UNIT V

MEMORY AND I/O SYSTEMS

Memory hierarchy - Memory technologies – Cache basics – Measuring and improving cache performance - Virtual memory, TLBs - Input/output system, programmed I/O, DMA and interrupts, I/O processors.

5.1 Memory Hierarchy

- Memory hierarchy is the structure that uses multiple levels of memories; as the distance from the CPU increases, the size of the memories and the access time both increase.
- By implementing the memory of a computer as a memory hierarchy the advantage of principal of locality can be obtained.
- The principle of locality states that programs access a relatively small portion of their address space at any instant of time. There are two different types of locality:
    1. Temporal locality
    2. Spatial locality

Temporal locality (locality in time):

- The principle states that if a data location is referenced then it will tend to be referenced again soon. If you recently brought a book to your desk to look at, you will probably need to look at it again soon.

Spatial locality (locality in space):

- The locality principle states that if a data location is referenced, data locations with nearby addresses will tend to be referenced soon. For example, Libraries put books on the same topic together on the same shelves to increase spatial locality.
- There are three primary technologies used in building memory hierarchies.

- Main memory is implemented from DRAM (dynamic random access memory), while levels closer to the processor (caches) use SRAM (static random access memory).
- DRAM is less costly per bit than SRAM, although it is substantially slower.
- The price difference arises because DRAM uses significantly less area per bit of memory.
- The third technology, used to implement the largest and slowest level in the hierarchy, is magnetic disk.
- The access time and price per bit vary widely among these technologies, as the table below shows, using typical values for 2004:

Memory technology	Typical access time	$ per GB in 2008
SRAM	0.5–2.5 ns	$2000–$5000
DRAM	50–70 ns	$20–$75
Magnetic disk	5,000,000–20,000,000 ns	$0.20–$2

- Because of these differences in cost and access time, it is advantageous to build memory as a hierarchy of levels.
- The below figure shows the faster memory is close to the processor and the slower, less expensive memory is below it. The goal is to present the user with as much memory as is available in the cheapest technology, while providing access at the speed offered by the fastest memory.

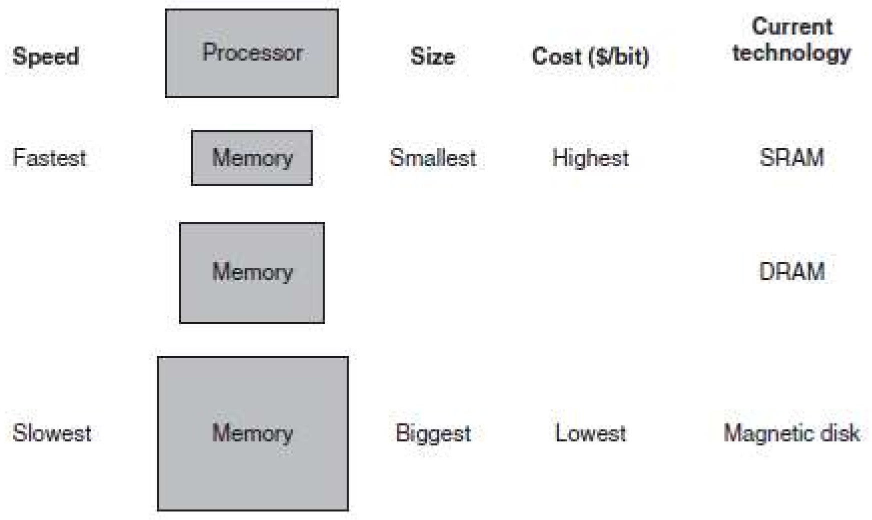


Figure: The basic structure of a memory hierarchy.

- By implementing the memory system as a hierarchy, the user has the illusion of a memory that is as large as the largest level of the hierarchy, but can be accessed as if it were all built from the fastest memory.
- Flash memory has replaced disks in many embedded devices, and may lead to a new level in the storage hierarchy for desktop and server computers.
- The memory system is organized as a hierarchy: a level closer to the processor is generally a subset of any level further away, and all the data is stored at the lowest level.
- A memory hierarchy can consist of multiple levels with different speed and size, but data is copied between only two adjacent levels at a time, they are the upper level and the lower layer.
- The upper level is one which is closer to the processor. It is smaller and faster than the lower level.

5.1.1 Two-level Hierarchy:

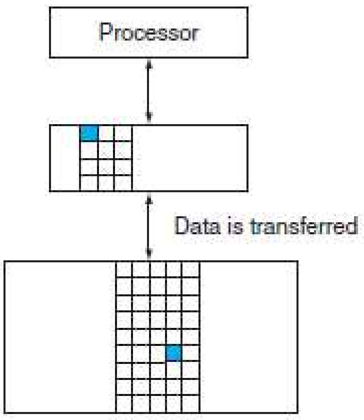


Figure: Pair of levels in the memory hierarchy.

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:

- The **hit rate, or hit ratio**, 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:

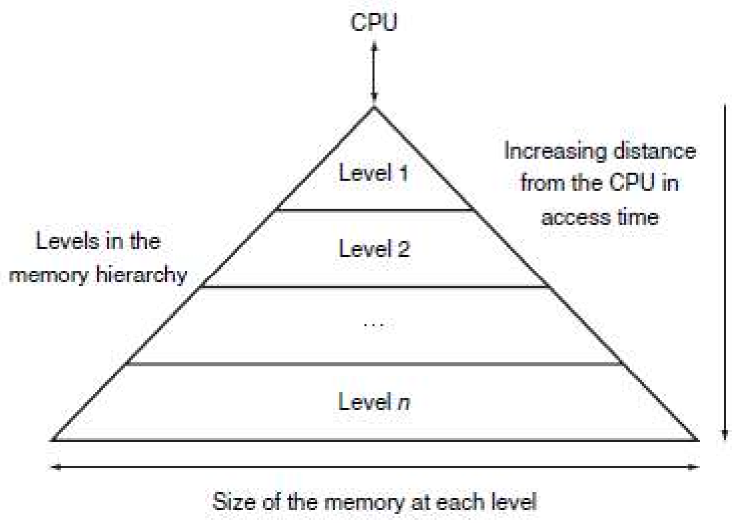
- The **miss rate** (1 – hit rate) is the fraction of memory accesses not found in the upper level. Since performance is the major reason for having a memory hierarchy, the time to service hits and misses is important.

Hit time:

- **Hit time** 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:

- The **miss penalty** 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.


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

-

- Because the upper level is smaller and built using faster memory parts, the hit time will be much smaller than the time to access the next level in the hierarchy, which is the major component of the miss penalty.
- The above figure shows that a memory hierarchy uses smaller and faster memory technologies close to the processor.
- Thus, accesses that hit in the highest level of the hierarchy can be processed quickly. Accesses that miss go to lower levels of the hierarchy, which are larger but slower.
- If the hit rate is high enough, the memory hierarchy has an effective access time close to that of the highest (and fastest) level and a size equal to that of the lowest (and largest) level.

5.2 Memory Technologies

- There are four primary technologies used in memory hierarchies such as
  1. SRAM technologies
  2. DRAM technologies
  3. FLASH Memory technologies
  4. Disk Memory technologies

5.2.1 SRAM Technology

- SRAM –Static Random Access is a part of the Random Access Memory (RAM). It is maim memory and located very close to the processor.
- SRAM's are simply integrated circuits that are memory array with a single access port and it can provide either a read or a write. It has a fixed access time to any through the read and write access times may differ.
- SRAM's no need to refresh so the access time is very close to the cycle time. It typically use six to eight transistors per bit to prevent the information from being disturbed when read. It needs only minimal power to retain the charge in standby mode.
- Most PCs and server systems used separate SRAM chips for either their primary, secondary or even caches.

5.2.2 DRAM Technology

- In a SRAM, as long as power is applied the value can be kept indefinitely. In a Dynamic RAM (DRAM) the values kept in a cell is stored as a charge in a capacitor.
- DRAM has single transistor used to access this stored charge either to read the value or to overwrite the charge stored there. Because it uses only a single transistor per bit of storage they are much denser and cheaper than SRAM.
- DRAM's stores all the charge on a capacitor so it cannot be kept indefinitely and must be periodically refreshed. That is why this memory structure is called dynamic.
- In DRAM to refresh the cell we read its contents and write it back . The charge can be kept for several milliseconds.
- If every bit had to be read out of the DRAM and then written back individually, so we constantly be refreshing the DRAM by leaving no time for accessing it.
- DRAM uses a two level decoding structure and it will allow us to refresh the entire row with a read followed immediately by a write cycle.

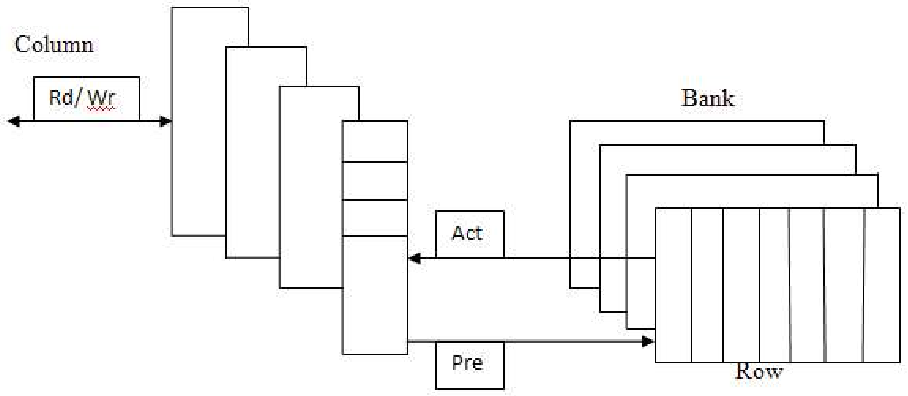


Figure :Internal organization of a DRAM

- DRAM's are organized in banks, typically four for DDR3.Each bank consists of a series of rows. Sending a PRE (Prechange) command opens or close a bank.
- A row address is sent with an Act (active) which causes the row to transfer to a buffer, when the row is in the buffer it can be transferred by successive column addresses as whatever the width of the DRAM is or by specifying a block

transfer and the starting address. Each command as well as block transfers are synchronized with a clock.

- The row organization helps with refresh as well as with performance. To improve performance DRAM's buffer rows for repeated access. The buffer acts like as SRAM by changing the address, random bits can be accessed in the buffer until the next row access.
- This capability improves the accesss time significantly because the access time to bits in the row is much lower.Making the chip wider also we can improves the memory bandwidth of the chip.
- To improve the interface to processor's DRAM's added clocks with its and are called synchronous DRAM's or SDRAM's

Advantages of SDRAM

- The use of a clock eliminates the time for the memory and processor to synchronize.
- The ability to transfer the bits in the burst without having to specify additional address bits.
- The clock transfers the successive bits in a burst

DDR- Double Data Rate

- The fastest version of SDRAM is called Double Data Rate. It means data transfers on both the rising and falling edge of the clock, thereby getting twice as much bandwidth as we expect based on the clock rate and the data width.
- The latest version of this technology is called DDR4. A DDR4-3200. DRAM can do 3200 million transfers per second which means it has a 1600 MHz clock.
- To obtain much bandwidth requires clever organization inside the DRAM. Instead of a faster row buffer the DRAM can be internally organized to read or write from multiple banks and each bank must have its own row buffer.
- Sending and address to several banks permits them all to read or write simultaneously. For example, with four banks there is one access time and then accesses rotate between the four banks to supply four times the bandwidth. This rotating access scheme is called address inter leaving

- Personal mobile devices like the ipad use individual DRAM's and memory for servers are commonly sold on small boards called dual inline memory modules (DIMM'S).
- Dual inline memory modules contain 4-16 DRAM and they are normally organized to be 8 bytes wide for server systems. A DIMM using DDR4-3200 SDRAM cloud transfer at 8 3200 25,600 megabytes per second. Such DIMM'S are named after their bandwidth PC25600.
- DIMM can have so many chips and only a portion of them are used for a particular transfer. To avoid confusion with the internal DRAM names of row and banks they used memory rank for subset of chips in a DIMM.

Year introduced	Chip size	$ per GIB	'otal access time to a new row/ column	/erage column access time to existing row
1992	l6 Mebibit	$15,000	90 ns	30 ns
1996	54 Mebibit	$10,000	60 ns	12 ns
1998	28 Mebibit	$4,000	60 ns	10 ns
2000	56 Mebibit	$1,000	55 ns	7 ns
2004	12 Mebibit	$ 250	50 ns	5 ns
2007	1 Gibibit	$ 50	45 ns	1.25ns
2010	2Gibibit	$ 30	40 ns	1 ns
2012	4 Gibibit	$ 1	35 ns	0.8 ns

5.2.3 Flash Memory Technology

- Flash memory is a type of Electrically Erasable Programmable Read Only Memory (EEPROM)
- Unlike disks and DRAM, the EEPROM technologies the writes can wear out flash memory bits most flash products include a controller to spread the writes

by remapping blocks that have been written many times to less trodelen blocks. This technique is called wear leveling

- With wear leveling, personal mobile devices are very unlikely to exceed the write limits in the flash. Flash controllers perform wear leveling and it also improves the mapping out memory cells that were manufactured incorrectly.

5.3.4 Disk Memory Technology

- 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 entire drive permanently sealed to control the environment inside the drive,which in turn allows the disk head to be much closer to the drive surface.
- Each disk surface is divided into concentric circles called tracks. There are typically tens of thousands of tracks per surface . Track is one of thousands of concentric circles that make up the surface of a magnetic disk.
- Each track is in divided into sector that contain the information. Each track may have thousands of sectors and it has 512 to 4096 bytes in size.
- Sector is one of the segments that make up a track on a magnetic disk and it is the smallest amount of information that is read or written on a disk.
- 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.
- To access data, the operating system must direct the disc through a three stage process.
- The first step is to position the head over the proper track. This operation is called a seek and time to move the head to desired track is called the seek time.
- Disk manufacturer report minimum seek time, maximum seek time and average seek time in their manuals.
- Minimum and maximum seek time are easy to measure but the average seek time is open to wide interpretation because it depends on the seek distance.

- The industry calculates average seek time as

$$\text{Average seek time} = \frac{\text{sum of the time for all possible seeks}}{\text{Number of possible seeks}}$$

- Average seek times are usually advertised as 3 ms but depending on the application and scheduling of disk requests, the actual average seek time may be only 25% to 33% of the advertised number because of locality of disk references.

- This locality arises because of following two reasons:
  1. Successive accesses to the same file.
  2. Operating system tries to schedule such accesses together.
- Second step is 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.
- Rotational latency also called rotational delay. The time required for the desired sector of a disk to rotate under the read /write head usually assumed to be half the rotation time.
- Disks rotate at 5400 rpm to 15,000 rpm. The average rotational latency at 5400 rpm is

$$\text{Average rotational latency} = \frac{0.5 \text{ rotation}}{5400 \text{ rpm}}$$

$$= \frac{0.5 \text{ rotation}}{5400 \text{ rpm}/\left(60\frac{\text{seconds}}{\text{minute}}\right)}$$

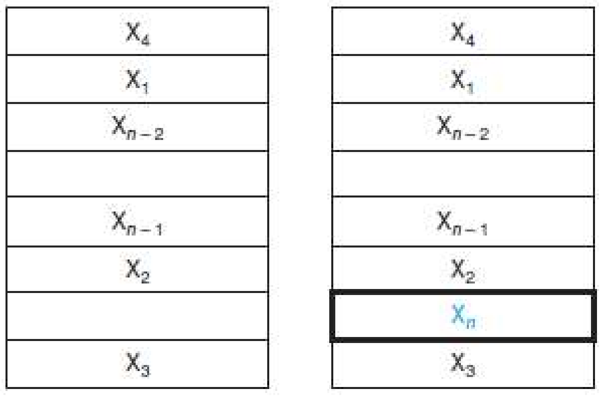
= 0.0056 seconds

= 5.6 ms

- Third step of a disk access is transfer time. it 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.

5.4 Cache Basics

- Cache is a safe place for hiding or storing things.
- Cache was the name chosen to represent the level of the memory hierarchy between the processor and main memory in the first commercial computer to have this extra level.
- The word cache is also used to refer any storage managed to take advantage of locality of access.
- Caches first appeared in research computers in the early 1960s and in production computers later in that same decade; every general-purpose computer built today, from servers to low-power embedded processors, includes caches.
- In this section,very simple cache in which the processor requests are each one word and the blocks also consist of a single word.



a. Before the reference to X_n b. After the reference to X_n

- The above figure shows such a simple cache, before and after requesting a data item that is not initially in the cache.
- Before the request, the cache contains a collection of recent references X_1, X_2, . . . ,X_{n-1}, and the processor requests a word X*n*that is not in the cache.
- This request results in a miss, and the word X_n is brought from memory into cache.

- If each word can go in exactly one place in the cache, then it is straightforward to find the word if it is in the cache.
- The simplest way to assign a location in the cache for each word in memory is to assign the cache location based on the addressof the word in memory. This cache structure is called **direct mapped**, since each memory location ismapped directly to exactly one location inthe cache.
- The typical mapping between addresses and cache locations for a direct-mapped cache is usually simple. For example, almost all direct-mapped caches use the mapping

  (Block address) modulo (Number of cache blocks in the cache)
- This mapping is attractive because if the number of entries in the cache is a powerof two, then modulo can be computed simply by using the low-order $\log_2$ (cache size in blocks) bits of the address; hence the cache may be accessed directly with the low-order bits.

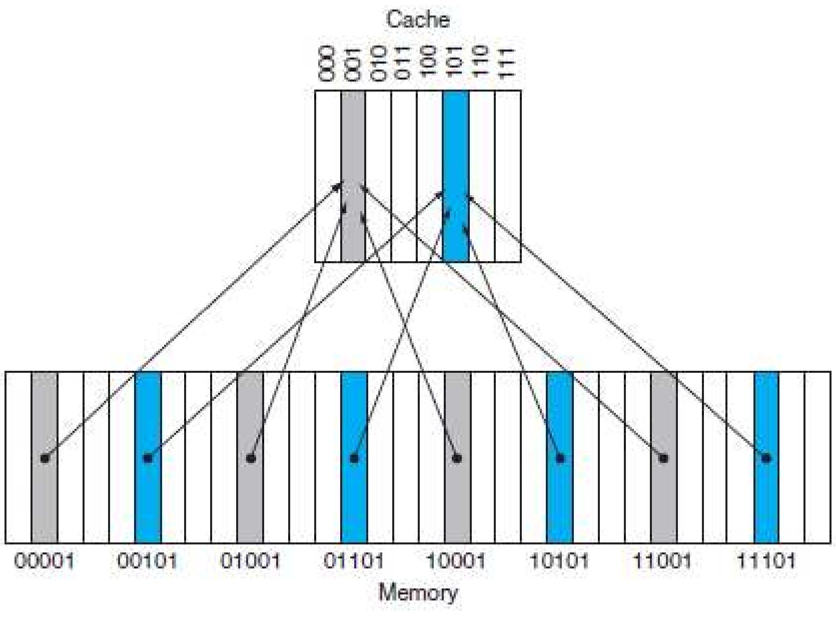


Figure: A direct-mapped cache with eight entries

- The above figure shows the addresses of memory words between 0 and 31 that map to the same cache locations.

- ecause there are eight words in the cache, an address X maps to the direct-mapped cache word X modulo 8. That is, the low-order $\log_2(8) = 3$ bits are used as the cache index.
- Thus, addresses 00001_{two}, 01001_{two}, $10001t_{wo}$, and 11001_{two} all map to entry 001_{two} of the cache, while addresses 00101_{two}, 01101two, 10101_{two}, and 11101_{two} all map to entry 101_{two} of the cache.
- It shows how the memory addresses between 1_{ten} (00001_{two}) and 29_{ten} (11101_{two}) map to locations 1_{ten} (001_{two}) and 5ten (101_{two}) in a direct-mapped cache of eight words because each cache location can contain the contents of a number of different memory locations.
- **Tag** is a field in a table used for a memory hierarchy that contains the address information required to identify whether the associated block in the hierarchy corresponds to a 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.
- Even after executing many instructions, some of the cache entries may still be empty as shown above. Thus, we needto know that the tag should be ignored for such entries.
- The most common method is to add a **valid bit** to indicate whether an entry contains a valid address.
- If the bit is not set, there cannot be a match for this block.

5.4.1 Accessing a Cache

- Below table shows the contents of an eight-word direct-mapped cache as it responds to a series of requests from the processor.
- Since there are eight blocks in the cache, the low-order 3 bits of an address give the block number. Here is the action for each reference

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (7.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (7.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (7.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (7.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (7.6f)	$(10010_{two} \bmod 8) = 010_{two}$

- When the word at address 18 (10010_{two}) is brought into cache block 2 (010_{two}), the word at address 26 (11010_{two}), which was in cache block 2 (010_{two}), must be replaced by the newly requested data.
- This behavior allows a cache to take advantage of temporal locality.

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}	Memorily (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})

Figure: The cache contents are shown after each reference request.

- The cache is initially empty, with all valid bits (V entry in cache) turned off (N).
- The processor requests the following addresses: 10110_{two} (miss), 11010_{two} (miss), 10110_{two} (hit), 11010_{two} (hit), 10000_{two} (miss), 00011_{two} (miss), 10000_{two} (hit), and 10010_{two} (miss).
- The figures show the cache contents after each miss in the sequence has been handled. When address 10010_{two} (18) is referenced, the entry for address 11010_{two} (26) must be replaced, and a reference to 11010_{two} will cause a subsequent miss.
- The tag field will contain only the upper portion of the address. The full address of a word contained in cache block *i* with tag field *j* for this cache is *j* ¥ 8 + *i,* or equivalently the concatenation of the tag field *j* and the index *i*.

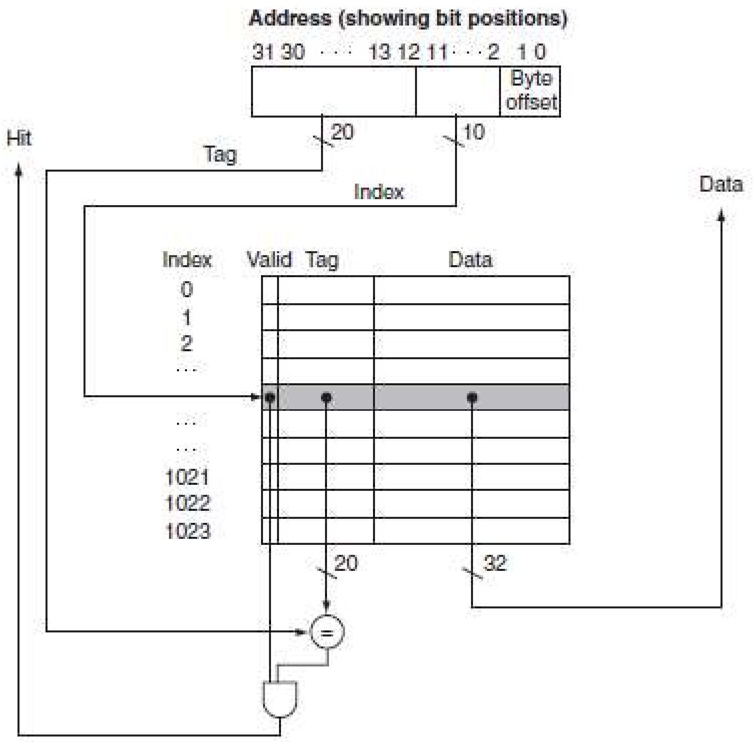


Figure: For this cache, the lower portion of the address is used to select a cache entry consisting of a data word and a tag.

- For example, in cache f above, index 010 has tag 10 and corresponds to address 10010.
- In the cache for each possible address: the low-order bits of an address can be used to find the unique cache entry to which the address could map.
- Theabove figure shows how a referenced address is divided into
    1. Cache index, which is used to select the block
    2. Tag field, which is used to compare with the value of the tag field of the cache
- 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 1 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.
- The index of a cache block, together with the tag contents of that block, uniquelyspecifies the memory address of the word contained in the cache block.
- Because the index field is used as an address to access the cache and because an *n*-bit field has 2*n* values, the total number of entries in a direct-mapped cache must be a power of 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.

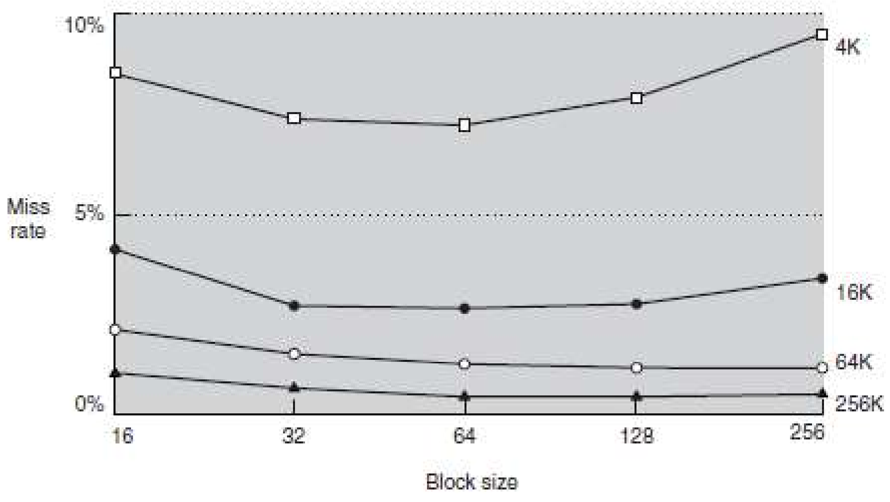


Figure: Miss rate versus block size.

- Note that miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size.
- A more serious problem associated with just increasing the block size is that the cost of a miss increases.
- The miss penalty is determined by the time required to fetch the block from the next lower level of the hierarchy and load it into the cache.
- The time to fetch the block has two parts: the latency to the first word and the transfer time for the rest of the block.hence the miss penalty—will increase as the block size grows.Furthermore, the improvement in the miss rate starts to decrease asthe blocks become larger.
- The result is that the increase in the miss penalty overwhelms the decrease in the miss rate for large blocks, and cache performance thus decreases.if we design the memory to transfer larger blocks more efficiently, we can increase the block size and obtain further improvements in cache performance.
- The major disadvantage of increasing the block size is that the cache miss penalty increases.
- For making the miss penalty effectively smaller the simplest method called early restart can be used it is simply to resume execution as soon as the requested word of the block is returned, rather than wait for the entire block.

- This technique is usually less effective for data caches because it is likely that the words will be requested from the block in a less predictable way, and the probability that the processor will need another word from a different cache block before the transfer completes is high.
- An even more sophisticated scheme is to organize the memory so that the requested word is transferred from the memory to the cache first.
- The remainder of the block is then transferred, starting with the address after the requested word and wrapping around to the beginning of the block. This technique, called requested word first, or critical word first, can be slightly faster than early restart, but it is limited by the same properties that limit early restart.

5.4.2 Handling Cache Misses

- **Cache miss** is the request for data from the cache that cannot be filled because the data is not present in the cache.
- The control unit must detect a miss and process the miss by fetching the requested data from memory. If the cache reports a hit, the computer continues using the data as if nothing had happened.
- The cache miss handling is done with the processor control unit and with a separate controller that initiates the memory access and refills the cache.
- The processing of a cache miss creates a stall, similar to the pipeline stalls.
- For a cache miss, we can stall the entire processor, essentially freezing the contents of the temporary and programmer-visible registers, while we wait for memory instruction misses are handled for either the multi-cycle or pipelined data path; the same approach can be easily extended to handle data misses.
- If an instruction access results in a miss, then the content ofthe Instruction register is invalid.
- To get the proper instruction into the cache, we must be able to instruct the lower level in the memory hierarchy to perform a read. Since the program counter is incremented in the first clock cycle of execution in both the pipelined and multi-cycle processors, the address of the instruction that

generates an instruction cache miss is equal to the value of the program counter minus 4.

- Once we have the address, we need to instruct the main memory to perform a read. We wait for the memory to respond (since the access will take multiple cycles), and then write the words into the cache.

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.

- The control of the cache on a data access is essentially identical: on a miss, we Simply stall the processor until the memory responds with the data.

5.4.3 Handling Writes

- The simplest way to keep the main memory and the cache consistent is to always write the data into both the memory and the cache. This scheme is called **write-through** write miss. We first fetch the words of the block from memory.
- After the block is fetched and placed into the cache, we can overwrite the word that caused the miss into the cache block. We also write the word to main memory using the full address. Although this design handles writes very simply, it would not provide very good performance.
- With a write-through scheme, every write causes the data to be written to main memory. These writes will take a long time, likely at least 100 processor clock cycles, and could slow down the processor considerably. One solution to this problem is to use a **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.
- The rate at which writes are generated may also be *less* than the rate at which the memory can accept them, and yet stalls may still occur.
- This can happen when the writes occur in bursts. To reduce the occurrence of such stalls, processors usually increase the depth of the write buffer beyond a single entry. The alternative to a write-through scheme is a scheme called **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.

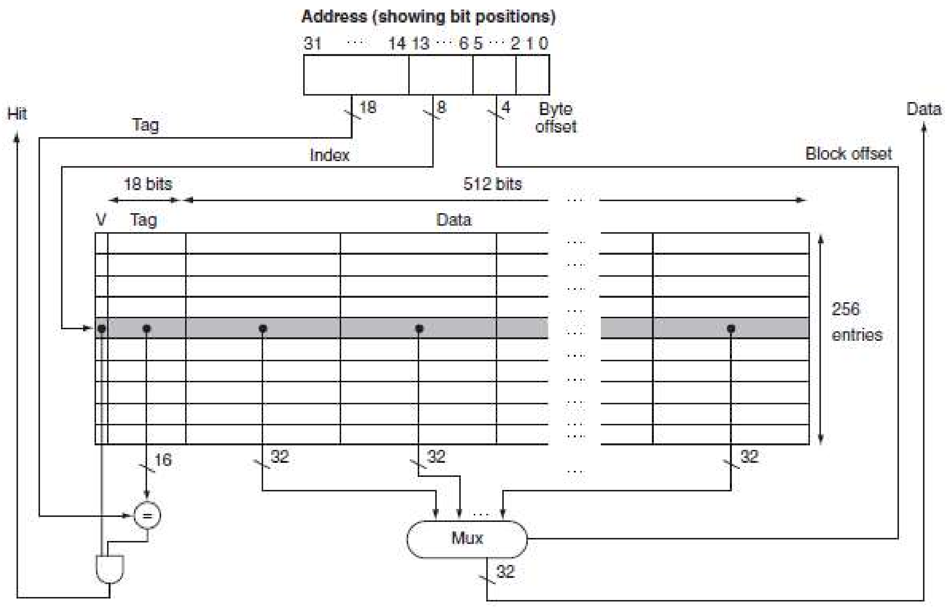


Figure: The 16 KB caches in the Intrinsity FastMATH processor

-
-

- Write-back schemes can improve performance, especially when processors can generate writes as fast or faster than the writes can be handled by main memory; a write-back scheme is, however, more complex to implement than write-through.
- The tag field is 18 bits wide and the index field is 8 bits wide, while a 4-bit field (bits 5−2) is used to index the block and select the word from the block using a 16-to-1 multiplexer.
- In practice, to eliminate the multiplexer, caches use a separate large RAM for the data and a smaller RAM for the tags, with the block offset supplying the extra address bits for the large data RAM. In this case, the large RAM is 32 bits wide and must have 16 times as many words as blocks in the cache.

Steps for a read request to either cache are as follows:

1. Send the address to the appropriate cache. The address comes either from the PC (for an instruction) or from the ALU (for data).
2. If the cache signals hit, the requested word is available on the data lines. Since there are 16 words in the desired block, we need to select the right one. A block index field is used to control the multiplexer which selects the requested word from the 16 words in the indexed block.
3. If the cache signals miss, we send the address to the main memory. When the memory returns with the data, we write it into the cache and then read it to fulfill the request.

5.5 Measuring and Improving Cache Performance

- There are two different techniques for improving cache performance.
    1. Reducing the miss rate by reducing the probability that two different memory blocks will contend for the same cache location.
    2. Reducing the miss penalty by adding an additional level to the hierarchy.

- This technique, called multilevel caching. Cache performance depends on the function of CPU and its execution time.

- 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.

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

- In real processors, the stalls generated by reads and writes can be quite complex, and accurate performance prediction usually requires very detailed simulations of the processor and memory system.
- Memory-stall clock cycles can be defined as the sum of the stall cycles coming from reads plus those coming from writes:

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 require that 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:

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

- Write buffer stalls depend on the proximity of writes, and not just the frequency.
- In systems with a reasonable write buffer depth and a memory capable of accepting writes at a rate that significantly exceeds the average write frequency in programs.
- The write buffer stalls will be small, and we can safely ignore them.

- If a system did not meet write buffer stall then it must be 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.
- In most write-through cache organizations, the read and write miss penalties are the same. If we assume that the write buffer stalls are negligible then we can combine the reads and writes by using a single miss rate and the miss penalty:

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

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

- Let's consider a simple example to help us understand the impact of cache performance on processor performance.

Calculating Cache Performance:

Example:

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%.

Solution:

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

Instruction miss cycles = I x 2% x 100 = 2.00 x I

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

Data miss cycles = I x 36% x 4% x 100 = 1.44 x 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. 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

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

$$= \frac{CPI_{stall}}{CPI_{perfect}} = \frac{5.44}{2}$$

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

What happens if the processor is made faster, but the memory system is not? The amount of time spent on memory stalls will take up an increasing fraction of the execution time. Suppose we
speed-up the computer in the previous example by reducing its CPI from 2 to 1 without changing the clock rate, which might be done with an improved pipeline. The system with cache misses would then have a CPI of 1 + 3.44 = 4.44, and thebnsystem with the perfect cache would be

$$\frac{4.44}{1} = 4.44 \text{ times faster.}$$

The amount of execution time spent on memory stalls would have risen from

$$\frac{3.44}{5.44} = 63\%$$

$$\frac{3.44}{4.44} = 77\%.$$

- Similarly, increasing the clock rate without changing the memory system also increases the performance lost due to cache misses. The previous examples and equations assume that the hit time is not a factor in determining cache performance.

- \If the hit time increases, the total time to access a word from the memory system will increase and it causing an increase in the processor cycle time.
- A larger cache can have a longer access time and it will increase the hit time.
- The increased hit time adds another stage to the pipeline, it may take multiple cycles for a cache hit.
- It is more complex to calculate the performance impact of a deeper pipeline, at some point the increase in hit time for a larger cache could dominate the improvement in hit rate, leading to a decrease in processor performance.
- The time to access data for both hits and misses affects performance, designers use average memory access time (AMAT) as alternative cache designs.
- Average memory access time is the average time to access memory considering both hits and misses and the frequency of different accesses; it is equal to the following:

$$\text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}$$

Calculating Average Memory Access Time:

Example:

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.

Solution:

The average memory access time per instruction is

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

Reducing Cache Misses by More Flexible Placement of Blocks

- Two types of schemes to reduce cache misses.
    1. Direct mapped
    2. Fully associative.

Direct mapped:

- A block can be placed exactly one place in the cache. Here we can perform direct mapping from any block address in memory to a single location in the upper level of the hierarchy.

Fully Associative:

- A block can be placed in any location in the cache. Such a scheme is called fully associative, because a block in memory may be associated with any entry in the cache.
- 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.
- Search is done in parallel with a comparator associated with each cache entry. These comparators significantly increase the hardware cost, effectively making fully associative placement practical only for caches with small numbers of blocks.

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. 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.
- Thus, a set-associative placement combines direct-mapped placement and fully associative placement: a block is directly mapped into a set, and then all the blocks in the set are searched for a match.
- Below figure shows where block 12 may be placed in a cache with eight blocks total, according to the three block placement policies.

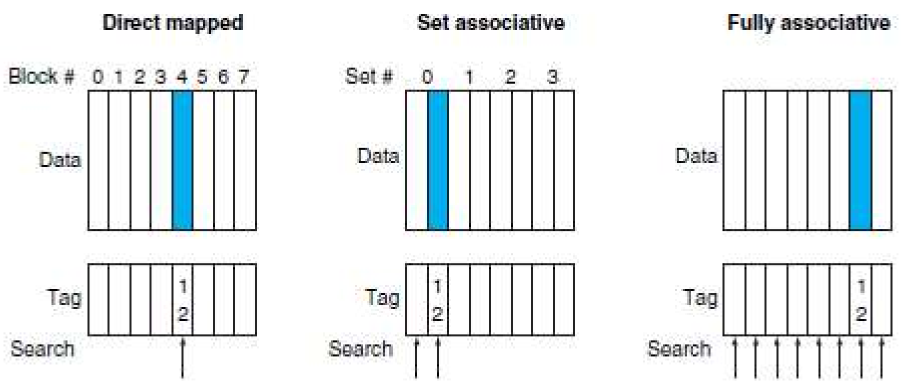


Figure: The location of a memory block in direct mapped, set-associative, and fully associative.

- In direct-mapped placement, there is only one cache block where memory block 12 can be found, and that block is given by (12 modulo 8) = 4.
- In a two-way set-associative cache, there would be four sets, and memory block 12 must be in set (12 mod 4) = 0; the memory block could be in either element of the set.
- In a fully associative placement, the memory block for block address 12 can appear in any of the eight cache blocks.
- Remember that in a direct-mapped cache, the position of a memory block is given by

  (Block number) modulo (Number of blocks in the cache)
- In a set-associative cache, the set containing a memory block is given by

  (Block number) modulo (Number of sets in the cache)
- Since the block may be placed in any element of the set, all the tags of all the elements of the set must be searched.
- In a fully associative cache, the block can go anywhere, and all tags of all the blocks in the cache must be searched.
- The total size of the cache in blocks is equal to the number of sets times the associativity. Thus, for a fixed cache size, increasing the associativity decreases the number of sets while increasing the number of elements per set.

- With eight blocks, an eight-way set-associative cache is the same as a fully associative cache.

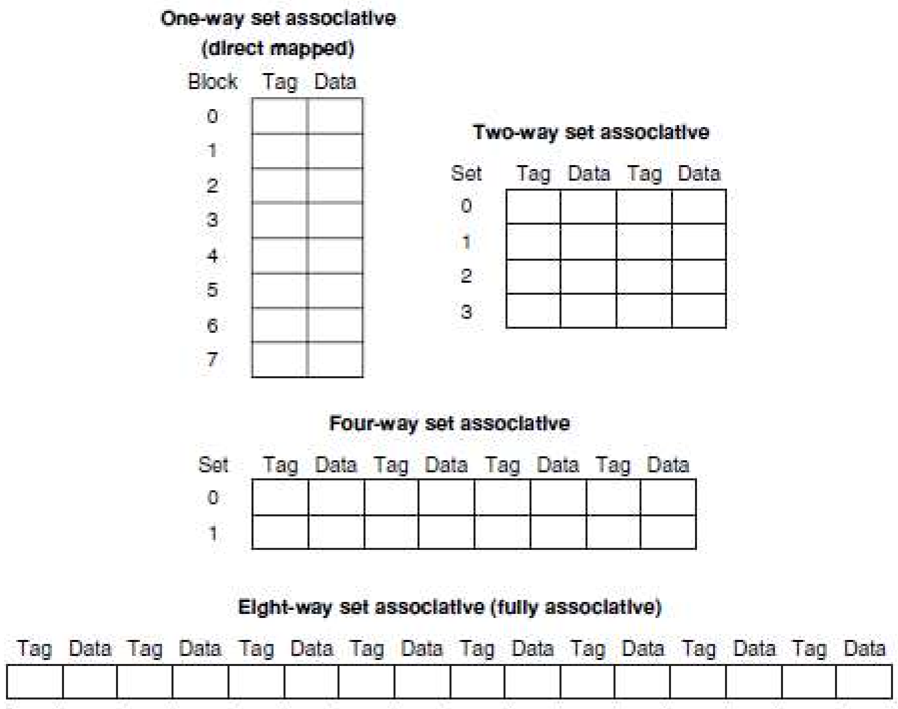

One-way set associative (direct mapped)
Block Tag Data
0 1 2 3 4 5 6 7
Two-way set associative
Set Tag Data Tag Data
0 1 2 3
Four-way set associative
Set Tag Data Tag Data Tag Data Tag Data
0 1
Eight-way set associative (fully associative)
Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data

Figure: An eight-block cache configured as direct mapped, two-way set associative,four-way set associative, and fully associative.

Misses and Associativity in Caches:

Example:

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:

The direct-mapped case is easiest. First, let's determine to which cache block each block address maps:

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

- Now we can fill in the cache contents after each reference, using a blank entry to mean that the block is invalid, colored text to show a new entry added to the cache for the associated reference, and plain text to show an old entry in the cache:

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]	

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

- Because we have a choice of which entry in a set to replace on a miss, we need a replacement rule. Set-associative caches usually replace the least recently used block within a set; that.
- Using this replacement rule, the contents of the set-associative cache after each reference looks like this:

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]		

- 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.
- Direct-mapped cache has five misses and set associative cache has one less than the direct mapped cache.
- 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]	

- Three misses is the best we can do, because three unique block addresses are accessed.
- If we had eight blocks in the cache, there are no replacements in the two-way set-associative cache and all three caches have the same number of misses as the fully associative cache. For 16 blocks, all 3 caches have the same number of misses.
- This example shows that cache size and associativity are not independent in determining cache performance.

5.5.1 Locating a Block in the Cache

- Let's consider the task of finding a block in a cache that is set associative. In a direct-mapped cache, each block in a set-associative cache includes an address tag that gives the block address.
- The tag of every cache block within the appropriate set is checked to see if it matches the block address from the processor.

Tag	Index	Block offset

Figure: Three portions of an address in a set-associative or direct-mapped cache.

- If the total cache size is kept the same, increasing the associativity increases the number of blocks per set, which is the number of simultaneous compares needed to perform the search in parallel: each increase by a factor of 2 in associativity doubles the number of blocks per set and halves the number of sets.
- Each factor-of-2 increase in associativity decreases the size of the index by 1 bit and increases the size of the tag by 1 bit.

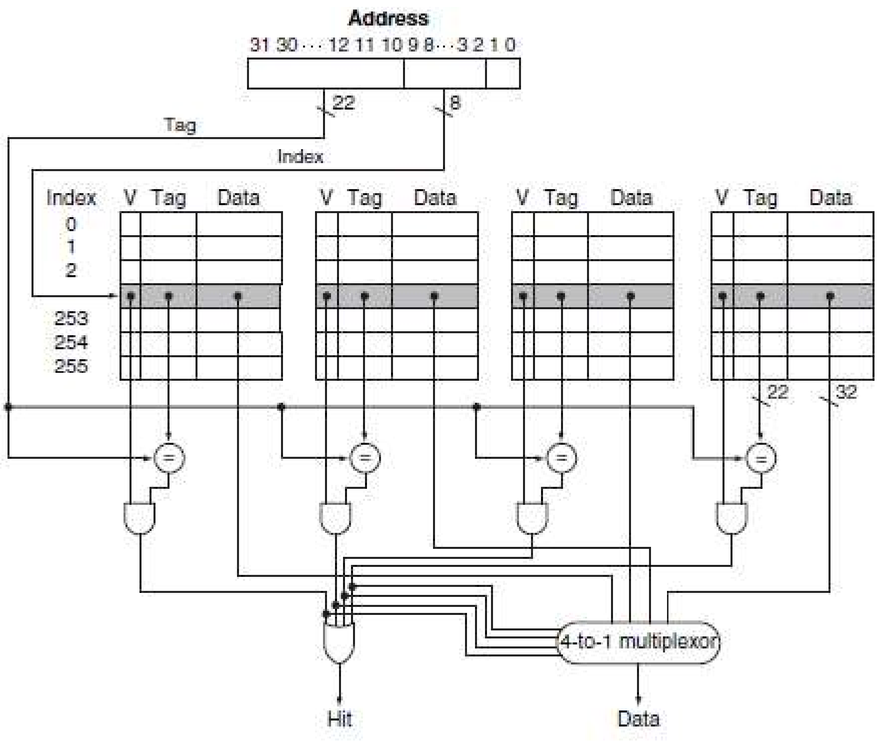


Figure: The implementation of a four-way set-associative.

- In a fully associative cache, a sequential search would make the hit time of a set-associative cache too slow. It has only one set, and all the blocks must be checked in parallel. thus, there is no index, and the entire address, excluding the block offset, is compared against the tag of every block. In this we search the entire cache without any indexing.
- In a direct-mapped cache, only a single comparator is needed, because the entry can be in only one block, and we access the cache simply by indexing.

- The implementation of 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 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 multiplexer with a decoded select signal. In some implementations, the Output enable signal can be used to select the entry in the set that drives the output.
- The Output enable signal will eliminates the need for the multiplexer.
- The cache access consists of indexing the appropriate set and then searching the tags of the set.
- The choice among direct-mapped, set-associative, or fully associative mapping in any memory hierarchy will depend on the cost of a miss versus the cost of implementing associativity, both in time and in extra hardware.

5.5.2 Choosing Which Block to Replace

- When a miss occurs in a direct-mapped cache, the requested block can go in exactly one position, and the block occupying that position must be replaced.
- In an associative cache, we have a choice of where to place the requested block, and hence a choice of which block to replace. In a fully associative cache, all blocks are candidates for replacement.
- In a set-associative cache, we must choose among the blocks in the selected set.

Least recently used:

- The most commonly used scheme is least recently used(LRU).
- In LRU scheme, the block replaced is the one that has been unused for the longest time.
- As associativity increases, implementing LRU gets harder. It is implemented by keeping track of when each element in a set was used relative to the other element in the set.

5.5.3 Reducing the Miss Penalty Using Multilevel Caches:

- All modern computers make use of caches. To close the gap further between the fast clock rates of modern processors and the increasingly long time required to access DRAMs, most microprocessors support an additional level of caching.
- This second-level cache is usually on the same chip and is accessed whenever a miss occurs in the primary cache.
- If the second-level cache contains the desired data, the miss penalty for the first-level cache will be essentially the access time of the second-level cache, which will be much less than the access time of main memory.
- If neither the primary nor the secondary cache contains the data, a main memory access is required, and a larger miss penalty is incurred.
- The design considerations for a primary and secondary cache are significantly different, because the presence of the other cache changes the best choice versus a single-level cache.
- In particular, a two-level cache structure allows the primary cache to focus on minimizing hit time to yield a shorter clock cycle or fewer pipeline stages, while allowing the secondary cache to focus on miss rate to reduce the penalty of long memory access times.
- The effect of these changes on the two caches can be seen by comparing each cache to the optimal design for a single level of cache. In comparison to a single level cache, the primary cache of a multilevel cache is often smaller.
- Furthermore, the primary cache may use a smaller block size, to go with the smaller cache size and also to reduce the miss penalty.
- In comparison, the secondary cache will be much larger than in a single-level cache, since the access time of the secondary cache is less critical.
- With a larger total size, the secondary cache may use a larger block size than appropriate with a single-level cache. It often uses higher associativity than the primary cache given the focus of reducing miss rates.

5.6 Virtual Memory

- It is a technique that uses main memory as a "cache" for secondary storage.
- The main memory can act as a "cache" for the secondary storage, usually implemented with magnetic disks. This technique is called virtual memory.
- There were two major motivations for virtual memory:
    1. To allow efficient and safe sharing of memory among multiple programs
    2. To remove the programming burdens of a small, limited amount of main memory.

Efficient and safe sharing of memory among multiple programs:

- Consider a collection of programs running all at once on a computer.
- Of course, to allow multiple programs to share the same memory, we must be able to protect the programs from each other, ensuring that a program can only read and write the portions of main memory that have been assigned to it.
- Main memory need to contain only the active portions of the many programs, just as a cache contains only the active portion of one program. Thus, the principle of locality enables virtual memory as well as caches.
- Virtual memory allows us efficiently to share the processor as well as the main memory.
- We cannot know which programs will share the memory with other programs when we compile them.
- The programs sharing the memory change dynamically while the programs are running.
- Because of this dynamic interaction, we would like to compile each program into its own address space—a separate range of memory locations accessible only to this program.
- 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 programs.

Physical address:

- It is an address in main memory.

Protection:

- 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.

Single user program to exceed the size of primary memory:

- The second motivation for virtual memory is to allow a single user program to exceed the size of primary memory.
- If a program became too large for memory, it was up to the programmer to make it fit.
- Programmers divided programs into pieces and then identified the pieces that were mutually exclusive.
- These overlays are loaded or unloaded under user program control during execution, with the programmer ensuring that the program never tried to access an overlay that was not loaded and that the overlays loaded never exceeded the total size of the memory.

- Overlays are organized as modules, each containing both code and data.
- A virtual memory block is called a page, and a virtual memory miss is called a page fault.
- With virtual memory, the processor produces a virtual address, which is translated by a combination of hardware and software to a physical address, which in turn can be used to access main memory.
- The below figure shows the virtually addressed memory with pages mapped to main memory. This process is called address mapping or address translation.
- The two memory hierarchy levels controlled by virtual memory are usually DRAMs and magnetic disks.

- If we return to our library comparison, we can think of a virtual address as the title of a book and a physical address as the location of that book in the library.

Page fault:

- It is an event that occurs when an accessed page is not present in main memory.

Virtual address:

- This is an address that corresponds to a location in virtual space and is translated by address mapping to a physical address when memory is accessed.

Address translation:

- It is also called address mapping. The process by which a virtual address is mapped to an address used to access memory.

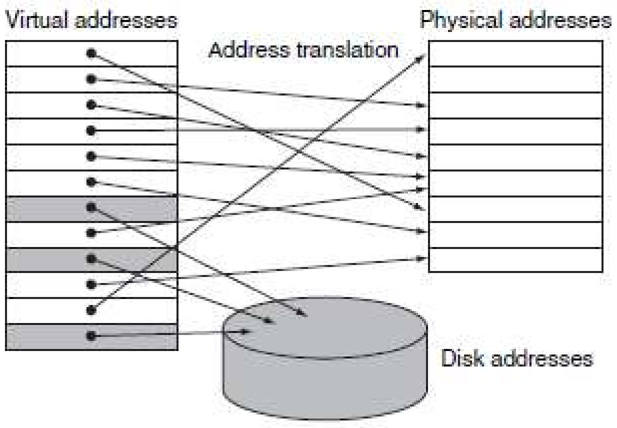


Figure : In virtual memory, blocks of memory (called pages) are mapped from one set of addresses (called virtual addresses) to another set (called physical addresses).

- The processor generates virtual addresses while the memory is accessed using physical addresses.
- Both the virtual memory and the physical memory are broken into pages, so that a virtual page is mapped to a physical page.
- It is also possible for a virtual page to be absent from main memory and not be mapped to a physical address; in that case, the page resides on disk.

- 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.
- Virtual memory also simplifies loading the program for execution by providing relocation.
- Relocation maps the virtual addresses used by a program to different physical addresses before the addresses are used to access memory. This relocation allows us to load the program anywhere in main memory.
- In virtual memory, the address is broken into a virtual page number and a page offset.
- The below 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 offset, which is not changed, constitutes the lower portion.
- The number of bits in the page offset 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.

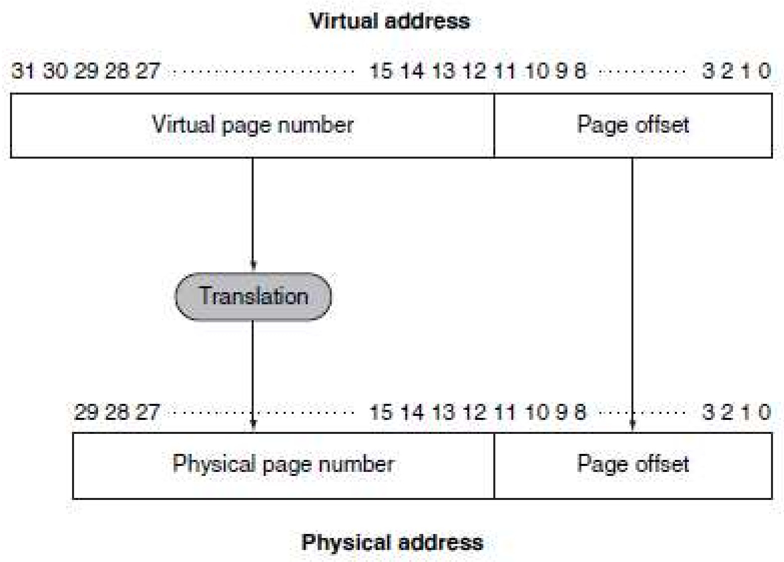


Figure: Mapping from a virtual to a physical address

Reasons for designing Virtual Memory:

- Organizations that reduce the page fault rate are attractive. Page faults can be handled in software because the overhead will be small compared to the disk access time
- Write-through will not work for virtual memory, since writes take too long. Instead, virtual memory systems use write-back.

Segmentation:

- 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 offset.

5.6.1 Placing a Page

- To reduce page the designer must optimizing page placement in more attractive way.
- If virtual page to be mapped to any physical page, the operating system can then choose to replace any page it wants when a page fault occurs.
- 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 see the corresponding physical page number.
- Each program has its own page table, which maps the virtual address space of that program to main memory.
- 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 that the page table is in a fixed and contiguous area of memory.

Page table :

- The table containing the virtual to physical addresses translations in a virtual memory system.

- The table, which is stored in memory, it 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.

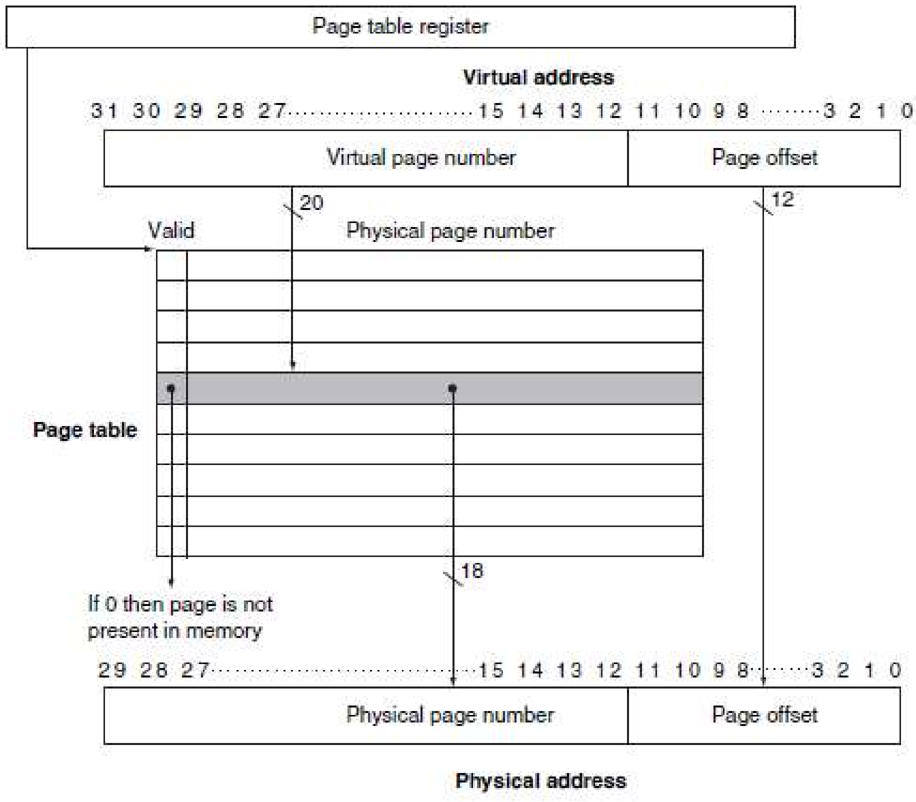


Figure: The page table is indexed with the virtual page number to obtain the corresponding portion of the physical address.

- The above figure uses the page table register, the virtual address, and the indicated page table to show how the hardware can form a physical address.
- A valid bit is used in each page table entry, just as we did in a cache.
- If the bit is off, the page is not present in main memory and a page fault occurs.
- If the bit is on, the page is in memory and the entry contains the physical page number.

5.6.2 Page Faults

- If the valid bit for a virtual page is off, a page fault occurs. The operating system must be given control.
- Once the operating system gets control, it must find the page in the next level of the hierarchy (usually magnetic disk) and decide where to place the requested page in main memory.
- The virtual address alone does not immediately tell us where the page is on disk.
- In a virtual memory system, we must keep track of the location on disk of each page in virtual address space.
- Because we do not know ahead of time when a page in memory will be replaced, the operating system usually creates the space on disk for all the pages of a process when it creates the process. This disk space is called the swap space.
- At that time, it also creates a data structure to record where each virtual page is stored on disk.
- This data structure may be part of the page table or may be an auxiliary data structure indexed in the same way as the page table.
- Below figureshows the organization when a single table holds either the physical page number or the disk addresses.
- The operating system also creates a data structure that tracks which processes and which virtual addresses use each physical page.
- The virtual page number is used to index the page table. If the valid bit is on, the page table supplies the physical page number to the virtual page.
- If the valid bit is off, the page currently resides only on disk, at a specified disk address. Pages in main memory and the pages on disk are the same size.
- The operating system also creates a data structure that tracks which processes and which virtual addresses use each physical page.
- When a page fault occurs, if all the pages in main memory are in use, the operating system must choose a page to replace.

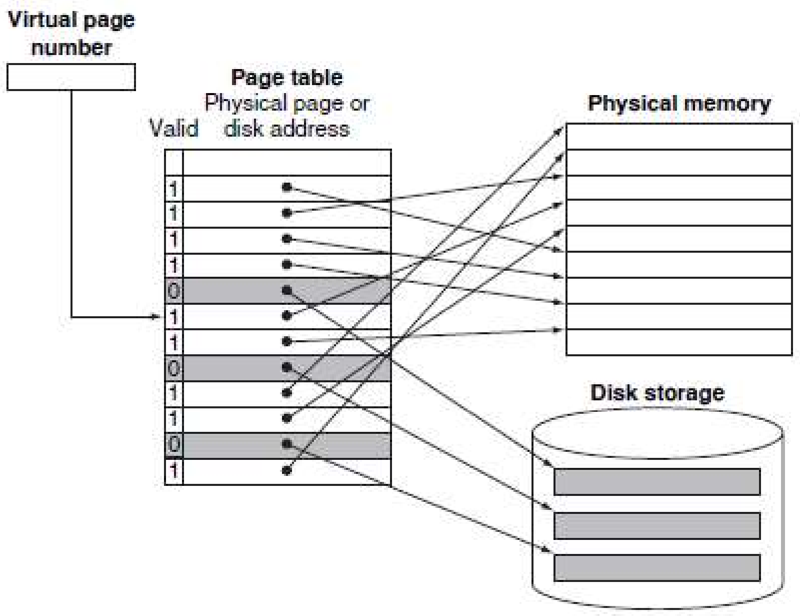


Figure: The page table maps each page in virtual memory to either a page in main memory or a page stored on disk.

- 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 replaced pages are written to swap space on the disk.

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.

Example:

With a 32-bit virtual address, 4 KB pages, and 4 bytes per page table entry, we can compute the total page table size:

$$\text{Number of page table entries} = \frac{2^{32}}{2^{12}} = 2^{20}$$

$$\text{Size of page table} = 2^{20} \text{ page table entries} \times 2^{2} \frac{\text{bytes}}{\text{page table entry}} = 4 \text{ MB}$$

- That is, we would need to use 4 MB of memory for each program in execution at any time.

Writes in Virtual Memory:

- In a virtual memory system, writes to the next level of the hierarchy can take millions of processor clock cycles; therefore, building a write buffer to allow the system to write-through to disk would be completely impractical.
- So virtual memory systems must use write-back, performing the individual writes into the page in memory, and copying the page back to disk when it is replaced in the memory.
- A write-back scheme has another major advantage in a virtual memory system. Because the disk transfer time is small compared with its access time, copying back an entire page is much more efficient than writing individual words back to the disk.
- A write-back operation, although more efficient than transferring individual words, is still costly. Thus, we would like to know whether a page needs to be copied back when we choose to replace it.
- To track whether a page has been written since it was read into the memory, a dirty bit is added to the page table.
- The dirty bit is set when any word in a page is written. If the operating system chooses to replace the page, the dirty bit indicates whether the page needs to be written out before its location in memory can be given to another page. Hence, a modified page is often called a dirty page.

5.7 Translation Look -aside Buffer (TLB)

- Page tables are stored in main memory; every memory access by a program can take at least twice as long: one memory access to obtain the physical address and a second access to get the data.
- To improve access performance we need to have a reference to the page table. So 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 look-aside buffer (TLB).

- It is a cache that keeps track of recently used address mappings to try to avoid an access to the page table.
- Below 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.

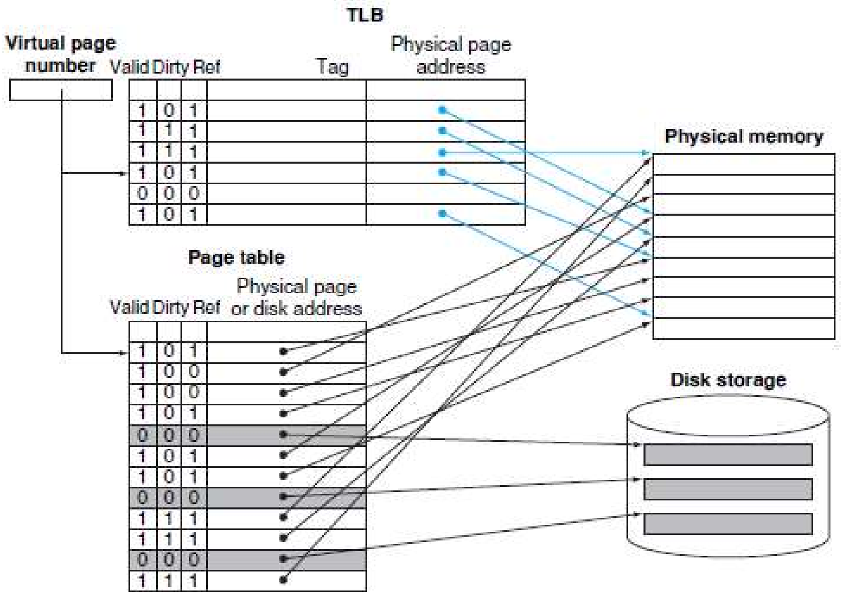


Figure: The TLB acts as a cache of the page table.

- The TLB contains a subset of the virtual-to-physical page mappings that are in the page table.
- The TLB mappings are shown in color. Because the TLB is a cache, it must have a tag field.If there is no matching entry in the TLB for a page, the page table must be examined.
- The page table either supplies a physical page number for the page or indicates that the page resides on disk, in which case a page fault occurs.
- On every reference, we look up the virtual page number in the TLB. If we get a hit, the physical page number is used to form the address, and the corresponding reference bit is turned on.

- 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 merely 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.
- 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. Both cases have basic operations and only little performance difference between the two approaches.
- 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. Some typical values for a TLB might be
    1. TLB size: 16–512 entries
    2. Block size: 1–2 page table entries (typically 4–8 bytes each)
    3. Hit time: 0.5–1 clock cycle
    4. Miss penalty: 10–100 clock cycles
    5. Miss rate: 0.01%–1%
- Designers have used a wide variety of associativities in TLBs, many systems use fully associative TLBs.
- Because a fully associative mapping has a lower miss rate and it has low cost. Because with a fully associative mapping, choosing the entry to replace becomes complex because implementing a hardware LRU scheme is too expensive.

- TLB misses are much more frequent than page faults and thus must be handled more cheaply, we cannot afford an expensive software algorithm.

5.7.1 The Intrinsity Fast MATH TLB

- The Intrinsity Fast MATH has the memory system as 4 KB pages and a 32-bit address space; thus, the virtual page number is 20 bits long.
- The physical address is the same size as the virtual address. The TLB contains 16 entries, it is fully associative.

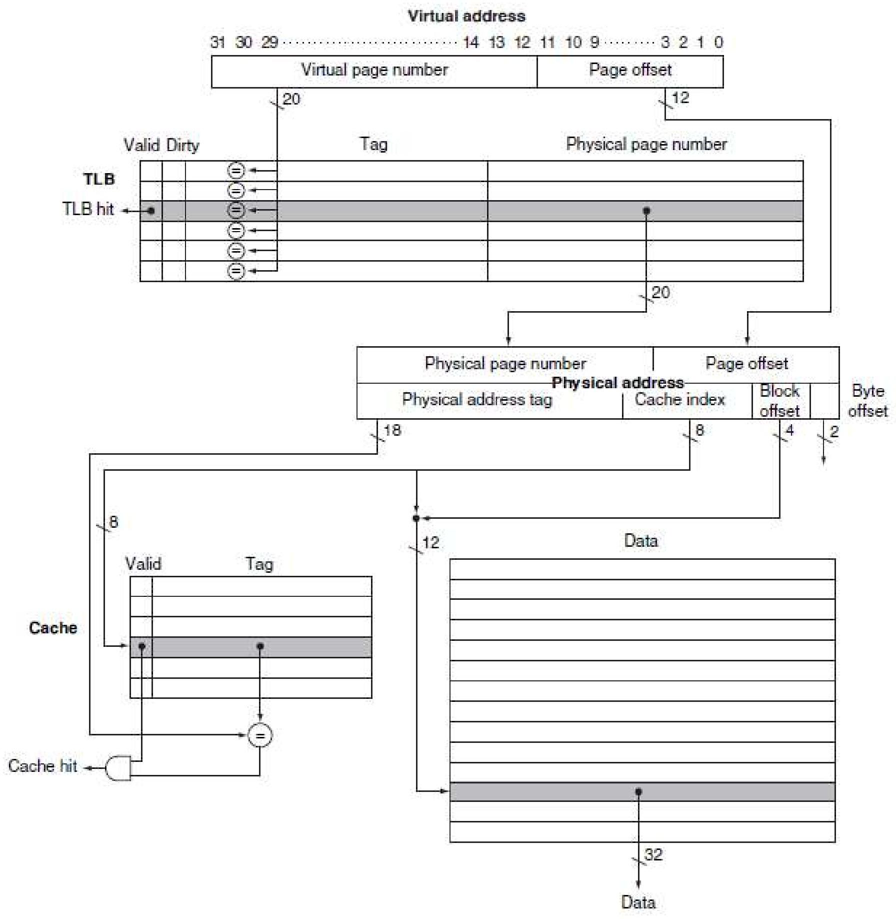
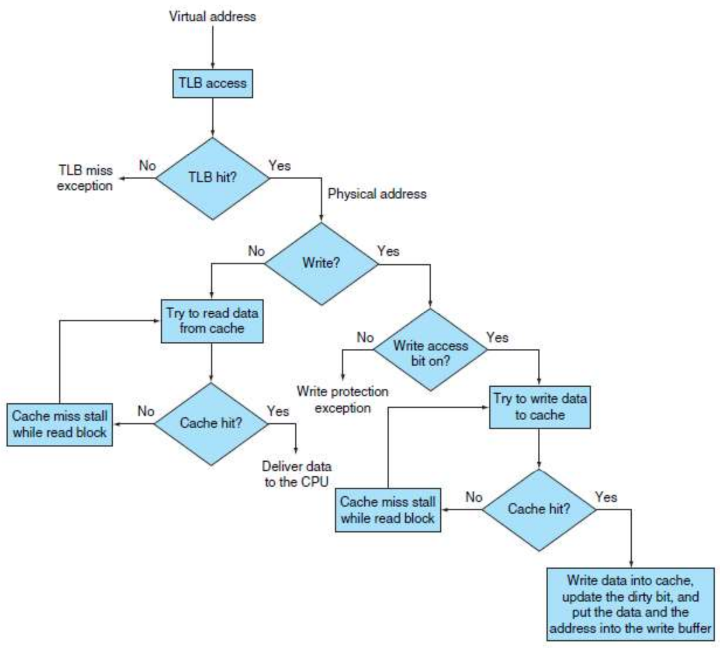


Figure : TLB and cache implement the process in Intrinsity Fast MATH.

- TLB entries are shared between the instruction and data references. Each entry is 64 bits wide and contains a 20-bit tag, the corresponding physical page number, a valid bit, a dirty bit, and other bookkeeping bits. Most MIPS systems software is used to handle TLB misses.

- When a TLB miss occurs, the MIPS hardware saves the page number of the reference in a special register and generates an exception.
- The exception invokes the operating system, which handles the miss in software. To find the physical address for the missing page the following task has been taken.
- The TLB miss routine indexes the page table using the page number of the virtual address. The page table register indicates the starting address of the active process page table.
- Using a special set of system instructions that can update the TLB, the operating system places the physical address from the page table into the TLB.
- A true page fault occurs if the page table entry does not have a valid physical address.The hardware maintains an index that indicates the recommended entry to replace; the recommended entry is chosen randomly.
- For write requests, the write access bit in the TLB must be checked. This bit prevents the program from writing into pages for which it has only read access.
- If the program attempts a write and the write access bit is off, an exception is generated.

Steps in processing a read or a write request:


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 hit?
No
Yes
Cache miss stall while read block
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

5.7.2 Integrating Virtual Memory, TLBs, and Caches

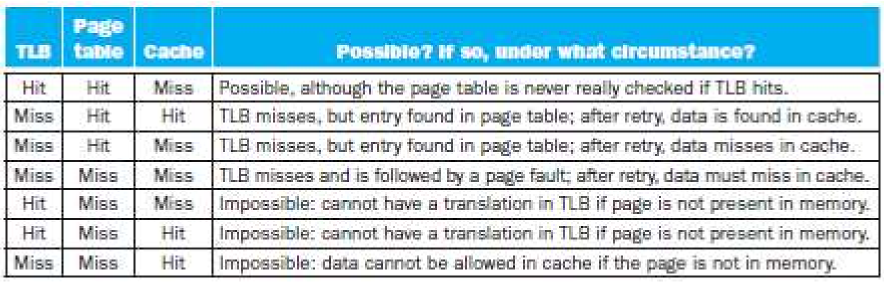
- Virtual memory and cache systems work together as a hierarchy, so that data cannot be in the cache unless it is present in main memory.
- The operating system helps maintain this hierarchy by flushing the contents of any page from the cache when it decides to migrate that page to disk.
- At the same time, the OS modifies the page tables and TLB, so that an attempt to access any data on the migrated page will generate a page fault.
- Under the best of circumstances, a virtual address is translated by the TLB and sent to the cache where the appropriate data is found, retrieved, and sent back to the processor.
- In the worst case, a reference can miss in all three components of the memory hierarchy: the TLB, the page table, and the cache.

Overall Operation of a Memory Hierarchy

Example:

- In a memory hierarchy which includes a TLB and a cache organized as shown, a memory reference can encounter three different types of misses: a TLB miss, a page fault, and a cache miss.
- Consider all the combinations of these three events with one or more occurring. For each possibility, state whether this event can actually occur and under what circumstances.

Solution:



TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

Figure shows all combinations and whether each is possible in practice.

5.7.3 Implementing Protection with Virtual Memory

- The most important function of virtual memory is to allow sharing of a single main memory by multiple processes, while providing memory protection among these processes and the operating system.
- The protection mechanism must ensure that although multiple processes are sharing the same main memory, one renegade process cannot write into the address space of another user process or into the operating system either intentionally or unintentionally.
- The write access bit in the TLB can protect a page from being written. Without this level of protection, computer viruses would be even more widespread.
- To enable the operating system to implement protection in the virtual memory system, the hardware must provide at least the three basic capabilities.

1. Support at least two modes that indicate whether the running process is a user process or an operating system process, variously called a supervisor process, a kernel process, or an executive process.
2. Provide a portion of the processor state that a user process can read but not write. This includes the user/supervisor mode bit, which dictates whether the processor is in user or supervisor mode, the page table pointer, and the TLB. To write these elements, the operating system uses special instructions that are only available in supervisor mode.
3. Provide mechanisms whereby the processor can go from user mode to supervisor mode and vice versa.
    - The first direction is typically accomplished by a system call exception, implemented as a special instruction (syscallin the MIPS instruction set) that transfers control to a dedicated location in supervisor code space.
    - As with any other exception, the program counter from the point of the system call is saved in the exception PC (EPC), and the processor is placed in supervisor mode.
    - To return to user mode from the exception, use the return from exception (ERET) instruction, which resets to user mode and jumps to the address in EPC.

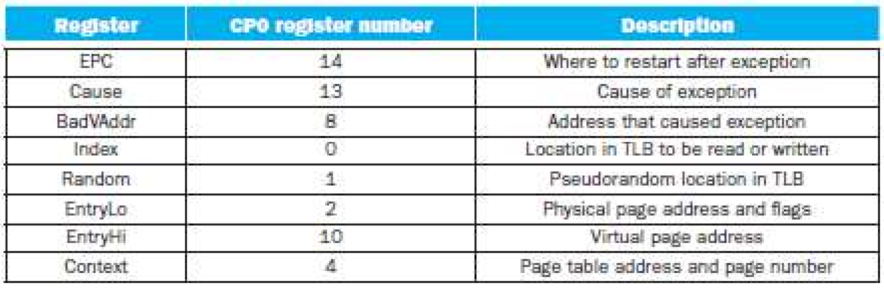
- We also prevent a process from reading the data of another process. Once we begin sharing main memory, we must provide the ability for a process to protect its data from both reading and writing by another process; otherwise, sharing the main memory will be a mixed blessing! Each process has its own virtual address space.
- If the operating system keeps the page tables organized so that the independent virtual pages map to disjoint physical pages, so one process will not be able to access another's data. It is more important because user process cannot change the page table mapping.
- The operating system must assure if it prevents the user process from modifying its own page tables. The operating system must be able to modify the page tables.
- When processes want to share information in a limited way, the operating system must assist them. Because accessing the information of another process requires changing the page table of the accessing process.
- The write access bit can be used to restrict the sharing to just read sharing, and, like the rest of the page table, this bit can be changed only by the operating system.
- To allow another process, say, P1,to read a page owned by process P2, P2 would ask the operating system to create a page table entry for a virtual page in P1's address space that points to the same physical page that P2 wants to share.
    - The operating system could use the write protection bit to prevent P1 from writing the data, if that was P2's wish.
    - Any bits that determine the access rights for a page must be included in both the page table and the TLB, because the page table is accessed only on a TLB miss.

5.7.4 Handling TLB Misses and Page Faults

- The translation of virtual to physical addresses with a TLB is straightforward when we get a TLB hit, handling TLB misses and page faults is more complex. A TLB miss occurs when no entry in the TLB matches a virtual address. A TLB miss can indicate one of two possibilities:

1. The page is present in memory, and we need only create the missing TLB entry.
2. The page is not present in memory, and we need to transfer control to the operating system to deal with a page fault.

- MIPS traditionally handles a TLB miss in software. It brings in the page table entry from memory and then re-executes the instruction that caused the TLB miss.
- Upon re-executing, it will get a TLB hit. If the page table entry indicates the page is not in memory, this time it will get a page fault exception.
- Handling a TLB miss or a page fault requires using the exception mechanism to interrupt the active process, transferring control to the operating system, and later resuming execution of the interrupted process.
- A page fault will be recognized sometime during the clock cycle used to access memory. To restart the instruction after the page fault is handled, the program counter of the instruction that caused the page fault must be saved.
- A TLB miss or page fault exception must be asserted by the end of the same clock cycle that the memory access occurs, so that the next clock cycle will begin exception processing rather than continue normal instruction execution.
- If the page fault was not recognized in this clock cycle, a load instruction could overwrite a register, and this could be disastrous when we try to restart the instruction.
- We must prevent the write into memory from actually completing when there is a page fault; this is usually done by deasserting the write control line to the memory.



Register	CP0 register number	Description
EPC	14	Where to restart after exception
Cause	13	Cause of exception
BadVAddr	8	Address that caused exception
Index	0	Location in TLB to be read or written
Random	1	Pseudorandom location in TLB
EntryLo	2	Physical page address and flags
EntryHi	10	Virtual page address
Context	4	Page table address and page number

Figure :MIPS control registers.

- Once the operating system knows the virtual address that caused the page fault, it must complete three steps:
    1. Look up the page table entry using the virtual address and find the location of the referenced page on disk.
    2. Choose a physical page to replace; if the chosen page is dirty, it must be written out to disk before we can bring a new virtual page into this physical page.
    3. Start a read to bring the referenced page from disk into the chosen physical page.

- Page fault exceptions for data accesses are difficult to implement properly in a processor because of a combination of three characteristics:
    1. They occur in the middle of instructions, unlike instruction page faults.
    2. The instruction cannot be completed before handling the exception.
    3. After handling the exception, the instruction must be restarted as if nothing had occurred.

- When a TLB miss occurs, the MIPS hardware saves the page number of the reference in a special register called BadVAddr and generates an exception.
- The exception invokes the operating system, which handles the miss in software.

Handler:

- It is the name of a software routine invoked to handle an exception or interrupt. Control is transferred to address 8000 0000hex, the location of the TLB miss handler.
- To find the physical address for the missing page, the TLB miss routine indexes the page table using the page number of the virtual address and the page table register, which indicates the starting address of the active process page table.
- MIPS code for TLB miss handler is shown below.

TLB miss:

```
mfc0 $k1,Context          # copy address of PTE into temp $k1
lw   $k1, 0($k1)          # put PTE into temp $k1
mtc0  $k1,EntryLo         # put PTE into special register Entry Lo
tlbwr                     # put Entry Lo into TLB entry at Random
eret                      # return from TLB miss exception
```

- To make this indexing fast, MIPS hardware places everything you need in the special Context register: the upper 12 bits have the address of the base of the page table, and the next 18 bits have the virtual address of the missing page.
- Each page table entry is one word, so the last 2 bits are 0.
- Thus, the first two instructions copy the Context register into the kernel temporary register $k1 and then load the page table entry from that address into $k1.
- The instruction tlbwr copies from control register Entry Lo into the TLB entry selected by the control register Random.
- Random implements random replacement, so it is basically a free-running counter. A TLB miss takes about a dozen clock cycles.
- The TLB miss handler does not check to see if the page table entry is valid. Because the exception for TLB entry missing is much more frequent than a page fault, the operating system loads the TLB from the page table without examining the entry and restarts the instruction.
- If the entry is invalid, another and different exception occurs, and the operating system recognizes the page fault.

5.8 Input / Output System

- The data processing function of a computer involve its processor and Main Memory.
- The processor fetches instructions and data from Main memory, process them and store the results back in Memory. The other system components are secondary memory, user interface device and so on, constitute the IO (Input-Output) system.

IO Control Methods:

- Input-Output operations are distinguished based on the involvement of CPU in their execution.
- Input-Output operation refers data transfer between CPU and M, (or) between Input/Output and CPU.
- It is completely controlled by the CPU such kind of process is called programmed IO.
- IO operations are controlled by IO devices such process is called DMA.

Direct Memory Access:

- The block of information can transfer from M to IO and Vice Versa, without CPU intervention.
- The IO device interface control circuit is called a DMA controller is placed between Memory and IO.
- When IO devices need any data transfer from Memory without CPU, it activate DMA request signal to the DMA controller for requesting Direct Memory access.
- In response to DMA request, the DMA controller interrupts the CPU for requesting to give up the system BUS by sending HOLD signal.
- The CPU controls the system bus to DMA controller by sending acknowledgment signal (HLDA). The CPU is still responsible for initiating each block transfer, that is, CPU load starting address of block transfer and count value (number of data to be transferred) in DMA control register.
- Then DMA controller takes the control of system bus and act as bus master to allow the data transfer between IO and Memory. Once data transfer is over, the controller interrupt the CPU to indicate that data transfer is over, in response to that once again CPU take the control of system bus.
- Therefore, a DMA controller has partial control of IO operation. Another technique that has complete control of IO operation can given up by the CPU using IOP processor.

- An IOP processor like DMA has direct access to main memory without CPU. Usually IOP is connected to the devices and it is controlled by a separate system bus called IO bus.

5.9 Programmed IO

- This technique is useful in small, low speed systems where hardware costs must be minimized.
- In this method, all IO operations can be executed under control of CPU i.e., every data transfer operation involving IO device requires the execution of an instruction by the CPU.
- The IO devices do not have direct access to main memory M. A data transfer from an IO devices to M requires the CPU to execute several instruction, including
- Input Instruction to transfer a word from the IO device to the CPU and
- Store (Out) instruction to transfer the word from the CPU to M.

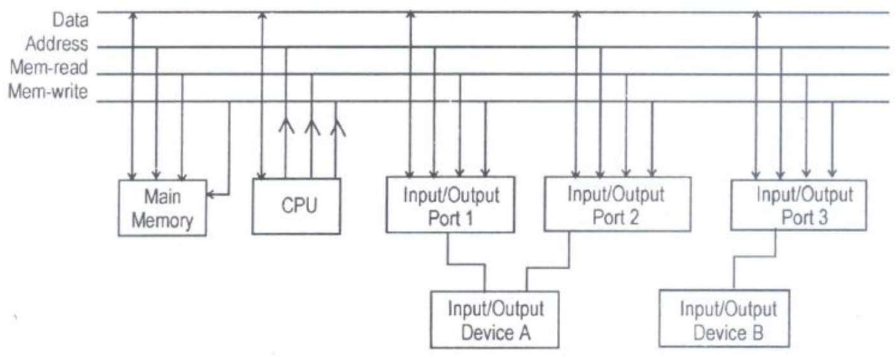
5.9.1 IO Addressing

- In systems employing programmable 10, the CPU, M and 10 devices usually communicate via the system bus.
- The address lines of the system bus are used to select Memory Location or IO devices.
- An IO device is connected to the bus via IO ports (called addressable data register). Simplest IO ports are buffer or latch.
- A most commonly used technique to assign address space to the IO ports are:
  - Memory-Mapped IO
  - IO Mapped IO.

Memory mapped I/O:

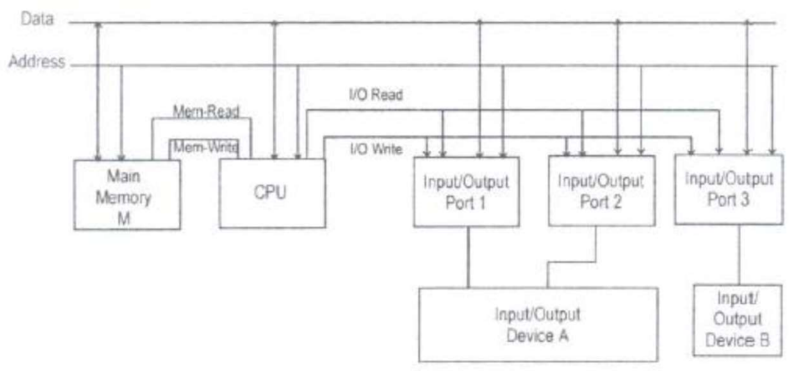
- Main memory space is used to assign IO ports.
- Memory instruction is used for data transfer between IO and CPU.
- CPU communicates with IO device like one of the memory location.

- Memory read/write control signals are activated by the CPU when processing memory reference instruction.



IO mapped IO/Isolated I/O:

- Main memory and IO space are separate.
- Separate IO instruction for data transfer between IO and CPU.
- CPU activate IO read/write control signal when executing IO instruction.



5.9.2 IO Instruction

- The two IO instructions are implemented in programmed IO. They are
- IN X; transfer a word from specified input device to the accumulator register A.
- OUT X; transfer the word from A register to output device specified, where X is the IO port address.

- When the CPU executes an IO instruction, the addressed IO port is expected to be ready to respond to the instruction.
- Therefore, the IO device must transfer data to or form the CPU-IO data bus within a specified period.
- If a slow peripheral device is connected to CPU, it is necessary to check IO status whether the device is ready or not ready for data transfer.
- With programmed IO the CPU can be programmed to test the IO device's status before initiating an IO data transfer.
- The following steps to be performed by CPU to determine the status of an IO devices:
    - Read the IO device's status bit.
    - Test the status bit to determine if the device is ready to begin transferring data.
    - If not ready, return to step 1; otherwise, proceed with the data transfer following program shows transfer a data word from an 10 device to the CPU's A register.

Program to read one word form IO device

	Instruction	*Comments*
Wait:	IN Port1	Read IO device status into A register
	CPI Ready	Compare immediate word Really to A; If equal Set $Z = 1$; otherwise $Z = 0$
	JNZ Wait	If $Z \neq 1$ (IO device not ready), jump to wait
	IN Port2	Read data word into A register.

- The digital PDP-8 is an early minicomputer, it has an IO instruction called TSK and it tests the status of the IO device and modifies the CPU program counter based on the test outcome.
- TSK means Test IO device Status Flag and Skip the next instruction if the status flag is set .It can be implemented by two control lines linking the CPU and the IO device.

- On executing TSK instruction the CPU sends a signal called TEST STATUS to the IO device. If the device status flag is set then a return is sent on the SKIP line.

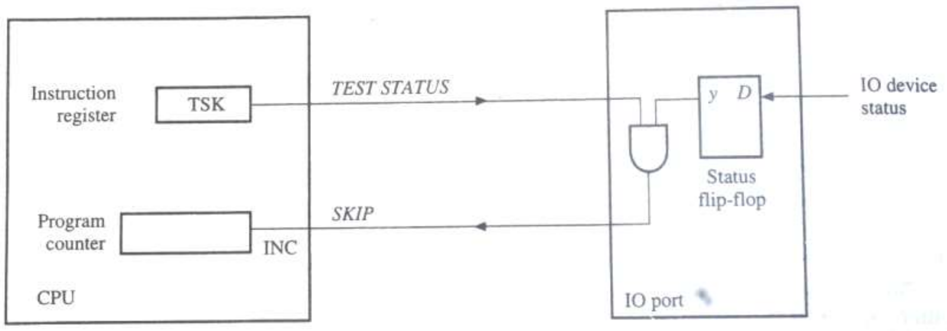


Figure: Implementation of the test status and skip IO instruction

- SKIP line increments the PC value so it skip the next instruction. Using this instruction the program to read one word from an IO device can be reduced to the following way

```
Wait:  TSK   1
       JMP   WAIT
       IN    2
```

- The task of common IO programming is to transfer a block of words between an IO device and a contiguous region of memory.

5.9.3 IO Interface Circuits

- To connect IO device to a computer system we need to use standard IC's known as IO interface circuits, peripheral interface adapter, etc.
- These circuits allow IO devices with different characteristics to be connected to a standard bus with a minimum of special purpose hardware or software.
- The simplest interface circuit is a one word, addressable register that serves as an IO port.
- The major microprocessor families contain various general and special purpose IO interface circuits.

- Most basic IO interface circuits are programmable to act as serial or parallel ports. Serial port has many types of slow peripheral devices ranging from secondary memory units to network connections.
- Parallel ports are designed to interface with IO devices employing multi-bit, bi-directional data paths.

Intel 8255 programmable peripheral interface circuit:

- This IC was designed for interfacing IO devices with the Intel 8085 and other small microprocessors. It has 40 pin package among that
    - 8 pins connect the 8255 to an 8 bit bidirectional CPU data bus.
    - 24 Io pins can be attached to several IO devices.
- These IO pins are programmable and functions of those pins are determined by a control word issued by a CPU instruction and stored internally in the 8255.
- This control word can specify a variety of operating modes involving either synchronous or asynchronous data transfers.
- In 8255 ,24 pins on the IO side is divided into 3 groups and each groups contain 8 bit they are
    1. Group A
    2. Group B
    3. Group C
- All these three groups act as an independent IO port. The C lines are further divided into two 4-bit groups C_A and C_B.
- Groups C_A and C_B are commonly used as status or handshaking lines in conjunction with the A and B ports.
- Address lines A0 and A1 are used to select one of the three ports A, B and C for use in an IO operation.
- The fourth address combination is used in conjunction with an output instruction of the form OUT CW to store an 8-bit user specified control word in the 8255's internal control.
- This control word has following two functions:
    1. It specifies whether the A, B and C ports are act as input, or output or in the case of A and B only, act as bidirectional IO ports.

2. It programs certain C lines to generate handshaking and interrupt signals automatically in response to actions by an IO device.

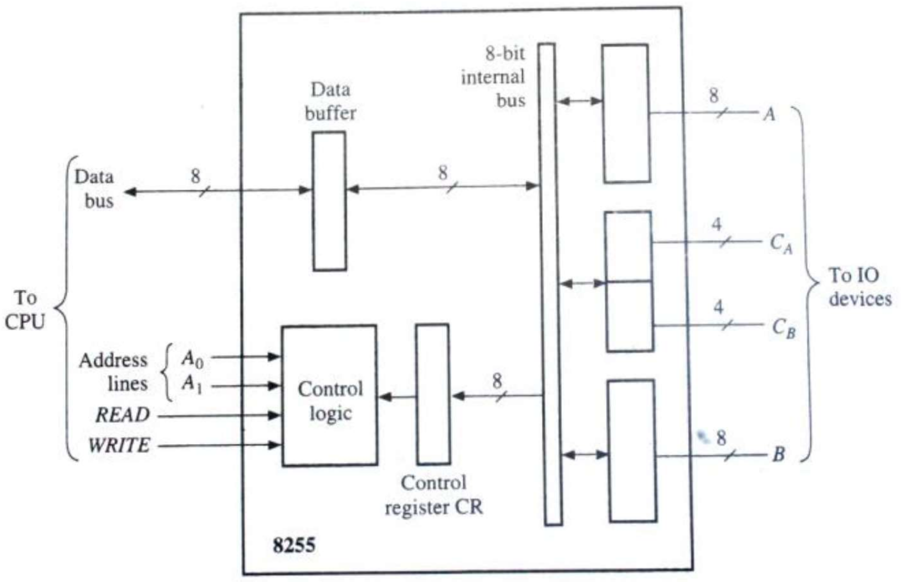


Figure: Intel 8255 programmable peripheral interface circuit

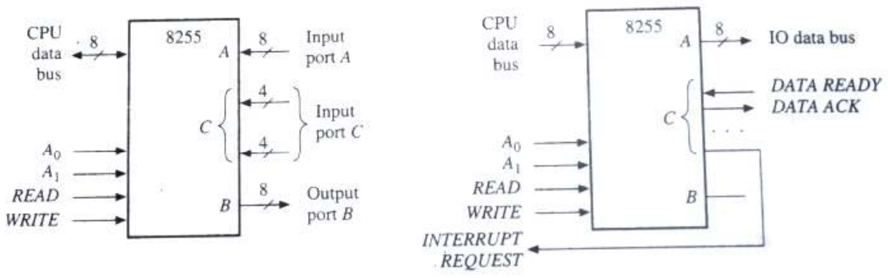


Figure: Two configurations of 8255

- The above figure shows one of the many possible configurations in the A, B and C lines are programmed as simple IO ports with no handshaking or interrupt capability.
- It is another configuration in that A port is programmed to be an input port with asynchronous timing signals generated by the C lines.
- The DATA READY line is used by the IO device to strobe a word into the buffer register at port A. Then 8255 automatically generates a response signal

on another C line, which can be sent to the IO device as an ACK signal if the IO device requires two-way control.

- The third C line generates an interrupt signal which is sent to the CPU to indicate the presence of data at IO port A.

5.10 DMA and Interrupts

5.10.1 DMA

- The programmed IO method has two limitations:
    1. The speed with which the CPU can test and service IO device limits IO data transfer rates.
    2. The time that the CPU spends for testing IO device status and executing IO data transfers can often be better spent on other tasks.
- DMA and interrupt circuits increase the speed of IO operation by eliminating most of the role played by the CPU.
- DMA and interrupt circuits use special control lines it has generic names as DMA REQUEST and INTERRUPT REQUEST, these two lines are connected to the IO devices to the CPU.
- Signals on these lines cause the CPU to suspend its current activities at appropriate breakpoints and attend to the DMA or interrupt request.
- These special request lines eliminate the need for the CPU to execute routines that determine IO device status.
- DMA also allows IO data transfers to take place without the execution of IO instructions by the CPU.
- DMA request by an IO device only requires the CPU to grant control of the memory (system) bus to the requesting device.
- The instruction cycle is the collection of number of CPU cycles and several cycles are required to use the system bus. During the instruction cycle there are five breakpoints when the CPU can respond to a DME request.
- When such arequest is reeceived by the CPU , it waits until the next breakpoint, release the system bus and signals the requesting IO device by activating a DMA ACKNOWLEDGE control line.

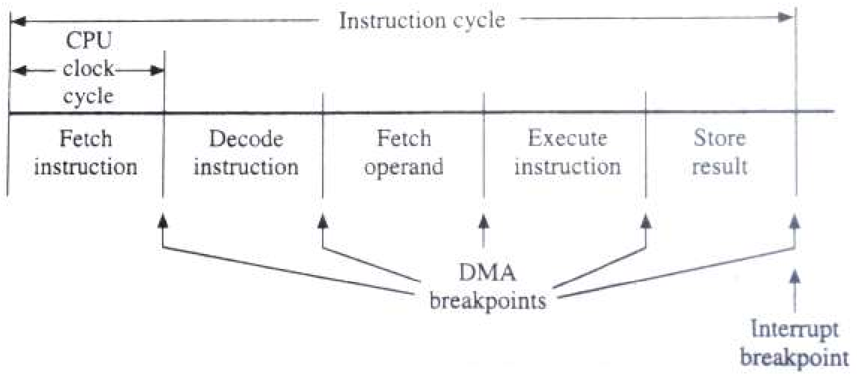


Figure: DMA and interrupt breakpoints during instruction processing.

- Interrupts are requested and acknowledged in same way as DMA requests. An interrupt is not a request for bus controls rather it asks the CPU to begin executing an interrupt service program.
- The interrupt program performs tasks such as initiating an IO operation or responding to an error encountered by the IO device.
- The CPU transfers control to interrupt programs as the same way how to control transfers to a subroutine. The CPU responds to interrupt programs only between instructions cycles.

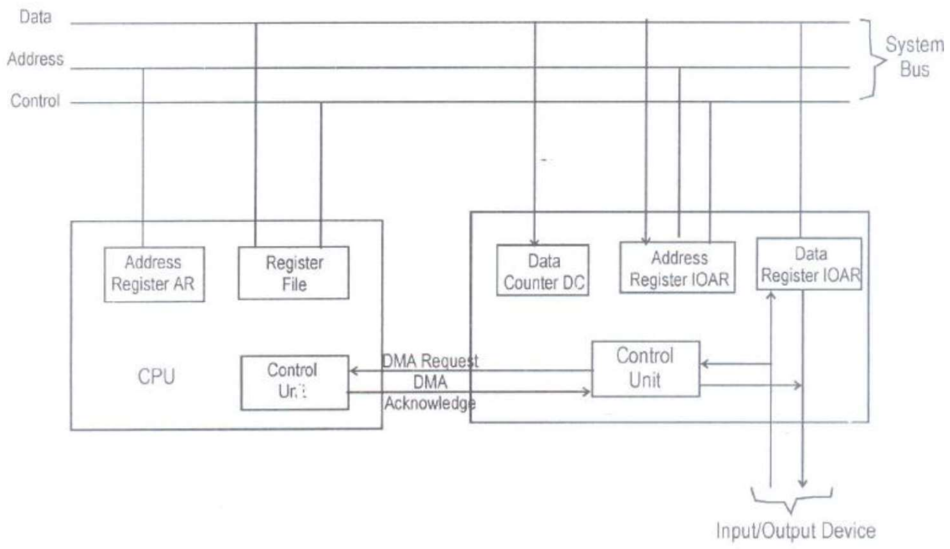


Figure: Block diagram for direct memory access(DMA)

- To transfer block of data directly between an external device and the main memory without continuous intervention by the processor a special control unit is provided. This approach is called Direct Memory Access or DMA.
- Three ways of DMA Data Transfer:
    - DMA block transfer.
    - Cycle Stealing.
    - Transparent DMA.
- DMA transfers are performed by a control circuit that is part of input/output device interface. This circuit is DMA controller.
- DMA controller can transfer data without intervention by the processor and its operation must be under the control of a program executed by the processor.
- DMA controller contains a data buffer register.IODR to store the data, as in the programmed IO, it also controls an address register.
- IOAR to store the address of next word to be transferred, it is automatically incremented or decremented after each transferred.
- Data counter DC stores the number of words that remain to be transferred. It is automatically decremented and tested for zero.

DMA block transfer:

- The DMA controller may be given limited access to the main memory to transfer of block without interruption. This is known as 'block' or 'burst mode'. Block DMA transfer supports the fastest IO data-transfer rates.

Cycle Stealing

- The DMA controller will steal memory cycles from the process; hence this technique is called cycle stealing.
- Cycle stealing allows the DMA controller to use the system bus to transfer one data word, after it returns the control of the bus to the CPU.
- Cycle stealing reduces the maximum IO transfer rate, but if also reduces the interference by the DMA controller in the CPU's memory access.
- It is possible to eliminate this interference completely by designing the DMA interface so that bus cycles are stolen only when the CPU is not actually using the system bus this is transparent DMA.

DMA Operation

- The CPU executes two IO instructions, which load the DMA registers IOAR and DC with their initial values.
    1. IOAR should contain the starting address of the memory block.
    2. DC should contain the number of words to be transferred.
- When the DMA controller is ready to transmit or receive data, it activates the DMA REQUEST line to the CPU,
    1. The CPU waits for the next DMA breakpoint.
    2. DMA controllers give up the control of data and address lines and activate DMA ACKNOWLEDGE.
    3. DMA requests from several DMA controllers are resolved by bus-priority control techniques.
- The DMA controller transfers data directly to or from main memory. After a word is transferred, IOAR and DC are updated.
- If DC has not yet reached zero but the IO device is not ready to send or receive the next batch of data then DMA controller will release the system bus to the CPU by deactivating the DMA REQUEST line.
    1. The CPU responds by deactivating DMA ACKNOWLEDGE and resuming control of the system bus.
- If DC is decremented to zero then the DMA controller again relinquishes 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.

5.10.2 Interrupt

- The word interrupt is used to cause a CPU to temporarily transfer control from its current program to another ptrogram.
- Interrupts are used to obtain the services of the CPU by IO devices and its significantly improve a computer's IO performance by giving Io devices direct , rapid access to the CPU , by freeing the CPU from the need to check the status of its IO devices.

- Various internal and external sources generate interrupts to the CPU. IO interrupts are external requests to the CPU to initiate or terminate an IO operation.
- Interrupts are also produced by hardware or software error detection circuits. It invokes error handling routines within the operating system.
- For instance if a power supply failure occurs then interrupt can be generated and interrupt handler designed to save critical data about the systems state. It is a kind of hardware interrupt.
- An instruction to divide by zero ,or to execute a privileged instruction when not in the privileged state are examples of software interrupt.
- An operation system also generates an interrupt a user program that has exceeded its alloted time.
- The basic method of interrupting the CPU is byadding a control line with the generic name INTERRUPT REQUEST that connects the interrupt source to the CPU.
- An interrupt indicator is stored in a CPU register and it will test the CPU peroidically at the end of every instruction cycle.
- Once interrupt occur the CPU executes a specific interrupt handling program. Each interrupt source requires execution of a different program, so the CPU musty determine the address of the interrupt program to be used.
- If two or more interrupt requests appear at the same time means we need to use priorities among the interrupts. In that which is having higher proirity that has to be handled first.
- The CPU responds to an interrupt request has the following steps:
    1. The CPU identifies the source of an interrupt, for example by polling IO devices.
    2. The CPU obtains the memory address of the required interrupt handler. This address can be provided by the interrupting device along with its interrupt request.
    3. The program counter PC and other CPU status information are saved as in a subroutine call.

4. The PC is loaded with the address of the interrupt handler. Execution starts until a return instruction isd encountered and it transfers control back to the interrupted program.

- Instruction sets usually include instructions to selectively disable or mask interrupt requests so CPU can ignore certain interrupts.
- Without these controls, IO device generated interrupts must require too much CPU's time and interface with the CPU's other tasks.
- When a high priority interrupt is serviced then all the lower priority interrupts must be disabled.
- An interrupt enable instruction must be subsequently executed to give the lower priority interrupts access to the CPU.

5.10.2.1 Interrupts Selection

- Selecting the interrupt among multiple interrupt is a major problem.
- Interrupt selection are made in different ways. One is single line and another is multiple line interrupt.
- The interrupt selection method requiring the least hardware is the single line method that is shown below.

Single-line interrupt system:

- 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.
- The connection sequence of this line determines the interrupt priority of each device. Alternatively the CPU can execute a program that polls each IO device interrupt requesting status information.
- Main advantage of polling is allowing the interrupt priority to be programmed.

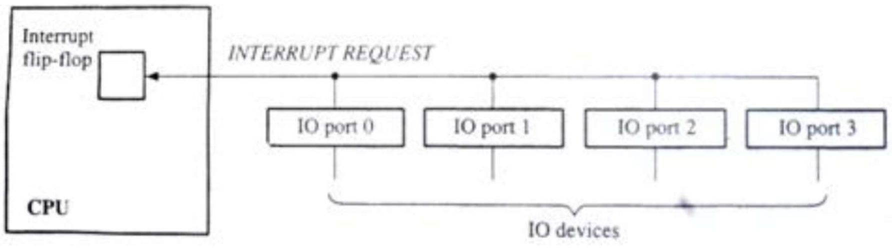


Figure: Single-line interrupt system

Multiple-line interrupt system:

- Multiple-line interrupt also called multilevel interrupt.
- Each interrupt request line is assigned a unique priority.
- The source of the interrupt is immediately known to the CPU so it eliminates the need for a hardware or software scan of the Io ports.
- Unless further measures are taken, the CPU still have to execute a program that fetches the address of the interrupt service program to be used. This step can be eliminated by another technique called vectoring of interrupts.

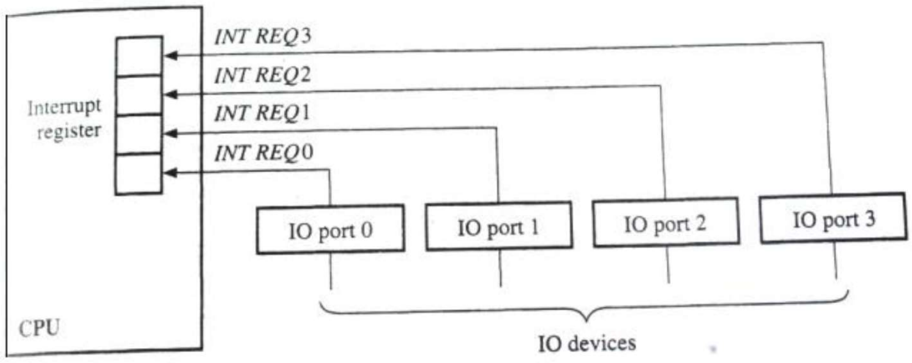


Figure: Multiple-line interrupt system

5.10.2.2 Vectored interrupts

- The most flexible response to the interrupts is to obtain when an interrupt request from a particular device causes a direct, hardware-implemented transition to the correct interrupt handling program.

- Once the interrupt occurs the interrupting device must supply the CPU with the starting address or interrupt vector of that program.
- The below 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 [log $_2$ K] bit address and it is inserted into PC.

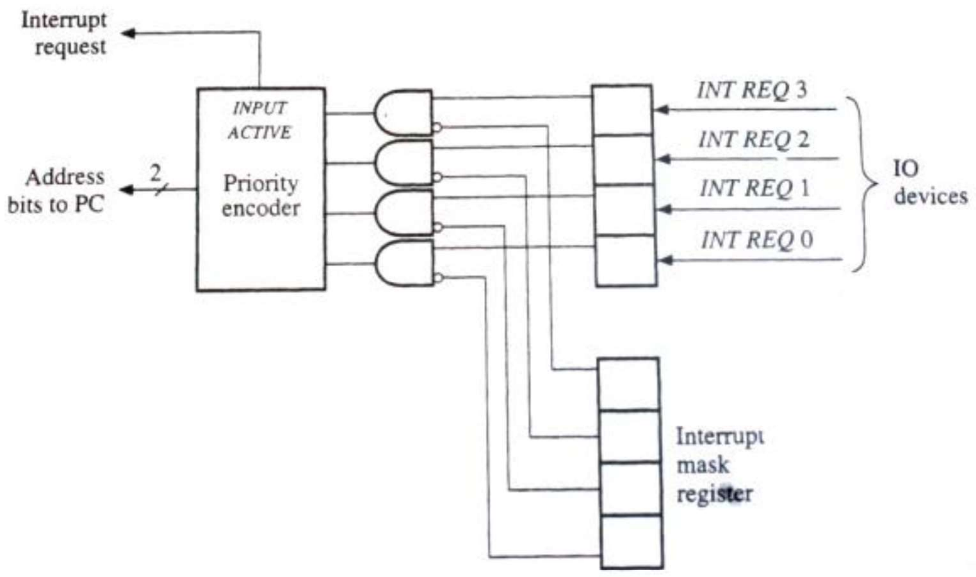


Figure: Vectored interrupt scheme

- Three devices are connected to four IO ports and assume when an interrupt request from IO port I is accepted then the 2 bit address I is generated by the priority encoder and inserted into the program counter PC.
- In below figure it shows the locations of the interrupt handlers and contents of these locations are the user assigned start addresses of the interrupt handling routines.

- The routines itself has an arbitrary length and that can be located anywhere in memory M.

Limitation:

- This method is a one-to-one correspondence between interrupt request lines and interrupt handlers.
- If IO device require the services of K distinct programs then it needs K distinct interrupt request lines.

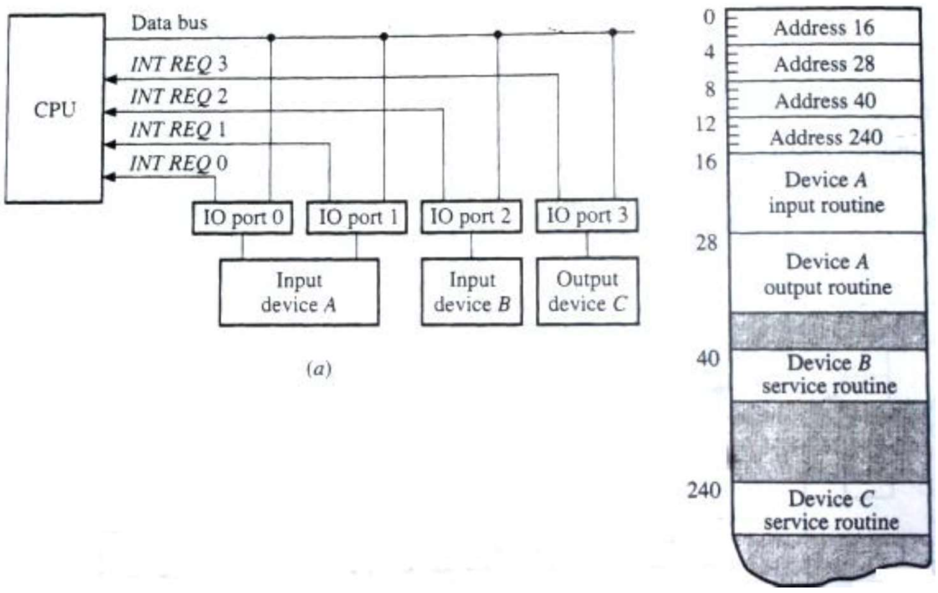


Figure: System with vectored I/O interrupts

- Each IO port can request the services of many different programs. So multiple interrupt request lines are used but each IO port has its own interrupt acknowledge line.
- When the CPU activates an acknowledge line in response to an interrupt request following task has to be taken.
    1. The IO port places the address of the desired interrupt handler on the main data bus.
    2. It transfers the address to the CPU, where it modifies the program counter.
    3. This approach requires the interrupting IO port to be able to generate at least partial memory addresses and act as a bus master.

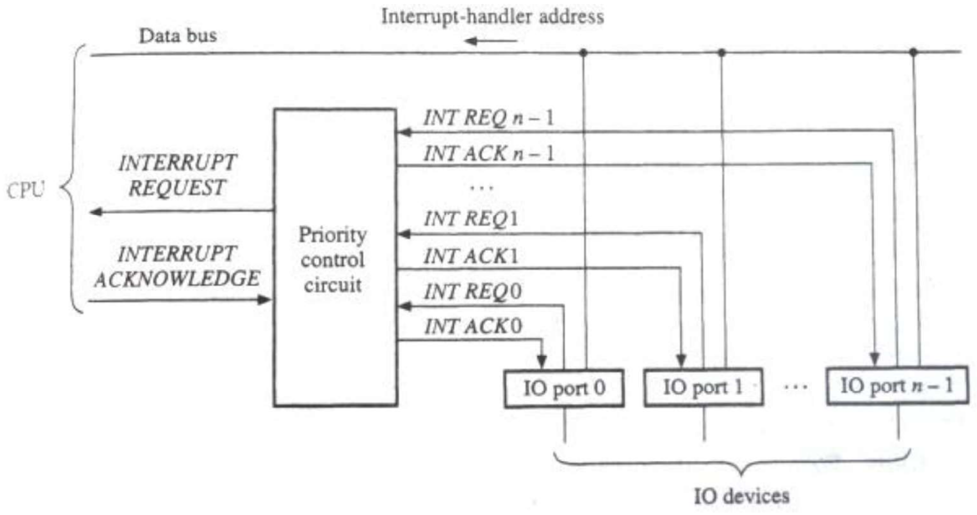


Figure: Another implementation of vectored interrupts

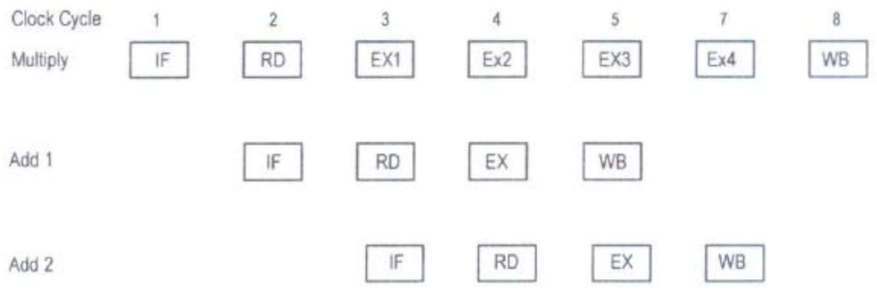
5.10.2.3 PCI Interrupts

- PCI local bus provides general support for interrupt handling. The PCI bus has four interrupt request lines. PCI is designed to easily interface with different microprocessor families, main memory and a very wide range of IO devices.
- Many of the PCI bus lines are optional so it can be attached to bus units with as few as 47 pins and as many as 100.It can support either 32 bit or 64 bit data transfers.
- Each PCI device is required to implement a set of registers called its configuration registers and whose format is defined in the PCI bus specification.
- When the system is first powered up, all such registers are accessed by the system control software to determine which IO devices are currently attached to the PCI bus and their basic communication requirements.
- A single function IO device with interrupt capability must use $\overline{\text{INTA}}$ line as interrupt request and multifunction IO devices can use all four lines.
- The $\text{INT}x$ interrupt request lines and the interrupt capability acknowledge command can implement the request acknowledge signal exchange at the time of interrupt transaction over the PCI bus.

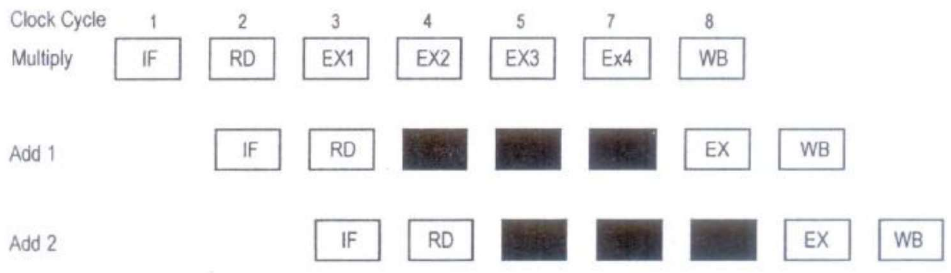
- Configuration register is a 16 bit wide, in that first 8 bit is called as Interrupt pin register and second 8-bit is called as Interrupt line register.
- Interrupt pin register is used to tell the system controller which IO devices used the interrupt request line.
- Interrupt line register specifies the system controller input line that is connected to $\overline{INTx}$ line, once the connection has made the routing of the interrupt request lines are programmable.
- The system controller uses the above two registers to find out the IO devices interrupt request priority and to access its interrupt vectors.
- The CR registers form a small address space that is separate from the main memory and IO address spaces.
- These small address space indicated by the existence of configuration read and configuration write in the command set specified for the PCI bus.

5.10.2.4 Pipeline interrupts

- When an interrupts occurs in a system, to control the CPU must be able to identify the interrupting instruction and the register contents needed for execution of ISR.
- This is not a problem when instructions are executed in sequence and only one instruction is executed at a time.
- However, in a pipeline processor several instructions are process concurrently.
- In this technique, it is possible for instructions to finish out of sequence. i.e., an instruction can finish sooner than another instruction that



Out-of order



In order

- Where three floating point instructions [MUL and 2 ADD] are being processed by one or more pipeline unit.
  7 - Clock cycle required for - Multiply and
  2 - Clock cycle are taken for add instructions.
- The first add Instruction (add1) is completed before the multiply instruction. This completion order is acceptable when there are no data dependencies.
- Registers affected by add1, instruction can be further modified by the multiply interrupt routine so that proper execution of add 1 interrupt subroutine may not be possible.
- In this situation, the CPU state is imprecise. To avoid such imprecise CPU state has to satisfy the conditions listed below.
  1) All instructions issued prior to I have completed their execution.
  2) No instruction has been issued after I.
  3) The PC contains I's address.

5.11 IOP Processors

- The IO process has the ability to execute instructions, which gives it faily complete control over IO operations. Like a CPU, an IOP is an instruction-set-producer but it has a more restricted instruction set.
- IOPs are primarily communication control unit designed to link IO devices to a computer.

- These are also known as peripheral processing units (PPUs). In system with programmed IO peripheral devices are controlled directly by the CPU.

5.11.1 I/O Instruction Types

- In a computer with an IOP, the CPU does not normally execute I/O data transfer instructions such instructions are contained in I/O program that are stored in DMA and are fetched by the IOP.
- The CPU does not execute a few I/O instructions that allow it to initiate and terminate the execution of I/O programs via the I/O and also used to test the status of the I/O system.
- There are three instructions that has been executed by the IOP such as
    1. Data transfer instruction
    2. Branch instruction
    3. I/O device control instruction
- The instructions executed by the IOP are called channel command words (CCW'S) that has following functions:

Data transfer instruction

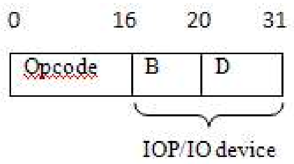
- These include input (read), output (write) and sense (read status).
- They cause the number of bytes in the data count field to be transferred between the specified memory region and the previously selected I/O device.

Branch instruction:

- These cause the IOP to fetch the next CCW from the specified memory address rather than from the next sequential location.

I/O device control instruction:

- These are transmitted to the I/O device and specify functions unusual to that device.



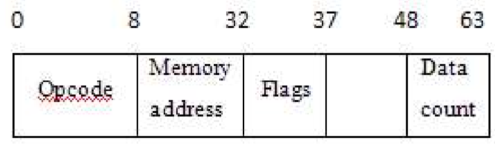


Figure: Formats of system 360 I/O instructions a) by a CPU b) by an IOP

- There are three major instructions of this type:
    1. START IO
    2. HALT IO
    3. TEST IO
- The START IO instruction initiates an IO operation and it provides the IOP it names with the memory address of the IO program to be executed by the IOP.
- The HALT IO instruction causes the IOP to terminate IO program execution, while test Io allows the CPU to determine the status of the named IO device and IOP.
- Status conditions of interest include available, busy, not operational and interrupt pending.
- Opcode of the data transfer instruction can be transmitted directly to the IO device as the command byte while the IO operation is being ser up.
- If the IO device requires more control information, it is supplied through an output data transfer.

- The flags field of the CCV modifies the operation specified by the opcode.
- For example, a program control flag PCI can be set to instruct the IOP to generate an IO interrupt and make the current IOP status available to the CPU.
- Another flag specifies command chaining, which means that the current CCW is followed by another CCW that is to be executed immediately.
- If this flag is not set, the IOP causes IO program executing the current CCW.
- Every CCW contain four fields are separated by commas, which correspond to the opcode, memory address, flags and data count field.

5.11.2 IOP Organization

- The structure of a system containing an IOP .The IO and CPU share access to a common memory M via the system bus.
- M stores separate programs for execution by the CPU and the IOP.
- It also contains a communication region IOCR for passing information in the form of messages between the two processors.
- The CPU can place there the parameter of an IO task, for example, the addresses of the IO programs to be executed and the identity of the IO devices to be used.
- The CPU and IOP also communicate with each other directly via control lines. Standard DMA and bus grant/acknowledge lines are used for arbitration of the system bus between two processors.
- By activating the ATTENTION line, the CPU can attract the IOP's attention and execute the START IO instruction.
- In response IOP begins execution of an IOP program whose specifications have been placed in the IOCR communication area.

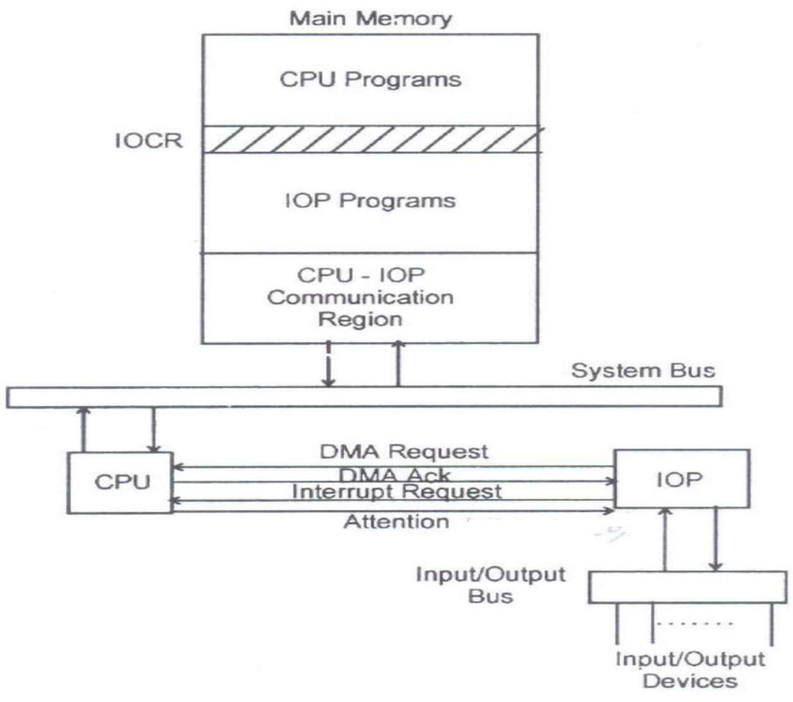


Figure: Computer containing an IOP

- INTERRUPT REQUEST line is activated, IOP attracts the CPU's attention, causing the CPU to execute an interrupt handler that typically response to the IOP by identifying a new IO program for the IOP to execute..

```
WAIT: If ATTENTION = 1 then
        begin
        Fetch Parameters from IOCR.
SETUP: Setup DMA control registers:
        Begin to Program execution.
        Send Command(s) to Input/Output device:
SEND: Transmit data word:
        If transmission error then go to EXIT
        If not end of IO program then go to SETUP.
EXIT: Place termination status in IOCR
        end:
        goto WAIT.
```

Example: The Intel 8089 IO processor

- IOP has two DMA channels, each of which can control an independent IO operation.
- 8089's DMA channels have their own program counters and other circuits necessary to execute an instruction set that is specialized toward IO operation in addition to the usual address and data count registers found in DMA controllers.
- Thus the 8089 can execute two unrelated IO programs concurrently and logically appears to the CPU like two independent IOPs. The DMA channels share a 20-bit ALU intended mainly for processing memory addresses.
- They also share bus interface circuits for communication with memory and IO devices.

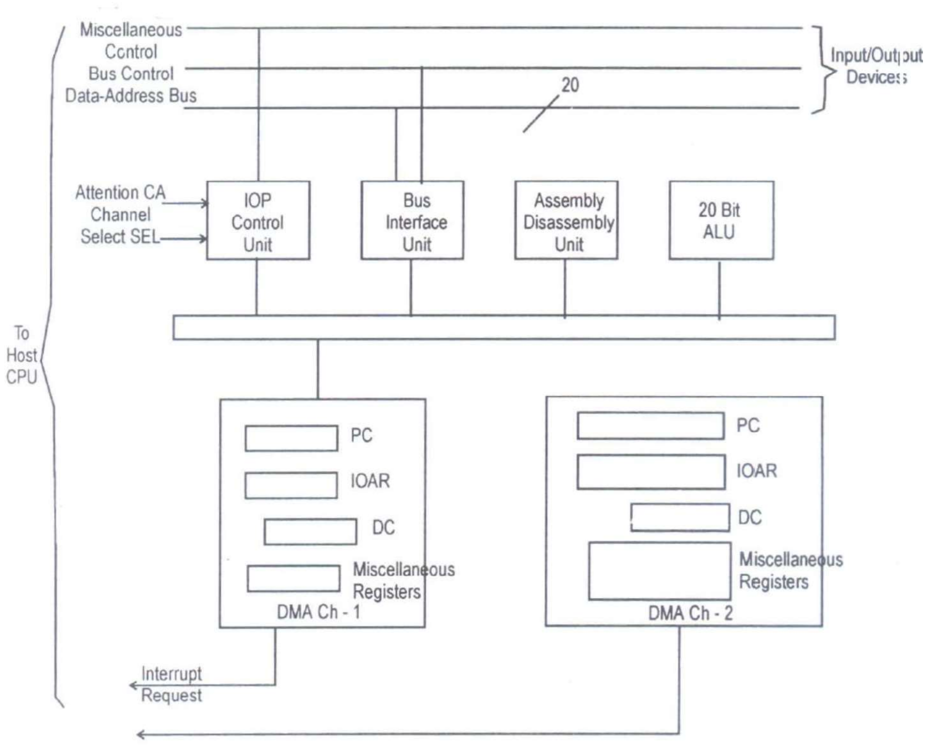


Figure: Structure of the Intel 8089 IOP

- Each DMA channel has an associated parameter block PB containing a pointer to the channel's current IO program that is a channel address word.
- PB also contains application specific input parameters for the IO program as well as output parameters for variables that the channel is to return to the CPU.
- These parameters are used to identify the IO buffer regions in main memory, IO devices names, data addresses in secondary memory devices and so on.
- The location of the two PBs is stored in a channel control block CB, which is created by the CPU when the system is powered up or reset.
- CB stores status information and a command from the CPU for each channel. These 1 byte commands fill essentially the same role as the START, TEST and HALT IO instructions of the system /360-370 series.

- The CPU also uses them to enable, disable or deactivates the channels interrupt request line. The CPU supervises each IOP channel by writing into its PB region and into its portion of CB.

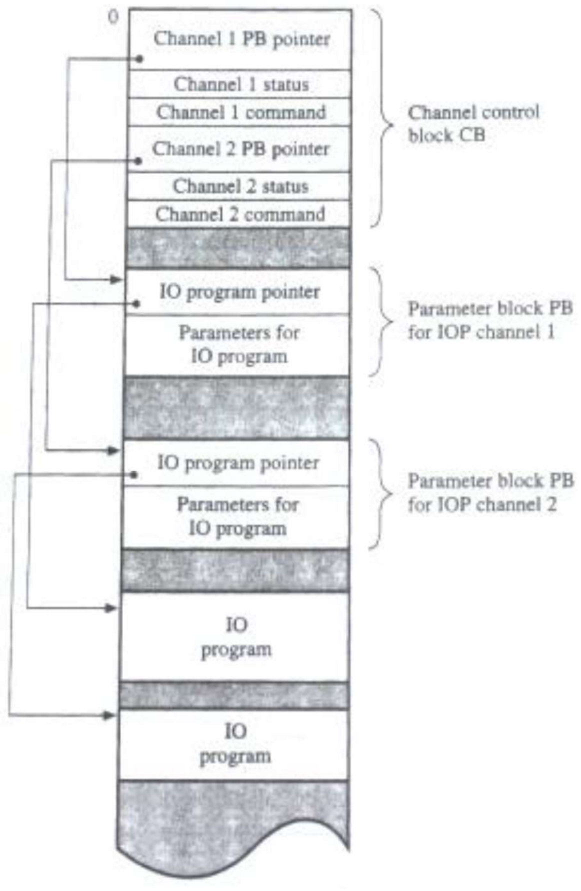


Figure: Memory organization for 8089 IOP