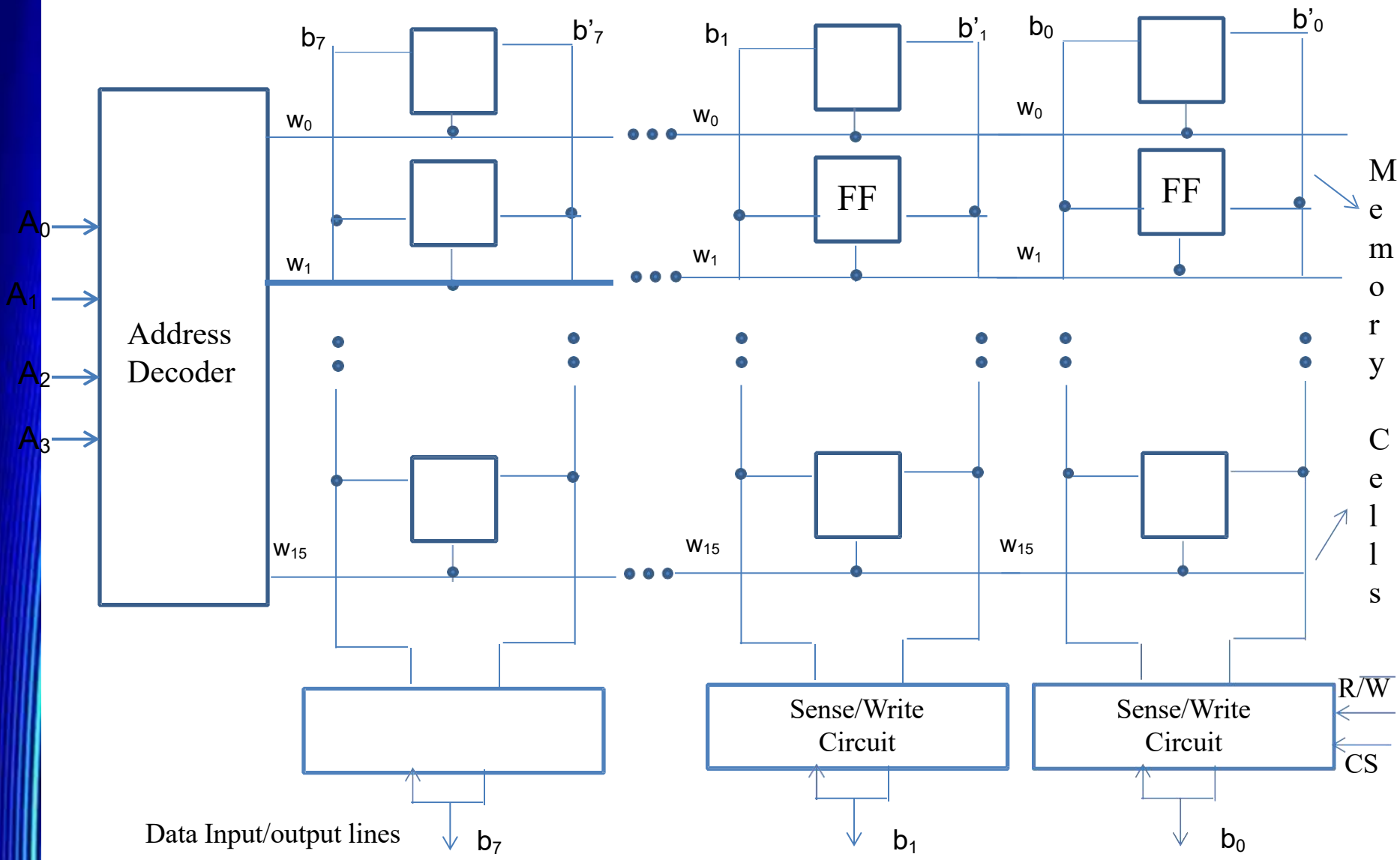
The time gap between the initiation of two consecutive memory operations is called as the memory cycle time.

Some Basic concepts

- An important design issue is to provide a computer system with as large and fast a memory as possible, within a given cost target.

- Several techniques to increase the effective size and speed of the memory:
  - Cache memory (to increase the effective speed).
  - Virtual memory (to increase the effective size).

Organization of bit cells in a 16 x 8 Memory chip



Internal organization of memory chips

Each memory cell can hold one bit of information.

Memory cells are organized in the form of an array.

One row is one memory word.

All cells of a row are connected to a common line, known as the "word line".

Word line is connected to the address decoder.

Sense/write circuits are connected to the data input/output lines of the memory chip.

No of external pins required to connect a memory chip

For 16 x 8 chip

4 address lines

8 data lines

2 (R/W + CS)

2 (Power Supply+G_{ND})

---

16 (Total)

No of external pins required to connect a memory chip

For 128 x 8 chip

7 address lines

8 data lines

2 (R/W + CS)

2 (Power Supply+G_{ND})

---

19 (Total)

No of external pins required to connect a memory chip

For 1K x 1 chip

10 address lines

1 data line

2 (R/W + CS)

2 (Power Supply+G_{ND})

---

15 (Total)

No of external pins required to connect a memory chip

For 64 x 16 chip

6 address lines

16 data line

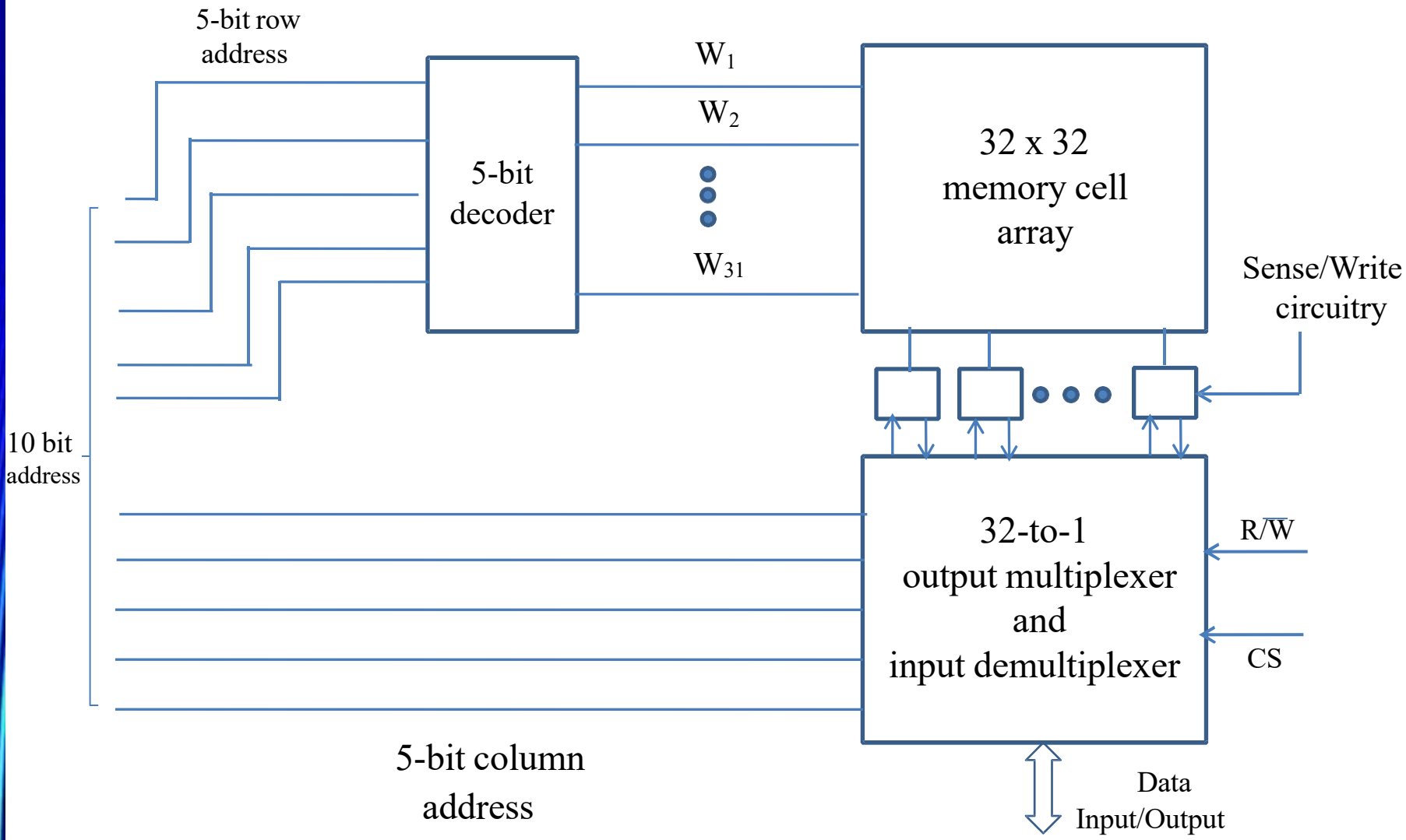
2 (R/W + CS)

2 (Power Supply+G_{ND})

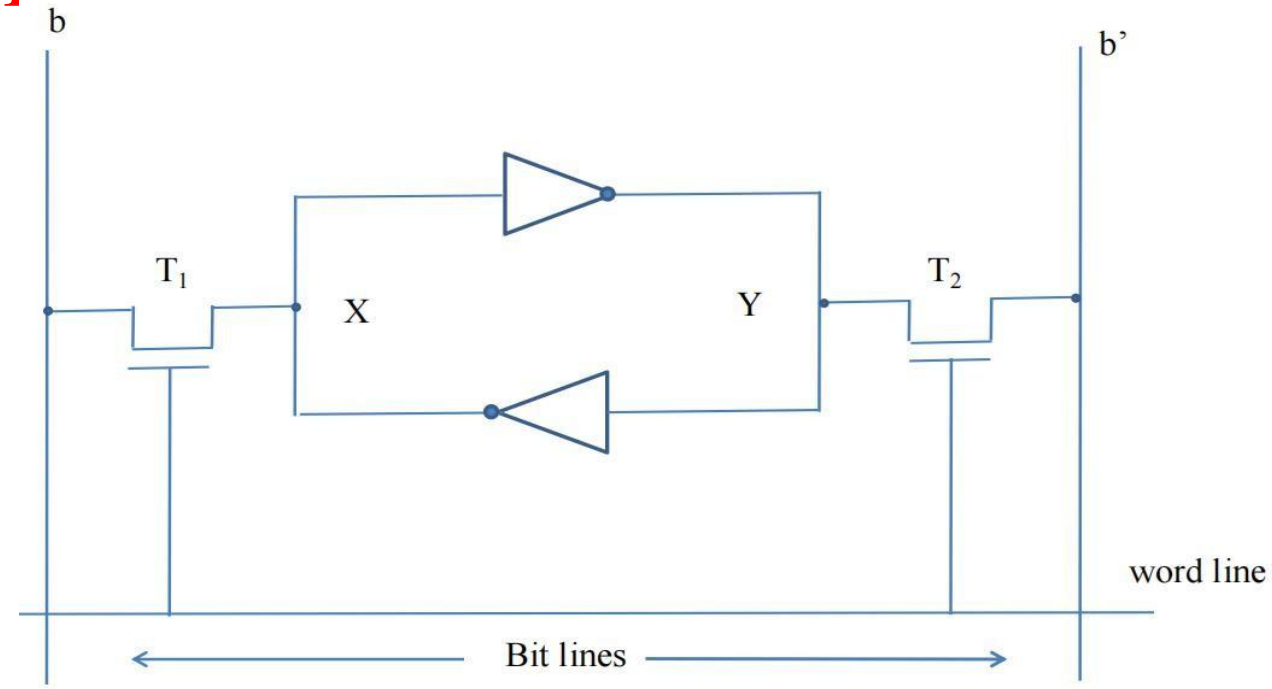
---

26 (Total)

Internal organization of 1K x 1 memory chip



Implementation of a SRAM Cell

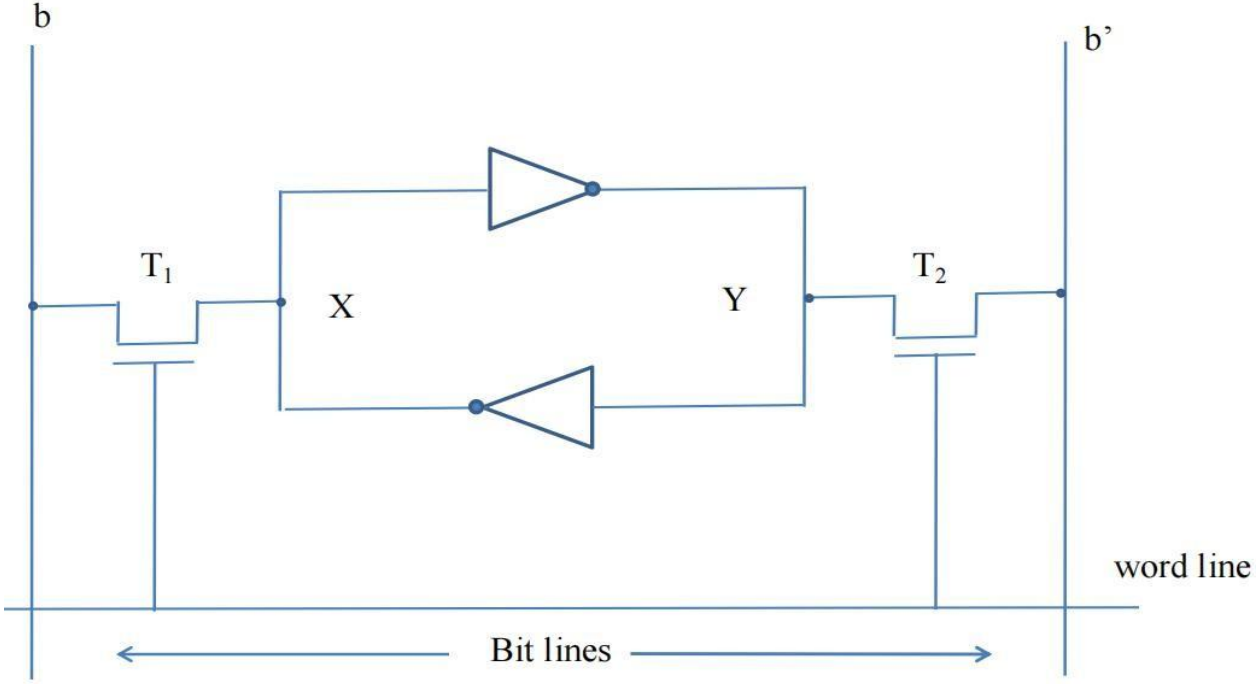


Two inverters are cross connected to implement a basic storage element **"latch"**.

The cell is connected to one word line and two bits lines by transistors T1 and T2.

When word line is at ground level, the transistors are turned off and the latch retains its state.

Implementation of a SRAM Cell



Read operation:

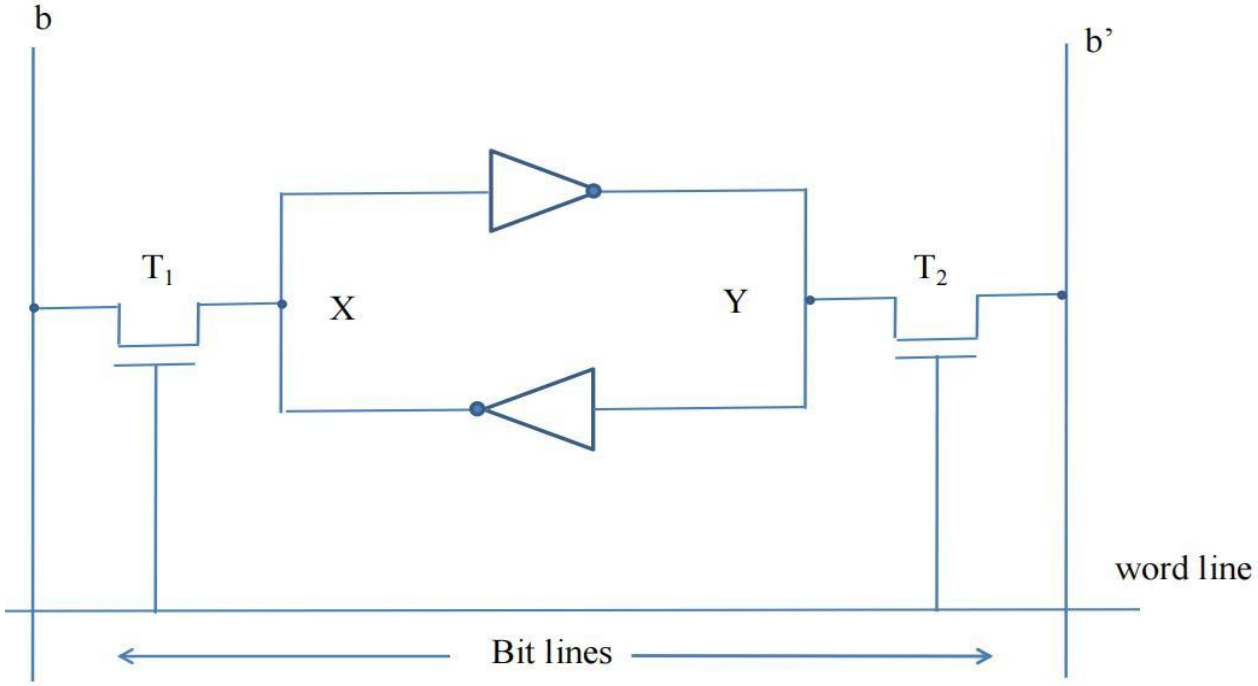
1.In order to read state of SRAM cell, the word line is activated to close switches T1 and T2. Sense/Write circuits at the bottom monitor the state of b and b'

2.Sense/Write circuits at the bottom monitor the state of b and b' and set the output accordingly.

3.If the cell is in **state 1**, the signal on bit line b is high and the signal on bit line b' is low.

4. The opposite is true if the cell is in **state 0.**

Implementation of a SRAM Cell



Write operation:

1. The state of the cell is set by placing the appropriate value on bit line b and its complement on b', and then activating the word line.

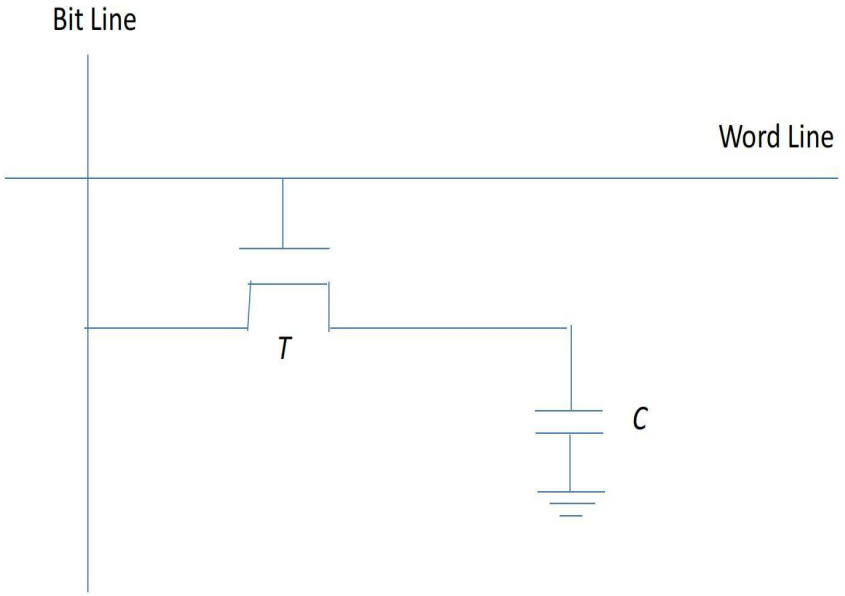
2. This forces the cell into the corresponding state.

3. The required signals on the bit lines are generated by the Sense/Write ckt.

Implementation of a DRAM Cell

Dynamic RAM (DRAM): slow, cheap, and dense memory.

Typical choice for main memory.



Cell Implementation:

1-Transistor cell (pass transistor)

Trench capacitor (stores bit)

Bit is stored as a charge on capacitor. The charge can be maintained for only tens of milliseconds.

But the cell is required to store the information for a much longer time.

Hence, the contents must be refreshed by restoring the capacitor charge to its full value.

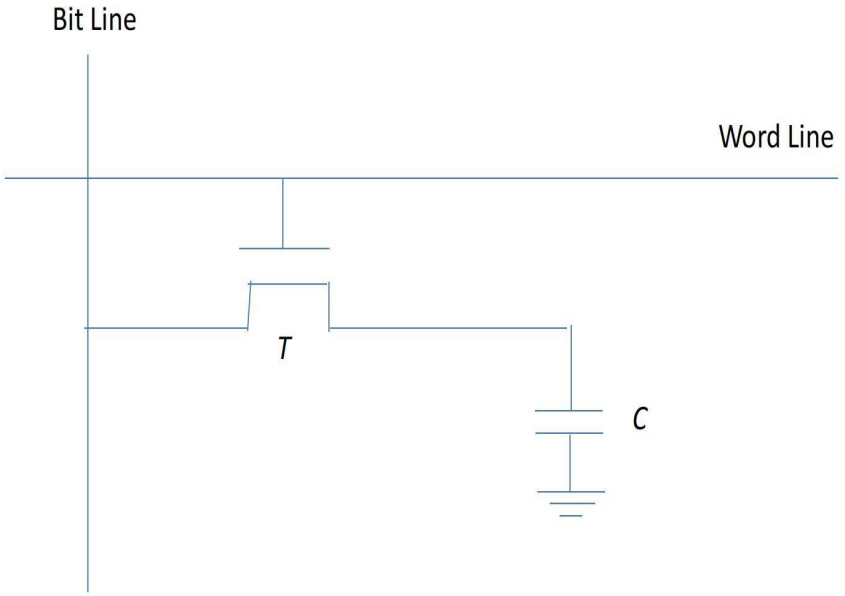
Implementation of a DRAM Cell

Why periodic refreshing is required??

After the transistor is turned off, the capacitor begins to discharge.

1. This happens due to the capacitor's own leakage resistance.

2. The transistor continues to conduct a tiny amount of current measurd in picoamperes, after it is turned off.



Implementation of a DRAM Cell

How periodic refresh operation is perfomed??

During a read operation, the transistor in a selected cell is turned on.

A sense amplifier connected to the bit line detects whether the charge stored on the capacitor is above the threshold value.

If so, it drives the bit line to a full voltage that represents logic value 1. This voltage recharges the capacitor to the full charge that corresponds to logic value 1.

If the sense amplifier detects that the charge on the capacitor is below the threshold value , it pulls the the bit line to ground level, which ensures that the capacitor will have no charge, representing logic value 0.

Hence, reading the contents of the cell automatically refreshes its contents.

SRAM Vs DRAM Cell

Static RAMs (SRAMs):

Consist of circuits that are capable of retaining their state as long as the power is applied.

Volatile memories, because their contents are lost when power is interrupted.

Access times of static RAMs are in the range of few nanoseconds.

Requires low power to retain bit. As power is consumed, only when the cell is accessed.

However, the cost is usually high.

Dynamic RAMs (DRAMs):

Do not retain their state indefinitely.

Contents must be periodically refreshed.

Contents are be refreshed while accessing them for reading.

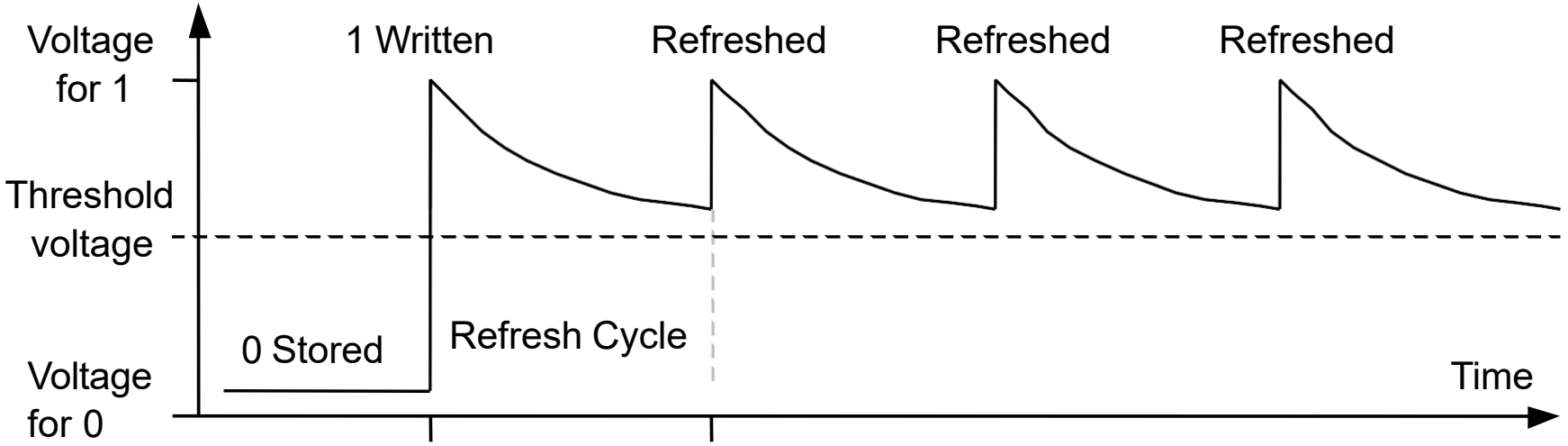
DRAM Refresh Cycles

Refresh cycle is about tens of milliseconds.

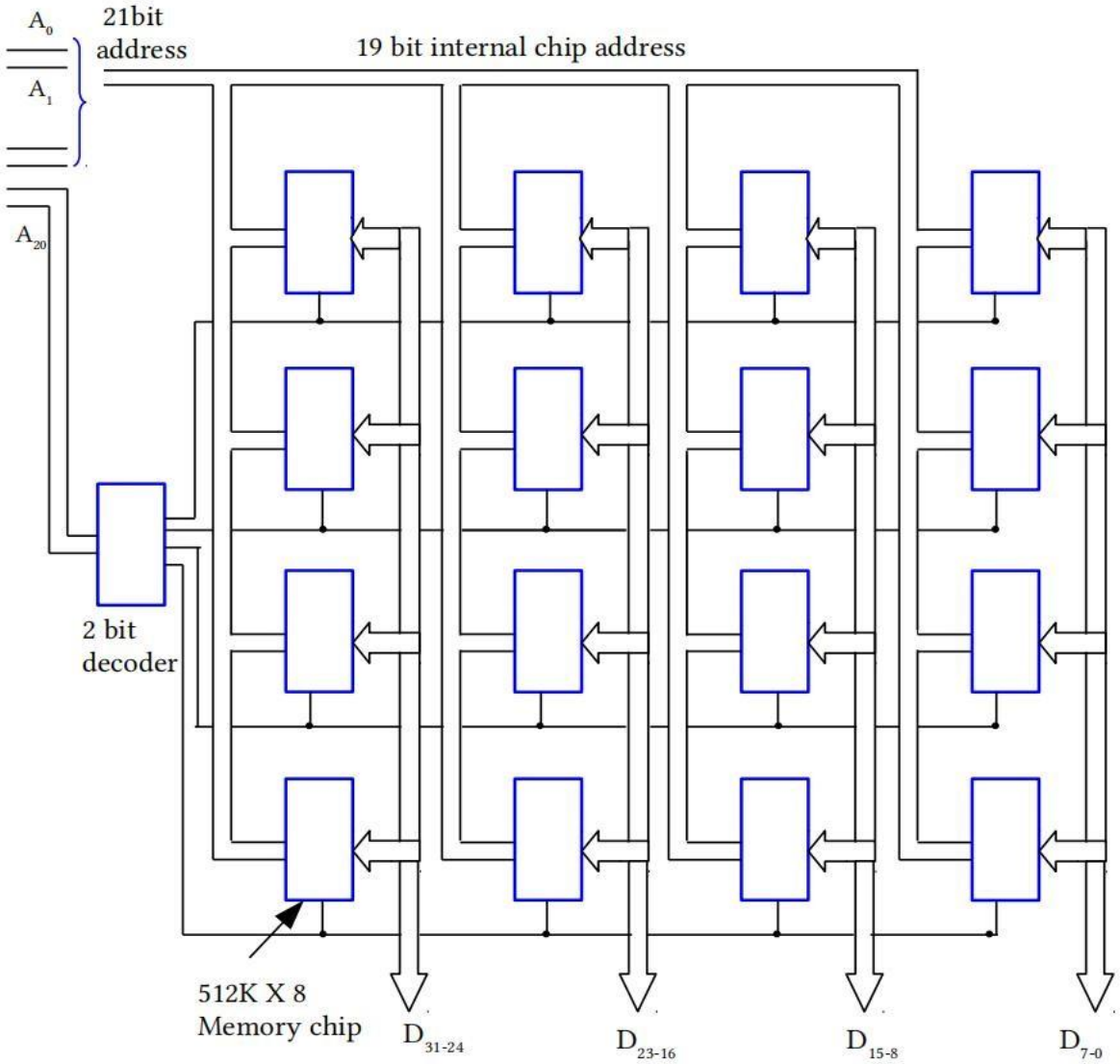
Refreshing is done for the entire memory.

Each row is read and written back to restore the charge.

Some of the memory bandwidth is lost to refresh cycles.



2M X 32 Using 512K X 8 Chips



2M X 32 Using 512K X 8 Chips

Step 1: Find out , how many smaller size chips are required to meet the required size:

Divide the total required size/ Size of the smaller chip

2M X 32/ 512K x 8= 2^{21} X 2^{5} / 2^{19} X 2^{3} = 2^{26}/ 2^{22}= 2^{4}= 16 Chips

Step2 : Find, how many smaller size chips need to connect in paralell to meet the required data size.[finding the no of columns in the matrix arrangement.

Here, in 2M x 32, 32 bits of data is required, and in 512K x 8 chip, 8 bits of data can be communicated from one location.

So, 4, 512K x 8 chips need to be connected in parallel, to meet the 32 bits data size.

i.e., we need to connect 512K x 8 chips in a matrix form., where no of columns will be= size of one location in the bigger size/ size of one location in the smaller size.

Number of columns = 32/ 8= 4.

2M X 32 Using 512K X 8 Chips

Step 3: Find out the no of rows:

No of rows x No of Columns = Total no of elements

#rows x 4 =16

#rows =16/4=4

How to connect the address lines:

For 2M x 32 , memory, 21($2M=2^{21}$) address lines are required, and for 512K x 8, 19($512K=2^{19}$) address lines are required. So, out of 21 address lines, the 1st 19 lines will be connected to all the 512K x 8 memory chips.

Then to select a row, out of 4 rows of 512K x 8 memory chips, the higher order 2 address lines (out of 21 address lines) are connected to a decoder, and the output of the decoder will select a particular row. From the selected row, 4 chips of 512K x 8 will give/ take 8 bits of data each, meeting the required size of 32 bits.

Numerical : Design 8M X 32 bits memory using 512KX8 bits memory chip.

Step 1: Find out , how many smaller size chips are required to meet the required size:

Divide the total required size/ Size of the smaller chip

8M X 32/ 512K x 8= 2^{23} X 2^5 / 2^{19} X 2^3 = 2^{28}/ 2^{22}= 2^6= 64 Chips

Step2 : Find, how many smaller size chips need to connect in paralell to meet the required data size.[finding the no of columns in the matrix arrangement.]

Here, in 8M x 32, 32 bits of data is required, and in 512K x 8 chip, 8 bits of data can be communicated from one location.

So, 4, 512K x 8 chips need to be connected in parallel, to meet the 32 bits data size.

i.e., we need to connect 512K x 8 chips in a matrix form., where no of columns will be= size of one location in the bigger size/ size of one location in the smaller size.

Number of columns = 32/ 8= 4.

Numerical : Design 8M X 32 bits memory using 512KX8 bits memory chip.

Step 3: Find out the no of rows:

No of rows x No of Columns = Total no of elements

#rows x 4 =64

#rows =64/4=16

How to connect the address lines:

For 8M x 32 , memory, 23($8M=2^{23}$) address lines are required, and for 512K x 8, 19($512K=2^{19}$) address lines are required. So, out of 23 address lines, the 1st 19 lines will be connected to all the 512K x 8 memory chips.

Then to select a row, out of 16 rows of 512K x 8 memory chips, the higher order 4 address lines (out of 23 address lines) are connected to a decoder, and the output of the decoder will select a particular row. From the selected row, 4 chips of 512K x 8 will give/ take 8 bits of data each, meeting the required size of 32 bits.

Numericals

A computer employs RAM chips of 256 X 8 . The computer system needs 2K bytes of RAM . Design the memory module of above configuration

Step 1: Find out , how many smaller size chips are required to meet the required size:

Divide the total required size/ Size of the smaller chip

$2K \times 8 / 256 \times 8 = 2^{11} \times 2^3 / 2^8 \times 2^3 = 2^3 = 8$ Chips

Step2 : Find, how many smaller size chips need to connect in paralell to meet the required data size.[finding the no of columns in the matrix arrangement.

Here, in 2K x 8, 8 bits of data is required, and in 256 x 8 chip, 8 bits of data can be communicated from one location.

So, 1, 256 x 8 chip needs to be connected in a column to meet the 8 bits data size.

No of columns = 1

Numerical : Design 2K X 8 bits memory using 256X8 bits memory chip.

Step 3: Find out the no of rows:

No of rows x No of Columns = Total no of elements

#rows x 1 =8

#rows = 8

How to connect the address lines:

For 2K x 8 , memory, 11($2K=2^{11}$) address lines are required, and for 256 x 8, 8 ($256=2^{8}$) address lines are required. So, out of 11 address lines, the 1st 8 lines will be connected to all the 256 x 8 memory chips.

Then to select a row, out of 8 rows of 256 x 8 memory chips, the higher order 3 address lines (out of 11 address lines) are connected to a decoder, and the output of the decoder will select a particular row. From the selected row, 1 chip of 256 x 8 will give/ take 8 bits of data , meeting the required size of 8 bits.

Numericals

A computer uses RAM chips of 256 X 4 capacity. Design a memory capacity of 1KB by using available chip.

Step 1: Find out , how many smaller size chips are required to meet the required size:

Divide the total required size/ Size of the smaller chip

1K X 8/ 256 x 4= 2^{10} X 2^3 / 2^8 X 2^2 = 2^3= 8 Chips

Step2 : Find, how many smaller size chips need to connect in paralell to meet the required data size.[finding the no of columns in the matrix arrangement.]

Here, in 1K x 8, 8 bits of data is required, and in 256 x 4 chip, 4 bits of data can be communicated from one location.

So, 2, 256 x 8 chip needs to be connected in parallel to meet the 8 bits data size.

i.e., we need to connect 256 x 4 chips in a matrix form., where no of columns will be= size of one location in the bigger size/ size of one location in the smaller size.

Number of columns= 8/4 = 2

Numerical : Design 1K X 8 bits memory using 256X4 bits memory chip.

Step 3: Find out the no of rows:

No of rows x No of Columns = Total no of elements

#rows x 2 =8

#rows = 4

How to connect the address lines:

For 1K x 8 , memory, 10($1K=2^{10}$) address lines are required, and for 256 x 4, 8 ($256=2^{8}$) address lines are required. So, out of 10 address lines, the 1st 8 lines will be connected to all the 256 x 4 memory chips.

Then to select a row, out of 8 rows of 256 x 4 memory chips, the higher order 2 address lines (out of 10 address lines) are connected to a decoder, and the output of the decoder will select a particular row. From the selected row, 2 chips of 256 x 4 will give/ take 4 bits of data each , meeting the required size of 8 bits.

Typical Memory Hierarchy

Registers are at the top of the hierarchy. Fastest storage element, present inside the CPU. But, limited in number. Access time < 0.5 ns

Level 1 Cache : On chip cache, designed using SRAM technology. Typical size is in the range::(8 – 64 KB) and access time: 1 ns

L2 Cache : Off chip cache. Typical size is in the range::(512KB – 8B) and access time: 3 to 10 ns

Main Memory: Designed using SRAM technology. Typical size is in the range::(8 – 16 GB) and access time: 50ns to 100 ns

Disk Storage: Serial access memory.Typical size is > 200 GB and access time: 5 – 10 ms

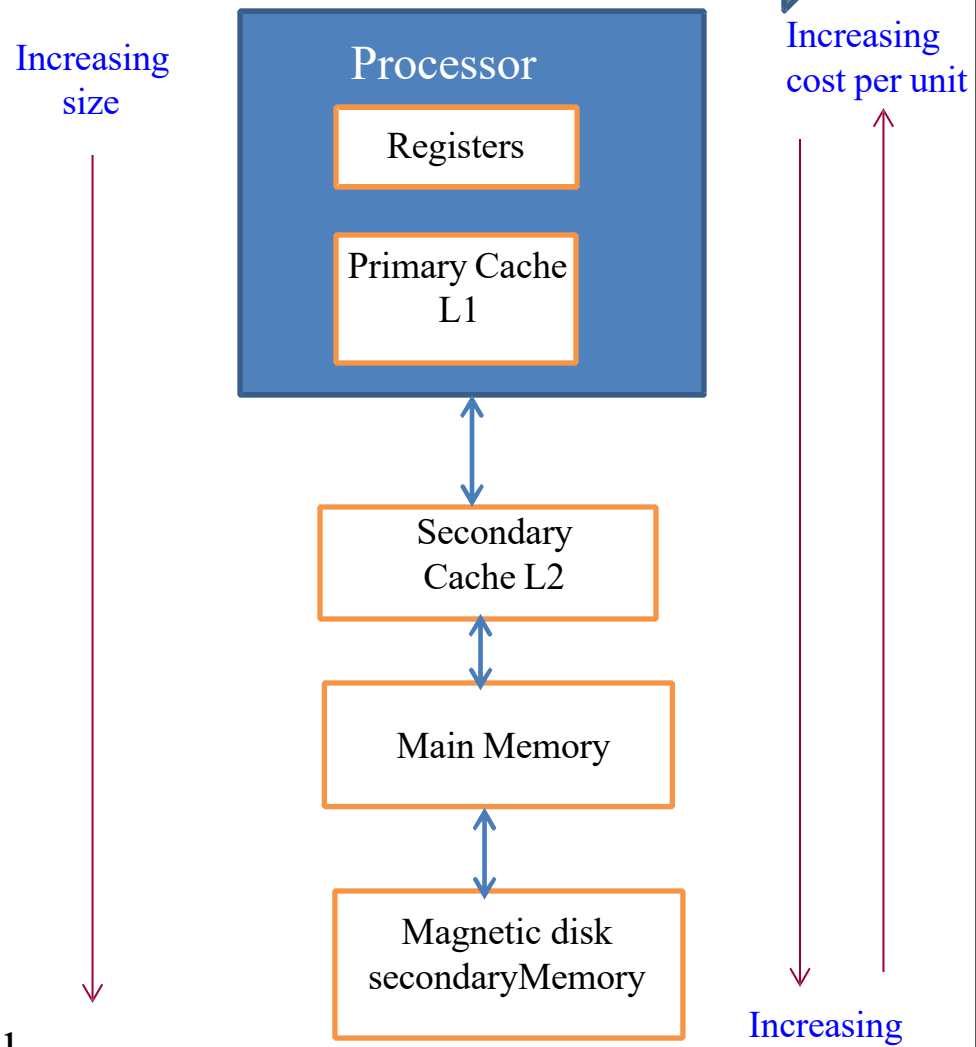


Fig. Memory Hierarchy

Why Cache Memory is required?

Processor is much faster than the main memory.

- As a result, the processor has to spend much of its time waiting while instructions and data are being fetched from the main memory.
- Major obstacle towards achieving good performance.

Speed of the main memory cannot be increased beyond a certain point.

Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is.

Cache memory is based on the property of computer programs known as **"locality of reference".**

Why Cache Memory has become possible?

Analysis of programs indicates that many instructions in localized areas of a program are executed repeatedly during some period of time, while the others are accessed relatively less frequently.

These instructions may be the ones in a loop, nested loop or few procedures calling each other repeatedly.

This is called **"locality of reference".**

Temporal locality of reference:

- Recently executed instruction is likely to be executed again very soon.
- If an instruction is executed at time instant t, then most likely the same instruction will be executed at time instant $t + \Delta t$.

Spatial locality of reference:

- Instructions with addresses close to a recently instruction are likely to be executed soon.
- If an instruction at address 'i' is executed at time instant t, then most likely the s instruction at address 'i+1' will be executed at time instant $t + \Delta t$.

What is a Cache Memory ?

Small and fast (SRAM) memory technology

Stores the subset of instructions & data currently being accessed.

Used to reduce average access time to memory.

Caches exploit temporal locality by …

Keeping recently accessed data closer to the processor.

Caches exploit spatial locality by …

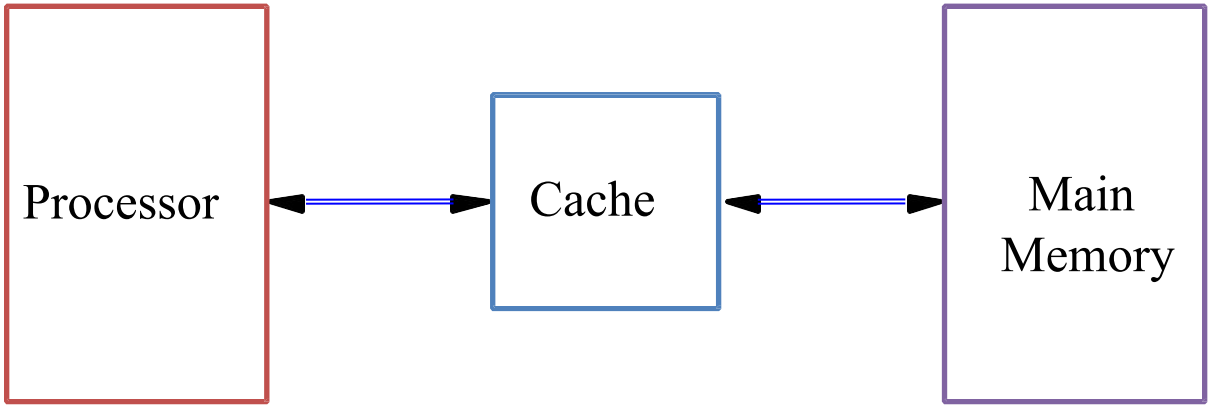
Moving blocks consisting of multiple contiguous words.

Goal is to achieve

Fast speed of cache memory access.

Balance the cost of the memory system.

The Basics of Caches



Processor issues a Read request, a block of words is transferred from the main memory to the cache, one word at a time.

Subsequent references to the data in this block of words are found in the cache.

At any given time, only some blocks in the main memory are held in the cache. **Mapping functions** determine where a main memory block will be placed in the cache.

When the cache is full, and a block of words needs to be transferred from the main memory some block of words in the cache must be replaced. This is determined by a **"replacement algorithm".**

The Basics of Caches: Cache Hit

Existence of a cache is transparent to the processor. The processor issues Read and Write requests in the same manner.

If the data is in the cache it is called a **Read or Write hit.**

Read hit:

- The data is obtained from the cache.

Write hit:

- Cache has a replica of the contents of the main memory.
- Contents of the cache and the main memory may be updated simultaneously.

  This is the **write-through** protocol.

- Update the contents of the cache, and mark it as updated by setting a bit known as the **dirty bit or modified** bit. The contents of the main memory are updated when this block is replaced. This is **write-back or copy-back** protocol.

The Basics of Caches: Cache Miss

If the data is not present in the cache, then a **Read miss or Write miss** occurs.

Read Miss:

1. Block of words containing the requested word is transferred from the memory. After the block is transferred, the desired word is forwarded to the processor. This is called **load-back.**

2. The desired word may also be forwarded to the processor as soon as it is transferred without waiting for the entire block to be transferred. This is called **load-through or early-restart.**

Block Placement

What is a block??

A block is a set of consecutive memory locations content.

When processor generates an address for any read/write operation, first we check whether that address's content is present in the cache memory or not.

If yes, then we perform our read/ write operation from the cache memory.

If not, then we bring the block containing the address that we are trying to access for our read/write operation from the main memory. i.e., the unit of transfer between main memory and cache is a block.

The main memory is bigger compared to the cache, hence the length of the main memory address is more compared to the cache memory's address.

In block placememt/ mapping functions we will see how to access a byte within a block in the cache memory, for a given main memory address.

Mapping functions

Mapping functions determine how memory blocks are placed in the cache.

Three mapping functions:

1. **Direct mapping**
2. **Associative mapping**
3. **Set-associative mapping.**

Direct-Mapped Cache

A memory address is divided into

Block address: identifies block in memory

Block offset: to access bytes within a block

A block address is further divided into

- Index: used for direct cache access
- Tag: most-significant bits of block address

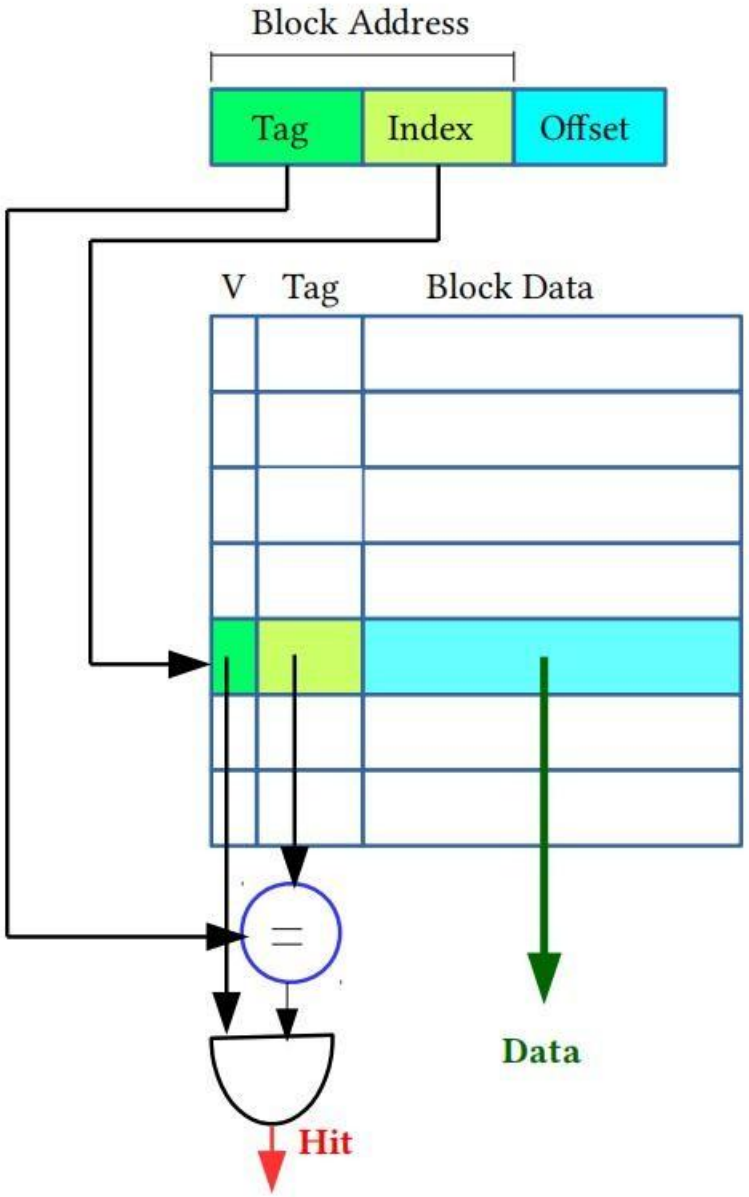
Index = Block Address* mod *Cache Blocks

Tag must be stored also inside cache

For block identification

A valid bit is also required to indicate

Whether a cache block is valid or not



Mapping functions

A simple processor example:

1. Cache consisting of 128 blocks of 16 words each.
2. Total size of cache is 2048 (2K) words.
3. Main memory is addressable by a 16-bit address.
4. Main memory has 64K words.
5. Main memory has 4K blocks of 16 words each.

Direct Mapping

Block j of the main memory maps to

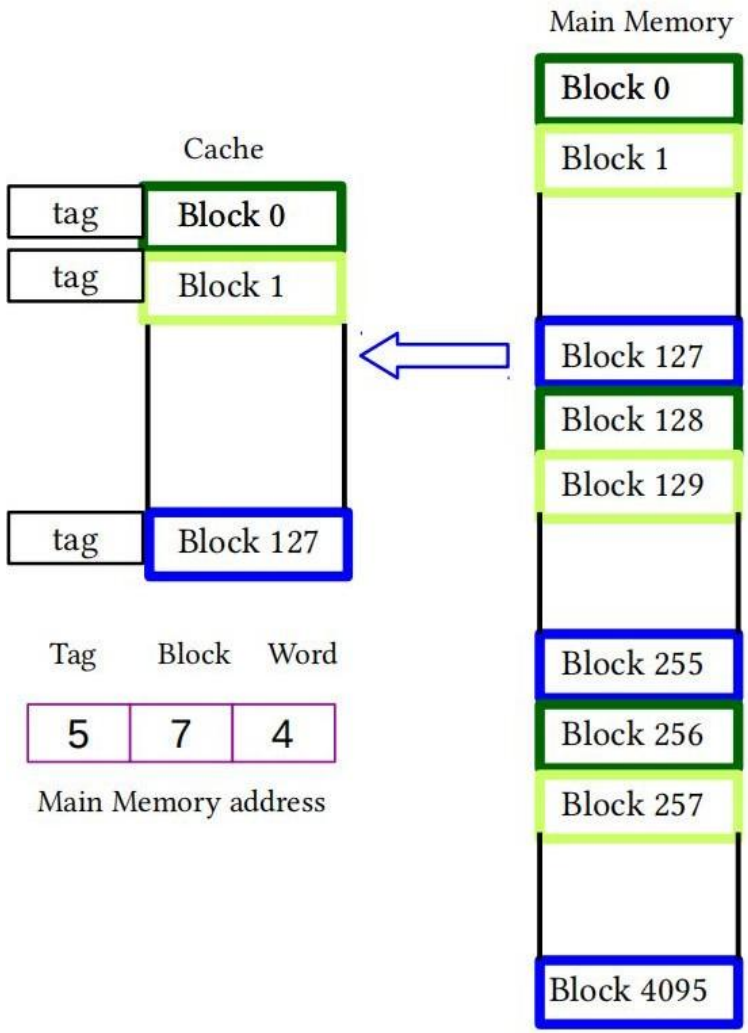
j modulo 128

So, 0 maps to 0, 129 maps to 1.

More than one memory block is mapped onto the same position in the cache.

May lead to **contention** for cache blocks even if the cache is not full.

Resolve the contention by allowing new block to replace the old block, leading to a **trivial** replacement algorithm.



Direct Mapping

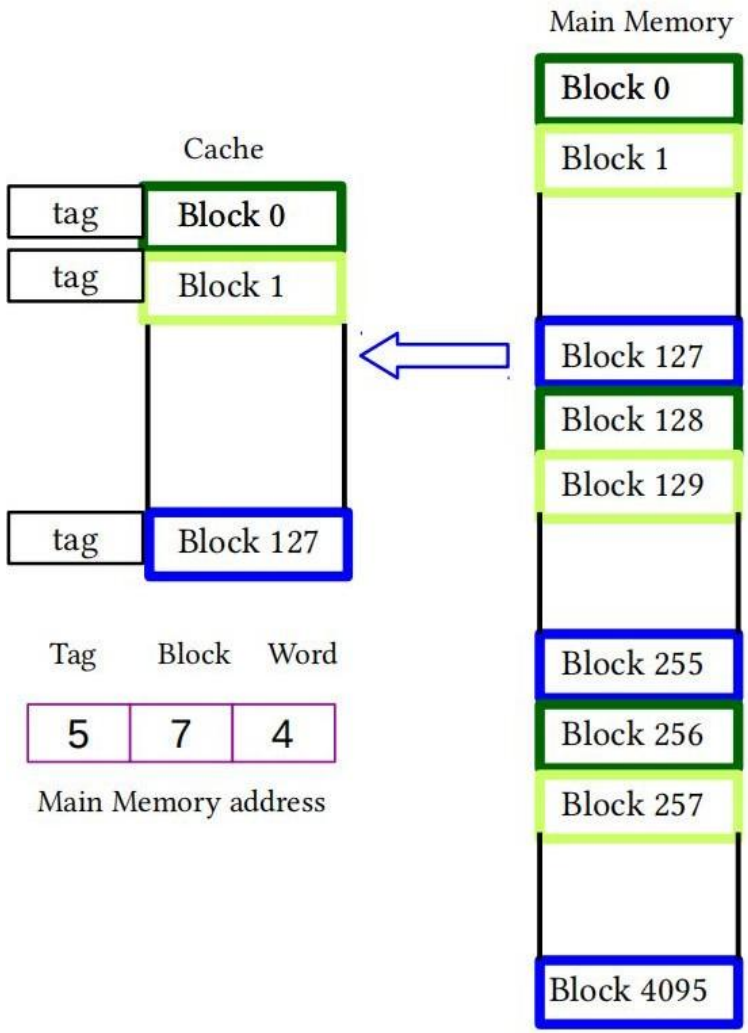
Memory address is divided into three fields:

1. Low order 4 bits determine one of the 16 words in a block.

2. When a new block is brought into the cache, the the next 7 bits determine which cache block this new block is placed in.

3. High order 5 bits determine which of the possible 32 blocks is currently present in the cache. These are tag bits.

Simple to implement but not very flexible.



Problem

A computer system uses 16-bit memory addresses. It has a 2K-byte cache organized in a direct-mapped manner with 64 bytes per cache block. Assume that the size of each memory word is 1 byte.

(a)Calculate the number of bits in each of the Tag, Block, and Word fields of the memory address.
(b)When a program is executed, the processor reads data sequentially from the following word addresses:

128, 144, 2176, 2180, 128, 2176

All the above addresses are shown in decimal values. Assume that the cache is initially empty. For each of the above addresses, indicate whether the cache access will result in a hit or a miss.

Problem: Solution

Block size = 64 bytes = 2^6 bytes = 2^6 words (since 1 word = 1 byte)
Therefore, Number of bits in the **Word field = 6**

Cache size = 2K byte = 2^{11} bytes
Number of cache blocks = Cache size / Block size = $2^{11}/2^6 = 2^5$
Therefore, Number of bits in the **Block field = 5**

Total number of address bits = 16
Therefore, Number of bits in the **Tag field = 16 - 6 - 5 = 5**

For a given 16-bit address, the 5 most significant bits, represent the Tag, the next 5 bits represent the Block, and the 6 least significant bits represent the Word.

Problem: Solution

b) The cache is initially empty. Therefore, all the cache blocks are **invalid.**

Access # 1:

Address = $(128)_{10}$ = $(0000000010000000)_2$

(Note: Address is shown as a 16-bit number, because the computer uses 16-bit addresses)

For this address, **Tag = 00000, Block = 00010, Word = 000000**

Since the cache is empty before this access, this will be a **cache miss**

After this access, **Tag field for cache block 00010 is set to 00000**

Problem: Solution

Access # 2:

Address = $(144)_{10} = (0000000010010000)_2$

For this address, **Tag = 00000, Block = 00010, Word = 010000**

Since tag field for cache block 00010 is 00000 before this access, this will be a **cache hit** (because **address tag = block tag)**

Problem: Solution

Access # 3:

Address = $(2176)_{10}$ = $(0000100010000000)_2$

For this address, **Tag = 00001, Block = 00010, Word = 000000**

Since tag field for cache block 00010 is 00000 before this access, this will be a **cache miss** (address tag ≠ block tag)

After this access, Tag field for **cache block 00010 is set to 00001**

Access # 4:

Address = $(2180)_{10}$ = $(0000100010000100)_2$

For this address, **Tag = 00001, Block = 00010, Word = 000100**

Since tag field for cache block 00010 is 00001 before this access, this will be a **cache hit**

(address tag=block tag)

Problem: Solution

Access # 5:

Address = $(128)_{10}$ = $(0000000010000000)_2$

For this address, **Tag = 00000, Block = 00010, Word = 000000**

Since tag field for cache block 00010 is 00001 before this access, this will be a **cache miss** (address tag ≠ block tag)

After this access, Tag field for **cache block 00010 is set to 00000**

Access # 6:

Address = $(2176)_{10}$ = $(0000100010000000)_2$

For this address, **Tag = 00001, Block = 00010, Word = 000000**

Since tag field for cache block 00010 is 00001 before this access, this will be a **cache miss** (address tag ≠block tag)**.** After this access, Tag field for **cache block 00010 is set to 00001**

Numericals on Direct Mapping

Direct Mapping Question: Assume a computer has 32 bit addresses. Each block stores 16 words. A direct-mapped cache has 256 blocks. In which block (line) of the cache would we look for each of the following addresses? Addresses are given in hexadecimal for convenience.
a. 1A2BC012 b. FFFF00FF c. 12345678 d. C109D532

Block size = 16 words = 2^4 words
Therefore, Number of bits in the **Word field = 4**

Number of cache blocks = 2^8
Therefore, Number of bits in the **Block field = 8**

Hence, in the 32 bit main memory address, the 1st lower order 4 bits (1st lower order hexadecimal digit represents the offset of the word inside a block).

Then the next 8bits (2nd and 3rd lower order hexadecimal digit represents the block no to which the incoming address will be mapped to).

So, in the address 1A2BC012, 01 is the block no in the cache, to which this address will be mapped to ie. we will check block no 01 in the cache for a hit or a miss.

Numericals on Mapping

Direct Mapping Question: Assume a computer has 32 bit addresses. Each block stores 16 words. A direct-mapped cache has 256 blocks. In which block (line) of the cache would we look for each of the following addresses? Addresses are given in hexadecimal for convenience.
a. 1A2BC012 b. FFFF00FF c. 12345678 d. C109D532

So, in the address FFFF00FF, 0F is the block no in the cache, to which this address will be mapped to, i.e., we will check block no 0F in the cache for a hit or a miss.

So, in the address 12345678, 67 is the block no in the cache, to which this address will be mapped to, i.e., we will check block no 67 in the cache for a hit or a miss.

So, in the address C109D532 , 53 is the block no in the cache, to which this address will be mapped to, i.e., we will check block no 53 in the cache for a hit or a miss.

Numericals

Consider a direct mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find the number of bits in tag field and the size of the tag directory.

Solution:

Assumption: The memory is byte addressable.
The size of the main memory= 128 KB = 2^{17}B.
Hence, the no of bits in the main memory address is 17 bits.

Block size = 256 Bytes = 2^8 Bytes
Therefore, Number of bits in the **Word field = 8**

# of Blocks in the cache =Size of the cache memory/ Block size
= 16KB / 256B = $2^{14}/2^8 = 2^6$
i.e., 6 bits are required to represent a block no in the cache

# of tag bits = total address length - (block field length + word field length)
=17-(6+8)
=3 bits

Numericals

Consider a direct mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find the number of bits in tag field and the size of the tag directory.

Solution:

For each block in the cache, tag bits are stored.

Size of the tag directory = no of blocks in the cache x no of bits in the tag field.

= 64 x 3 bits

= 192 bits

= 24 Bytes.

Numericals

Consider a direct mapped cache of size 256 KB with block size 1 KB. There are 8 bits in the tag. Find the size of main memory and the size of the tag directory.

Solution:

Assumption: The memory is byte addressable.

Block size = 1K B = 2^{10} Bytes

Therefore, Number of bits in the **Word field = 10**

# of Blocks in the cache =Size of the cache memory/ Block size

= 256KB / 1KB = $2^{18}/2^{10} = 2^8$

i.e., 8 bits are required to represent a block no in the cache

# of tag bits = 8 bits

total address length# of tag bits = # of tag bits + block field length + word field length

=8 + 8 + 10 = 26 bits

The size of main memory = 2^{26} bytes = 64MB

Size of the tag directory = no of blocks in the cache x no of bits in the tag field.

= 256 x 8 bits

= 256 Bytes

Numericals

Consider a direct mapped cache with block size 2 KB. The size of main memory is 64 GB and there are 10 bits in the tag. Find the size of cache memory and the size of tag directory.

Solution:

Assumption: The memory is byte addressable.

Block size = 2KB = 2^{11} Bytes

Therefore, Number of bits in the **Word field = 11**

The size of the main memory= 64 GB = 2^{36}B.

Hence, the no of bits in the main memory address is 36 bits.

# of tag bits = 10 bits

block field length = total address length - (# of tag bits+ word field length) = 36-(10+11)
=15 bits

The size of cache memory = # of blocks x Size of each block = 2^{15} x 2^{11} Bytes =2^{26} B
=64MB

Size of the tag directory = no of blocks in the cache x no of bits in the tag field.
= 2^{15} x 10 bits
= 40960 Bytes

Numericals

Consider a machine with a byte addressable main memory of 2^{32} bytes divided into blocks of size 32 bytes. Assume that a direct mapped cache having 1K cache lines is used with this machine. What is the size of the tag field ?

Solution:

Assumption: The memory is byte addressable.

Block size = 32 bytes = 2^5 Bytes

Therefore, Number of bits in the **Word field = 5**

# of Blocks in the cache =1K =2^{10}

i.e., 10 bits are required to represent a block no in the cache

The size of the main memory= 2^{32}B.

Hence, the no of bits in the main memory address is 32 bits.

# of tag bits = total address length - (block field length + word field length)

=32-(10+5)

=17 bits

Numericals

An 16 KB direct-mapped write back cache is organized as multiple blocks, each of size 16 bytes. The processor generates 32 bit addresses. The cache controller maintains the tag information for each cache block comprising of the following-

1 valid bit, 1 modified bit 2 replacement bits and as many bits as the minimum needed to identify the memory block mapped in the cache.

What is the total size of memory needed at the cache controller to store meta data (tags) for the cache?

Solution:

Assumption: The memory is byte addressable.

Block size = 16 bytes = 2^4 Bytes

Therefore, Number of bits in the **Word field = 4**

# of Blocks in the cache =Size of the cache memory/ Block size

= 16KB / 16B = $2^{14}/2^4 = 2^{10}$

i.e., 10 bits are required to represent a block no in the cache

# of tag bits = total address length - (block field length + word field length)

=32-(10+4)

=18 bits

Numericals

An 16 KB direct-mapped write back cache is organized as multiple blocks, each of size 16 bytes. The processor generates 32 bit addresses. The cache controller maintains the tag information for each cache block comprising of the following-

1 valid bit, 1 modified bit 2 replacement bits and as many bits as the minimum needed to identify the memory block mapped in the cache.

What is the total size of memory needed at the cache controller to store meta data (tags) for the cache?

Solution:

Size of the memory required to store the meta data= # of blocks in the cache x (tag bits + 1valid bit + 1 modified bit + 2 replacement bits)

$= 2^{10}$ x (18+ 1+1 +2) bits

$= 2^{10}$ x 22 bits

= 2816 Bytes

Fully Associative Cache

A block can be placed anywhere in the cache.

i.e., there is no fixed/specific position in the cache, for a block in the main memory.

In this mapping, replacement of a block is not required in the cache, until and unless cache is completely full.

But to find a main memory block i, in the cache, tag of the incoming block need to be compared with all the tags stored in the cache. Searching time is more.

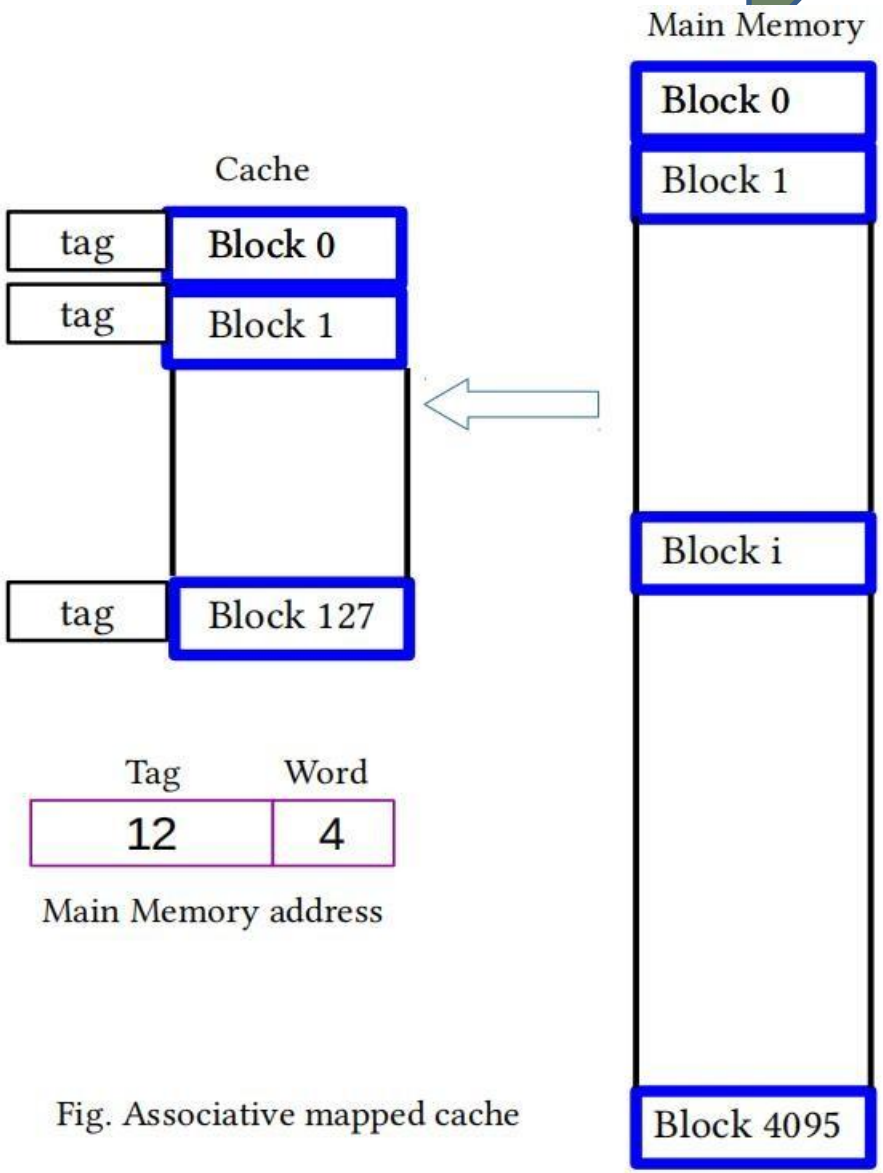


Fig. Associative mapped cache

Set-Associative Cache

Here, the position of a main memory block is not completely fixed and it cannot occupy any free block in the cache as well.

A main memory block can occupy any of the free blocks from a specific set of blocks.

Cache memory is organized as a set of blocks. there may be 2 blocks in set/ 4 blocks per set/ m number of blocks per set. The number of blocks present in a set is called as a way number.

Here a main memory block i will be mapped to a specific set j, where

j= i mod (# no of sets in the cache)

The main memory block i will be mapped to any of the blocks present in the set j.

To find a main memory block i, in the cache, the tag of block i will be compared ony with the tags of the blocks present in the set j.

Set-Associative Cache

Replacement of a block is required only when all the blocks in a set is occupied and an incoming block is mapped into that set.

For direct mapped cache, way number is 1.

For associative mapped cache, way number is the number of blocks in the cache.

Here, main memory address will be divided into 3 parts: tag, set, word.

To find the bits in the set field, we need to find the no of sets in the cache: no of blocks in the cache/ way no. Then expressing that result in the powers of 2 and taking the exponent as the length of the set field.

To find the bits in the tag field, we need to find the number of blocks in the main memory/the no of sets in the cache. Then expressing that result in the powers of 2 taking the exponent as the length of the tag field.

Set Associative Mapped Cache

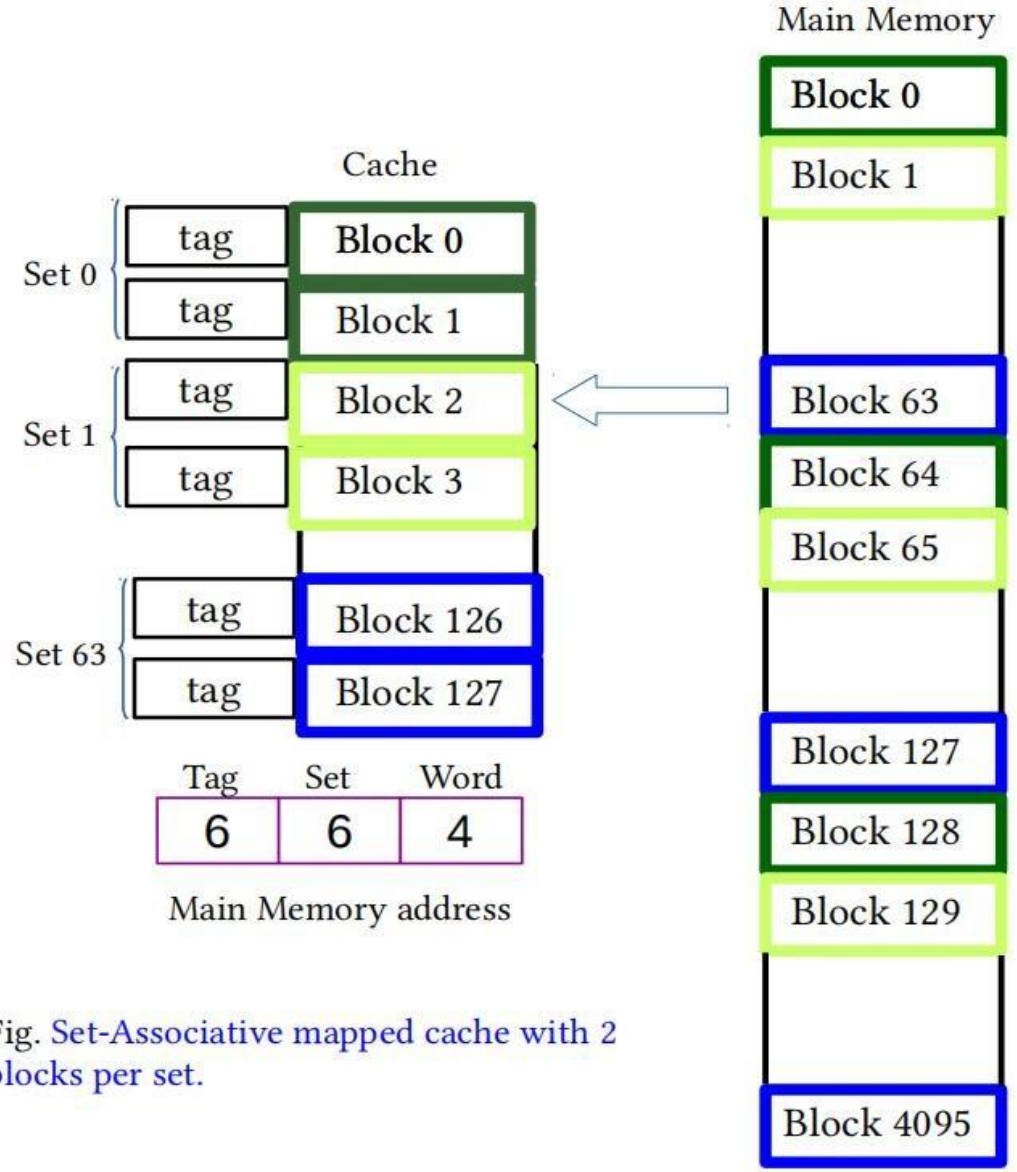


Fig. Set-Associative mapped cache with 2 blocks per set.

Problem

A computer system uses 16-bit memory addresses. It has a 2K-byte cache organized in a 2-wat set associative manner with 64 bytes per cache block. Assume that the size of each memory word is 1 byte.

(a)Calculate the number of bits in each of the Tag, set, and Word fields of the memory address
(b)When a program is executed, the processor reads data sequentially from the following word addresses:

128, 144, 2176, 2180, 128, 2176

All the above addresses are shown in decimal values. Assume that the cache is initially empty For each of the above addresses, indicate whether the cache access will result in a hit or a miss.

Problem: Solution

Block size = 64 bytes = 2^6 bytes = 2^6 words (since 1 word = 1 byte)
Therefore, Number of bits in the **Word field = 6**

Cache size = 2K-byte = 2^{11} bytes
Number of cache blocks = Cache size / Block size = $2^{11}/2^6 = 2^5$
Number of cache sets = total no of Cache blocks / way no = $2^5/2 = 2^4$

Therefore, Number of bits in the **Set field = 4**

Total number of address bits = 16
Therefore, Number of bits in the **Tag field = 16 - 6 - 4 = 6**

For a given 16-bit address, the 6 most significant bits, represent the Tag, the next 4 bits represent the Set, and the 6 least significant bits represent the Word.

Problem: Solution

b) The cache is initially empty. Therefore, all the cache blocks are **invalid.**

Access # 1:

Address = $(128)_{10}$ = $(0000000010000000)_2$

(Note: Address is shown as a 16-bit number, because the computer uses 16-bit addresses)

For this address, **Tag = 000000, Set = 0010, Word = 000000**

Since the cache is empty before this access, this will be a **cache miss**

After this access, **Tag field for the first block in the cache set 0010 is set to 000000**

Problem: Solution

Access # 2:

Address = $(144)_{10} = (0000000010010000)_2$

For this address, **Tag = 000000, Set = 0010, Word = 010000**

Since tag field for the first cache block in the set 0010 is 00000 before this access, this will be a **cache hit** (because **address tag = block tag)**

Problem: Solution

Access # 3:

Address = $(2176)_{10}$ = $(0000100010000000)_2$

For this address, **Tag = 000010, Set = 0010, Word = 000000**

The tag field for this address does not match the tag field for the first block in set 0010. The second block in set 0010 is empty. Therefore, this access will be a cache **miss.** After this access, Tag field for the second block **in set 0010 is set to 000010**

Access # 4:

Address = $(2180)_{10}$ = $(0000100010000100)_2$

For this address, **Tag = 000010, Set = 0010, Word = 000100**

Since tag field for the 2nd cache block in the set 0010 is 00001 before this access, this will be a **cache hit** (address tag = block tag)

Problem: Solution

Access # 5:

Address = $(128)_{10} = (0000000010000000)_2$

For this address, **Tag = 000000, Set = 0010, Word = 000000**

The tag field for this address matches the tag field for the first block in set 0010. Therefore, this access will be a cache **hit.**

Access # 6:

Address = $(2176)_{10} = (0000100010000000)_2$

For this address, **Tag = 000010, Set = 0010, Word = 000000**

The tag field for this address matches the tag field for the second block in set 0010. Therefore, this access will be a cache **hit.**

Numericals

Consider a fully associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find the number of bits in tag field and the size of the tag directory.

Solution:

Assumption: The memory is byte addressable.

The size of the main memory= 128 KB = 2^{17}B.

Hence, the no of bits in the main memory address is 17 bits.

Block size = 256 Bytes = 2^8 Bytes

Therefore, Number of bits in the **Word field = 8**

# of tag bits = total address length - word field length

=17- 8

=9 bits

Size of the tag directory = no of blocks in the cache x no of bits in the tag field

# of Blocks in the cache =Size of the cache memory/ Block size

= 16KB / 256B = $2^{14}/2^8 = 2^6$

Size of the cache directory = 64 x 9 bits = 72Bytes

Numericals

Consider a fully associative cache of size 256 KB with block size 1 KB. There are 18 bits in the tag. Find the size of main memory and the size of the tag directory.

Solution:

Assumption: The memory is byte addressable.

Block size = 1K B = 2^{10} Bytes

Therefore, Number of bits in the **Word field = 10**

# of Blocks in the cache =Size of the cache memory/ Block size

= 256KB / 1KB = $2^{18}/2^{10} = 2^8$

i.e., 8 bits are required to represent a block no in the cache

# of tag bits = 18 bits

total address length = # of tag bits + word field length

=18 + 10 = 28 bits

The size of main memory = 2^{28} bytes = 256MB

Size of the tag directory = no of blocks in the cache x no of bits in the tag field.

= 256 x 8 bits

= 256 Bytes

Numericals

Consider a fully associative mapped cache with block size 4 KB. The size of main memory is 16 GB. Find the number of bits in tag.

Solution:

Assumption: The memory is byte addressable.

Block size = 4KB = 2^{12} Bytes

Therefore, Number of bits in the **Word field = 12**

The size of the main memory= 16 GB = 2^{34}B.

Hence, the no of bits in the main memory address is 34 bits.

# of tag bits = total address length - word field length = 34 - 12

=22 bits

Numericals

Cache/Memory Layout: A computer has an 8 GByte memory with 64 bit word sizes. Each block of memory stores 16 words. The computer has a direct-mapped cache of 128 blocks. The computer uses word level addressing. What is the address format? If we change the cache to a 4-way set associative cache, what is the new address format?

Solution:

Assumption: The memory is WORD addressable.

Block size = 16 words = 2^4 words

Therefore, Number of bits in the **Word field = 4**

The size of the main memory= 8 GB = 2^{33}B.

Hence, the no of bits in the main memory address is 33 bits.

# of blocks in the cache = 128 = 2^7

Main memory address : 33 bits.

word field : 4 bits

block field : 7 bits

Tag field : 33- (4+7) bits = 22 bits

Numericals

Cache/Memory Layout: A computer has an 8 GByte memory with 64 bit word sizes. Each block of memory stores 16 words. The computer has a direct-mapped cache of 128 blocks. The computer uses word level addressing. What is the address format? If we change the cache to a 4-way set associative cache, what is the new address format?

Solution: Part 2: For 4-way set associative cache

Assumption: The memory is WORD addressable.

Block size = 16 words = 2^4 words

Therefore, Number of bits in the **Word field = 4**

The size of the main memory= 8 GB = 2^{33}B.

Hence, the no of bits in the main memory address is 33 bits.

# of blocks in the cache = 128 = 2^7

# of sets in the cache = 2^7 / 4 = 2^5

Main memory address : 33 bits.

word field : 4 bits

set field: 5 bits

Tag field : 33- (4+5) bits = 24 bits

Numericals

A two-way set associative cache memory uses block of 4 words. The cache can have a total of 2048 words from main memory. The main memory size is 128K X 32.

i)Draw the format of main memory address.ii)What is the size of cache with tag bits

Solution:

Assumption: The memory is WORD addressable.

Block size = 4 words = 2^2 words

Therefore, Number of bits in the **Word field = 2**

The size of the main memory= 128K x 32 = 2^{17}words.

Hence, the no of bits in the main memory address is 17 bits.

# of blocks in the cache = No of words / block size = 2^{11} / 2^2 = 2^9

# of sets in the cache = 2^9 / 2 = 2^8

Main memory address : 17 bits.

word field : 2 bits

set field : 8 bits

Tag field : 17- (2+8) bits = 7 bits

The size of cache with tag bits = No of blocks x no of tag bits x block size

= 2^9 x 7 x 4 words

=14336 words

Numericals

Consider a 4-way set associative mapped cache of size 16 KB with block size 512 bytes. The size of main memory is 256 KB. Find the number of bits in tag field and the size of the tag directory.

Solution:

Assumption: The memory is byte addressable.

Block size = 512 B = 2^9 Bytes

Therefore, Number of bits in the **Word field = 9**

# of Blocks in the cache =Size of the cache memory/ Block size

= 16KB / 512B = $2^{14}/2^9 = 2^5$

# of Sets in the cache =No of blocks in the cache memory/ Way no(# of blocks in a set)

= $2^5/4 = 2^3$

The size of main memory = 256KB = 2^{18} bytes

total address length = # of tag bits + # of set bits + word field length

# of tag bits = 18 -(3+9) = **6 bits**

Size of the tag directory = no of blocks in the cache x no of bits in the tag field.

= 32 x 6 bits

= **24 Bytes**

Numericals

Consider a 4-way set associative mapped cache of size 512 KB with block size 1 KB. There are 9 bits in the tag. Find the size of main memory and the size of the tag directory

Solution:

Assumption: The memory is byte addressable.

Block size = 1KB = 2^{10} Bytes

Therefore, Number of bits in the **Word field = 10**

# of Blocks in the cache =Size of the cache memory/ Block size

= 512KB / 1KB = $2^{19}/2^{10} = 2^9$

# of Sets in the cache =No of blocks in the cache memory/ Way no(# of blocks in a set)

= $2^9/4 = 2^7$

total address length = # of tag bits + # of set bits + word field length

= 9+ 7+ 10 = 26 bits

The size of main memory = 2^{26} bytes = **64MB**

Size of the tag directory = no of blocks in the cache x no of bits in the tag field.

= 512 x 9 bits

= **576 Bytes**

Numericals

Consider a 4-way set associative mapped cache with block size 4 KB. The size of main memory is 16 GB and there are 10 bits in the tag. Find the size of cache memory and the size of the tag directory.

Solution:

Assumption: The memory is byte addressable.

Block size = 4 KB = 2^{12} Bytes

Therefore, Number of bits in the **Word field = 12**

The size of main memory = **16GB** = 2^{34} bytes

total address length = # of tag bits + # of set bits + word field length

# of set bits = 34 - (10 + 12) = 12 bits

Size of the cache memory = # of sets in the cache x way no x size of a block

= 2^{12} x 4 x 2^{12} Bytes = 2^{26} Bytes = **64MB**

Size of the tag directory = no of blocks in the cache x no of bits in the tag field.

= 2^{12} x 4 x 10 bits

= **20,480 Bytes**

Numericals on Mapping

A cache consists of a total of 128 blocks. The main memory contains 2K blocks, each consisting of 32 words.

(I)How many bits are there in each of the TAG, BLOCK and WORD field in case of direct mapping?

(ii)How many bits are there in each of the TAG, SET, and WORD field in case of 4-way set-associative mapping?

Solution:

Assumption: The memory is word addressable.

Block size = 32 words = 2^5 words

Therefore, Number of bits in the **Word field = 5**

# of Blocks in the cache =128 =2^7

i.e., 7 bits are required to represent a block no in the cache

# of Blocks in the main memory =2K =2^{11}

To find the tag bits, # of Blocks in the main memory/ # of Blocks in the cache

= 2^{11}/ 2^7 = 2^4

The no of **tag bits=4**

Numericals on Mapping

A cache consists of a total of 128 blocks. The main memory contains 2K blocks, each consisting of 32 words.

(I)How many bits are there in each of the TAG, BLOCK and WORD field in case of direct mapping?

(ii)How many bits are there in each of the TAG, SET, and WORD field in case of 4-way set-associative mapping?

Solution: 2nd Part

Assumption: The memory is word addressable.

Block size = 32 words = 2^5 words

Therefore, Number of bits in the **Word field = 5**

# of Blocks in the cache =128 =2^7

# of sets in the cache =# of Blocks in the cache/ way no = $2^7/4 = 2^5$

i.e., 5 bits are required to represent a set no in the cache

# of Blocks in the main memory =2K =2^{11}

To find the tag bits, # of Blocks in the main memory/ # of sets in the cache

$2^{11}/ 2^5 = 2^6$

The no of **tag bits = 6**

Numericals

Consider a 16-way set associative mapped cache. The size of cache memory is 512 KB and there are 11 bits in the tag. Find the size of main memory.

Solution:

Assumption: The memory is byte addressable.
Let the no of sets in the cache = 2^x Bytes
Let the size of a block = 2^y Bytes

Size of the cache memory = No of sets x way no x block size

$$2^{19} = 2^x \times 2^4 \times 2^y$$

$$\Rightarrow x+y = 15 \ldots\ldots\ldots\ldots\ldots\ldots <1>$$

Length of the main memory address = No of tag bits + bits in the set field + no of bits in word field

$$= 11 + x+y$$

Now replacing the value of x+y from equation <1 > , we get
Length of the main memory address = 11+ 15 = **26 bits**
Size of the main memory = 2^{26} Bytes = **64MB**

Numericals

Consider a 4-way set associative mapped cache. The size of main memory is 64 MB and there are 11 bits in the tag. Find the size of cache memory.

Solution:

Assumption: The memory is byte addressable.
Let the no of sets in the cache = 2^x Bytes
Let the size of a block = 2^y Bytes

Size of the main memory = 64MB = 2^{26} B

total address length = # of tag bits + # of set bits + word field length

$26 = 10 + x + y$

$\Rightarrow x+y = 15$.....<1>

Size of the cache memory = No of sets x way no x block size

$= 2^x \times 2^2 \times 2^y$

$= 2^{(2+x+y)}$

Now replacing the value of x+y from equation <1 > , we get

Size of the cache memory $=2^{(2+15)}$Bytes

$= 2^{17}$ Bytes = 128 KB

Numericals

A computer has a 256 KB, 4-way set associative, write back data cache with block size of 32 bytes. The processor sends 32 bit addresses to the cache controller. Each cache tag directory entry contains in addition to address tag, 2 valid bits, 1 modified bit and 1 replacement bit. What is the width if tag field and the size of the tag directory?

Solution:

Assumption: The memory is byte addressable.

Block size = 32B = 2^5 Bytes

Therefore, Number of bits in the **Word field = 5**

# of Sets in the cache =No of blocks in the cache memory/ Way no(# of blocks in a set)

= (Size of the cache memory/ Block size) / Way no

= (256 KB / 32 B) / 4

= ($2^{18}/2^5$) /2^2 = $2^{13}/2^2$ =2^{11}

total address length = # of tag bits + # of set bits + word field length

# of TAG bits = 32 - (11 + 5) = 16 bits

Width of tag field = address tag bits + 2 valid bits + 1 modified bit + 1 replacement bit

= 16 + 2 + 1 +1 = 20bits

Size of the tag directory = no of blocks in the cache x no of bits in the tag field.

= 2^{13} x 20 bits

= **20,480 Bytes**

Numericals

Consider a direct mapped cache with 8 cache blocks (0-7). If the memory block requests are in the order-

3, 5, 2, 8, 0, 6, 3, 9, 16, 20, 17, 25, 18, 30, 24, 2, 63, 5, 82, 17, 24

Which of the following memory blocks will not be in the cache at the end of the sequence?

3 18 20 30 Also, calculate the hit ratio and miss ratio.

Numericals

Consider a fully associative cache with 8 cache blocks (0-7). The memory block requests are in the order-

4, 3, 25, 8, 19, 6, 25, 8, 16, 35, 45, 22, 8, 3, 16, 25, 7

If LRU replacement policy is used, which cache block will have memory block 7?

Also, calculate the hit ratio and miss ratio.

Numericals

Consider a 4-way set associative mapping with 16 cache blocks. The memory block requests are in the order-

0, 255, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155

If LRU replacement policy is used, which cache block will not be present in the cache?

3
8
129
216

Also, calculate the hit ratio and miss ratio.

Hit Rate and Miss Penalty

Hit Rate= Number of Successful attempts/Number of total attempts

Miss Rate= Number of Unsuccessful attempts/Number of total attempts

Miss Penalty= time required to bring desired information to cache

Average Access Time= t_{avg}= hc +(1-h)M

h=hit ratio

c= time to access the cache memory

M= Miss Penalty

Numericals

What is the hit ratio of a cache memory if cache memory access time is 30ns and main memory access time is 150ns and average access time is 42ns?

Solution:

Average Access Time= t_{ave}= hc +(1-h)M

Given, c=30ns, t_{avg}= 42ns, M=150ns

t_{avg}= hc +(1-h)M

42= h x 30 + (1-h) x 150

h= 0.9

Numericals

In a cache organization if the cache memory has an access time of 8nsec and hit rate as 0.98, then find out Average Memory Access time (AMAT) for the whole arrangement. Assume the access time for the main memory is 1 .0 msec

Solution:

Average Access Time= t_{ave}= hc +(1-h)M

Given, c=8ns, h= .98, M=1ms = 1000ns

t_{avg}= hc +(1-h)M

= (0.98 x 8 + 0.02 x 1000) ns

t_{avg}= 27.84 ns

Numericals

Consider a computer C1, with no cache memory , that takes 10 cc to read from memory.

Consider another computer C2, with cache memory and interleaved main memory, which takes 17cc to transfer a block from memory to cache, on a cache miss.

It is found that 30% instructions executed are data reference instructions. hit ratio of instruction cache is 95% and data cache is 90%. cache memory access time is 1cc for both the caches.

Find the **improvement in performance due to use of caches over the non cached one.**

Solution:

30% instructions are data reference instructions, i.e., when we have executed 100 instructions, 30 are again referring to memory for data access.

So, total memory references = 100 + 30

When we are executing 100 instructions, for each of them too we are refererring to memory for instruction fetch

e taken = Number of accesses x Time taken for each access

= 130 x 10CC = 1300CC

Numericals

Consider a computer C1, with no cache memory , that takes 10 cc to read from memory.

Consider another computer C2, with cache memory and interleaved main memory, which takes 17cc to transfer a block from memory to cache, on a cache miss.

It is found that 30% instructions executed are data reference instructions. hit ratio of instruction cache is 95% and data cache is 90%. cache memory access time is 1cc for both the caches.

Find the **improvement in performance due to use of caches over the non cached one.**

Solution:

With Cache time taken,

tav = 100(0.95 x 1 + 0.05 x 17) + 30 (0.90 x 1 + 0.1 x 17) CC = 258CC

Improvement = Time Taken with No cache / Time Taken with cache

=1300/258

Improvement = 5.04

The Average Access Time with Two Levels of Caches

$t_{avg} = h_1C_1 + (1-h_1)\text{Miss Penalty}$

$t_{avg} = h_1C_1 + (1-h_1)[\ h_2C_2 + (1-h_2)\ M]$

$t_{avg} = h_1C_1 + (1-h_1)\ h_2C_2 + (1-h_1)(1-h_2)\ M]$

Numericals

Consider a two level memory system such that the level-1 having hit ratio 75%. The level-1 memory is 20 times faster than level-2 memory. The average access time of level-2 memory is 52ns. If the average access time of level-2 memory is changed or increased by 20% of 52ns. Compute the followings.

(i) What is the access time of level-1 memory?

(ii)What is the new hit ratio?

(iii)What is the percentage of change in hit ratio?

Solution:

Given, h=0.75 T_{avg}=52ns , $t_2=20t_1$

$t_{avg}= ht_1 +(1-h)t_2$

$52= 0.75t_1 + 0.25 \times 20\ t_1$

t1=9.04ns

t_2= 180.8ns

Numericals

Consider a two level memory system such that the level-1 having hit ratio 75%. The level-1 memory is 20 times faster than level-2 memory. The average access time of level-2 memory is 52ns. If the average access time of level-2 memory is changed or increased by 20% of 52ns. Compute the followings.

(i) What is the access time of level-1 memory?

(ii)What is the new hit ratio?

(iii)What is the percentage of change in hit ratio?

Solution:

$T_{avg2}=T_{avg1}$+20% of 52=52+ 10.4=62.4ns

$t_{avg2}= h't_1 +(1-h')t_2$

62.4= h' x 9.04 + (1-h') x 180.8

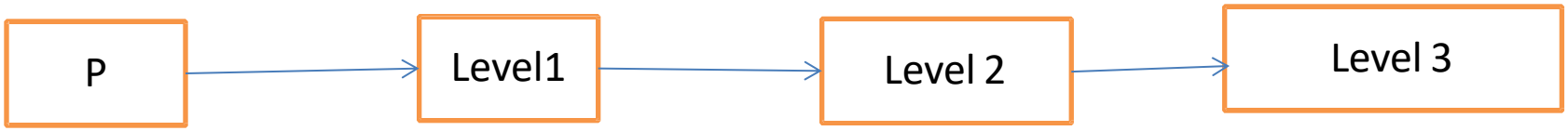
Percentage of change in hit ratio= (old-new)/old x 100%

Percentage of change in hit ratio= (0.75-0.689)/.75 x 100%

= 8.1 %

Numericals

Consider a three level memory system with access times per word 20ns, 40ns, 100ns. Hit ratios are 0.7, 0.8 and 1 respectively. If the referred word is not available in level1 get the two word block from level2 to level1 and supply the desired word to the processor. If it is not available in level2 then get a 4 word block from level3 to level2 and transfer the associated block from level2 to level1. Handover the desired word to processor from level1. what is the average access time?



$t_{avg} = h_1 t_1 + (1-h_1)[h_2(t_B+t_1)+(1-h_2)(t_B'+t_B+t_1)]$

Numericals

Consider a three level memory system with access times per word 20ns, 40ns, 100ns. Hit ratios are 0.7, 0.8 and 1 respectively. If the referred word is not available in level1 get the two word block from level2 to level1 and supply the desired word to the processor. If it is not available in level2 then get a 4 word block from level3 to level2 and transfer the associated block from level2 to level1. Handover the desired word to processor from level1. what is the average access time?

Solution:

h_1=.7, h_2=.8

t_1= 20ns, t_B= 40ns x 2= 80ns, t_B' = 100ns x 4 =400ns

$t_{avg} = h_1 t_1 + (1-h_1)[h_2(t_B+t_1)+(1-h_2)(t_B'+t_B+t_1)]$

t_{avg}= 0.7 x 20 + 0.3 [0.8(80+ 20) + 0.2(400 + 80 + 20)] ns

Thank You