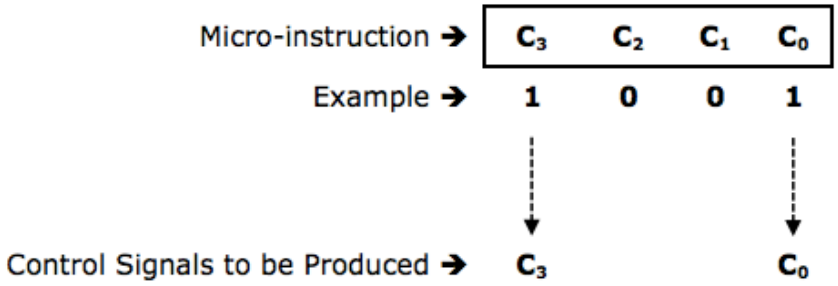
MICRO-INSTRUCTION FORMAT

The main part of the micro-instruction is its control field.
It determines the control signals to be produced.
It can be of two different formats: Horizontal or Vertical.

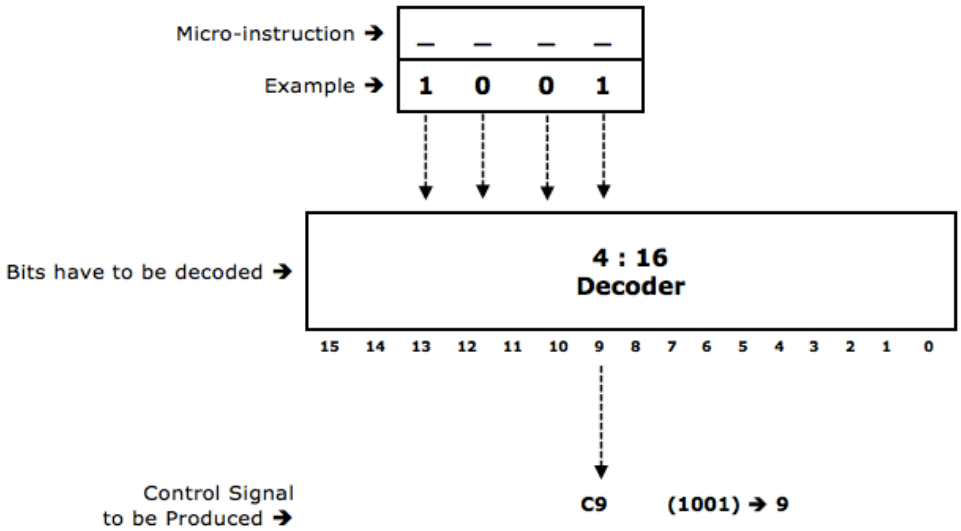
1) HORIZONTAL MICRO-INSTRUCTION

Here every bit of the micro-instruction corresponds to a control signal.
Whichever bit is "1", that particular control signal will be produced by the micro-instruction.



2) VERTICAL MICRO-INSTRUCTION

Here bits of the micro-instruction have to be decoded.
The decoded output decides the control signal to be produced.

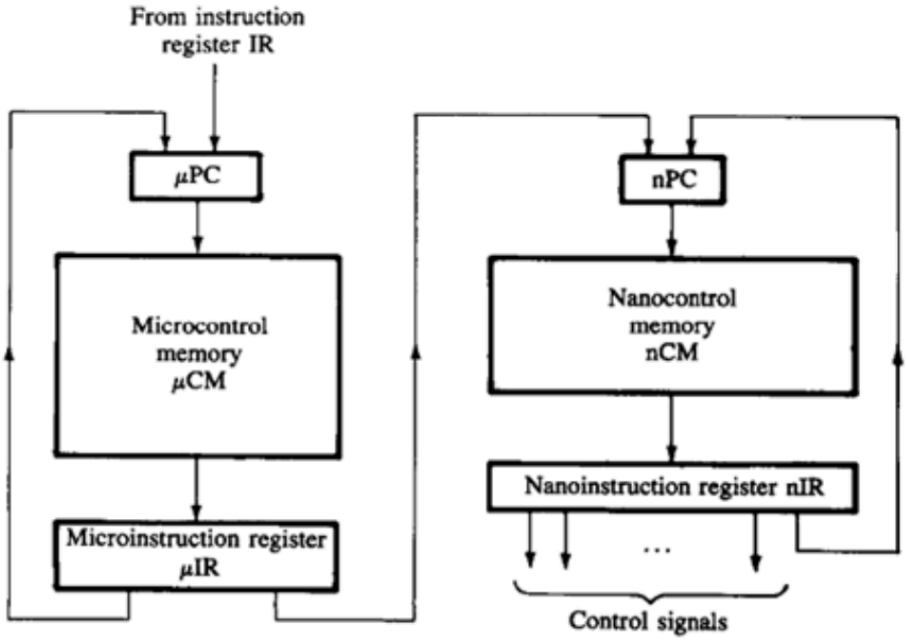


	HORIZONTAL MICRO-INSTRUCTION	VERTICAL MICRO-INSTRUCTION
1	Every bit of the micro-instruction corresponds to a control signal.	Bits of the micro-instruction have to be decoded to produce control signals.
2	Does not require a decoder.	Needs a decoder.
3	N bits in the micro-instruction will totally produce N control signals.	N bits in the micro-instruction will totally produce 2^N control signals.
4	Multiple control signals can be produced by one micro-instruction.	Only one control signal can be produced by one micro-instruction.
5	As the control signals increase, the micro-instruction grows wider. Hence the Control Memory grows Horizontally.	To produce more control signals, more number of micro-instructions are needed. Hence the Control Memory grows Vertically.
6	Executes faster as no decoding needed.	Executes slower as decoding is needed.
7	Micro-instruction are very wide. Hence Control memory is large.	Micro-instruction are much narrower. Hence Control memory is small.
8	Circuit is simpler as a decoder is not needed.	Circuit is more complex as a decoder is needed.

As seen from the above comparison, both methods have their pros and cons.
So a combination of both is used together called Nano-Programming.

NANO-PROGRAMMING *(Very Important)*

1) **Horizontal** µ-instructions can produce **multiple control signals** simultaneously, but are **very wide**.
2) This makes the Control Memory **very large in size**.
3) **Vertical** micro-instructions are **narrow**, **but on decoding** can **produce** only **one control signal**.
4) This makes the Control Memory **small** but the execution is **slow**.
5) Hence a **combination of both techniques** is needed called **Nano-Programming**.
6) Here we have a **two level control memory**.
7) The instruction is fetched from the **main memory into IR**.
8) Using its **opcode** we load **address of its first micro-instruction** into **µPC**,
9) Using this address we **fetch the micro-instruction** from µ-Control Memory (**µCM**) **into µIR**.
10) This is in **vertical form** and has to be decoded.
11) The decoded output **loads a new address** in a Nano program counter (**nPC**).
12) Using this address we **fetch the Nano-instruction** from Nano-Control Memory (**nCM**) into **nIR**.
13) This is in **horizontal form** and can **directly generate control signals**.
14) Such a combination **gives advantage of both techniques**.
15) The size of the Control Memory is **small** as µ-instructions are **Vertical**.
16) Multiple control signals can be **produced simultaneously** as Nano-instructions are **Horizontal**.



	HARDWIRED CONTROL UNIT	MICROPROGRAMMED CONTROL UNIT
1	**Control signals are generated using hardware**.	**Control signals are generated using software (Microprogram).**
2	Since **hardware** is used, the circuit is **rigid**.	Since **software** is used, the circuit is **flexible.**
3	**Modification** to the Control Unit requires **re-design** of the entire **hardware**.	**Modification** to the Control Unit simply requires **re-programming** of μ-instructions.
4	**Ideally suited** for processors with **small and simple instruction sets**.	**Ideally suited** for processors with **large and complex instruction sets**.
5	**Debugging** a large Hardwired Control Unit is **very difficult.**	As micro-programs are software, **debugging is much easier.**
6	**Emulation is not possible.**	**Emulation is possible.**
7	Executes **faster** as control signals are directly generated by hardware.	Executes **slower** as **time** is **wasted** in **fetching and decoding μ-instructions**.
8	Does not need a **Control Memory**.	Needs a **Control Memory**.
9	**Cost is lower** as Control Memory is not needed.	**Cost is higher** as Control Memory is needed inside the processor.
10	Preferred in **RISC processors**.	Preferred in **CISC processors**.

As seen from the above comparison, both methods have their pros and cons.
Hence modern processors use a combination of both.
Simple and regularly used instructions are decoded by a Hardwired Control Unit as they are faster.
Complex instructions are decoded by a Microprogrammed Control Unit as they are easier to design..

7) CACHE MAPPING TECHNIQUES

Blocks are loaded from Main Memory to Cache Memory.
Cache Mapping decides which block of Main Memory comes into which block of Cache Memory.
There are several mapping techniques trying to balance between Hit Ratio, Search-time and Tag size.
Every cache block has a **Tag** indicating which block of Main Memory is mapped into that block.
A collection of such tags is called the **cache director**y, very similar to a page table.
Cache directory is a part of the cache.
Since Cache Memory is **very expensive**, we need the cache directory to be as small as possible.
That means the **Tag must be of minimum size**.
The Main Memory address, issued by the processor contains the desired block number.
This is compared to the Tag of a Cache Block, which gives the Block number that is present.
If they are equal, **it's a Hit**. If Not, the search may have to be repeated for several other blocks.
It is obvious to understand, **the number of searches must be as low as possible.**
Finally, the Mapping technique must yield maximum utilization of the Cache memory space, so that the **Hit ratio remains as High as possible**.
There are three popular Cache Mapping Techniques:

1) **Associative Mapping** also called Fully Associative Mapping
2) **Direct Mapping** also called One-Way Set Associative Mapping
3) **Set Associative Mapping** also called Two-Way Set Associative Mapping

ASSOCIATIVE MAPPING (FULLY ASSOCIATIVE MAPPING)

During memory operations, Blocks are loaded from Main Memory to Cache Memory.
Cache Mapping decides which block of Main Memory comes into which block of Cache Memory.
Fully Associative Mapping technique states:
Any block of Main Memory can be mapped at Any available block of Cache Memory.
There are no rules restricting the mapping at all.
This means the Full Cache is available for mapping hence the name Fully Associative.

Consider Pentium Processor Cache

Size of Main Mem:	**4GB** = 2^{32}
Size of Cache Mem:	**8KB** = 2^{13}
Size of Cache Block (Line):	**32** bytes (words) = 2^5
No. of Blocks in Main Mem:	Size of Main Memory (2^{32}) ÷ Size of Block (2^5) = 2^{27}
No. of Blocks in Cache Mem:	Size of Cache Memory (2^{13}) ÷ Size of Block (2^5) = 2^8 = **256**
Main Mem address:	**32 bits** (because main Mem is of **4GB** = 2^{32})

Tag Size:
A block of Cache Memory can **contain any block** of main memory out of a possible 2^{27} **blocks**.
Hence, the **Tag** next to every block in Cache Memory must be of **27 bits**.

Searches:
A block of main Memory can be **mapped into any block** of Cache Memory out of **256 Blocks**.
Hence, we need to do **256 searches** in Cache Memory.

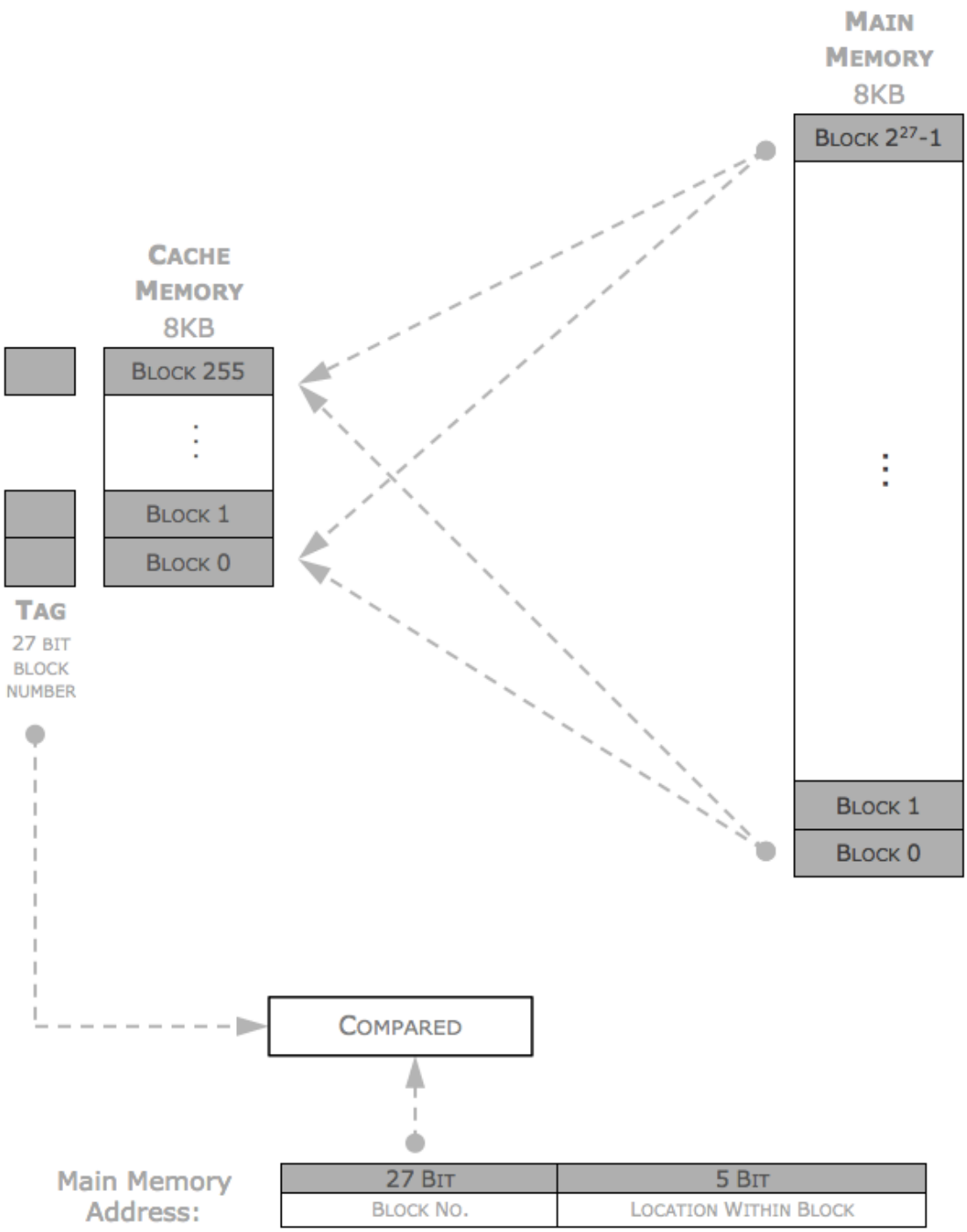
Method of Searching:
The Processor issues a 32 bit Main Memory address. It can be divided as:

Main Memory Address:	27 BIT	5 BIT
	Block No.	Location Within Block

This 27 bit Block number is the block number we need to search.
The Tag of each cache block also contains a 27-bit block number.
This is the block number that's present in that respective cache block.
These two block numbers are compared. **If they are equal, it's a HIT.**
If not equal, the search is repeated with the Tag of the next cache block.
This is done a total of 256 times as there are 256 blocks in Cache Memory.
If none of them match with the block number we are searching, then it's a Miss.

Advantage
Since the full cache is available for mapping, it causes maximum utilization of Cache Memory hence gives the **Best Hit Ratio.**

Drawback
Tag Size too big: **27bits.**
Searches are too many: **256.**


MAIN MEMORY
8KB
BLOCK 2^{27}-1
BLOCK 1
BLOCK 0
CACHE MEMORY
8KB
BLOCK 255
BLOCK 1
BLOCK 0
TAG
27 BIT BLOCK NUMBER
COMPARED
Main Memory Address:
27 BIT
BLOCK NO.
5 BIT
LOCATION WITHIN BLOCK

DIRECT MAPPING (ONE WAY SET ASSOCIATIVE MAPPING)

Direct Mapping technique states:
Any block of Main Memory can only be mapped at ONE block of Cache Memory.
Since there is only one way of Mapping, its also called One Way Set Associative Mapping.
We treat the entire Cache as One Set.
The Main Memory is divided into Sets which are then subdivided into Blocks.
A Block of Main Mem. (of any set), can only be mapped into the same Block No. in Cache Mem.
This means, Block 0 of Main Memory (of any set), can only be mapped into Block 0 of Cache Memory.
In other words, Block 0 of Cache Memory can only contain Block 0 of Main Memory but of any Set.

Consider Pentium Processor Cache

Size of Main Mem:	**4GB** = 2^{32}
Size of Cache Mem:	**8KB** = 2^{13} … this is treated as One Set
Hence Size of Set:	**8KB** = 2^{13}
Size of Cache Block (Line):	**32** bytes (words) = 2^5
No. of Blocks in a set:	Size of Set (2^{13}) ÷ Size of Block (2^5) = 2^8 = **256**
No. of Sets in Main Mem:	Size of Main Mem (2^{32}) ÷ Size of Set (2^{13}) = 2^{19}
No. of Sets in Cache Mem:	1
Main Mem address:	**32 bits** (because main Mem is of **4GB** = 2^{32})

Tag Size:
Since, Block 0 of Cache Memory can only contain Block 0 of Main Memory but of any Set, the Tag has to only indicate the Set No. of Main Memory, from which the block is present.
As Main Memory has 2^{19} sets, the **Tag** size is **19 bits**.

Searches:
If we need Block 0 if Main Memory, we only need to search Block 0 of Cache Memory.
Hence, we need to do only **1 search** in Cache Memory to know if it is a Hit or a Miss.

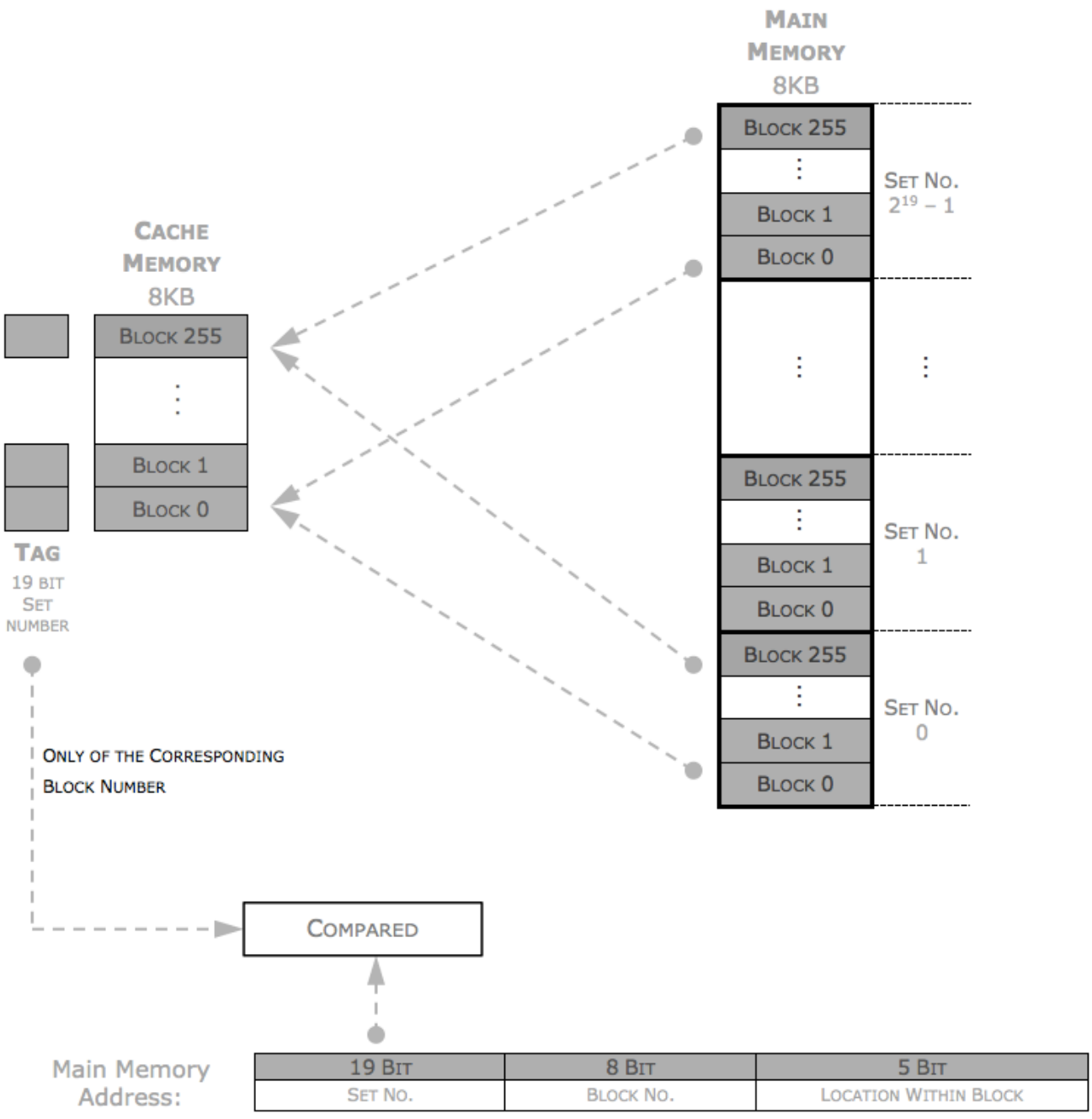
Method of Searching:
The Processor issues a 32 bit Main Memory address. It can be divided as:

Main Memory Address:	19 BIT	8 BIT	5 BIT
	Set No.	Block No.	Location Within Block

First we look at the block no. we need, to know where we need to search.
We then look at the Set No. that we need and compare it with the Tag of the corresponding Block no in the Cache Memory.
Assume the Main Memory address is 5:0:6. This means we need location 6 of Block 0 of set 5.
We go to Block 0 of Cache Memory.
It has a Tag, which gives the Set No of Main Memory whose Block 0 is present in Cache Memory.
These two set numbers are compared. **If they are equal, it's a HIT, else it's a Miss.**

Advantage
In **1 Search** we know if it is a Hit or a Miss. **Tag Size** = **19 bits**.

Drawback
Since the method is **very rigid**, the **Hit Ratio drops tremendously.**


MAIN MEMORY
8KB
BLOCK 255
BLOCK 1
BLOCK 0
SET NO. $2^{19} - 1$
SET NO. 1
SET NO. 0
CACHE MEMORY
8KB
TAG
19 BIT SET NUMBER
ONLY OF THE CORRESPONDING BLOCK NUMBER
COMPARED
Main Memory Address:
19 BIT
SET NO.
8 BIT
BLOCK NO.
5 BIT
LOCATION WITHIN BLOCK

SET ASSOCIATIVE MAPPING (TWO WAY SET ASSOCIATIVE MAPPING)

Two way Set Associative Mapping technique states:
A block of Main Memory can only be mapped into the same corresponding Block No. of Cache Memory, in any of the two sets.
Since there are two ways of Mapping, its called Two Way Set Associative Mapping.
We treat the entire Cache as **Two Sets**. The Main Memory is divided into Sets, subdivided into Blocks.
A Block of Main Mem. (of any set), can only be mapped into the same Block No. in Cache Memory again of any set. This means, Block 0 of Main Memory (of any set), can only be mapped into Block 0 of Cache Memory, into one of its two sets.
In other words, Block 0 of Cache Memory can only contain Block 0 of Main Memory but of any Set.

Consider Pentium Processor Cache (This is Actually how Pentium's Cache is implemented)

Size of Main Mem:	**4GB** = 2^{32}
Size of Cache Mem:	**8KB** = 2^{13} … this is treated as Two Sets
Hence Size of Set:	**4KB** = 2^{12}
Size of Cache Block (Line):	**32** bytes (words) = 2^5
No. of Blocks in a set:	Size of Set (2^{12}) ÷ Size of Block (2^5) = 2^7 = **128**
No. of Sets in Main Mem:	Size of Main Mem (2^{32}) ÷ Size of Set (2^{12}) = 2^{20}
No. of Sets in Cache Mem:	2
Main Mem address:	**32 bits** (because main Mem is of **4GB** = 2^{32})

Tag Size:
Since, Block 0 of Cache Memory can only contain Block 0 of Main Memory but of any Set, the Tag has to only indicate the Set No. of Main Memory, from which the block is present.
As Main Memory has 2^{20} sets, the **Tag** size is **20 bits**.

Searches:
If we need Block 0 if Main Memory, we only need to search Block 0 of Cache Memory, but in any of the two sets. Hence, we need **2 searches** in Cache Memory to know if it is a Hit or a Miss.

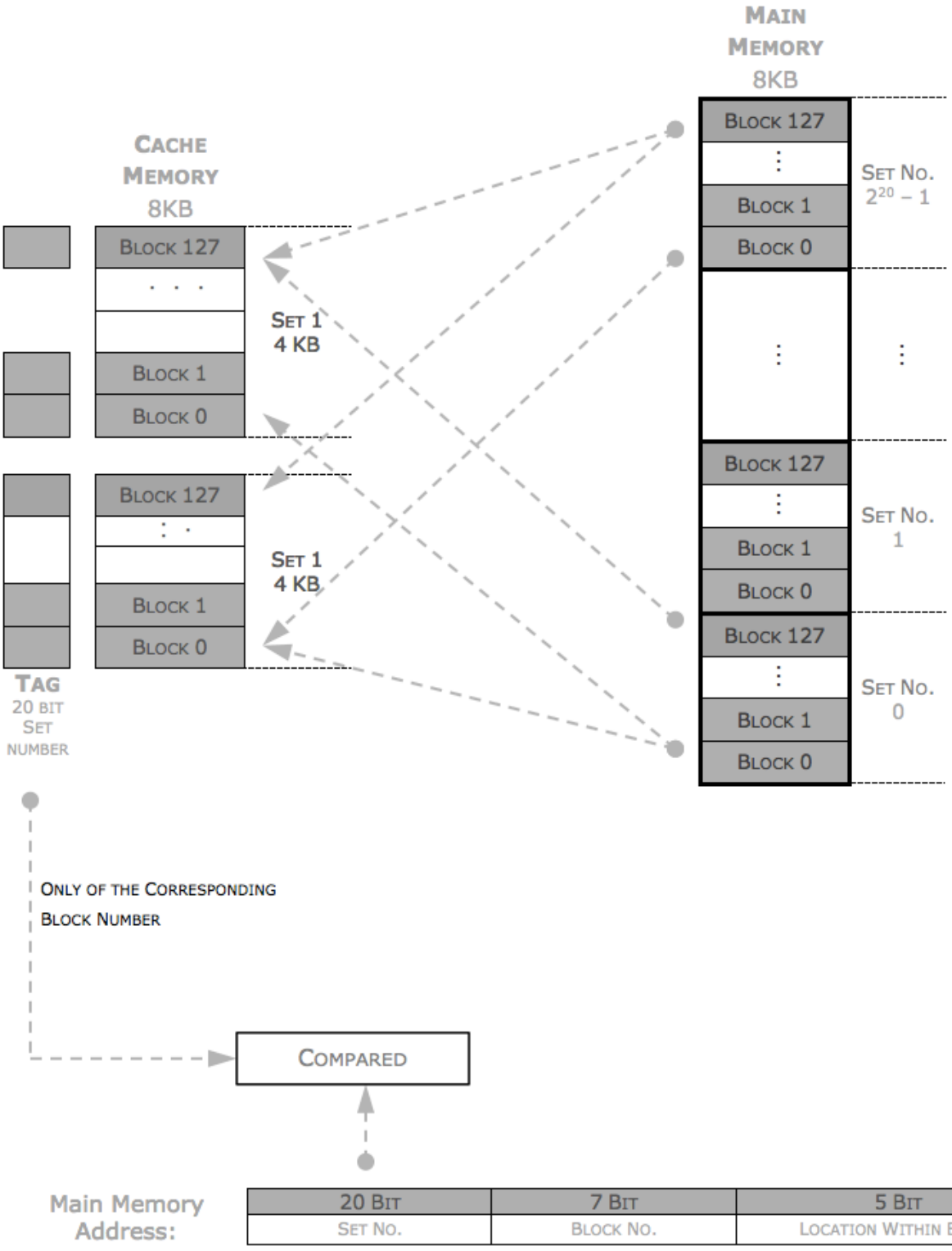
Method of Searching:
The Processor issues a 32 bit Main Memory address. It can be divided as:

Main Memory Address:	**20 BIT**	**7 BIT**	**5 BIT**
	Set No.	Block No.	Location Within Block

First we look at the block no. we need, to know where we need to search.
We then look at the Set No. that we need and compare it with the Tags of the corresponding Block no in the Cache Memory, in any of the two sets.
Assume the Main Memory address is 5:0:6. This means we need location 6 of Block 0 of set 5.
We go to Block 0 of Cache Memory, in both sets.
It has a Tag, which gives the Set No of Main Memory whose Block 0 is present in Cache Memory.
These required Set no. is compared with the two Tags. **If found in any of the 2, it's a HIT, else Miss.**

Advantage
In **2 Searches** we know if it is a Hit or a Miss. **Tag Size** = **20 bits**.

Drawback
Since the method is **flexible**, it significantly **increases** the **Hit Ratio.**


MAIN MEMORY
8KB
CACHE MEMORY
8KB
BLOCK 127
BLOCK 1
BLOCK 0
SET 1
4 KB
SET 1
4 KB
SET NO.
$2^{20} - 1$
SET NO.
1
SET NO.
0
TAG
20 BIT
SET
NUMBER
ONLY OF THE CORRESPONDING
BLOCK NUMBER
COMPARED
Main Memory
Address:
20 BIT
SET NO.
7 BIT
BLOCK NO.
5 BIT
LOCATION WITHIN BLOCK

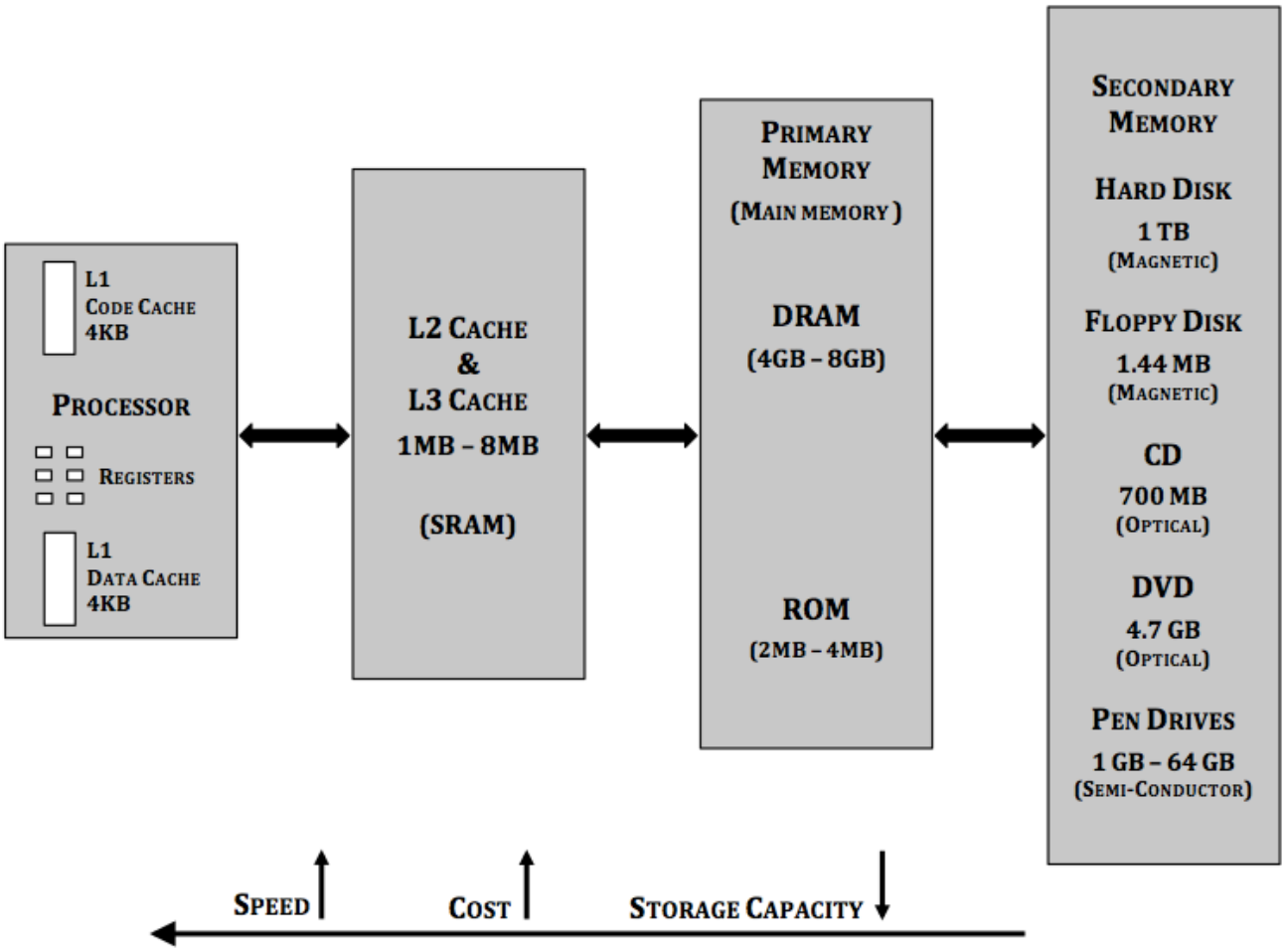
Expanding the logic of set associative cache further, we can derive the following conclusion:

NO OF WAYS	NO OF SEARCHES	TAG SIZE	NO OF BLOCKS IN A SET
2 Way	2	20 bits	128
4 Way	4	21 bits	64
8 Way	8	22 bits	32
16 Way	16	23 bits	16
32 Way	32	24 bits	8
64 Way	64	25 bits	4
128 Way	128	26 bits	2
256 Way	256	27 bits	1
This becomes exactly the same as Fully Associative: 256 Searches, 27-bit Tag			

MEMORY

MEMORY HIERARCHY

1) The purpose of any memory device is to **store programs and data**.
2) **Several types** of memory devices are used in the computer forming a **Memory Hierarchy**.
3) Each plays a specific role contributing to the **speed, cost effectiveness, portability** etc.



REGISTERS

1) Registers are **present inside the processor**.
2) They are basically a **set of flip-flops**.
3) They store **data and addresses** and can **directly take part** in **arithmetic and logic operations**.
4) They are very small in size typically **just a few bytes**.

PRIMARY MEMORY

1) It is the original form of memory also called as **Main memory**.
2) It comprises of **RAM and ROM**, both are **Semi-Conductor** memories. (chip memories)
3) **ROM is non-volatile**.
   It is used is storing permanent information like the **BIOS program**.
   It is typically of **2 MB - 4 MB** in size.
4) **RAM is writable** and hence is used for **day-to-day operations**.
   Every file that we access from secondary memory, is **first loaded into RAM**.
   To provide large amount of working space RAM is **typically 4 GB - 8 GB**.

SECONDARY MEMORY

1) The main purpose of Secondary Memory is to **increase the storage capacity, at low cost**.
2) Its biggest component is the **Hard Disk**.
   This is where all the files inside a computer **are stored**.
3) It is **writeable as well as non-volatile**.
4) Typical size of a **HD is 1 TB**.
5) Disk memories are much **slower than chip memories** but are also **much cheaper**.

PORTABLE SECONDARY MEMORY

1) These are required to **physically transfer files** between computers.
2) **Floppy Disk**: It is a **magnetic form** of storage. Typical **Size is 1.44 MB**.
3) **CD**: It is an **optical form** of storage. Typical **Size is 700 MB**.
4) **DVD**: It is an **optical form** of storage. Typical **Size is 4.7 MB**.
5) **Pen Drives & Memory Cards**: It is a **semi-conductor form** of storage.
   It is composed of FLASH ROM.
   It's a special type of ROM that's writable as well as non-volatile.
   Typical **Size ranges form 1 GB - 64 GB** depending upon the cost.

CACHE MEMORIES

1) It is the **fastest form of memory** as it uses SRAM (Static RAM).
2) The Main Memory uses DRAM (Dynamic RAM).
3) **SRAM uses flip-flops and hence is much faster than DRAM which uses capacitors.**
4) But SRAM is also **very expensive** as compared to DRAM.
5) Hence **only the current portion of the file** we need to access is copied from Main Memory (DRAM) to Cache memory (SRAM), to be directly accessed by the processor.
6) This gives **maximum performance and yet keeps the cost low**.
7) Typical size of Cache is around **2 MB – 8MB**.
8) If **code and data** are in the **same cache** then it is **unified cache** else its **called split cache**.
9) Depending upon the location of cache, it is of three types: L1, L2 and L3.
10) **L1** cache is **present inside the processor** and is a **split cache** typically **4-8 KB**.
11) L2 is present on the **same die as the processor** and is a **unified cache** typically **1 MB**.
12) L3 is present **outside the processor**. It is also **unified** and is typically of **2-8 MB**.

MEMORY CHARACTERISTICS

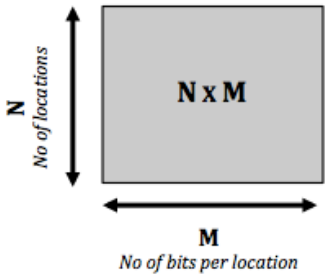
1) Location

Based on its physical location, memory is classified into three types.

- **On-Chip:** This memory is present **inside the CPU**. E.g.:: Internal Registers and **L1 Cache**.
- **Internal:** This memory is present **on the motherboard**. E.g.:: **RAM**.
- **External:** This memory is **connected to the motherboard**. E.g.:: **Hard disk**.

2) Storage Capacity

This indicates the **amount of data stored** in the memory.
Obviously it should be **as large as possible**.
It is represented as N x M.
Here,
N = **Number of memory locations** (no of words)
M = **Number of pits per memory location** (word size)
E.g.:: (4K x 8) means there are 4K locations of 8-bits each.



3) Transfer Modes

Data can be accessed from memory in two different ways.

- **Word Transfer:** Here, if CPU needs some data, it will transfer only that amount of data.
  E.g.:: Data accessed from **L1 Cache.**
- **Block Transfer:** Here, if CPU needs some data, it will transfer an entire block containing that data. This makes further access to remaining data of this block much faster. This is based on Principle of Spatial Locality. A processor is most likely to access data near the current location being accessed.
  E.g.:: On a **cache miss**, processor goes to **main memory** and copies a **block** containing that data.

4) Access Modes

Memories can allow data to be accessed in two different ways.

- **Serial Access:** Here locations are accessed one by one in a **sequential manner**.
  The access time depends on how far the target location is, from the current location.
  Farther the location, **more** will be its **access time**.
  E.g.:: Magnetic tapes.
- **Random Access:** Here **all locations** can be directly accessed in any **random order**.
  This means **all locations** have the **same access time** irrespective of their address.
  E.g.:: Most modern memories like RAM.

5) Physical Properties

There are various Physical attributes to memory.

- **Writeable: Contents** of the memory **can be altered**. E.g.:: **RAM**
- **Non-Writeable: Contents** of the memory **cannot be altered**. E.g.:: **ROM**
- **Volatile: Contents** of the memory are **lost** when power is **switched off**. E.g.:: **RAM**
- **Non-Volatile: Contents** of the memory are **retained** when power is **switched off**. E.g.:: **ROM**

Most secondary memories like Hard disk are Writable as well as non-volatile.

6) Access Time (t_A)

It is the time taken between **placing the request** and **completing the data transfer**.
It should be as **less as possible**.
It is also known as **latency**.

7) Reliability

It is the **time** for which the memory is expected to **hold the data without any errors**.
It is measured as **MTTF: Mean Time To Failure**.
It should be as **high as possible**.

8) Cost

This indicates the **cost of storing data** in the memory.
It is expressed as **Cost/bit**.
It must be **as low as possible**.

9) Average Cost

It is the total cost per bit, for the entire memory storage.
Consider a system having **two memories M_1 (RAM) & M_2 (ROM)**
If **C_1** is the cost of memory M_1 of size **S_1**
& **C_2** is the cost of memory M_2 of size **S_2**

Then the average cost of the memory is be calculated as:
$C_{AVG} = (C_1 S_1 + C_2 S_2)/ (S_1 + S_2)$

Small sizes of **expensive** memory and **large** size of **cheaper** memory **lowers** the **average cost**.

10) Hit Ratio (H)

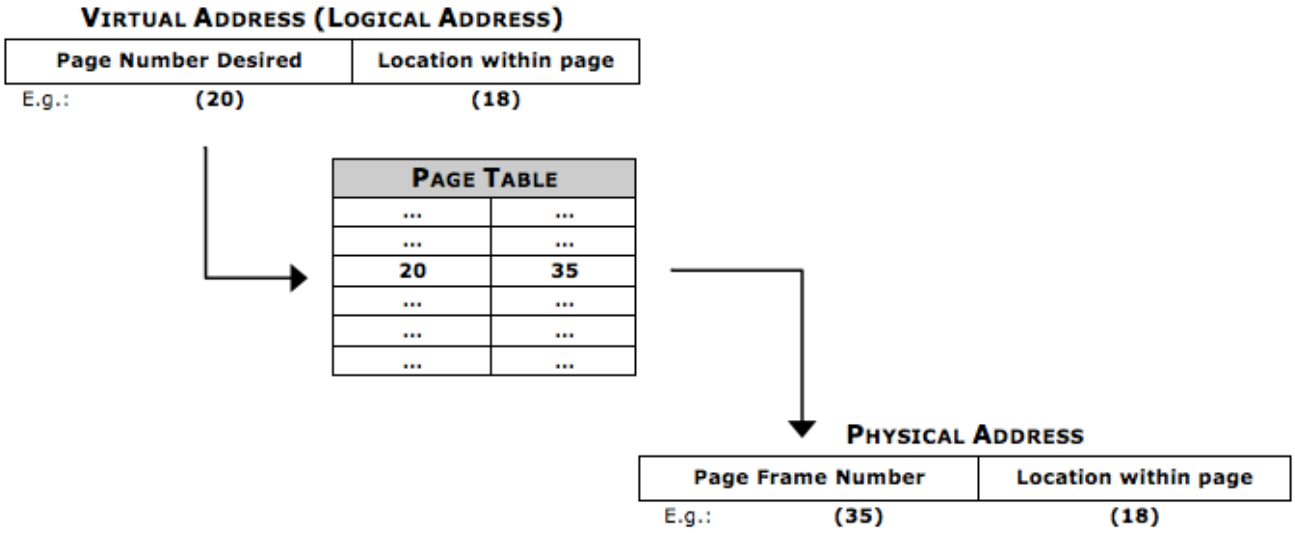
Consider two memories M_1 and M_2.
M_1 is **closer** to the processor E.g.:: **RAM**, than **M_2** E.g.:: **Hard disk**.
If the **desired data is found in M_1**, then it is called a **Hit**, else it is a **Miss**.
Let **N_1** be the number of **Hits** and **N_2** the number of **Misses**.
The **Hit Ratio** H is defined as **number of hits divided by total attempts**.
$H = (N_1) / (N_1 + N_2)$
It is expressed s a percentage.
H can never be 100%. In most computers it is maintained around 98%.

From the above discussion it is clear that no single memory can satisfy all the characteristics, **hence we need a hierarchy of memories**.
Cache memories are the **fastest** but also the **most costly**.
Hard disk is **writeable** as well as **non volatile** and is also very **inexpensive**, but is much **slower**.
CD/DVD etc. are needed for **portability**.
ROM is **nonvolatile**, and is used for **storing BIOS**.
DRAM is **writable**, f**aster than hard disk** and **cheaper than SRAM** hence forms **most part of Main Memory**.

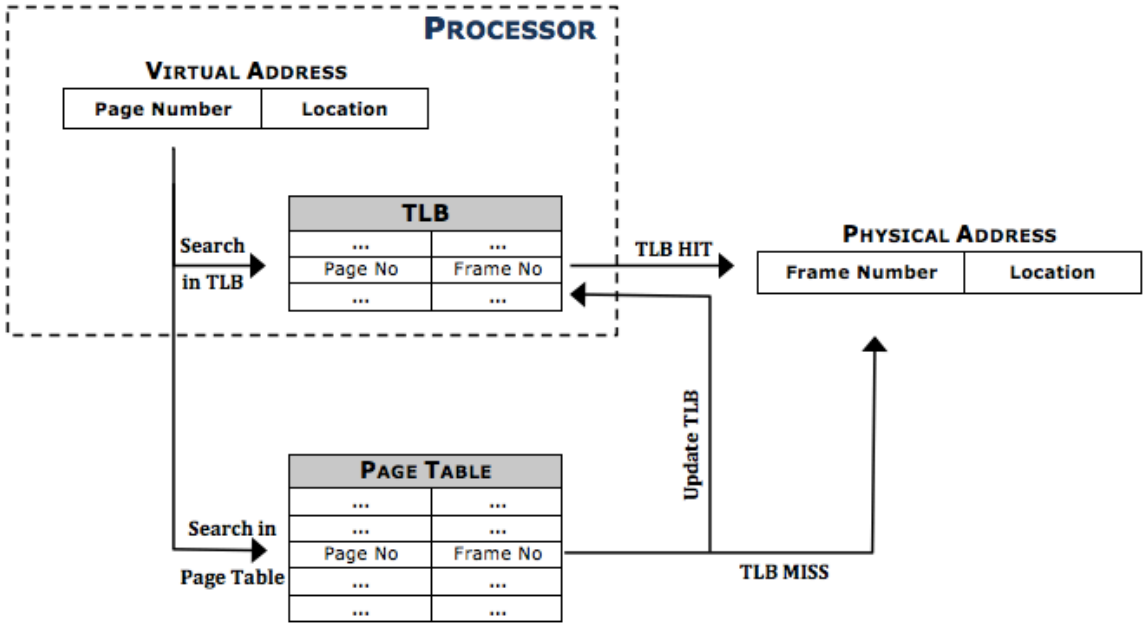
VIRTUAL MEMORY MANAGEMENT USING PAGING

1) The **Virtual Memory space** is divided into **equal size blocks** called **"Pages"**.
2) The **Physical Memory space** (also called Main Memory) is also divided into **equal size blocks** called **Page Frames** (also simply called pages).
3) **Pages are loaded** from Virtual Memory **into any** available page frame of Physical Memory.
4) Consider example of 80386 Processor. Physical Memory = 4 GB. Page Size = 4 KB.
   No of pages in Physical Memory = 4 GB / 4 KB = 1M. ($2^{32} / 2^{12} = 2^{20}$)
5) When a page is needed, Processor **checks** if the **desired page** is **present** in the **Physical Memory**.
6) If found, it is called a **"HIT"** and the operation is performed on the Physical Memory.
7) If the **desired page** is **not present** in the **Physical Memory** its called a **MISS** or a **Page Fault.**
8) On a Page Fault the desired page is loaded form Virtual Memory into Physical Memory.
9) If no empty page frame is available in Physical Memory, then a **"Page Replacement"** is performed.
10) An old page in Physical Memory is replaced with a new desired page from the Virtual Memory.
11) Popular **algorithms** like **FIFO** (First in first out), **LRU** (Least recently used) or **LFU** (Least frequently used) are used to determine which page of the Physical Memory must be replaced.
12) If there are **too many page faults**, the system is said to be **Thrashing**. This must be avoided.
13) While replacing, the **"Dirty Bit"** is checked to determine if a page is **modified** in Physical Memory.
14) **If Dirty bit = 1, then the page has been modified** (is "Dirty") and hence must be copied back into Virtual Memory before being replaced else the modified information will be lost.
15) **If Dirty bit = 0, then the page can be directly replaced**.
16) Since **any page** of Virtual Memory can be **loaded into any page frame** of Physical Memory, a **"Page Table"** is required to give the **mapping between** Virtual Memory **page number** and Physical Memory **page frame number**.
17) **Virtual Address** can be divided in two parts: **Desired Page Number | Location in page**
18) **Physical Address** can be divided in two parts: **Present Page Frame Number | Location in page**
19) The Page Table relates the desired page number to the page frame number (if present) in the Physical Memory. This is how address translation is performed.
20) **But this process is very slow**. To access any location, we must first access the corresponding entry in the page table, then access the page.
21) **Page Table is also in the memory**. This means, to access any memory location we must make an **additional trip** to the memory just to access the page table.
22) To speed up the process a **"Translation Look-aside Buffer"** is used (called TLB).
23) **The TLB is an on chip cache, present inside the processor.**
24) **It stores 32 most recently used Page table entries.**
25) This makes subsequent access to these pages (whose information is cached in the TLB) **much faster as there is no need to access the page table**. Processor can directly obtain the starting address of the page frame from the TLB and hence directly access the page.
26) Due to principle of **"Locality of reference"** most systems get a **Hit ratio of >98%** on the TLB, thus making the operations very fast.
27) An application may require several pages. All are not loaded into Physical Memory at once. Instead, pages are loaded as and when they are required by the processor. This is called **Demand Paging**.

PAGING MECHANISM:



TRANSLATION LOOK-ASIDE BUFFER (TLB)



PAGING V/S SEGMENTATION

	PAGING	SEGMENTATION
1	Paging is a **physical division** of the memory.	Segmentation is a **logical division** of the memory.
2	Pages are of a **fixed size**.	Segments are of **variable sizes**.
3	**Fragmentation** (wastage of unused space) **is high** as pages are of **fixed size.**	**Fragmentation is low** as segments are of **variable size.**
4	Completely **hidden from programmer**.	**Controlled by programmer.**
5	Does **not** offer **much control over protection mechanism**	As segments are controlled by the programmer, we can assign **different privilege levels to segments**. Hence provides better control over protection mechanism.
6	**Does not differentiate between code and data.**	Code and data segments are **treated differently**.
7	Pages are loaded based on **algorithms** such as **FIFO, LRU, LFU** etc.	Segments are loaded based on **algorithms** such as **First Fit, Best Fit** etc.
8	Does **not allow sharing** of code or data between different processes/ tasks.	Segments can be either global or local. **Global segments can be shared** among all tasks.

PAGE REPLACEMENT NUMERICALS *(Very Important)*

1) Consider Main Memory has 3 page frames (0,1,2).
Processor requires pages from Virtual Memory in the following sequence of page numbers:
2,3,2,1,5,2,4,5,3,2,5,2. Show and compare the implementation of FIFO, LRU and LFU.

FIFO	2	3	2	1	5	2	4	5	3	2	5	2	Hit Ratio = 0.25
Frame 0	2*	2*	2*	2*	5	5	5*	5*	3	3	3	3*	
Frame 1		3	3	3	3*	2	2	2	2*	2*	5	5	
Frame 2				1	1	1*	4	4	4	4	4*	2	
			HIT					**HIT**		**HIT**			

LRU	2	3	2	1	5	2	4	5	3	2	5	2	Hit Ratio = 0.42
Frame 0	2	2	2	2	2	2	2	2	3	3	3	3	
Frame 1		3	3	3	5	5	5	5	5	5	5	5	
Frame 2				1	1	1	4	4	4	2	2	2	
			HIT			**HIT**		**HIT**			**HIT**	**HIT**	

LFU	2	3	2	1	5	2	4	5	3	2	5	2	Hit Ratio = 0.50
Frame 0	2 (1)	2 (1)	2 (2)	2 (2)	2 (2)	2 (3)	2 (3)	2 (3)	2 (3)	2 (4)	2 (4)	2 (5)	
Frame 1		3 (1)	3 (1)	3 (1)	5 (1)	5 (1)	5 (1)	5 (2)	5 (2)	5 (2)	5 (3)	5 (3)	
Frame 2				1 (1)	1 (1)	1 (1)	4 (1)	4 (1)	3 (1)	3 (1)	3 (1)	3 (1)	
			HIT			**HIT**		**HIT**		**HIT**	**HIT**	**HIT**	

2) Consider Main Memory has 4 page frames (0,1,2,3)
Processor requires pages from Virtual Memory in the following sequence of page numbers:
7,5,3,2,1,0,4,1,6,7,4,2. Show and compare the implementation of FIFO, LRU and LFU.

FIFO	7	5	3	2	1	0	4	1	6	7	4	2	Hit Ratio = 0.16
Frame 0													
Frame 1													
Frame 2													
Frame 3													

LRU	7	5	3	2	1	0	4	1	6	7	4	2	Hit Ratio = 0.16
Frame 0													
Frame 1													
Frame 2													
Frame 3													

LFU	7	5	3	2	1	0	4	1	6	7	4	2	Hit Ratio = 0.16
Frame 0													
Frame 1													
Frame 2													
Frame 3													

OPTIMAL REPLACEMENT ALGORITHM

1. Another proposed replacement algorithm is called "**Optimal Replacement Algorithm**" (**OPT**)
2. We need to know beforehand, the order in which, pages will be used in the near future.
3. **The pages that will be used sooner, will be retained.**
4. **The pages that will not be used for the longest time will be replaced.**
5. Of course it is impossible to predict the pages to be used in the future.
6. But if we have had some **sample runs of the program in a simulator** then using that data as a reference, we can make safe predictions of the behavior of the program.

Consider Main Memory has 3 page frames (0,1,2).
Calculate Hits and Misses and suggest the best algorithm out of **FIFO, LRU and OPT**
Processor requires pages from Virtual Memory in the following sequence of page numbers:
4,7,3,0,1,7,3,8,5,4,5,3,4,7. (Sem 4 Comps IT Dec 2015 Exam question – 10 marks)

FIFO	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Frame 0	4*	4*	4*	0	0	0*	3	3	3*	4	4	4	4	4*
Frame 1		7	7	7*	1	1	1*	8	8	8*	8*	3	3	3
Frame 2			3	3	3*	7	7	7*	5	5	5	5*	5*	7
	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	**HIT**	MISS	**HIT**	MISS

Total Attempts: 14. Hits = 2. Misses (Page Faults) = 12. **Hit Ratio = 2/14 = 0.117**

LRU	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Frame 0	4	4	4	0	0	0	3	3	3	4	4	4	4	4
Frame 1		7	7	7	1	1	1	8	8	8	8	3	3	3
Frame 2			3	3	3	7	7	7	5	5	5	5	5	7
	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	**HIT**	MISS	**HIT**	MISS

Total Attempts: 14. Hits = 2. Misses (Page Faults) = 12. **Hit Ratio = 2/14 = 0.117**

OPT	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Frame 0	4	4	4	0	1	1	1	8	5	5	5	5	5	7
Frame 1		7	7	7	7	7	7	7	7	4	4	4	4	4
Frame 2			3	3	3	3	3	3	3	3	3	3	3	3
	MISS	MISS	MISS	MISS	MISS	**HIT**	**HIT**	MISS	Miss	Miss	**HIT**	**HIT**	**HIT**	Miss

Total Attempts: 14. Hits = 5. Misses (Page Faults) = 9. **Hit Ratio = 5/14 = 0.357**

BUS CONTENTION / BUS ARBITRATION / PRIORITY RESOLVING SCHEMES

In a Loosely coupled system all processors can use their local bus simultaneously. But the system bus can be used by only one module at a time. Hence there is contest for the system bus. This is called bus contention. It is resolved buy various arbitration schemes having different priority methods.

A) Daisy Chain Method

- **All** bus **masters use** the **same** line for Bus **Request**.
- **If** the **Bus Busy** line is **inactive**, the **Bus Controller gives** the **Bus Grant signal**.
- Bus **Grant** signal is **propagated serially** through all masters **starting** from **nearest** one.
- The bus **master**, which requires the system bus, **stops this signal**, **activates** the Bus **Busy** line and **takes control** of the system bus.

Advantage:

i. **Design** is **simple**.
ii. The **number of control lines is less**. Also **adding** new bus masters is **easy**.

Disadvantage:

i. **Priority** of bus masters is **rigid** and **depends** on the **physical proximity** of the bus masters with the bus arbiter i.e. The one nearest to the Bus Arbiter gets highest priority.
ii. Bus is granted **serially** and hence a **propagation delay** is induced in the circuit.
iii. **Failure of one** of the devices may **fail** the entire **system**.

B) Polling Method

- Here also **all bus masters** use the **same** line for **Bus Request**.
- Here the **controller generates** binary **address** for the master.
  Eg: To connect 8 bus masters we need 3 address lines ($2^3 = 8$).
- In **response** to a **Bus Request**, the **controller "polls"** the bus masters by **sending** a **sequence** of bus master **addresses** on the address lines.Eg: 000,010,100,011 etc
- The selected **master activates** the **Bus Busy** line and **takes control** of the bus.

Advantage:

i. The **Priority** is **flexible** and can easily be **changed** by **altering** the **polling sequence**.
ii. If **one** module **fails**, the entire **system does not fail**.

Disadvantage:

i. **Adding** more bus masters is **difficult** as **increases** the number of **address lines** of the circuit. Eg: In the above circuit to add the 9^{th} Bus Master we need 4 address lines.

C) Independent Request Method

- Here, **all bus masters** have their **individual Bus Request** and **Bus Grant lines**.
- The **controller** thus **knows which master** has **requested**, so bus is granted to that master.
- **Priorities** of the masters are **predefined** so on **simultaneous** Bus **Requests**, the bus is **granted based** on the **priority**, provided the Bus Busy line is not active.
- The **Controller** consists of **encoder** and **decoder** logic for the priorities.

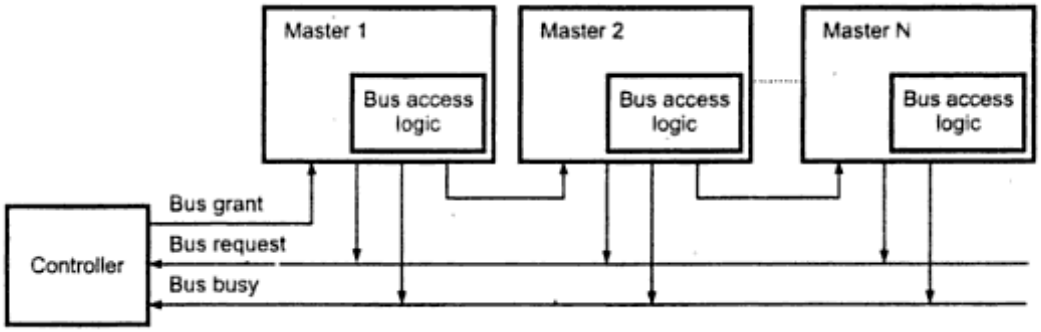
Advantage:

i. The Bus **Arbitration is fast**.
ii. The **speed of** Bus **Arbitration** is **independent** of the **number** of devices connected.

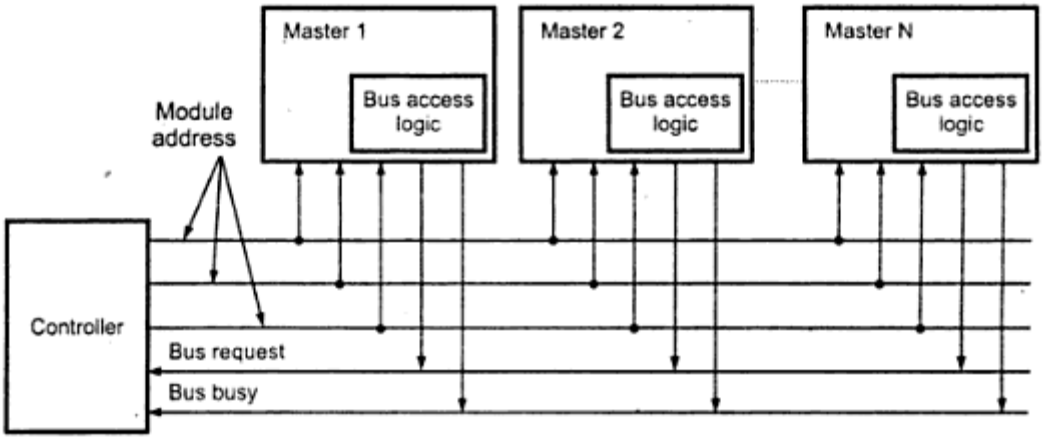
Disadvantage:

i. The **number of control lines required is more** (2n line required for n devices).

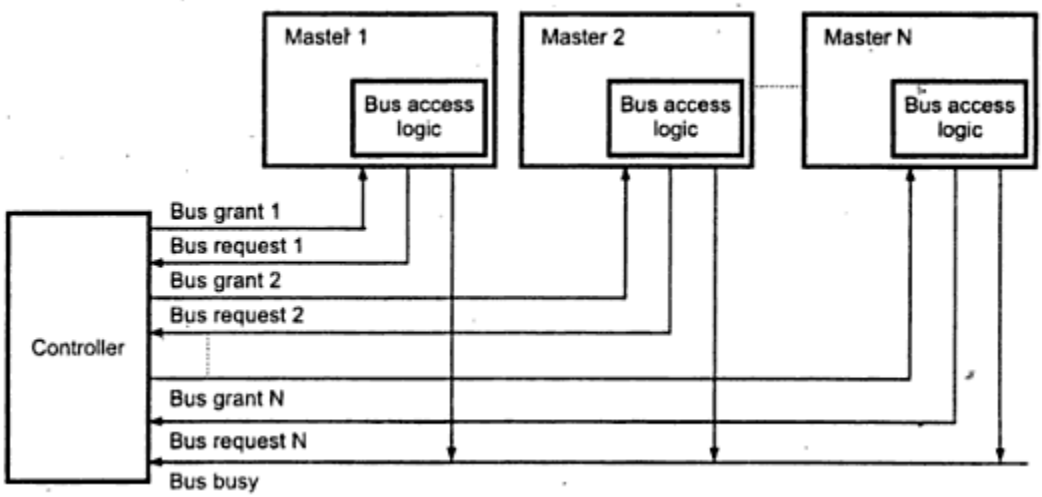
Daisy Chaining



Polling



Independent Requests

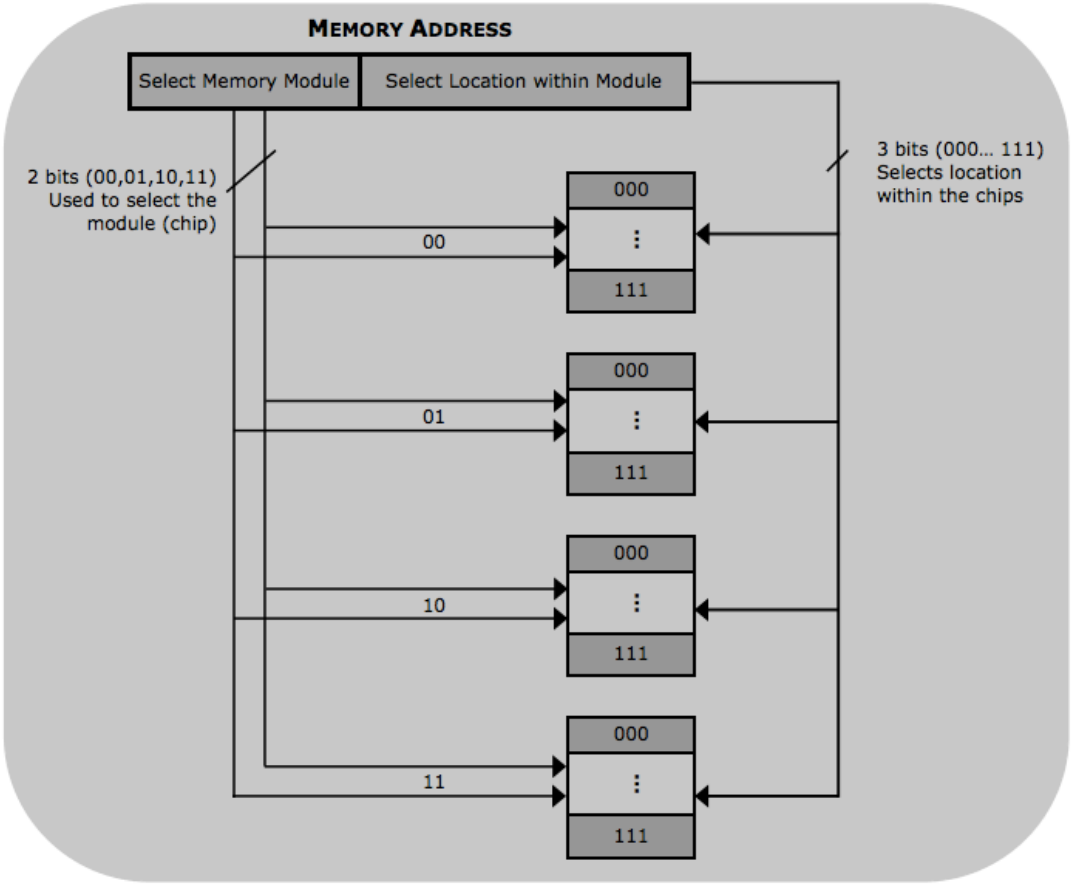


INTERLEAVED MEMORIES

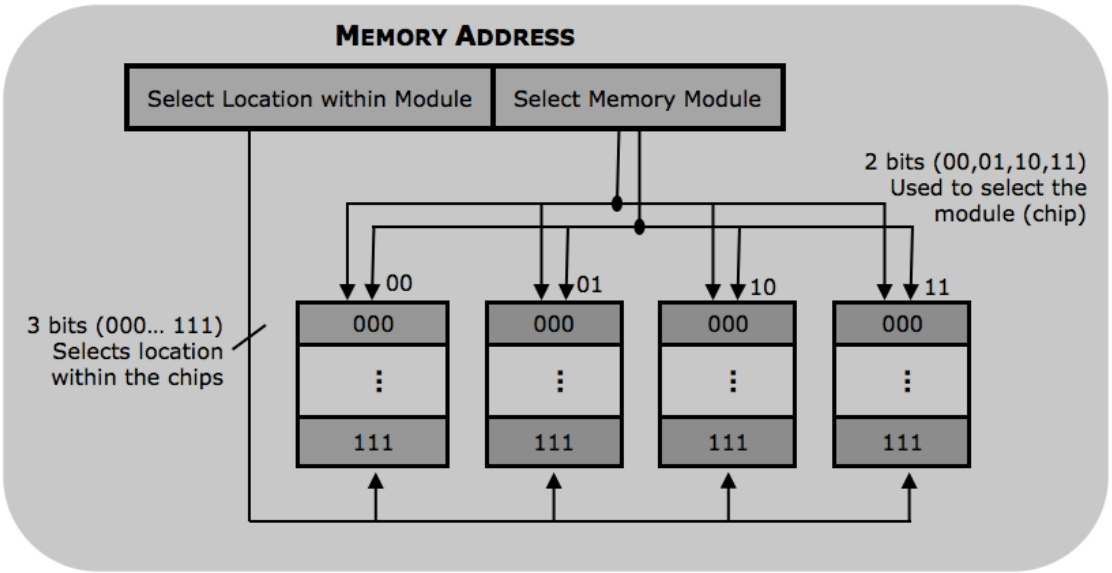
Memory Interleaving means **combining smaller memory modules to form one large memory.**
In most computer systems memory is not implemented as one single large module (chip).
Instead, it is formed by combining several smaller modules using the method of memory interleaving.
There are two interleaving techniques called Higher order and Lower order interleaving.

	HIGHER ORDER INTERLEAVING	LOWER ORDER INTERLEAVING
1	**Higher bits of the memory address are used to identify the memory module.**	**Lower bits of memory address are used to identify the memory module.**
2	Used to increase the **size of memory**.	Has **no effect** on size of memory.
3	Has **no effect** on the number of bits that can be transferred in one cycle.	**Increases the number of bits that can be transferred in one cycle**.
4	**Does not** increase the speed of the processor.	**Increases the speed** as it increases the number of bits transferred in 1 cycle.
5	**It is optional** and is only done when we need to increase the overall storage capacity.	**It is compulsory** in processors that need more bits in one cycle. E.g.:: 8086, 16-bit processor has two banks (Modules). 80386, 32-bit processor has 4 banks etc…
6	The **number of modules used are not fixed** and can be changed whenever required.	The **number of modules used is fixed** depending upon how many bits are needed in one cycle.
7	**Failure of one module does not effect the others.**	**Failure of one module effects all other modules** are data is "stripped" across modules.
8	**Reliability is high**	**Reliability is poor**
9	**Consecutive locations are in the same** module.	**Consecutive locations are in different** modules.
10	**Only one module** can be selected at a time.	**One or more modules can be selected simultaneously** depending upon the size of data needed to be accessed.

HIGHER ORDER INTERLEAVING



LOWER ORDER INTERLEAVING



3) DMA BASED I/O

DMA means **transferring data directly between memory and I/O.**
DMA transfers are **very fast** as compared to Processor based transfers due to two reasons.
1. They are **hardware based** so no time is wasted in fetching and decoding instructions.
2. Transfers are **directly between memory and I/O** without data going via the Processor.

To Perform a DMA transfer we need a **DMA Controller like 8237/ 8257.**
It is capable of taking control of the buses from the Processor.
The process is performed as follows.

1) By Default **Processor is the bus master**.
2) The DMA transfer parameters first initialized by the processor.
3) **Processor programs two registers inside the DMAC called CAR and CWCR** giving the starting address and the number of bytes to be transferred.
4) DMAC now ensures that the I/O device is ready for the transfer by checking the DREQ signal.
5) **If DREQ=1, then DMAC gives HOLD signal** to the Processor requesting control of the system bus.
6) **Processor releases control of the bus** after finishing the current machine (bus) cycle.
7) Processor **gives HLDA** informing DMAC that it is now the bus master.
8) **DMAC issues DACK#** (by default active low, but can be changed) to I/O device indicating that the transfer is about to begin.
9) Now DMAC **transfers one byte in one cycle**.
10) After every byte is transferred the **Address register and Count register are decremented by 1**.
11) This repeats till Count reaches **"0"** also called **Terminal Count**.
12) Now the **transfer is complete**.
13) DMAC **returns the system bus to Processor by making HOLD = 0**.
14) Processor once again **becomes bus master**.

Advantage of DMA
DMA transfers are very fast.

Drawback of DMA
DMAC becomes the bus master. Hence during DMA cycles, the processor cannot perform any operations as the bus is already being used for DMA. The processor remains in HOLD state.

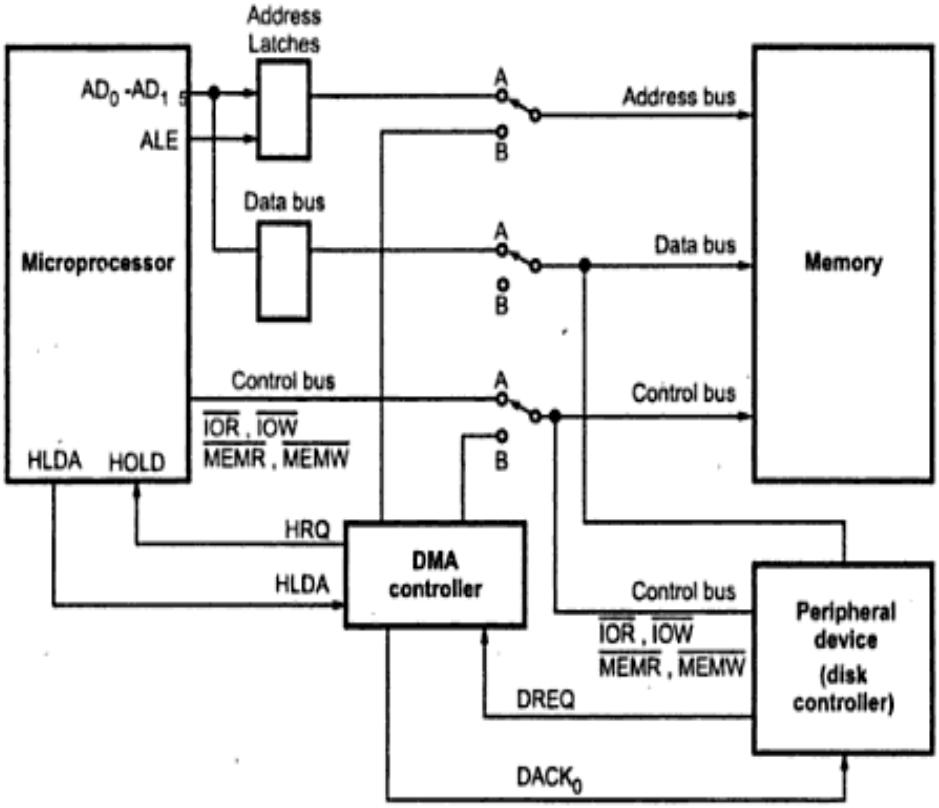
Difference between Interrupt Request and DMA request

When an interrupt occurs, the processor has to suspend the current program, execute the ISR and then **return to the next instruction** of the main program. Hence it is necessary that the **processor completes the current instruction** before servicing an interrupt request.

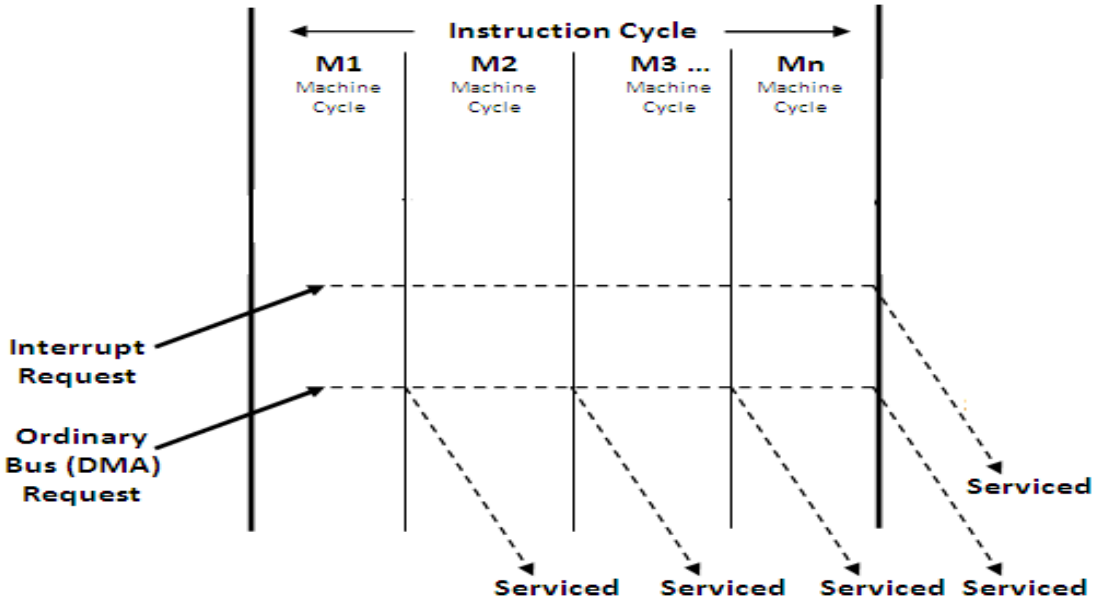
When a DMA request occurs, the processor has to simply relinquish (give away) control of the system bus and enter hold state. When ever it gets back the bus it can **resume from where it had left**. Hence the processor **need not finish the current instruction** before servicing a DMA request.
It simply has to **finish the current machine cycle**.

Hence Instruction cycles are Interrupt Breakpoints and Machine cycles are DMA breakpoints.

DMA Operation



Difference between Interrupt Request and DMA Request



TYPES / METHODS / TECHNIQUES OF DMA TRANSFERS

8237 has **four modes of data transfer**:

1) BLOCK TRANSFER MODE / BURST MODE.

In this mode, the DMAC is programmed to **transfer ALL THE BYTES** in one complete DMA operation.
After a byte is transferred, the **CAR and CWCR are adjusted** accordingly.
The **system bus** is **returned** to the **processor, ONLY after all the bytes are transferred.**
It is the **fastest** form of DMA but keeps the **processor inactive** for a long time.

2) SINGLE BYTE TRANSFER MODE/ CYCLE STEALING.

Once the DMAC becomes the bus master, it will transfer only **ONE BYTE** and return the bus to the processor. As soon as the processor performs one bus cycle, DMAC will once again take the bus back from the processor.
Hence both **DMAC and processor** are **constantly stealing bus cycles** from each other.
It is the **most popular** method of DMA, because it keeps the **processor active in the background**.
After a byte is transferred, the **CAR** and **CWCR** are **adjusted** accordingly.

3) DEMAND TRANSFER MODE.

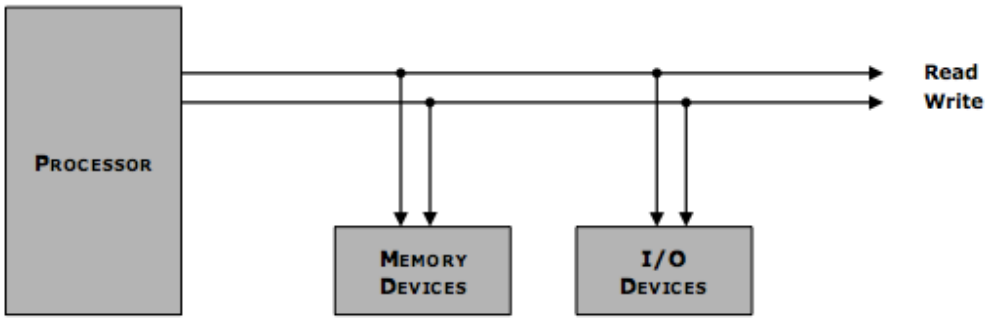
It is very **similar to Block Transfer**, **except** that the **DREQ must remain active throughout the DMA operation**.
If during the operation **DREQ goes low**, the **DMA operation is stopped** and the **busses** are **returned** to the **processor.** *#Please refer Bharat Sir's Lecture Notes for this ...*
In the meantime, the **processor** can **continue** with its own operations. **Once DREQ goes high again**, the **DMA operation continues** from where it had stopped.
This means, the transfer happens on demand from the I/O device, hence the name.

4) HIDDEN MODE / TRANSPARENT MODE.

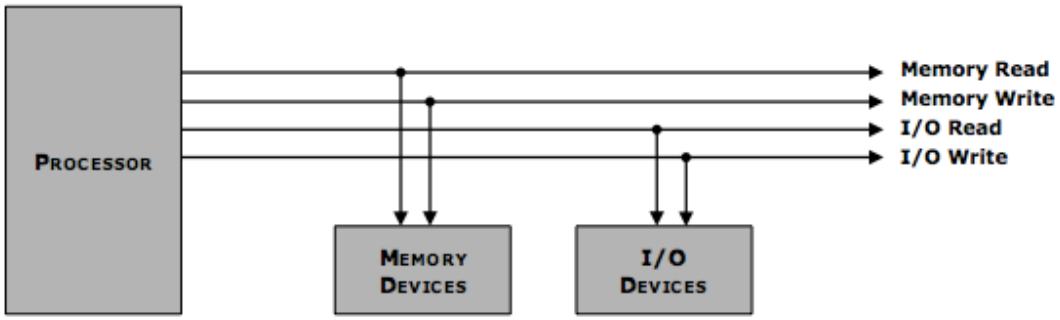
In this mode, **once the processor programs all parameters inside the DMAC, the DMAC** does not request the processor for the control of the bus. Instead it observes the processor. **It waits for the processor to enter idle state**. Once the processor is idle, the DMAC will take control of the bus and perform the Transfer. So the Transfer is **totally transparent to the processor** or hidden from the processor. Hence the name.

	MEMORY MAPPED I/O	I/O MAPPED I/O
1	**I/O devices are mapped into memory space.**	**I/O devices are mapped into I/O space.**
2	I/O devices are allotted **memory addresses**.	I/O devices are allotted **I/O addresses**.
3	Processor **does not differentiate** between memory and I/O. Treats I/O devices also like memory devices.	Processor **differentiates between I/O devices and memory**. It isolates I/O devices.
4	I/O addresses are as big as memory addresses. E.g.:: in 8085, **I/O addresses will be 16 bit** as memory addresses are also 16-bit.	I/O addresses are smaller than memory addresses. E.g.:: in 8085, **I/O addresses will be 8 bit** though memory addresses are 16-bit.
5	This allows us to increase the number of I/O devices. E.g.:: in 8085, we can access up to **2^{16} = 65536 I/O devices**.	This allows us to access limited number of I/O devices. E.g.:: in 8085, we can access only up to **2^{8} = 256 I/O devices**.
6	We can transfer data from I/O devices using **any instruction like MOV** etc.	We can transfer data from I/O device using **dedicated I/O instructions like IN and OUT ONLY**.
7	Data can be transferred using **any register** of the processor.	Data can be transferred only using a fixed register. E.g.:: in 8085 only **"A" register**.
8	We need **only two control** signals in the system: Read and Write.	We need **four control signals**: Memory Read, Memory Write and I/O Read and I/O Write
9	Memory addresses are big so **address decoding will be slower**.	I/O addresses are smaller so **address decoding will be faster**.
10	Address decoding will be **more complex and costly**.	Address decoding will be **simpler and cheaper**.

MEMORY MAPPED I/O



I/O MAPPED I/O

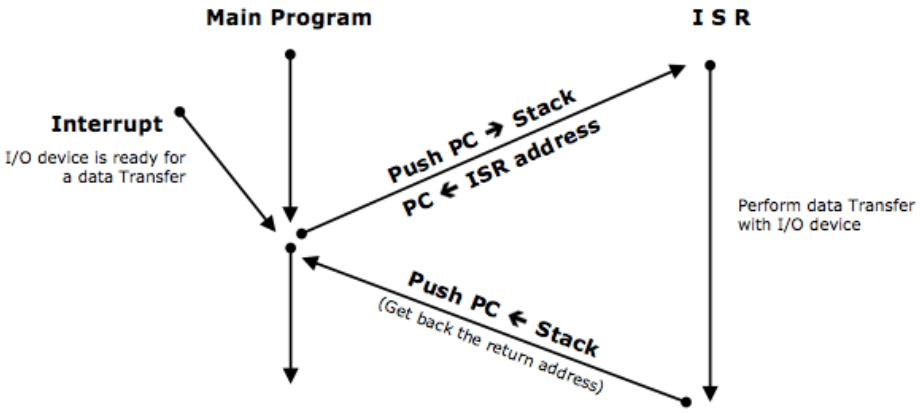


2) INTERRUPT DRIVEN I/O

1) In interrupt driven I/O, the transfer is not initiated by the processor.
2) Instead, **an I/O device which wants to perform a data transfer with the processor, must give an interrupt to the processor.**
3) An interrupt is a condition that makes the processor execute an ISR (Interrupt Service Routine).
4) In the ISR, processor will perform data Transfer with the I/O device.
5) This relieves the processor from periodically checking the status of every I/O device thereby saves as lot of time of the processor.
6) **The processor is free to carry on its own operations.**
7) Whenever a device wants to transfer data, it will interrupt the processor.
8) This is how many I/O devices Transfer data with the processor.
9) E.g.:: **Keyboard.** Instead of the processor checking all the time, whether a key is pressed, the keyboard interrupts the processor as an when we press a key. In the ISR of the keyboard, which is a part of the keyboard driver software, the processor will read the data from the keyboard.
10) **Hence interrupt driven I/O is much better than Polled I/O (Programmed I/O).**

INTERRUPT HANDLING MECHANISM

1) When an interrupt occurs, processor, firstly, finishes the current instruction.
2) It then suspends the current program and executes an ISR.
3) To do so, it Pushes the value of PC (address of next instruction), into the stack.
4) Now it loads the ISR address into PC and proceeds to execute the ISR.
5) At the end of the ISR, it POPs the return address from the stack and loads it back into PC.
6) This is how the processor return to the very next instruction in the program.



	INTERRUPT DRIVEN I/O	POLLING (PROGRAMMED I/O)
1	**I/O device interrupts the processor whenever it wants to perform a data Transfer.**	**Processor periodically checks (polls) the status of every I/O device to know if it wants to perform a data Transfer.**
2	**Processor is free** to carry on its own operations, hence **saves system time**.	**Processor is busy** in constantly checking all I/O devices, hence **system time wasted**.
3	**Additional hardware required** to handle interrupts. E.g.:: 8259 Programmable interrupt controller.	Additional hardware **not required**.
4	Increases **cost and complexity** of the system.	System is **cheaper and less complex**.
5	Interrupt **priority** has to be managed through software or through hardware.	No such issue.
6	Interrupt **vector addresses** (ISR Addresses) need to be stored in an Interrupt Vector table - **IVT**.	No such issue.

Types of interrupts

1. VECTORED AND NON VECTORED INTERRUPTS

A key element in interrupt handling is the ISR address.

If an interrupt has a fixed ISR address, it is called a Vectored interrupt.
Such interrupts are **executed faster** as the ISR address is known to the processor.
But such interrupts are **rigid**. Since they have a fixed ISR address they can serve only **one device**.
They cannot accept interrupts from multiple devices.
So they cannot expand the interrupt structure.
E.g.:: NMI interrupt of 8086 (Has a fixed vector number i.e. 2)

If an interrupt does not have a fixed ISR address, it is called a Non-Vectored interrupt.
Such interrupts are **executed slower**.
The ISR address is **obtained from the** interrupting **device**, usually an interrupt controller like **8259**.
But such interrupts are **flexible**.
Since they don't have a fixed ISR address they **can accept interrupts from multiple devices**.
So they can be used to **expand the interrupt structure**.
E.g.:: INTR interrupt of 8086 (Can service any vector number from 0… 255)

2. MASKABLE AND NON MASKABLE INTERRUPTS

Masking an interrupt means disabling it.

A **Maskable** interrupt is an interrupt that **can be disabled**.
If disabled, whenever this interrupt occurs, the processor will ignore it and simply continue the main program. Such interrupts are generally used to handle low priority, non-critical events like keyboard presses which can be easily disabled (Keypad can be locked)
E.g.:: INTR interrupt of 8086 (is disabled when Interrupt Flag is 0)

A **Non-Maskable** interrupt is an interrupt that **cannot be disabled**.
Whenever this interrupt occurs, the processor will have to service it.
Such interrupts are generally used to handle high priority, critical events like over-heating of the mother board, power failure etc.
E.g.:: NMI interrupt of 8086 (can never be disabled)

3. SOFTWARE AND HARDWARE INTERRUPTS

This is based on how the interrupt occurs.

If an interrupt is caused by **writing an instruction**, it is called a **software interrupt**.
Software interrupts are predictable events and are given by the programmer.
E.g.:: INT n instruction of 8086 (n can be anything between 0… 255)

If an interrupt is caused by **a signal on an external pin**, it is called a **hardware interrupt**.
Hardware interrupts are un-predictable events and are given by external devices.
E.g.:: NMI and INTR pins of 8086