JEPPIAAR INSTITUTE OF TECHNOLOGY

"Self-Belief | Self Discipline | Self Respect"

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

LECTURE NOTES

CS8491 – COMPUTER ARCHITECTURE

(Regulation 2017)

Year/Semester: II/IV CSE

2020 – 2021

Prepared by

Ms. R. Revathi

Assistant Professor/CSE

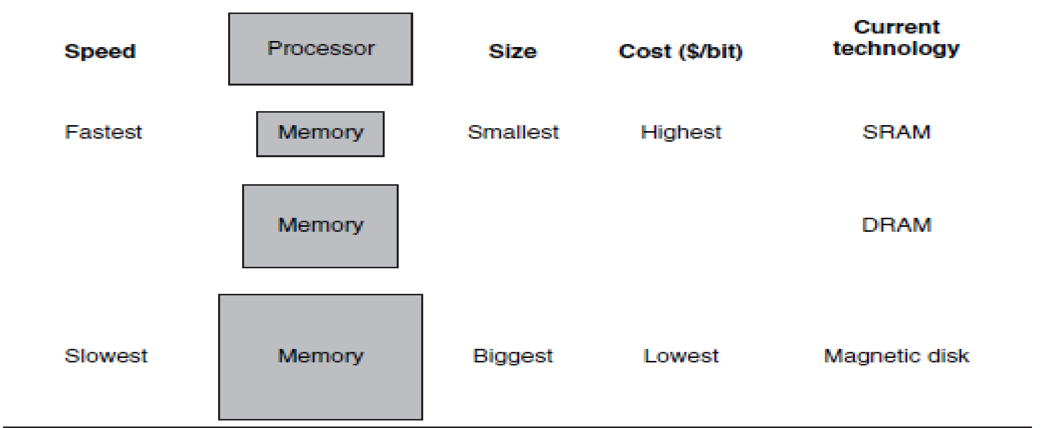
UNIT V

UNIT V MEMORY & I/O SYSTEMS 9

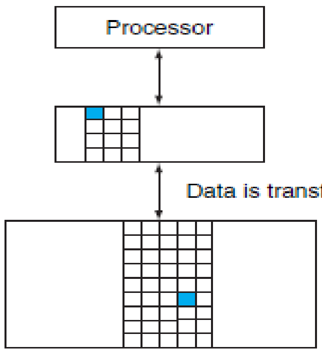
Memory Hierarchy - memory technologies – cache memory – measuring and improving cache performance – virtual memory, TLB‘s – Accessing I/O Devices – Interrupts – Direct Memory Access – Bus structure – Bus operation – Arbitration – Interface circuits - USB.

MEMORY HIERARCHY:

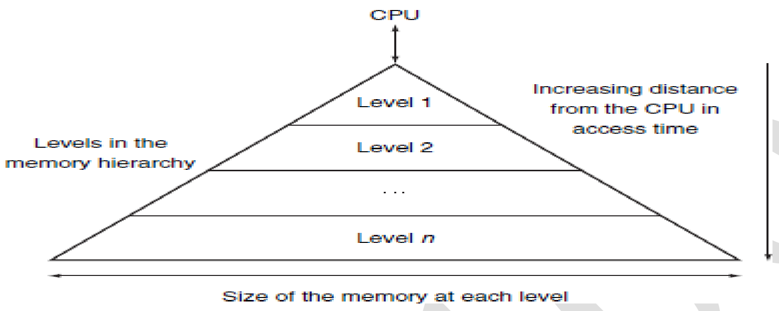
- A memory hierarchy consists of multiple levels of memory with different speeds and sizes.
- The faster memories are smaller and more expensive per bit than the slower memories.



- Figure shows the faster memory is close to the processor and the slower, less expensive memory is below it.
- A memory hierarchy can consist of multiple levels, but data is copied between only two adjacent levels at a time.
- The upper level is the one closer to the processor and smaller and faster than the lower level, since the upper level uses technology that is more expensive.
- **Block:** The minimum unit of information that can be either present or not present in the two-level hierarchy is called a **block or a line.**
- **Hit:** If the data requested by the processor appears in some block in the upper level, this is called a **hit**.
- **Miss:** If the data is not found in the upper level, the request is called a **miss.**
- The lower level in the hierarchy is then accessed to retrieve the block containing the requested data.
- **Hit rate or Hit ratio:** It is the fraction of memory accesses found in the upper level; it is often used as a measure of the performance of the memory hierarchy.
- **Miss rate**: Miss rate (1−hit rate) is the fraction of memory accesses not found in the upper level.
- **Hit time:** It is the time to access the upper level of the memory hierarchy, which includes the time needed to determine whether the access is a hit or a miss.
- **Miss penalty:** It is the time to replace a block in the upper level with the corresponding block from the lower level, plus the time to deliver this block to the processor.



Every pair of levels in the memory hierarchy can be thought of as having an upper and lower level.


CPU
Level 1
Level 2
. . .
Level n
Levels in the memory hierarchy
Increasing distance from the CPU in access time
Size of the memory at each level

- **Principle of locality:** It states that programs access a relatively small portion of their address space at any instant of time. There are two different types of locality:
- **Temporal locality:** The principle stating that if a data location is referenced then it will be likely to be referenced again soon.
- **Spatial locality:** The locality principle stating that if a data location is referenced, data locations with nearby addresses will be likely to be referenced soon.

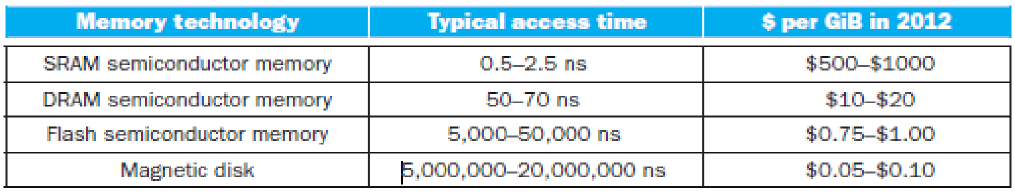
MEMORY TECHNOLOGY:

There are four primary technologies used today in memory hierarchies.

1. DRAM (Dynamic Random Access Memory)
2. SRAM (Static Random Access Memory).
3. Flash memory.
4. Magnetic disk.

- Main memory is implemented from DRAM (Dynamic Random Access Memory).
- Levels closer to the processor (caches) use SRAM (Static Random Access Memory).
- DRAM is less costly per bit than SRAM, although it is significantly slower. The price difference arises because DRAM uses significantly less area per bit of memory, and DRAMs thus have larger capacity for the same amount of silicon.
- The third technology is flash memory. This nonvolatile memory is the secondary memory in Personal Mobile Devices.
- The fourth technology, used to implement the largest and slowest level in the hierarchy in servers, is magnetic disk.
- The access time and price per bit vary widely among these technologies, as the table shows, below.

SRAM Technology



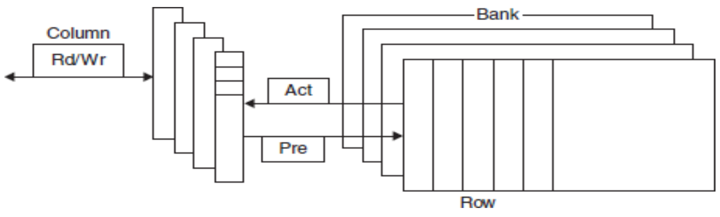
Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

- SRAMs are simply integrated circuits that are memory arrays with (usually) a single access port that can provide either a read or a write.
- SRAMs have a fixed access time to any datum, though the read and write access times may differ.
- SRAMs don't need to refresh and so the access time is very close to the cycle time.
- SRAMs typically use six to eight transistors per bit to prevent the information from being disturbed when read.
- SRAM needs only minimal power to retain the charge in standby mode.
- All levels of caches are integrated onto the processor chip.
- In a SRAM, as long as power is applied, the value can be kept indefinitely.

DRAM Technology

- In a dynamic RAM (DRAM), the value kept in a cell is stored as a charge in a capacitor.
- A single transistor is then used to access this stored charge, either to read the value or to overwrite the charge stored there. Because DRAMs use only a single transistor per bit of storage, they are much denser and cheaper per bit than SRAM.
- As DRAMs store the charge on a capacitor, it cannot be kept indefinitely and must periodically be refreshed. That is why this memory structure is called dynamic, as opposed to the static storage in an SRAM cell.
- To refresh the cell, we merely read its contents and write it back. The charge can be kept for several milliseconds.
- Figure shows the internal organization of a DRAM, and shows the density, cost, and access time of DRAMs.
- Modern DRAMs are organized in banks, typically four for DDR3. Each bank consists of a series of rows.
- When the row is in the buffer, it can be transferred by successive column addresses at whatever the width of the DRAM is (typically 4, 8, or 16 bits in DDR3) or by specifying a block transfer and the starting address.
- To further improve the interface to processors, DRAMs added clocks and are properly called **Synchronous DRAMs or SDRAMs.**
- The advantage of **SDRAMs** is that the use of a clock eliminates the time for the memory and processor to synchronize.
- The speed advantage of synchronous DRAMs comes from the ability to transfer the bits in the burst without having to specify additional address bits.

- The fastest version is called **Double Data Rate (DDR) SDRAM.** The name means data transfers on both the rising and falling edge of the clock, thereby getting twice as much bandwidth as you might expect based on the clock rate and the data width.



- Sending an address to several banks permits them all to read or write simultaneously. For example, with four banks, there is just one access time and then accesses rotate between the four banks to supply four times the bandwidth. This rotating access scheme is called **address interleaving.**

Flash Memory

- Flash memory is a type of electrically erasable programmable read-only memory (EEPROM).
- EEPROM technologies use flash memory bits for writing purpose.
- Most flash products include a controller to spread the writes by remapping blocks that have been written many times to less trodden blocks. This technique is called **wear leveling.**
- Personal mobile devices are very unlikely to exceed the write limits in the flash.

Disk Memory

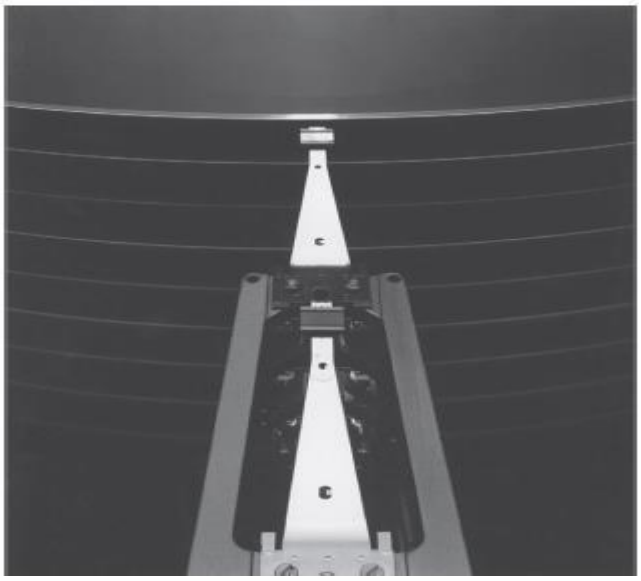
- A magnetic hard disk consists of a collection of platters, which rotate on a spindle at 5400 to 15,000 revolutions per minute.
- The metal platters are covered with magnetic recording material on both sides, similar to the material found on a cassette or videotape.
- To read and write information on a hard disk, a movable arm containing a small electromagnetic coil called a read-write head is located just above each surface.
- The disk heads to be much closer to the drive surface.
- **Tracks:** Each disk surface is divided into concentric circles, called **tracks.** There are typically tens of thousands of tracks per surface.
- **Sector:** Each track is in turn divided into **sectors** that contain the information; each track may have thousands of sectors. Sectors are typically 512 to 4096 bytes in size.
- The sequence recorded on the magnetic media is a sector number, a gap, the information for that sector including error correction code.
- The disk heads for each surface are connected together and move in conjunction, so that every head is over the same track of every surface.
- The term cylinder is used to refer to all the tracks under the heads at a given point on all surfaces.

- **Seek Time:** To access data, the operating system must direct the disk through a three-stage process. The first step is to position the head over the proper track. This operation is called a seek, and the time to move the head to the desired track is called the **seek time.**

- Average seek times are usually advertised as 3 ms to 13 ms, but, depending on the application and scheduling of disk requests.
- **Rotational Latency:** Once the head has reached the correct track, we must wait for the desired sector to rotate under the read/write head. This time is called the **rotational latency or rotational delay.**
- The average latency to the desired information is halfway around the disk. Disks rotate at 5400 RPM to 15,000 RPM. The average rotational latency at 5400 RPM is Average rotational latency 0.5 rotation.

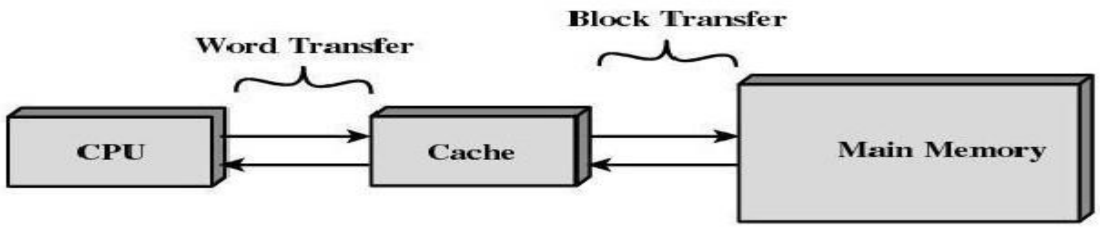
$$\text{Average rotational latency} = \frac{0.5\text{ rotation}}{5400\text{ RPM}} = \frac{0.5\text{ rotation}}{5400\text{ RPM}/\left(60\dfrac{\text{seconds}}{\text{minute}}\right)}$$
$$= 0.0056\text{ seconds} = 5.6\text{ ms}$$

- The last component of a disk access, transfer time, is the time to transfer a block of bits.
- The transfer time is a function of the sector size, the rotation speed, and the recording density of a track.
- In summary, the two primary differences between magnetic disks and semiconductor memory technologies are that disks have a slower access time because they are mechanical devices flash is 1000 times as fast and DRAM is 100,000 times as fast.
- Magnetic disks memory is cheaper per bit because they have very high storage capacity at a modest cost disk is 10 to 100 times cheaper.
- Magnetic disks are nonvolatile like flash, but unlike flash there is no write wear-out problem. However, flash is much more rugged and hence a better match to the jostling inherent in personal mobile devices.



CACHE MEMORY

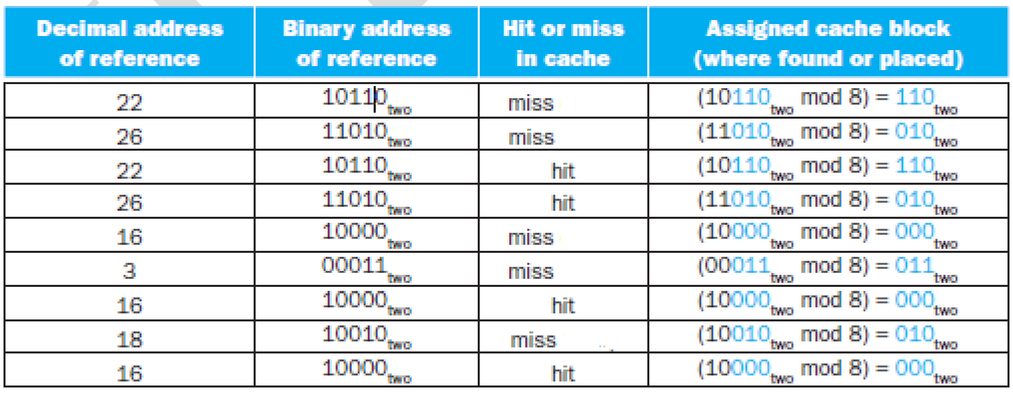
- Cache to represent the level of the memory hierarchy between the processor and main memory.



- If the number of entries in the cache is a power of 2, then modulo can be computed simply by using the low-order $\log_2$ (cache size in blocks) bits of the address.
- Because each cache location can contain the contents of a number of different memory locations.
- The tags contain the address information required to identify whether a word in the cache corresponds to the requested word.
- The tag needs only to contain the upper portion of the address, corresponding to the bits that are not used as an index into the cache. The lower 3-bit index field of the address selects the block.
- **Valid Bit:** The most common method is to add a valid bit a field in the tables of a memory hierarchy that indicates that the associated block in the hierarchy contains **valid data.** i.e., to indicate whether an entry contains a valid address. If the bit is not set, there cannot be a match for this block.

Accessing a Cache

- Below is a sequence of nine memory references to an empty eight-block cache, including the action for each reference.
- Figure shows how the contents of the cache change on each miss.

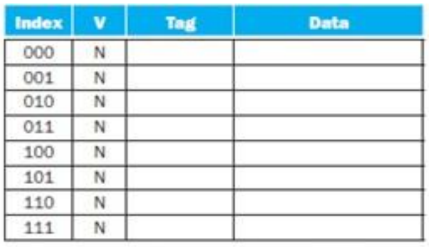


Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

- Since there are eight blocks in the cache, the low-order three bits of an address give the block number:
- The cache is initially empty, with all valid bits (V entry in cache) turned off (N). The processor requests the following addresses: 10110two (miss), 11010two (miss), 10110two (hit), 11010two (hit), and 10000two (miss), 00011two (miss), 10000two (hit), 10010two (miss), and 10000two (hit).
- The figures show the cache contents after each miss in the sequence has been handled.

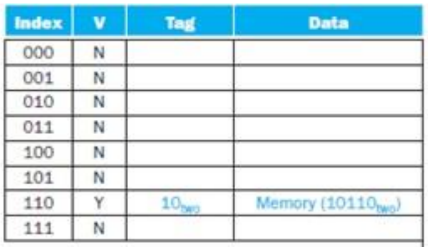
The tag field will contain only the upper portion of the address.

- A tag field, which is used to compare with the value of the tag field of the cache and a cache index, which is used to select the block.



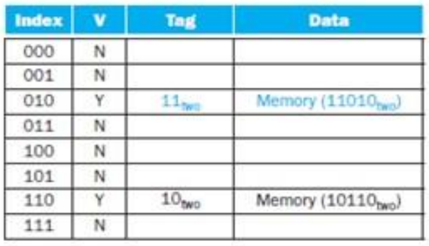
Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on



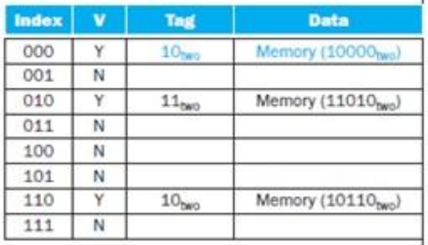
Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

b. After handling a miss of address (10110_{two})



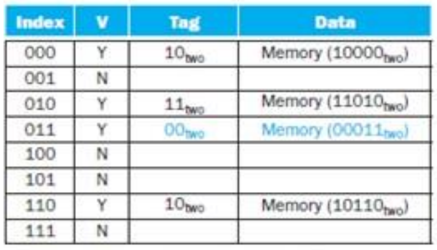
Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})



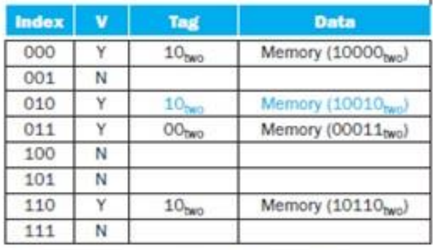
Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})



Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})



Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

f. After handling a miss of address (10010_{two})

- The total number of bits needed for a cache is a function of the cache size and the address size, because the cache includes both the storage for the data and the tags.
- The size of the block above was one word, but normally it is several. For the following situation:

   1. 32-bit addresses
   2. A direct-mapped cache
   3. The cache size is 2^n blocks, so n bits are used for the index
   4. The block size is 2^m words (2^{m+2} bytes), so m bits are used for the word within the block, and two bits are used for the byte part of the address the size of the tag field is,

$$\mathbf{32-(n+m+2)}$$

The total number of bits in a direct-mapped cache is

$$\mathbf{2^n \times (block\ size + tag\ size + valid\ field\ size).}$$

- This cache holds 1024 words or 4 KiB. We assume 32-bit addresses in this chapter. The tag from the cache is compared against the upper portion of the address to determine whether the entry in the cache corresponds to the requested address.
- Because the cache has 2^{10} (or 1024) words and a block size of one word, 10 bits are used to index the cache, leaving $32 - 10 - 2 = 20$ bits to be compared against the tag.
- If the tag and upper 20 bits of the address are equal and the valid bit is on, then the request

hits in the cache, and the word is supplied to the processor. Otherwise, a miss occurs.

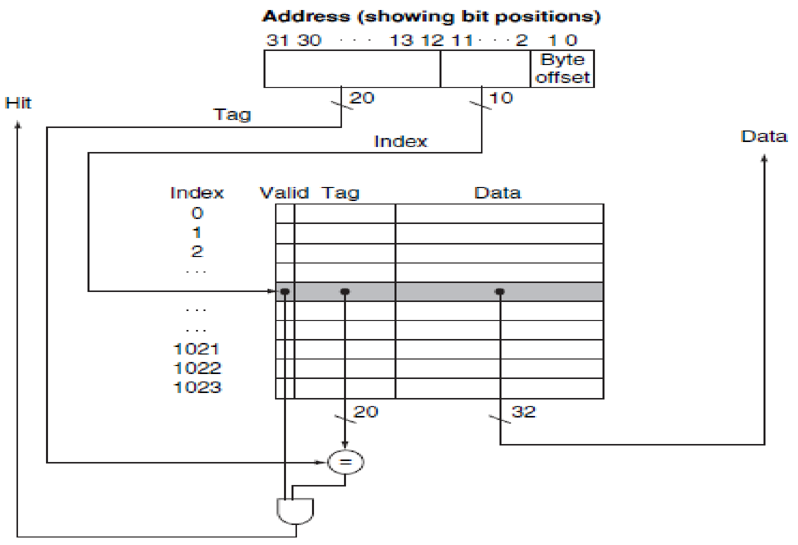


FIGURE For this cache, the lower portion of the address is used to select a cache entry consisting of a data word and a tag.

Example: 1

Mapping an Address to a Multiword Cache Block

Consider a cache with 64 blocks and a block size of 16 bytes. To what block number does byte address 1200 map?

We saw the formula on page 384. The block is given by

(Block address) modulo (Number of blocks in the cache)

where the address of the block is

$$\frac{\text{Byte address}}{\text{Bytes per block}}$$

Notice that this block address is the block containing all addresses between

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Bytes per block}$$

and

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Bytes per block} + (\text{Bytes per block} - 1)$$

Thus, with 16 bytes per block, byte address 1200 is block address

$$\left\lfloor \frac{1200}{6} \right\rfloor = 75$$

which maps to cache block number (75 modulo 64) = 11. In fact, this block maps all addresses between 1200 and 1215.

Example: 2

How many total bits are required for a direct-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?

Solution:

We know that 16 KiB is 4096 (2^{12}) words. With a block size of 4 words (2^2), there are 1024 (2^{10}) blocks. Each block has 4 * 32 or 128 bits of data plus a tag, which is (32-10-2-2) bits, plus a valid bit. Thus, the total cache size is

$$2^{10} \times (4 \times 32 + (32 - 10 - 2 - 2) + 1) = 2^{10} \times 147 = 147 \text{ Kibibits}$$

or 18.4 KiB for a 16 KiB cache. For this cache, the total number of bits in the cache is about 1.15 times as many as needed just for the storage of the data.

CACHE MEMORY MAPPING TECHNIQUES:

1. Direct Mapping

- **Direct-Mapped Cache:** A cache structure in which each memory location is mapped to exactly one location in the cache.

(Block address) modulo (Number of blocks in the cache)

The contention may occur,

- When the cache is full
- When more than one memory block is mapped onto a given cache block position.

The contention is resolved by allowing the new blocks to overwrite the currently resident block. Placement of block in the cache is determined from memory address. The memory address is divided into 3 fields. They are,

- **Low Order 4 bit field(word)->**Selects one of 16 words in a block.
- **7 bit cache block field->**When new block enters cache,7 bit determines the cache position in which this block must be stored.
- **5 bit Tag field->**The high order 5 bits of the memory address of the block is stored in 5 tag

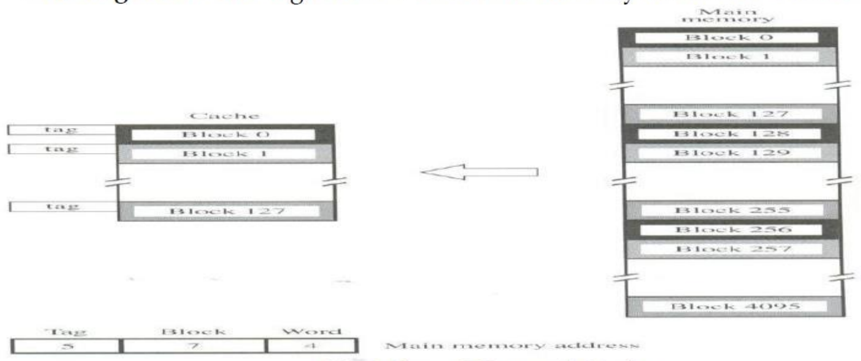


Fig: Direct Mapped Cache

Merit:
It is easy to implement
Demerit:
It is not very flexible.

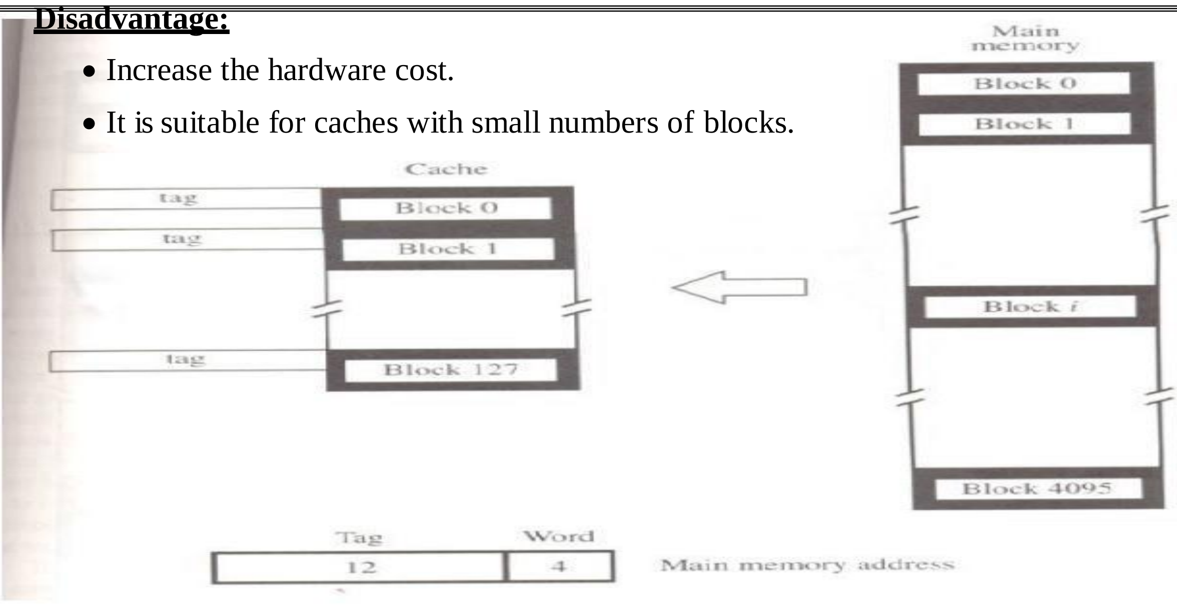
2. Fully Associative:

- Where a block can be placed in any location in the cache. Such a scheme is called **fully associative.** It is also called associative Mapping method.
- To find a given block in a fully associative cache, all the entries in the cache must be searched because a block can be placed in any one.

Advantage: More Flexible than Direct Mapping

Disadvantage:

- Increase the hardware cost.
- It is suitable for caches with small numbers of blocks.



ology

3. Set Associative:

- The middle range of designs between direct mapped and fully associative is called **set associative.**
- In a set-associative cache, there are a fixed number of locations where each block can be placed.
- A set-associative cache with n locations for a block is called an n-way set-associative cache.

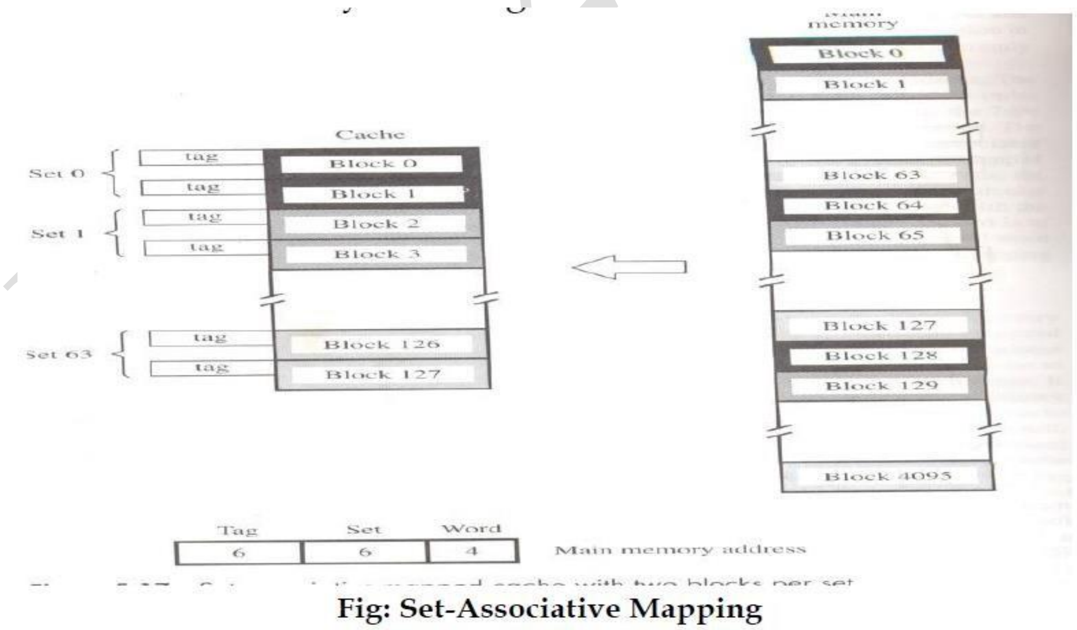


Fig: Set-Associative Mapping

- An n-way set-associative cache consists of a number of sets, each of which consists of n blocks.
- Each block in the memory maps to a unique set in the cache given by the index field, and a block can be placed in any element of that set.
- In a set-associative cache, the set containing a memory block is given by

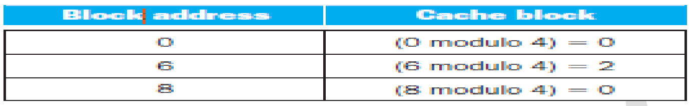
(Block number) modulo (Number of *sets* in the cache)

Example:1

Assume there are three small caches, each consisting of four one-word blocks. One cache is fully associative, a second is two-way set-associative, and the third is direct-mapped. Find the number of misses for each cache organization given the following sequence of block addresses: 0, 8, 0, 6, and 8.

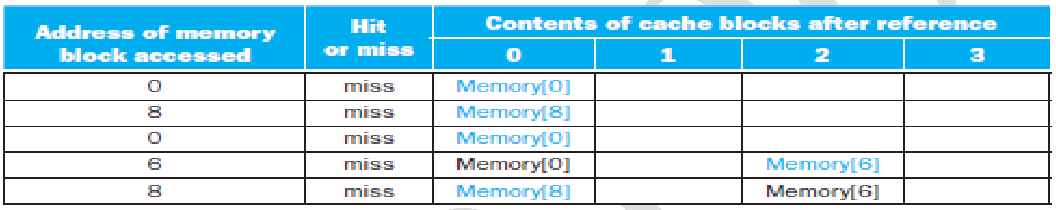
Solution:

rect Map: Number of blocks: 4



Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

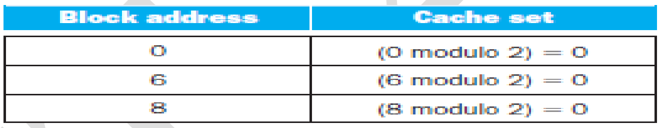
- The direct-mapped cache generates five misses for the five accesses.



Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

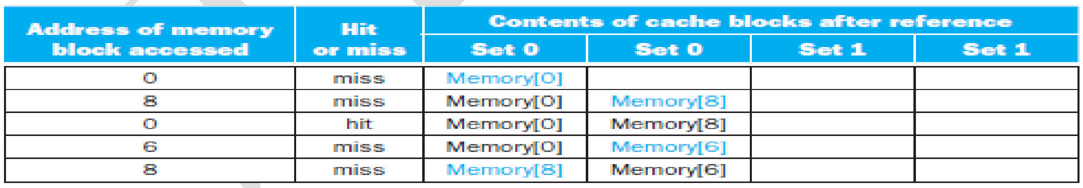
Set Associative:

- The set-associative cache has two sets (with indices 0 and 1) with two elements per set. Let's first determine to which set each block address maps:



Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

- Set-associative caches usually replace the least recently used block within a set;



Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

- Notice that when block 6 is referenced, it replaces block 8, since block 8 has been less recently referenced than block 0. The two-way set-associative cache has four misses, one less than the direct-mapped cache.

Fully Associative:

- The fully associative cache has four cache blocks (in a single set); any memory block can be stored in any cache block.
- The fully associative cache has the best performance, with only three misses:

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

The implementation of a four-way set-associative cache requires four comparators and a 4-to-1 multiplexor.

- The following figure shows that in a four-way set-associative cache, four comparators are needed, together with a 4-to-1 multiplexor to choose among the four potential members of the selected set.
- The comparators determine which element of the selected set (if any) matches the tag.
- The output of the comparators is used to select the data from one of the four blocks of the indexed set, using a multiplexor with a decoded select signal.
- In some implementations, the Output enables signals on the data portions of the cache RAMs can be used to select the entry in the set that drives the output.
- The Output enable signal comes from the comparators, causing the element that matches to drive the data outputs. This organization eliminates the need for the multiplexor.

Example:

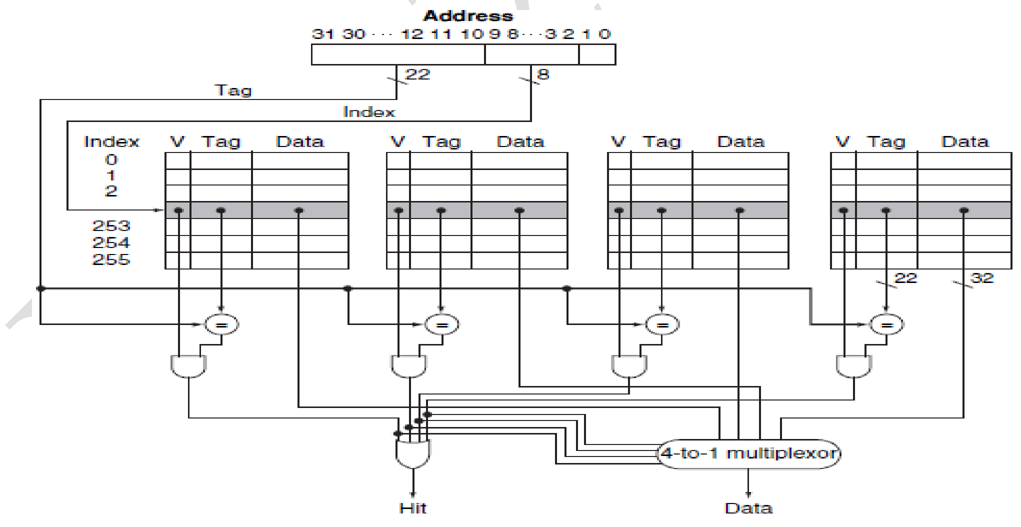


FIGURE The implementation of a four-way set-associative cache requires four comparators and a 4-to-1 multiplexor.

Increasing associativity requires more comparators and more tag bits per cache block. Assuming a cache of 4096 blocks, a 4-word block size, 16 bytes per block and a 32-bit address, find the total number of sets and the total number of tag bits for caches that are direct mapped, two-way and four-way set associative, and fully associative.

Solution:

- Number of bytes per block is : 16 [24]. So, 32-bit address yields 32-4=28 bits to be used for index and tag.

Direct Mapped cache:

- The direct-mapped cache has the same number of sets as blocks, and 12 bits of index (212=4096). Hence, the total number is (28-12) ×4096 =16 ×4096 =66 K tag bits.

Two-Way Set-Associative Cache:

- Thus, for a two-way set-associative cache, there are 2048 (211=2048) sets. The total number of tag bits is (28-11) × 2× 2048=34× 2048=70 Kbits.

Four-way set-associative cache:

- For a four-way set-associative cache, the total number of sets is 1024 (210=1024) sets. The total number is (28-10)×4×1024=72×1024=74 K tag bits.

Fully associative cache:

- For a fully associative cache, there is only one set with 4096 blocks, and the tag is 28 bits, leading to 28×4096×1=115 K tag bits.

Least Recently Used (LRU):

The algorithm which replaces the page that has not been used for the longest period of time is referred to as the *least recently used* (LRU) algorithm.

The result of applying LRU replacement to our example reference string is shown in Fig. The LRU algorithm produces 12 faults. It can be noticed that the first five faults are the same as the optimal replacement. When the reference to page 4 occurs, however, LRU replacement sees that, of the three frames in memory, page 2 was used least recently. The most recently used page is page 0, and just before that page 3 was used. Thus, the LRU algorithm replaces page 2, not knowing that page 2 is about to be used. When it then faults for page 2, the LRU algorithm replaces page 3 since, of the three pages in memory {0, 3, 4}, page 3 is the least recently used. Despite these problems, LRU replacement with 12 faults is still much better than FIFO replacement with 15.

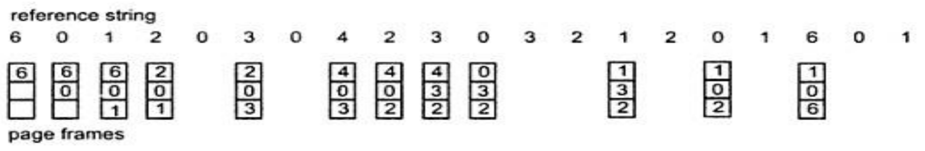


Fig. LRU page-replacement algorithm

First in First out (FIFO):

- A replacement scheme in which the block replaced is the one that has been entered first in the cache memory

For our example reference string, our three frames are initially empty. The first three references (6, 0, 1) cause page faults, and and brought into these empty frames.

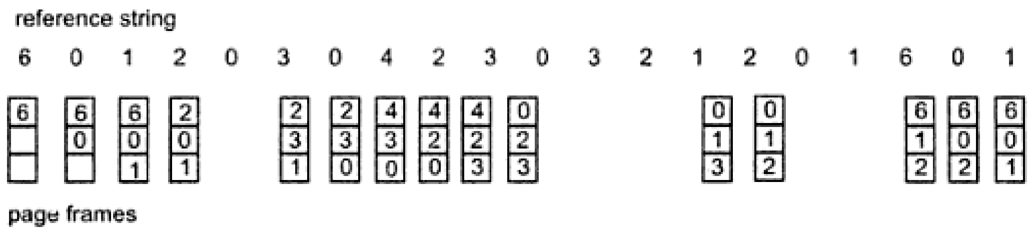


Fig. FIFO page-replacement algorithm

The next reference (2) replaces page 6, because page 6 was brought in first. Since 0 is the next reference and 0 is already in memory, we have no fault for this reference. The first reference to 3 results in page 0 being replaced, since it was the first of the three pages in memory (0, 1 and 2) to be brought in. This replacement means that the next reference, to 0, will fault. Page 1 is then replaced by page 0. This process continues as shown in Fig. 7.22. Every time a fault occurs, we show which pages are in out three frames. There are 15 faults algorithm.

MEASURING AND IMPROVING CACHE PERFORMANCE

- There are two different techniques for improving cache performance. The first one is **reducing the miss rate** by reducing the probability that two different memory blocks will contend for the same cache location.
- The second technique **reduces the miss penalty** by adding an additional level to the hierarchy. This technique, called multilevel caching.
- CPU time can be divided into the clock cycles that the CPU spends executing the program and the clock cycles that the CPU spends waiting for the memory system.
- Normally, we assume that the costs of cache accesses that are hits are part of the normal CPU execution cycles. Thus,

CPU time = (CPU execution clock cycles + Memory-stall clock cycles) × Clock cycle time

- The memory-stall clock cycles come primarily from cache misses. Memory-stall clock cycles can be defined as the sum of the stall cycles coming from reads plus those coming from write:

Memory-stall clock cycles = (Read-stall cycles + Write-stall cycles)

- The read-stall cycles can be defined in terms of the number of read accesses per program, the miss penalty in clock cycles for a read, and the read miss rate:

$$\text{Read-stall cycles} = \frac{\text{Reads}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

- Writes are more complicated. For a write-through scheme, we have two sources of stalls:
- **Write Misses:** we fetch the block before continuing the write and write buffer stalls, which occur when the write buffer is full when a write occurs. Thus, the cycles stalled for writes equals the sum of these two:

$$\text{Write-stall cycles} = \left(\frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty}\right) + \text{Write buffer stalls}$$

- Because the write buffer stalls depend on the proximity of writes. Fortunately, in systems with a reasonable write buffer depth (e.g., four or more words) and a memory capable of accepting writes at a rate that significantly exceeds the average write frequency in programs (e.g., by a factor of 2), the write buffer stalls will be small, and we can safely ignore them.
- If a system did not meet these criteria, it would not be well designed; instead, the designer should have used either a deeper write buffer or a write-back organization.
- **Write-back**: schemes also have potential additional stalls arising from the need to write a cache block back to memory when the block is replaced. If we assume that the write buffer stalls are negligible, we can combine the reads and writes by using a single miss rate and the miss penalty:

$$\text{Memory-stall clock cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

We can also factor this as

$$\text{Memory-stall clock cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Example:1

Calculating Average Memory Access Time

Find the AMAT for a processor with a 1 ns clock cycle time, a miss penalty of 20 clock cycles, a miss rate of 0.05 misses per instruction, and a cache access time (including hit detection) of 1 clock cycle. Assume that the read and write miss penalties are the same and ignore other write stalls.

The average memory access time per instruction is

$$\begin{aligned}\text{AMAT} &= \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}\\ &= 1 + 0.05 \times 20\\ &= 2 \text{ clock cycles}\end{aligned}$$

or 2 ns.

Handling Cache Miss

Cache miss

- It is a request for data from the cache that cannot be filled because the data is not present in the cache. We can now define the steps to be taken on an instruction cache miss:
  1. Send the original PC value (current PC – 4) to the memory.
  2. Instruct main memory to perform a read and wait for the memory to complete its access.
  3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.
  4. Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.

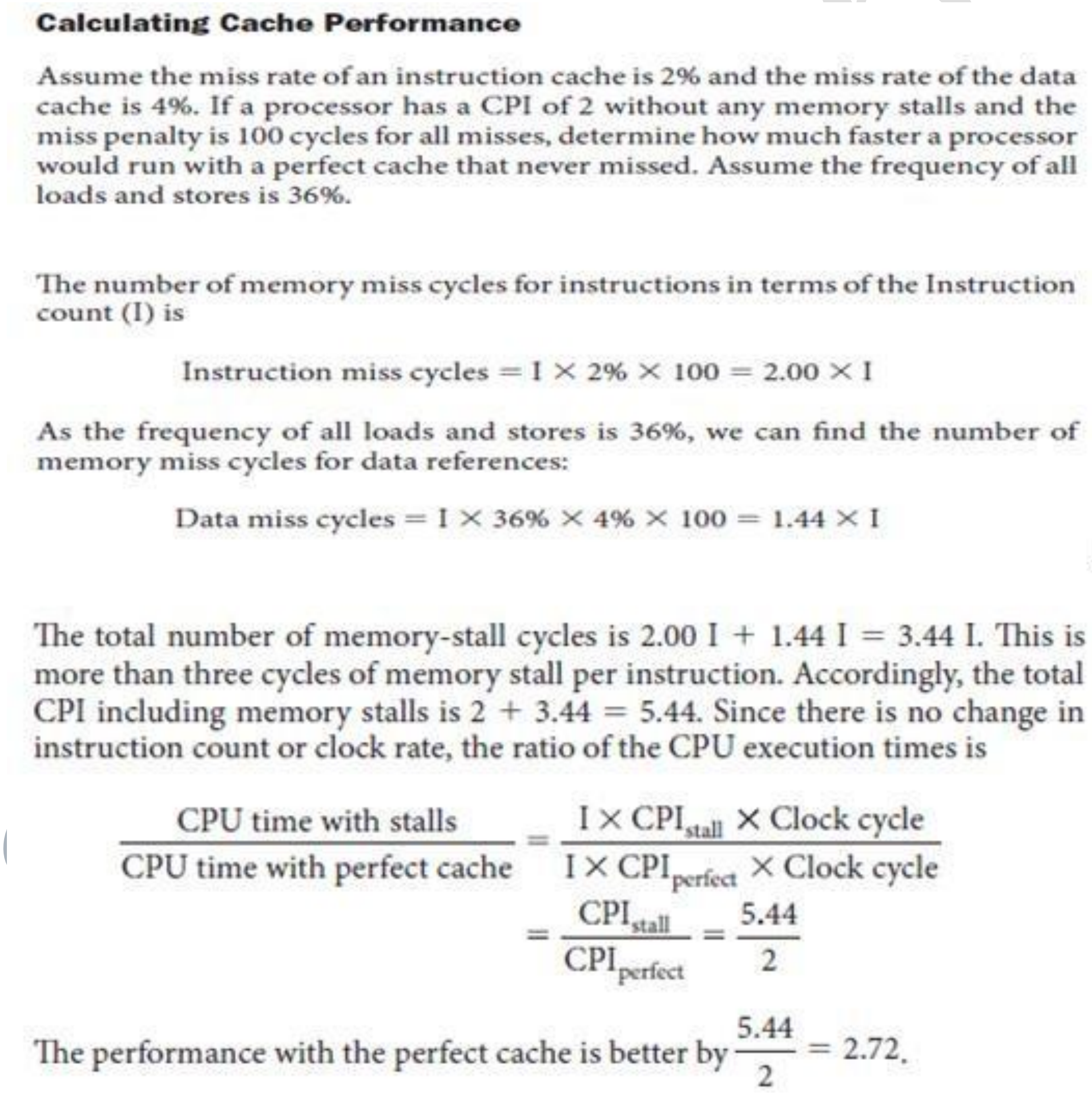
Handling Writes

- **Inconsistent:** After the write into the cache, memory would have a different value from that in the cache. In such a case, the cache and memory are said to be **inconsistent.**
- **Write Through:** The simplest way to keep the main memory and the cache consistent is always to write the data into both the memory and the cache. This scheme is called **write-through.**
- **Write Buffer:** A write buffer stores the data while it is waiting to be written to memory. After writing the data into the cache and into the write buffer, the processor can continue

execution.

- When a write to main memory completes, the entry in the write buffer is freed. If the write buffer is full when the processor reaches a write, the processor must stall until there is an empty position in the write buffer.
- **Write Back:** In a write-back scheme, when a write occurs, the new value is written only to the block in the cache. The modified block is written to the lower level of the hierarchy when it is replaced.
- Write-back schemes can improve performance and more complex to implement than write-through.
- **Split cache:** A scheme in which a level of the memory hierarchy is composed of two independent caches that operate in parallel with each other, with one handling instructions and one handling data.

Example:2



Calculating Cache Performance

Assume the miss rate of an instruction cache is 2% and the miss rate of the data cache is 4%. If a processor has a CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never missed. Assume the frequency of all loads and stores is 36%.

The number of memory miss cycles for instructions in terms of the Instruction count (I) is

$$\text{Instruction miss cycles} = \text{I} \times 2\% \times 100 = 2.00 \times \text{I}$$

As the frequency of all loads and stores is 36%, we can find the number of memory miss cycles for data references:

$$\text{Data miss cycles} = \text{I} \times 36\% \times 4\% \times 100 = 1.44 \times \text{I}$$

The total number of memory-stall cycles is 2.00 I + 1.44 I = 3.44 I. This is more than three cycles of memory stall per instruction. Accordingly, the total CPI including memory stalls is 2 + 3.44 = 5.44. Since there is no change in instruction count or clock rate, the ratio of the CPU execution times is

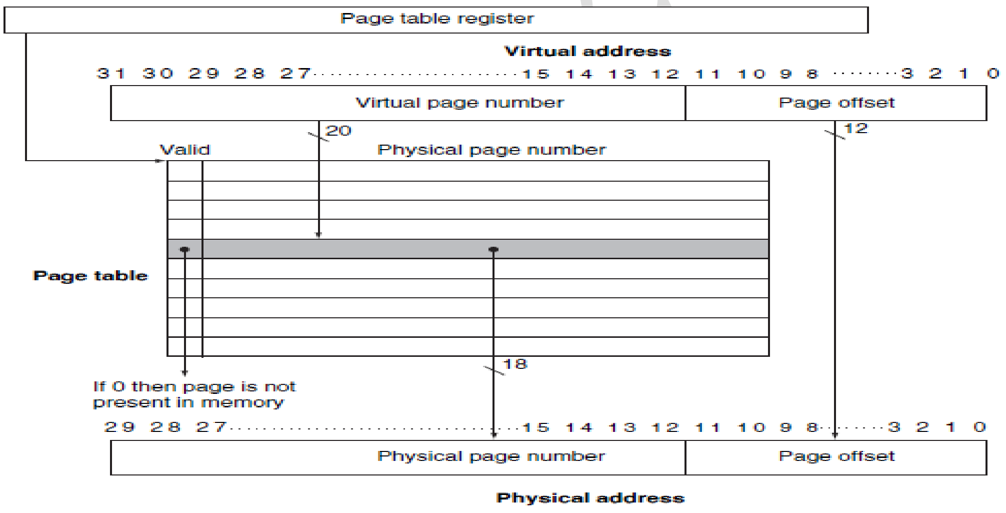
$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{\text{I} \times \text{CPI}_{\text{stall}} \times \text{Clock cycle}}{\text{I} \times \text{CPI}_{\text{perfect}} \times \text{Clock cycle}}$$

$$= \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{5.44}{2}$$

The performance with the perfect cache is better by $\frac{5.44}{2} = 2.72$.

VIRTUAL MEMORY:

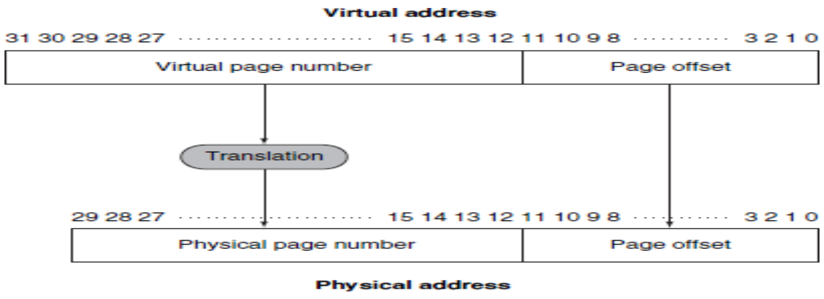
- **Virtual memory:** It is a technique that uses main memory as a "cache" for secondary storage.
- Virtual memory implements the translation of a program's address space to physical addresses. This translation process enforces protection of a program's address space from other virtual machines.
- **Protection**: It is a set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere, intentionally or unintentionally, with one another by reading or writing each other's data. These mechanisms also isolate the operating system from a user process.
- **Page fault:** It is an event that occurs when an accessed page is not present in main memory.
- **Address translation:** It is also called address mapping. The process by which a virtual address is mapped to an address used to access memory.
- In virtual memory, the address is broken into a virtual page number and a page offset. Figure shows the translation of the virtual page number to a physical page number.
- The physical page number constitutes the upper portion of the physical address, while the page off set, which is not changed, constitutes the lower portion.
- The number of bits in the page off set field determines the page size. The number of pages addressable with the virtual address need not match the number of pages addressable with the physical address.



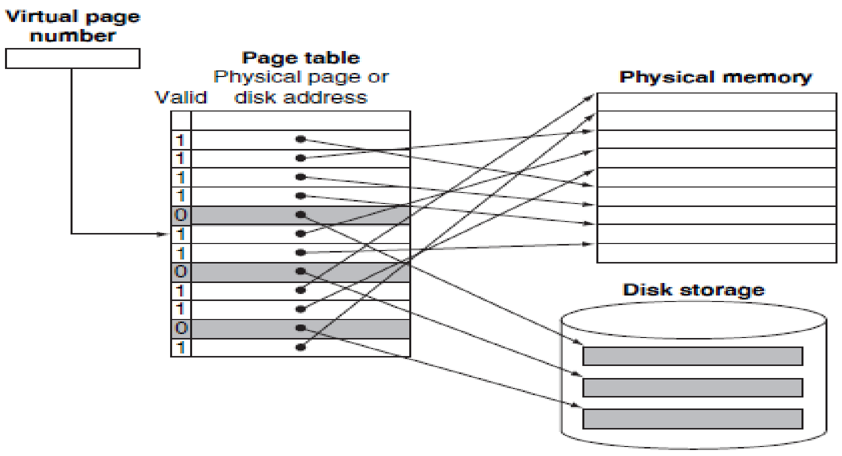
- **Pages:** Both the virtual memory and the physical memory are broken into pages, so that a virtual page is mapped to a physical page.
- Physical pages can be shared by having two virtual addresses point to the same physical address. This capability is used to allow two different programs to share data or code.
- **Segmentation**: It is a variable-size address mapping scheme in which an address consists of two parts: a segment number, which is mapped to a physical address, and a segment off set.
- **Page table**: It is the table containing the virtual to physical address translations in a virtual memory system. The table, which is stored in memory, is typically indexed by the virtual page number; each entry in the table contains the physical page number for that virtual

page if the page is currently in memory.

Mapping from virtual address into physical address

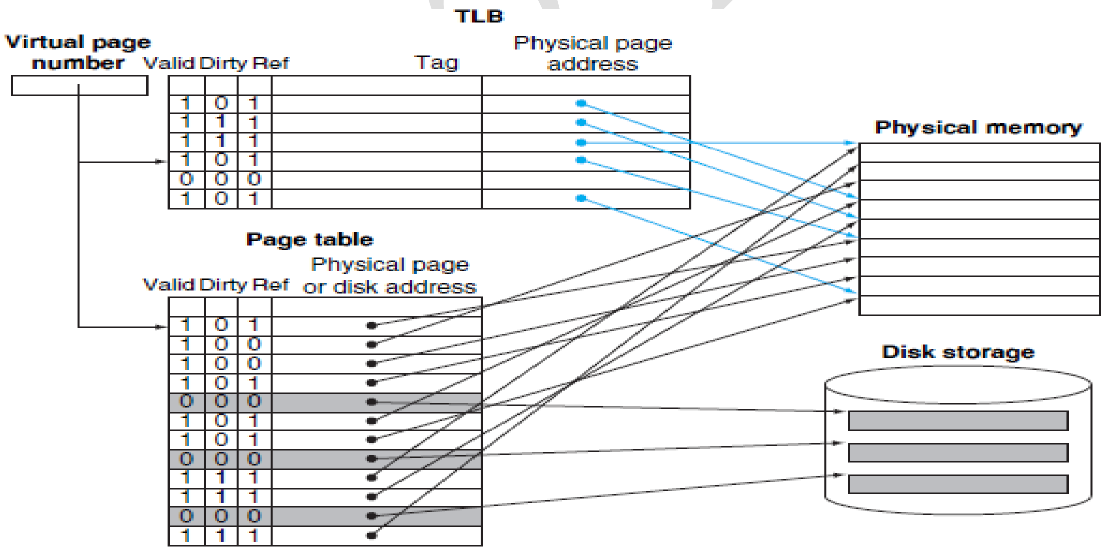


- The difficulty in using fully associative placement is in locating an entry, since it can be anywhere in the upper level of the hierarchy. A full search is impractical.
- In virtual memory systems, we locate pages by using a table that indexes the memory; this structure is called a page table, and it resides in memory.
- A page table is indexed with the page number from the virtual address to discover the corresponding physical page number. Each program has its own page table, which maps the virtual address space of that program to main memory.
- **Page Table Register:** To indicate the location of the page table in memory, the hardware includes a register that points to the start of the page table; we call this the **page table register**. Assume for now that the page table is in a fixed and contiguous area of memory.
- When a page fault occurs, if all the pages in main memory are in use, the operating system must choose a page to replace.
- Because we want to minimize the number of page faults, most operating systems try to choose a page that they hypothesize will not be needed in the near future.
- Using the past to predict the future, operating systems follow the least recently used (LRU) replacement scheme. The operating system searches for the least recently used page, assuming that a page that has not been used in a long time is less likely to be needed than a more recently accessed page. The replaced pages are written to swap space on the disk.
- **Swap space:** It is the space on the disk reserved for the full virtual memory space of a process.
- **Reference bit:** It is also called use bit. A field that is set whenever a page is accessed and that is used to implement LRU or other replacement schemes.



TRANSLATION-LOOKASIDE BUFFER (TLB):

- **Translation-lookaside buffer (TLB)** a cache that keeps track of recently used address mappings to try to avoid an access to the page table.
- Accordingly, modern processors include a special cache that keeps track of recently used translations. This special address translation cache is traditionally referred to as a **translation-lookaside buffer (TLB)**, although it would be more accurate to call it a translation cache.
- Figure shows that each tag entry in the TLB holds a portion of the virtual page number, and each data entry of the TLB holds a physical page number.

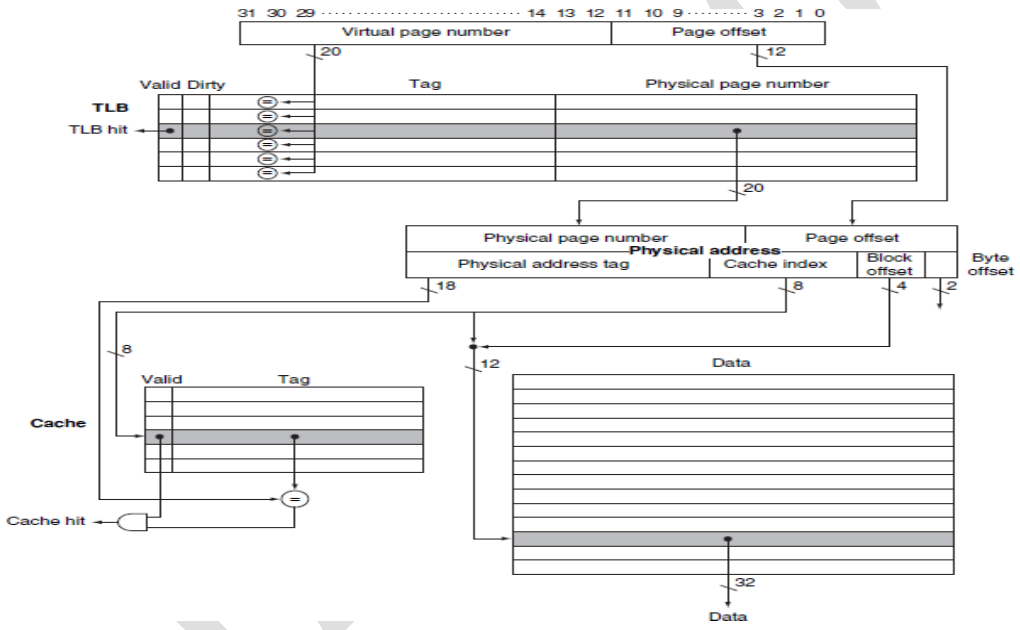


- Because we access the TLB instead of the page table on every reference, the TLB will need to include other status bits, such as the dirty and the reference bits. On every reference, we look up the virtual page number in the TLB.
- **Reference Bit:** If we get a hit, the physical page number is used to form the address, and the corresponding reference bit is turned on.
- **Dirty Bit:** If the processor is performing a write, the dirty bit is also turned on. If a miss in the TLB occurs, we must determine whether it is a page fault or purely a TLB miss.
- If the page exists in memory, then the TLB miss indicates only that the translation is missing. In such cases, the processor can handle the TLB miss by loading the translation

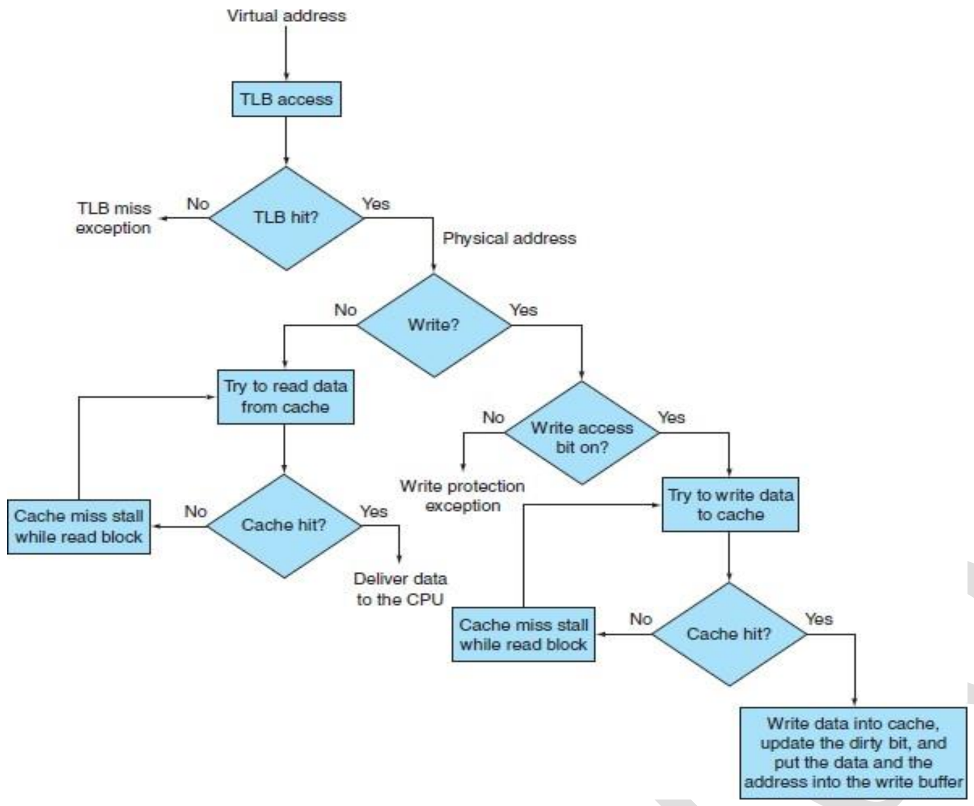
from the page table into the TLB and then trying the reference again.

- **True page Fault:** If the page is not present in memory, then the TLB miss indicates a true page fault. In this case, the processor invokes the operating system using an exception. Because the TLB has many fewer entries than the number of pages in main memory, TLB misses will be much more frequent than true page faults.
- TLB misses can be handled either in hardware or in software.
- After a TLB miss occurs and the missing translation has been retrieved from the page table, we will need to select a TLB entry to replace.
- Because the reference and dirty bits are contained in the TLB entry, we need to copy these bits back to the page table entry when we replace an entry. These bits are the only portion of the TLB entry that can be changed.
- Using write-back—that is, copying these entries back at miss time rather than when they are written—is very efficient, since we expect the TLB miss rate to be small.

- Some systems use other techniques to approximate the reference and dirty bits, eliminating the need to write into the TLB except to load a new table entry on a miss.

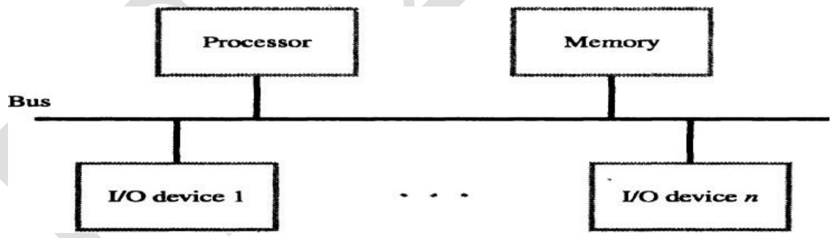


- **Virtually addressed cache** a cache that is accessed with a virtual address rather than a physical address.
- **Aliasing** a situation in which two addresses access the same object; it can occur in virtual memory when there are two virtual addresses for the same physical page.
- Physically addressed cache a cache that is addressed by a physical address.


Virtual address
TLB access
TLB hit?
No
Yes
TLB miss exception
Physical address
Write?
No
Yes
Try to read data from cache
Write access bit on?
No
Yes
Write protection exception
Try to write data to cache
Cache miss stall while read block
Cache hit?
No
Yes
Deliver data to the CPU
Cache miss stall while read block
Cache hit?
No
Yes
Write data into cache, update the dirty bit, and put the data and the address into the write buffer

ACCESSING I/O DEVICES

- A bus is a shared communication link, which uses one set of wires to connect multiple subsystems. Most modern computers use single bus arrangement for connecting I/O devices to CPU & Memory. The bus enables all the devices connected to it to exchange information.



Advantages

1. Versatility 2. Low cost.

Bus consists of 3 set of lines :

1. **Address:** Processor places a unique address for an I/O device on address lines.
2. **Data:** The data will be placed on Data lines. The data lines of the bus carry information between the source and the destination. This information may consist of data, complex commands, or addresses.
3. **Control:** The control lines are used to signal requests and acknowledgments, and to indicate what type of information is on the data lines. Processor requests for either Read / Write.

INPUT OUTPUT SYSTEM

1. Memory mapped I/O 2.Programmed I/O

1. Memory mapped I/O

- I/O devices and the memory share the same address space, the arrangement is called Memory-mapped I/O. In Memory-mapped I/O portions of address space are assigned to

I/O devices and reads and writes to those addresses are interpreted as commands to the I/O device.

2. Programmed I/O

- Programmed I/O is a method included in every computer for controlling I/O operation. It is most useful in small, low speed system where hardware cost must be minimized.
- Programmed I/O requires all I/O operation can be executed under the direct control of the CPU.
- Generally data transfer takes place between two registers. One is CPU register and other register is attached in I/O device. I/O device does not have direct access to main memory.
- A data transfer from an I/O device to memory required CPU to execute several instructions.
- Input instruction to transfer a word from the I/O device to the CPU and store instruction to transfer the word from the CPU to memory.

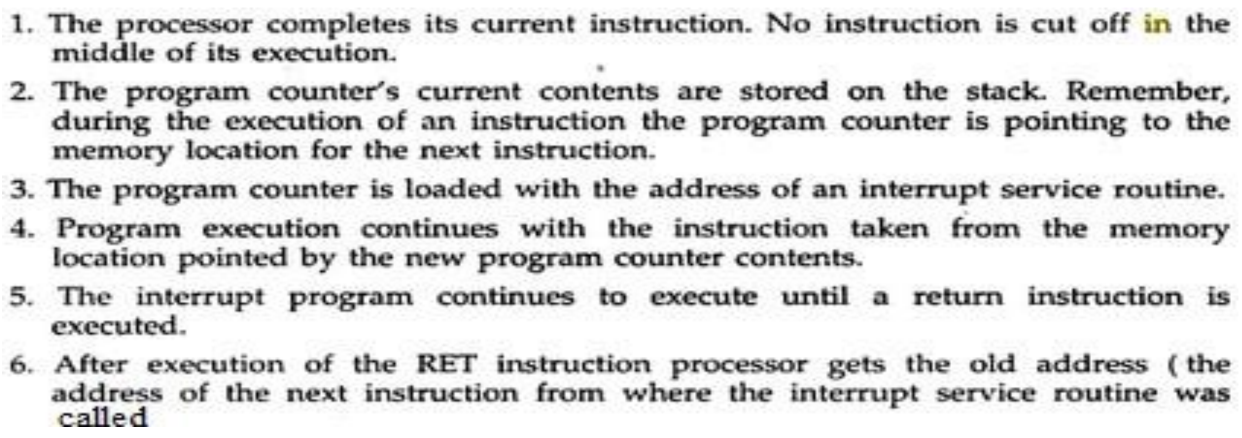
Techniques:

1. I/O Addressing 2. I/O-Mapped I/O 3. I/O Instruction 4. I/O Interface Circuit

INTERRUPTS

- The word interrupt is used to cause a CPU to temporarily transfer control from its current program to another program.
- When I/O Device is ready, it sends the INTERRUPT signal to processor via a dedicated controller line. In response to the interrupt, the processor executes the Interrupt Service Routine (ISR).
- All the registers, flags, program counter values are saved by the processor before running ISR.
- The time required to save status & restore contribute to execution overhead is called **"Interrupt Latency".**
- Various internal and external sources generate interrupts to the CPU. IO interrupts are external requests to the CPU to initiate or terminate an operation.
- If a power supply failure occurs then interrupt can be generated and interrupt handler to save critical data about the system's state, it is a kind of **hardware interrupt.**
- An instruction to divide by zero is an examples of **software interrupt.**

Steps:



1. The processor completes its current instruction. No instruction is cut off in the middle of its execution.
2. The program counter's current contents are stored on the stack. Remember, during the execution of an instruction the program counter is pointing to the memory location for the next instruction.
3. The program counter is loaded with the address of an interrupt service routine.
4. Program execution continues with the instruction taken from the memory location pointed by the new program counter contents.
5. The interrupt program continues to execute until a return instruction is executed.
6. After execution of the RET instruction processor gets the old address (the address of the next instruction from where the interrupt service routine was called

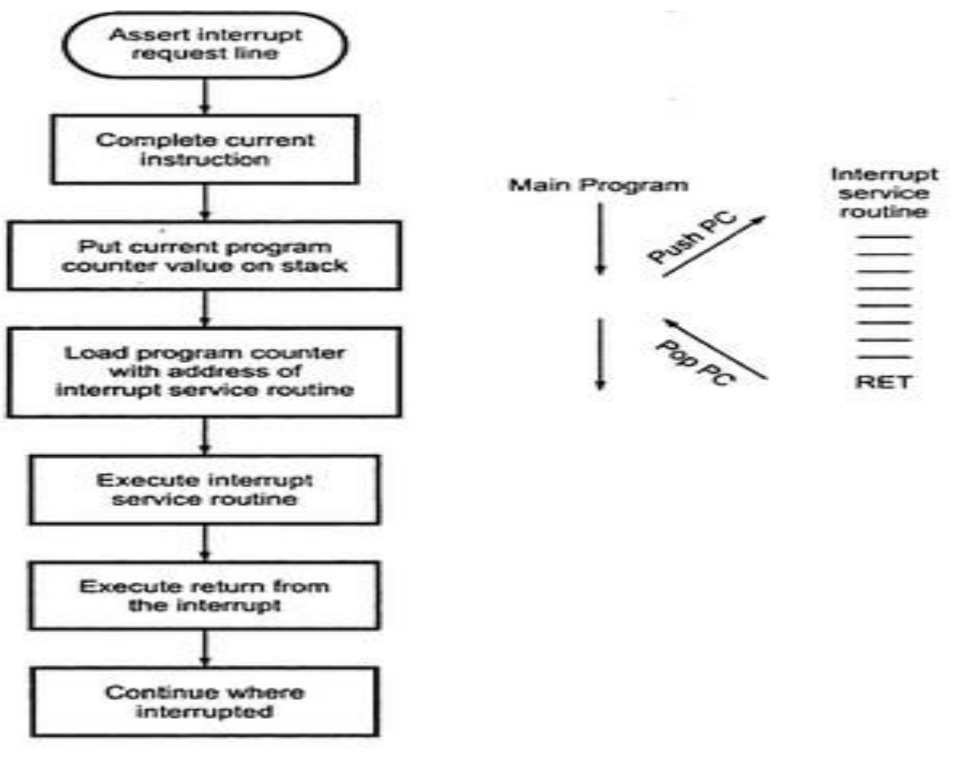

Assert interrupt request line
Complete current instruction
Put current program counter value on stack
Load program counter with address of interrupt service routine
Execute interrupt service routine
Execute return from the interrupt
Continue where interrupted
Main Program
Push PC
Pop PC
Interrupt service routine
RET

Fig. Response to an interrupt with the flowchart and diagram

<u>Handling Multiple Devices</u>

- Multiple devices can initiate interrupts. They uses the common interrupt request line Techniques are
    1. Maskable and Nonmaskable interrupt
    2. Single line interrupt(polling)
    3. Multilevel interrupt
    4. Vectored Interrupts
    5. Interrupt Nesting
    6. Daisy Chaining
    7. Daisy Chaining with priority group
    8. Pipeline Interrupt

1. <u>Maskable and Nonmaskable interrupt</u>

- Maskable interrupts are enabled and disabled under program control.
- By setting or resetting particular flip-flops in the processor, interrupts can be masked or unmasked.
- When masked processor does not respond to the interrupt even though the interrupt is activated. Most of the processor provides the masking facility.
- In the processor those interrupts which can be masked under software control are called **maskable interrupts.**
- In the processor those interrupts which cannot be masked under software control are called **nonmaskable interrupts.**
- Three methods of Controlling Interrupts (single device)

1. Ignoring interrupt
2. Disabling interrupts
3. Special Interrupt request line

1. Ignoring Interrupts

- Processor hardware ignores the interrupt request line until the execution of the first instruction of the ISR completed. Using an interrupt disable instruction after the first instruction of the ISR. A return from interrupt instruction is completed before further interruptions can occur.

2. Disabling Interrupts

- Processor automatically disables interrupts before starting the execution of the ISR.
- The processor saves the contents of PC and PS (status register) before performing interrupt disabling. The interrupt-enable is set to 0 means no further interrupts allowed.
- When return from interrupt instruction is executed the contents of the PS are restored from the stack, and the interrupt enable is set to 1

3. Special Interrupt line

- Special interrupt request line for which the interrupt handling circuit responds only to the leading edge of the signal Edge –triggered.
- Processor receives only one request regardless of how long the line is activated.
- No separate interrupt disabling instructions.

2. Single Line Interrupt

- In a single interrupts there can be many interrupting devices. But all interrupt requests are made via a single input pin of the CPU.
- In single line interrupt system, all IO ports share a single INTERRUPT REQUEST line. On responding to an interrupt request, the CPU must scan all the IO devices to determine the source of the interrupt.
- This procedure requires activating an INTERRUPT ACKNOWLEDGE line and that is connected in daisy chain fashion to all IO devices.
- Once the interrupting I/O port is identified, the CPU will service it and then return to task it was performing before the interrupt.
- Polling software routine that checks the logic state of each device. The Interrupt service routine polls the I/O devices connected to the bus.

- Main advantage of polling is it allowing the interrupt priority to be programmed.

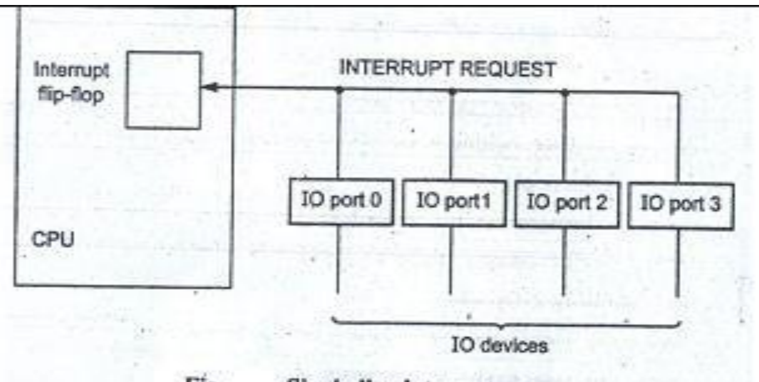


Fig. Single-line interrupt system

3. **Multilevel Interrupt**

- In multiple level interrupts, processor has more than one interrupt pins.
- The I/O devices are tied to the individual interrupt pins.
- The interrupts can be easily identified by the CPU upon receiving an interrupt request from it.
- This allows processor to go directly to that I/O device and service it without having to poll first.
- Advantage is saves the time in processing interrupt.

Multiple-Line Interrupt System

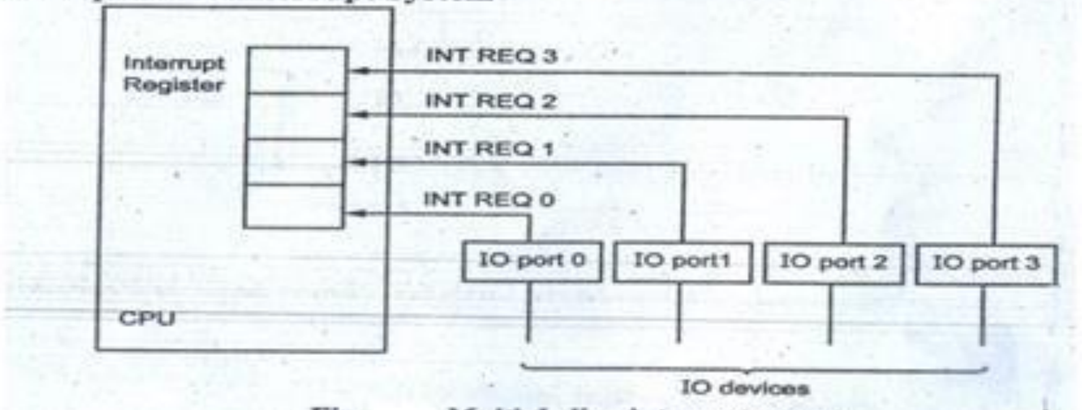


Fig. Multiple line interrupt system

4. **Vectored Interrupts**

- The following figure shows a basic way to derive interrupt vectors from multiple interrupt request lines.
- Each interrupt request line generates a unique fixed address and it is used to modify the CPU's program counter PC.
- Interrupt requests are stored on receipt in an interrupt register. The interrupt, mask register can disable any or all of the interrupt request lines under program control.
- The K masked interrupt signals are fed into a priority encoder that produces a [log2K] bit address and it is inserted into PC. PC finds the ISR address from the code.
- Device requesting an interrupt identifies itself directly to the processor. The device

sends a special code to the processor over the bus.

- The code contains the
    1. Identification of the device
    2. Starting address for the ISR
    3. Address of the branch to the ISR

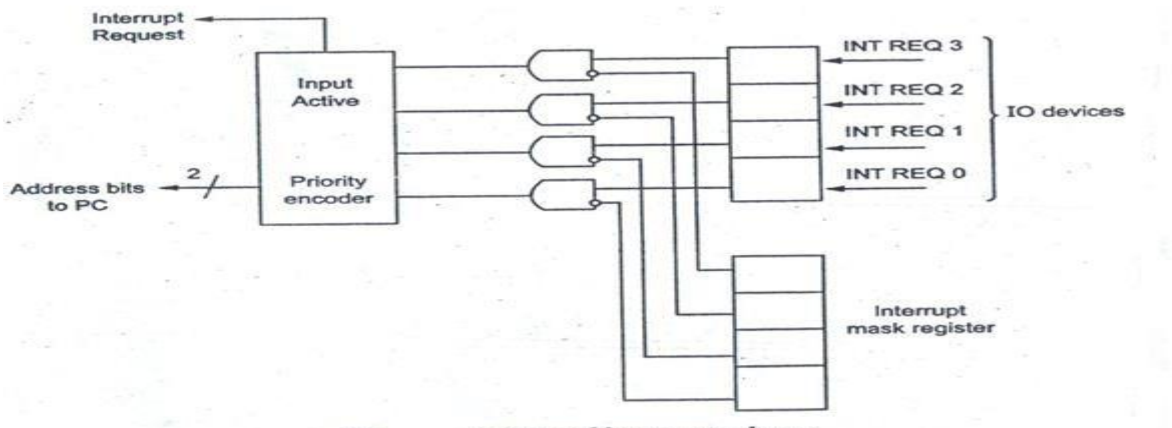
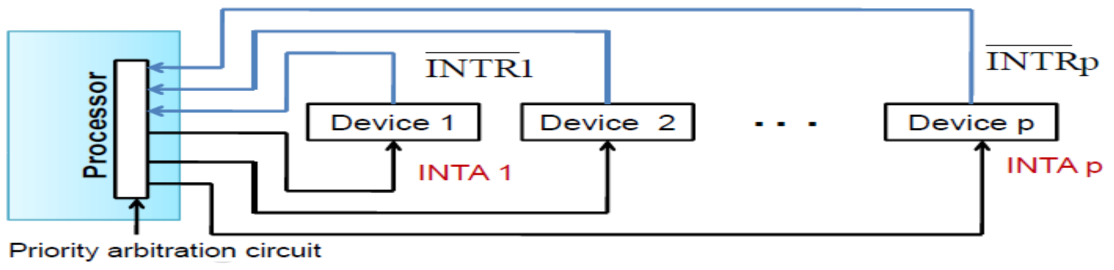


Fig. A vectored interrupt scheme

5. **Interrupt Nesting**

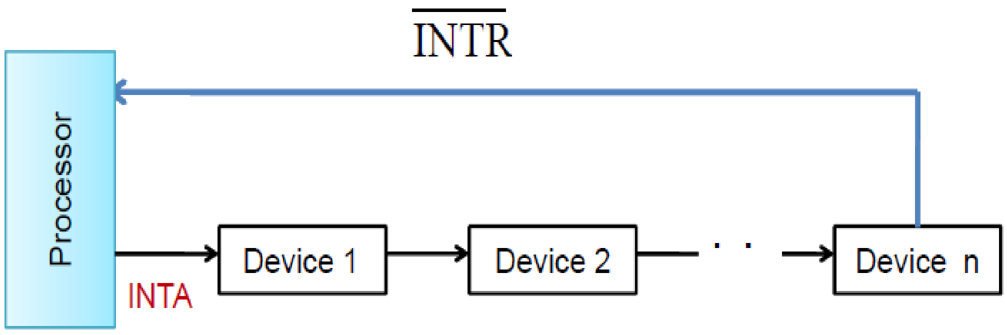
- Pre-Emption of low priority Interrupt by another high priority interrupt is known as Interrupt nesting.
- Need a priority of IRQ devices and accepting IRQ from a high priority device.
- The priority level of the processor can be changed dynamically.
- The privileged instruction write in the PS (processor status word), that encodes the processors priority. Organizing I/O devices in a prioritized structure.
- Each of the interrupt-request lines is assigned a different priority level. The processor is interrupted only by a high priority device.

Interrupt Nesting (contd.)


Processor
$\overline{\text{INTR}}1$
$\overline{\text{INTR}}$p
Device 1
Device 2
. . .
Device p
INTA 1
INTA p
Priority arbitration circuit

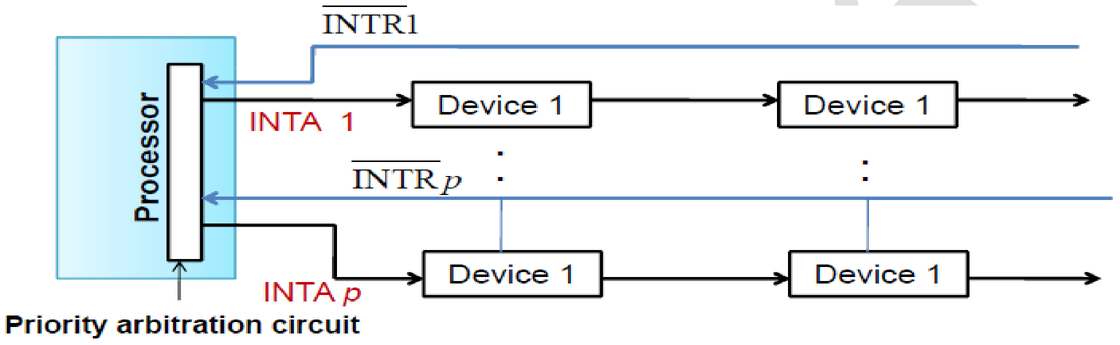
6. **Daisy Chaining**

- The interrupt request line INTR is common to all the devices.
- The interrupt acknowledgement line INTA is connected to devices in a DAISY CHAIN way.
- INTA propagates serially through the devices. Device that is electrically closest to the processor gets high priority.
- Low priority device may have a danger of STARVATION.



7. **Daisy Chaining with Priority Group**

- Combining Daisy chaining and Interrupt nesting to form priority group.
- Each group has different priority levels and within each group devices are connected in daisy chain way.



Arrangement of priority groups

8. **Pipeline Interrupts**

- In a pipelined processor, many instructions are executed at same time so it is difficult to find the interrupting instruction.
- In a pipelined process one instruction can finish sooner than another instruction that was issued earlier. Let's consider three different kinds of instructions floating point representation such as
    1. Multiply
    2. Add1
    3. Add2
- Multiply instruction has 7 clock cycles and Add 1 and 2 have four clock cycles.
- In Fig.(a) Add 1 instruction completes their execution before multiply instruction, even though multiply instruction starts their execution 1 cycle earlier than Add 1.
- Here no hazard occurs due to the data dependencies. Suppose add1 generates an interrupt due to a result overflows in its execution stage EX corresponding to cycle4.

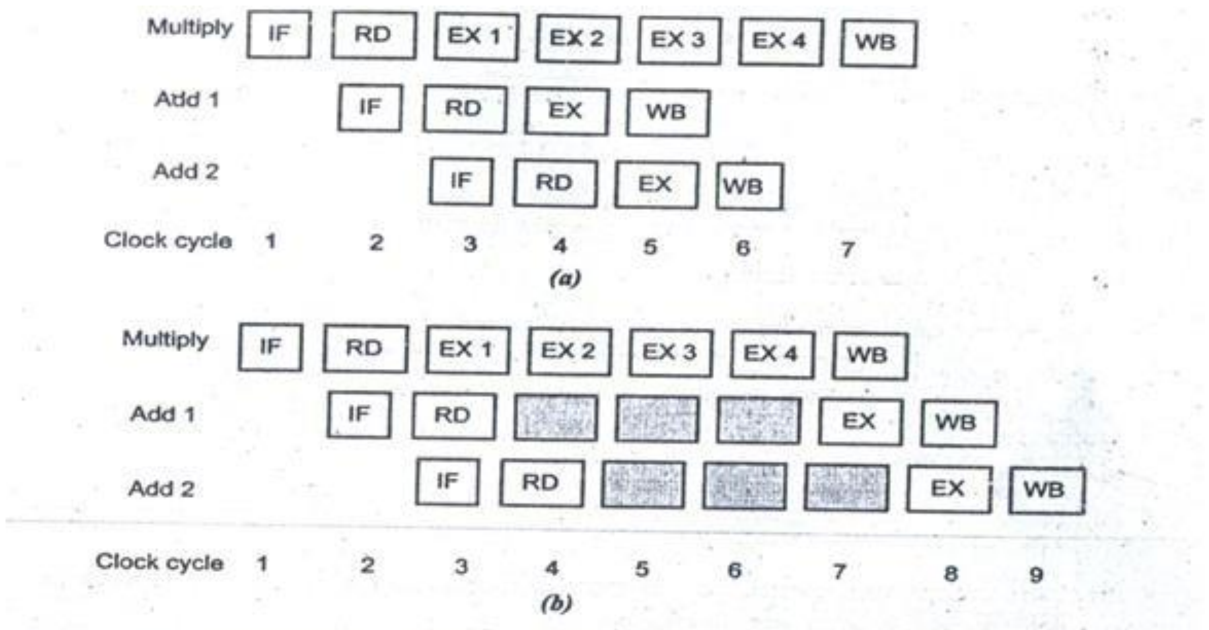
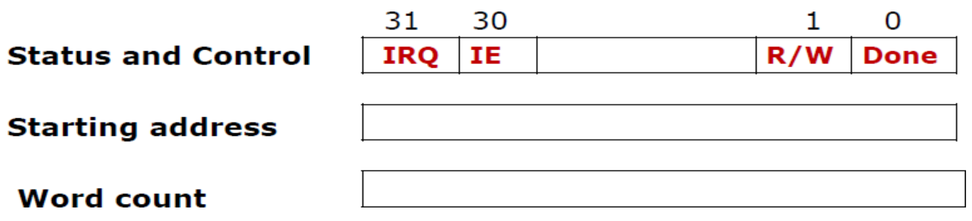


Fig. ***Instruction processing in a pipeline with (a) out-of-order completion (b) in-order completion***

- Now we need transfer the control to an interrupt handler designed to service adder overflow. It is also possible that the ongoing multiply instruction will generate another interrupt in cycle6.
- This second interrupt can change the CPU's state to prevent proper processing of the first interrupt.
- Registers affected by add1 instruction can be further modified by the multiply interrupt so proper recovery from the add1 instruction may not be possible.
- In such kinds of situation the CPU state is said to have become **imprecise.**
- **Precise interrupt** is one where the system state information or correct transfer of control to the interrupt handler and correct return to the interrupting program is always preserved.

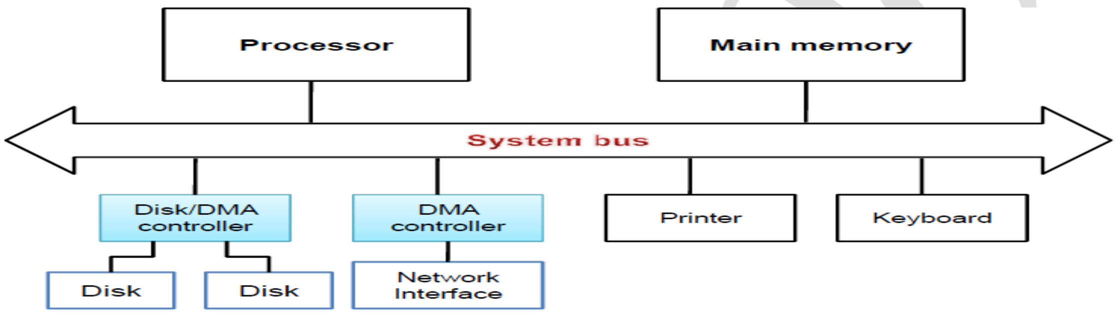
DIRECT MEMORY ACCESS (DMA)

- To transfer large blocks of data at high Speed, between EXTERNAL devices & Main Memory, DMA approach is often used.
- DMA controller allows data transfer directly between I/O device and Memory, with minimal intervention of processor.
- DMA controller acts as a Processor, but it is controlled by CPU.
- To initiate transfer of a block of words, the processor sends the following data to controller
    - The starting address of the memory block
    - The word count
    - Control to specify the mode of transfer such as read or write
    - A control to start the DMA transfer
- DMA controller performs the requested I/O operation and sends a interrupt to the processor upon completion



- In DMA interface First register stores the starting address, Second register stores Word count and Third register contains status and control flags.

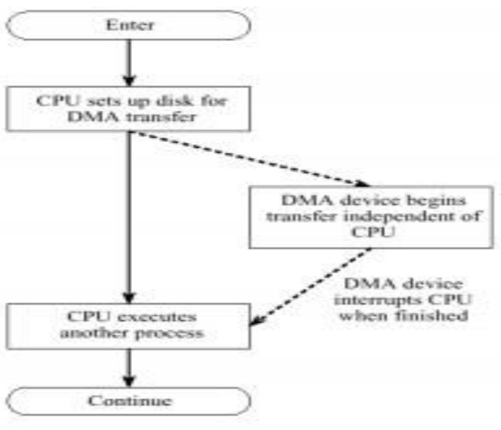
Bits and Flags	1	0
R/W	READ	WRITE
Done	Data transfer finishes	
IRQ	Interrupt request	
IE	Raise interrupt (enable) after Data Transfer	



Use of DMA Controller in a computer system

- Memory accesses by the processor and DMA Controller are interleaved.
- DMA devices have higher priority than processor over BUS control.
- **Cycle Stealing:** DMA Controller "steals" memory cycles from processor, though processor originates most memory access.
- **Block or Burst mode:** The DMA controller may give exclusive access to the main memory to transfer a block of data without interruption.

DMA Flowchart for a Disk Transfer


Enter
CPU sets up disk for DMA transfer
DMA device begins transfer independent of CPU
DMA device interrupts CPU when finished
CPU executes another process
Continue

- **Conflicts in DMA:**

1. Processor and DMA.
2. Two DMA controllers, try to use the Bus at the same time to access the main memory.

The hardware needed to implement DMA is shown in the following Figure.

- The circuit assumes that all access to main memory is via a shared system bus.
- The IO device is connected to the system bus via a special interface circuit called DMA controller.
- It contains a data buffer register IODR, it also controls an address register IOAR and a data count register DC.
- These registers enable the DMA controller to transfer data to or from a contiguous region of memory.
- IOAR stores the address of the next word to be transferred. It is automatically incremented or decremented after each word transfer.
- The data counter DC stores the number of words that remain to be transferred.
- Data counter is automatically incremented or decremented after each transfer and tested for zero, when DC reaches zero the DMA transfer halls.

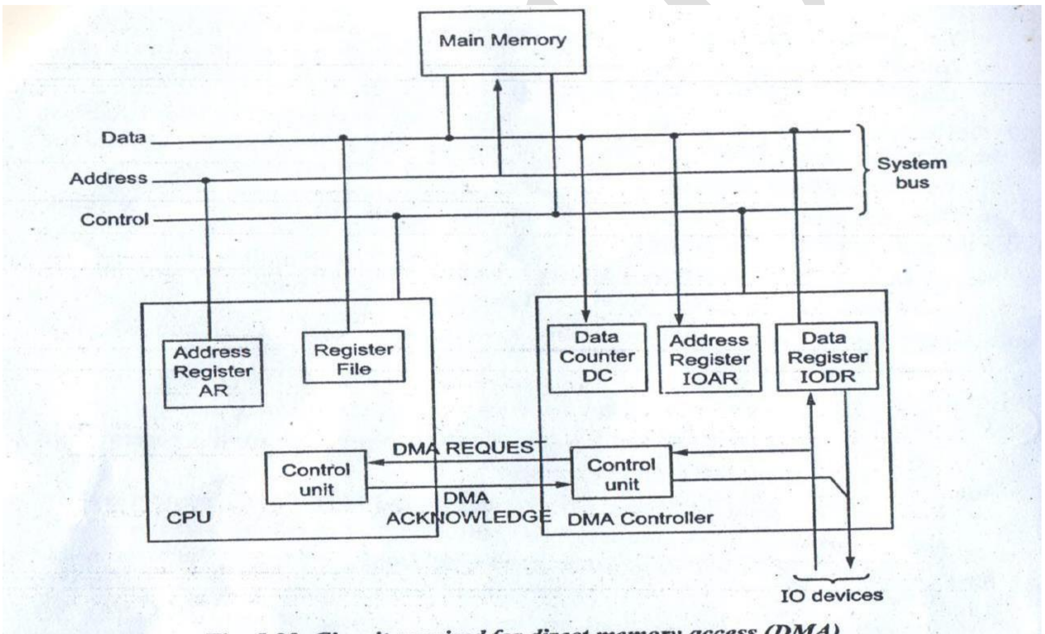


Fig. 5.33. Circuit required for direct memory access (DMA)

- The DMA controller is normally provided with interrupt capabilities to send an interrupt to the CPU to signal the end of the IO data transfer.
- A DMA controller can be designed to supervise DMA transfers involving several IO devices, each device has a different priority of access to the system bus. Under DMA control data can be transferred in several different ways.
- In a DMA block transfer a data word is a sequence of arbitrary length that is transferred in a single burst while the DMA controller is master of the memory bus.
- This DMA mode needs the secondary memories like disk drives, where data transmission cannot be stopped or slowed without loss of data.

- **Block DMA transfer** supports the fastest IO data transfer rates but it can make the CPU inactive for relatively long periods by tying up the system bus.

- **Cycle stealing:** An alternative technique called cycle stealing; it allows the DMA controller to use the system bus to transfer one word data after it must 'return control of the bus to the CPU.
- Cycle stealing reduces, the maximum IO transfer rate and also reduces the interference by the DMA controller in the CPU's memory access.
- **Transparent DMA:** It is possible to eliminate this interference completely by designing the DMA interfaces. CPU is not actually using the system bus. This is called transparent DMA.
- The above shown DMA circuit has the following DMA transfer process:

Step 1:

- The CPU executes two IO instructions, which load the DMA' registers IOAR and DC with their initial values.
- IOAR register should contain the base address of the memory region to be used in the data transfer.
- DC register should contain the number of words to be transferred to' or from that region.

Step 2:

- When the DMA controller is ready to transmit or receive data, it activates the DMA REQUEST line to the CPU.
- The CPU waits for the next DMA breakpoint.
- DMA controller relinquishes control of the data and address lines and activates DMA ACKNOWLEDGE.
- DMA REQUEST and DMA ACKNOWLEDGE are essentially used in BUS REQUEST and BUS GRANT lines for control of the system bus.
- DMA requests from several DMA controllers are resolved by bus priority control techniques.

Step 3:

- Now DMA controller transfers data directly to or from main memory. After transferring a word, IOAR and DC registers are updated.

Step 4:

- If DC has not yet reached zero but, the 10 device is not ready to send' or receive the next batch of data then the DMA-controller will release the system bus to the CPU by deactivating the DMA REQUEST line.
- The CPU responds by deactivating DMA ACKNOWLEDGE and resuming control of the system bus.

Step 5:

- If DC register is decremented to zero then the DMA controller again relinquished control of the system bus.
- It may also send an interrupt request signal to the CPU. The CPU responds by halting the IO device or by initiating a new DMA transfer.

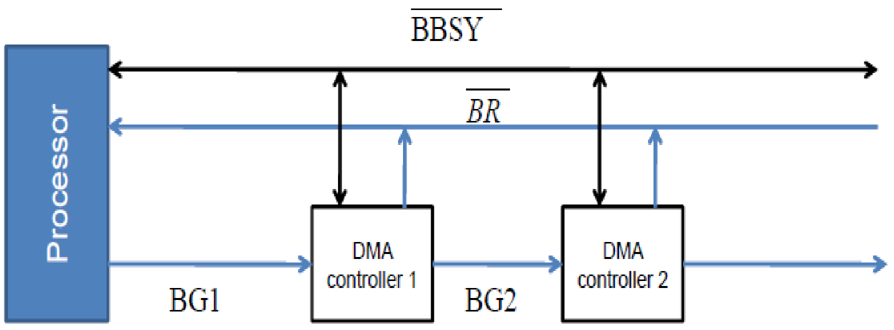
BUS ARBITRATION

- **Bus master:** Device that initiates data transfers on the bus.
- The next device can take control of the bus after the current master surrenders control.
- **Bus Arbitration:** Process by which the next device to become master is selected.

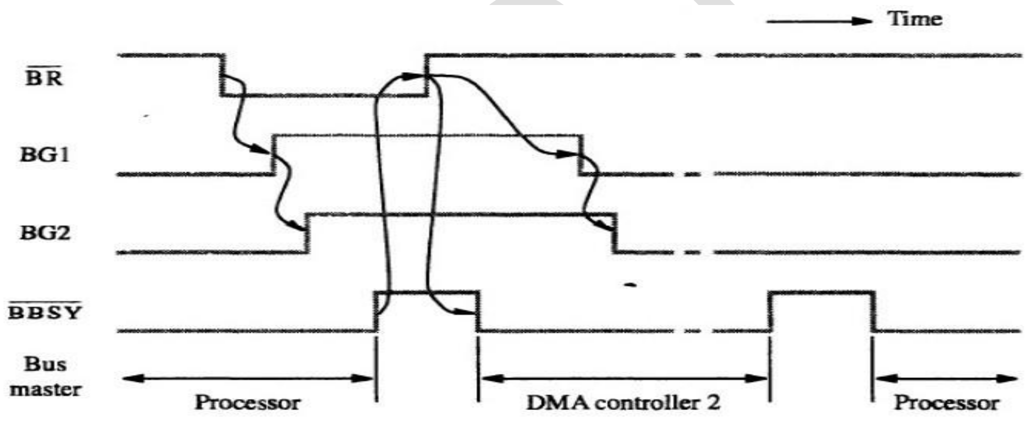
Types of bus arbitration:

1. Centralized and Distributed Arbitration
2. Distributed Arbitration

Centralized and Distributed Arbitration



A simple arrangement for bus arbitration using a daisy chain



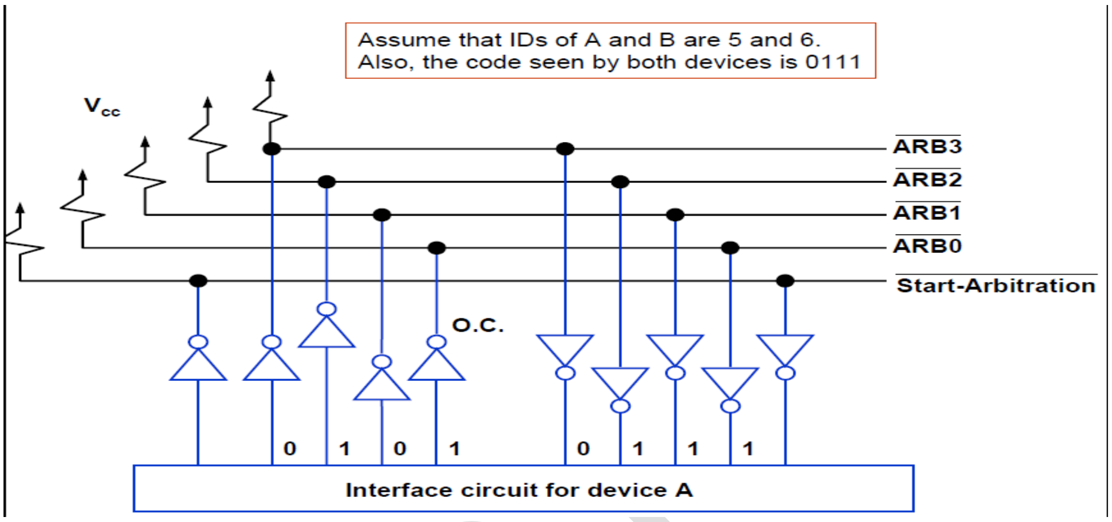
Sequence of signals during data transfer of bus mastership

- BR (Bus Request) line is the open drain line and the signal on this line is a logical OR of the bus request from all the DMA devices.
- BG (Bus Grant) line is processor activates this line indicating (acknowledging) to all the DMA devices (connected in daisy chain fashion) that the BUS may be used when it's free.
- BBSY (Bus Busy) line is an open collector line. The current bus master indicates devices that it is currently using the bus by signaling this line.
- Processor is normally the bus master, unless it grants bus mastership to DMA.
- For the timing/control, in the above diagram DMA controller 2 requests and acquires bus mastership and later releases the bus.
- During its tenure as the bus master, it may perform one or more data transfer operations, depending on whether it is operating in the cycle stealing or block mode.

- After it releases the bus, the processor resumes bus mastership.

Distributed Arbitration

- All devices waiting to use the bus have to carry out the arbitration process and it has no central arbiter.
- Each device on the bus is assigned with a 4-bit identification number.
- One or more devices request the bus by asserting the start-arbitration signal and place their identification number on the four open collector lines.


Assume that IDs of A and B are 5 and 6.
Also, the code seen by both devices is 0111
V_{cc}
$\overline{ARB3}$
$\overline{ARB2}$
$\overline{ARB1}$
$\overline{ARB0}$
$\overline{Start\text{-}Arbitration}$
O.C.
0
1
0
1
0
1
1
1
Interface circuit for device A

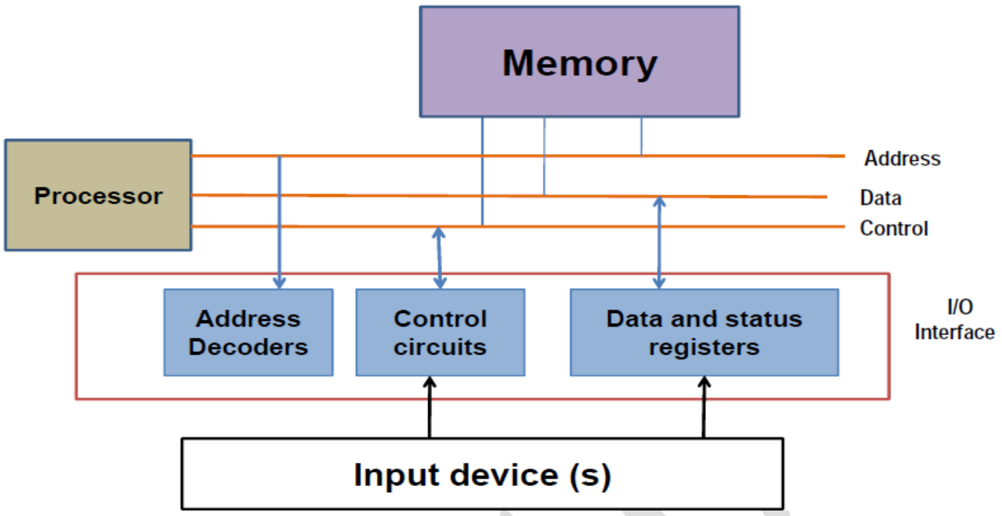
- ARB0 through ARB3 are the four open collector lines.
- One among the four is selected using the code on the lines and one with the highest ID number.
- Assume that two devices, A and B, having ID numbers 5 and 6, respectively, are requesting the use of the bus.
- Device A transmits the pattern 0101, and device B transmits the pattern 0110. The code seen by both devices is 0111.
- Each device compares the pattern on the arbitration lines to its own ID, starting from the most significant bit.
- If it detects a difference at any bit position, it disables its drivers at that bit position and for all lower-order bits. It does so by placing a 0 at the input of these drivers.
- In the case of our example, device A detects a difference on line ARB I. Hence, it disables its drivers on lines ARB 1 and ARBO.
- This causes the pattern on the arbitration lines to change to 0110, which means that B has won the contention.

Interface Circuit

- The interface circuit contains the following units. Namely,
  1. Address Decoder
  2. Control Circuits
  3. Data registers and Status registers
- The register in I/O Interface is buffer and control. There are two flags in Status

Registers, like SIN, SOUT and two Data Registers, like Data-IN, Data-OUT.

- Status register SIN, SOUT provides status information for keyboard and display unit.
- When the processor places the address and data on the memory bus, the memory system ignores the operation because the address indicates a portion of the memory space used for I/O.



- DATAIN is the address of the input buffer associated with the keyboard.

  Move DATAIN, R0; reads the data from DATAIN and stores them into processor registerR0

  Move R0, DATAOUT; sends the contents of register R0 to location DATAOUT
- The device controller, sees the operation, records the data, and transmits it to the device as a command.
- User programs are prevented from issuing I/O operations directly because the OS does not provide access to the address space assigned to the I/O devices and thus the addresses are protected by the address translation.