COMPUTER ORGANIZATION

Presented by
Dr. Lakshmi N S
Asst. Professor
IIIT Kottayam

Syllabus

Indian Institute of Information Technology Kottayam

- Memory hierarchy: caches, cache performance, virtual memory, common framework for memory hierarchies Input/output: I/O performance measures, types and characteristics of I/O devices, buses, interfaces in I/O devices, design of an I/O system, parallelism and I/O. Introduction to multicores and multiprocessors.

Fundamental principle in memory access

- 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:
- Temporal locality (locality in time): if an item is referenced, 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): if an item is referenced, items whose
- addresses are close by will tend to be referenced soon.

Memory hierarchy

- Memory hierarchy: A structure that uses multiple levels of memories; as the distance from the processor increases, the size of the memories and the access time both increase.
- Memory hierarchy consists of multiple levels of memory with different speeds and sizes.
- The faster memories are more expensive per bit than the slower memories and thus are smaller.

Memory characteristics

- there is a trade-off among the three key characteristics of memory: namely,
- capacity,
- access time, and
- cost
- A variety of technologies are used to implement memory systems, and across this spectrum of technologies, the following relationships hold:
- Faster access time, greater cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access time

Memory technologies

Indian Institute of Information Technology Kottayam

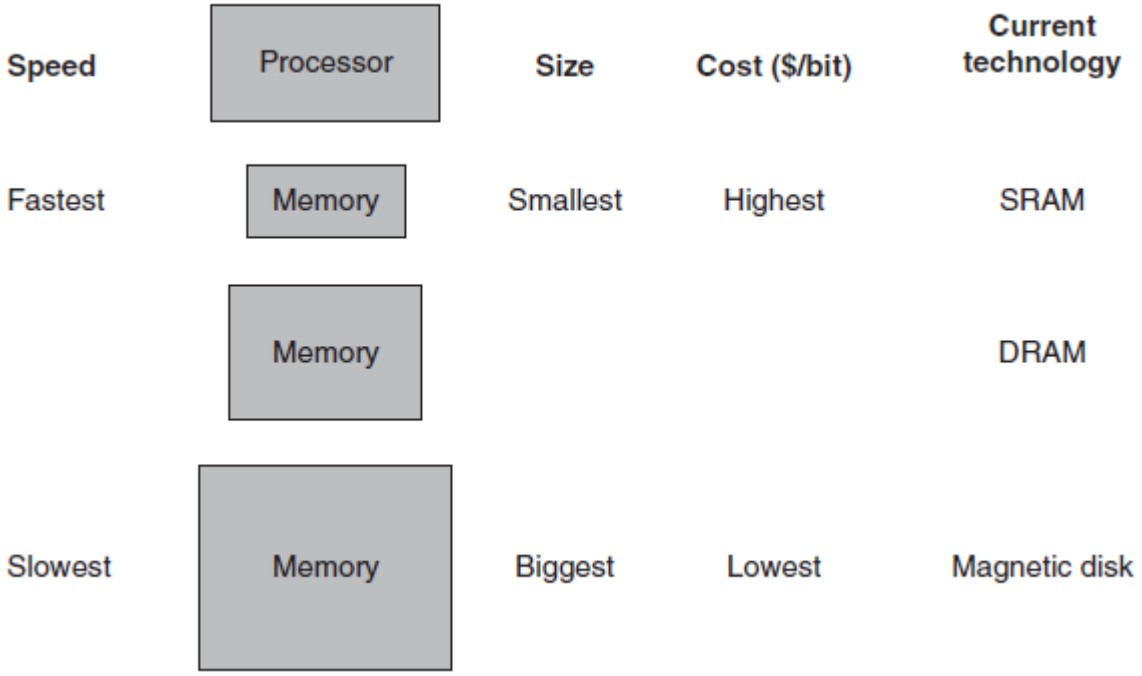
- There are three primary technologies used in building memory hierarchies.
- SRAM
- DRAM
- Magnetic Disk

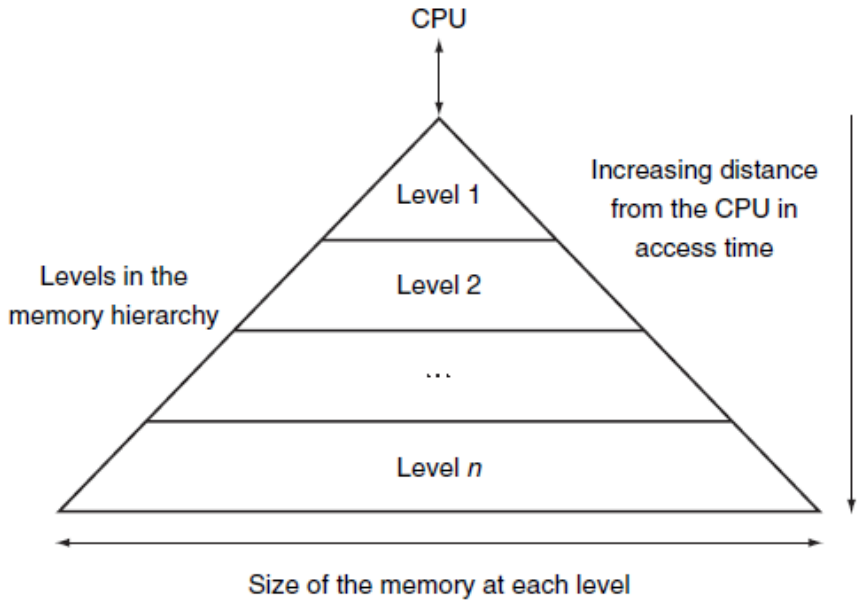
Memory technology	Typical access time	$ per GB in 2008
SRAM	0.5–2.5 ns	$2000–$5000
DRAM	50–70 ns	$20–$75
Magnetic disk	5,000,000–20,000,000 ns	$0.20–$2

Memory technologies

- 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, and DRAMs thus have larger capacity for the same amount of silicon; the speed difference arises from access mechanism

Memory hierarchy





as the distance from the processor increases, so does the size.

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

Memory hierarchy

- A memory hierarchy can consist of multiple levels, but data is copied between only two adjacent levels at a time,
- The upper level—the one closer to the processor—is smaller and faster than the lower level, since the upper level uses technology that is more expensive

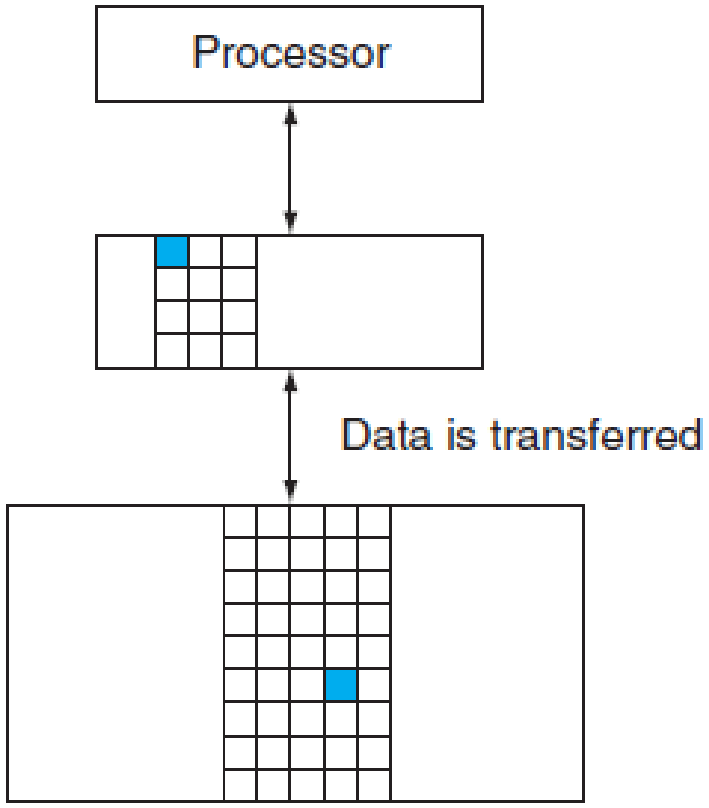
Memory hierarchy

- Memory hierarchies take advantage of temporal locality by keeping more recently accessed data items closer to the processor.
- Memory hierarchies take advantage of spatial locality by moving blocks consisting of multiple contiguous words in memory to upper levels of the hierarchy.

Memory hierarchy

- 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

Memory access



Hits
miss
Hit ratio
Miss ratio
Hit time
Miss penalty

Within each level, the unit of information that is present or not is called a block or a line. Usually we transfer an entire block when we copy something between levels

Performance metrics in memory hierarchy

- Hit rate -The fraction of memory accesses found in a level of the memory hierarchy.
- Miss rate -The fraction of memory accesses not found in a level of the memory hierarchy.
- Hit time -The time required to access a level of the memory hierarchy, including the time needed to determine whether the access is a hit or a miss.
- Miss Penalty : It can be defined as the addition clock cycles to service the miss, the extra time needed to carry the favored information into cache from main memory in case of miss in cache.
- Average memory access time = Hit Time + (Miss Rate X Miss Penalty)

Cache performance

- Hit Ratio (h)(hit rate) = Number of Hits / Total CPU references to CACHE
  = Number of hits / (Number of Hits + Number of Misses)

- Miss ratio(miss rate) =Number of misses / Total CPU references to cache
- = Number of misses/ (Number of hits + Number of misses)
- Miss Ratio = 1 – hit ratio(h)
- Average Memory Access Time (AMAT) :
- Average memory access time = Hit Time + (Miss Rate X Miss Penalty)
  - = h X tc + (1- h) X (tc + tm)
  - tc, h and tm denote the cache access time, hit ratio in cache and main memory access time respectively.

Cache performance metrics

- Hit ratio (hit rate)= hits/(hits+miss)
- Miss ratio (Miss rate)=Miss/(hits +miss)= 1-h
- Average memory access time= Average memory access time = Hit Time + (Miss Rate X Miss Penalty)
- This formula can be applied in multi level caches
- Suppose there are two levels of cache
- $AMAT = Hit\ time_{L1} + Miss\ ratio_{L1}(AMAT_{L2})$
- where $AMAT_{L2} = Hittime_{L2} + Miss\ ratio_{L2} * Miss\ penalty_{L2}$
- $AMAT = h_1 t_{C1} + (1-h_1)[(h_2 t_{c2} + (1-h_2)(t_{c2}+t_m)]$
- $= h_1 t_{C1} + (1-h_1)(h_2 t_{c2}) + (1-h_1)(1-h_2)(t_{c2}+t_m)$

Hit time L1

Miss ratio L1

Hit time L2

Miss ratio L1

Miss ratio L2

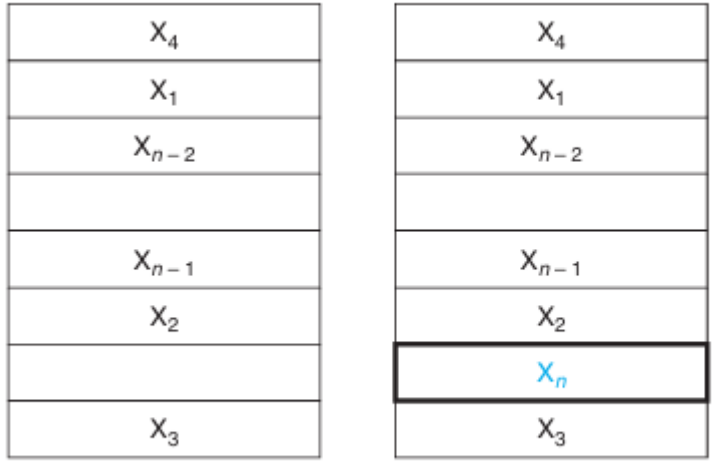
Miss penaltyL2

In high performance processors two levels of caches are used-L1 on processor chip L2 outside processor on memory

Basics of Cache

- Cache: level of hierarchy closest to processor
- Requesting Xn generates a miss and the word Xn will be brought from main memory to cache
- cache, the term is also used to refer to any storage managed to take advantage of locality of access.

How do we know that a data item is in cache?
- If so, How to find it?



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

Basics of Cache

- There are three different types of mapping used for the purpose of cache memory which is as follows:

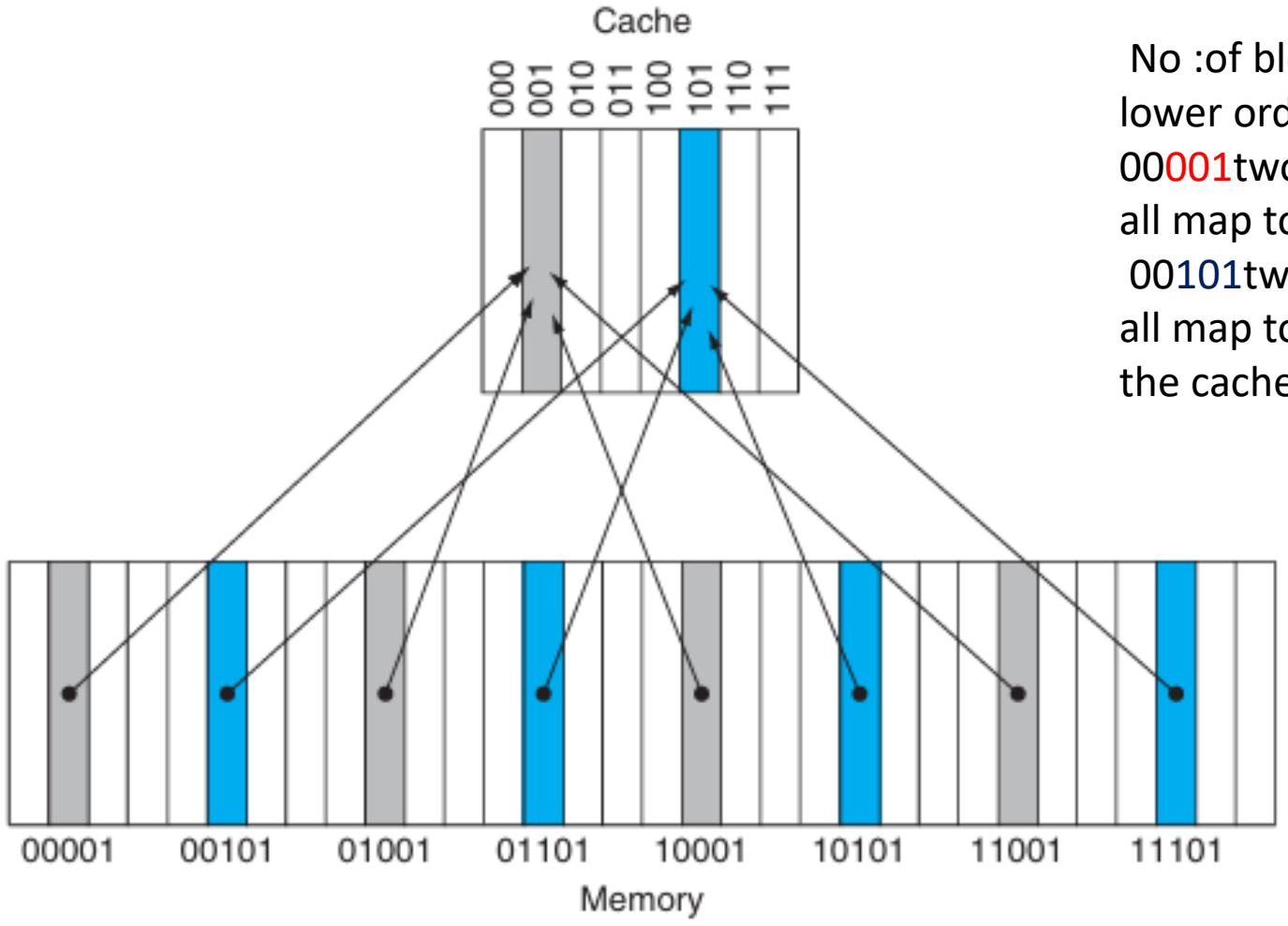
- Direct Mapping
- Associative Mapping
- Set-Associative Mapping

Direct Mapped cache and cache access

- Direct-mapped cache A cache structure in which each memory location is mapped to exactly one location in the cache
- All direct mapped caches use this mapping to find a block:

  (Block address) modulo (Number of blocks in the cache)
- If the number of entries in the cache is a power of 2, then modulo can be computed simply by using the lowerorder $\log_2$ (cache size in blocks) bits of the address.

Direct Mapped cache and cache access



No :of blocks in cache be 8
lower order $\log_2(8)$ bits of block address
00001two, 01001two, 10001two, and 11001two
all map to entry 001two of cache
00101two, 01101two, 10101two, and 11101two
all map to entry 101two of
the cache.

direct-mapped cache with eight entries showing the addresses of memory words between 0 and 31 that map to the same cache locations

Direct Mapped cache and cache access

Indian Institute of Information Technology Kottayam

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (5.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (5.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (5.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (5.6f)	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

Direct Mapped cache and cache access

Indian Institute of Information Technology Kottayam

Decimal address of reference
22
26
22
26
16
3
16
18
16

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

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

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

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

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

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

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

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

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

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

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

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

Direct Mapped cache and cache access

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	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

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

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	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	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	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	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

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

The cache is initially empty, with all valid bits (V entry in cache) turned off (N). The processor requests the following addresses: 10110two (miss), 11010two (miss), 10110two (hit), 11010two (hit), 10000two (miss), 00011two (miss), 10000two (hit), 10010two (miss), and 10000two (hit).

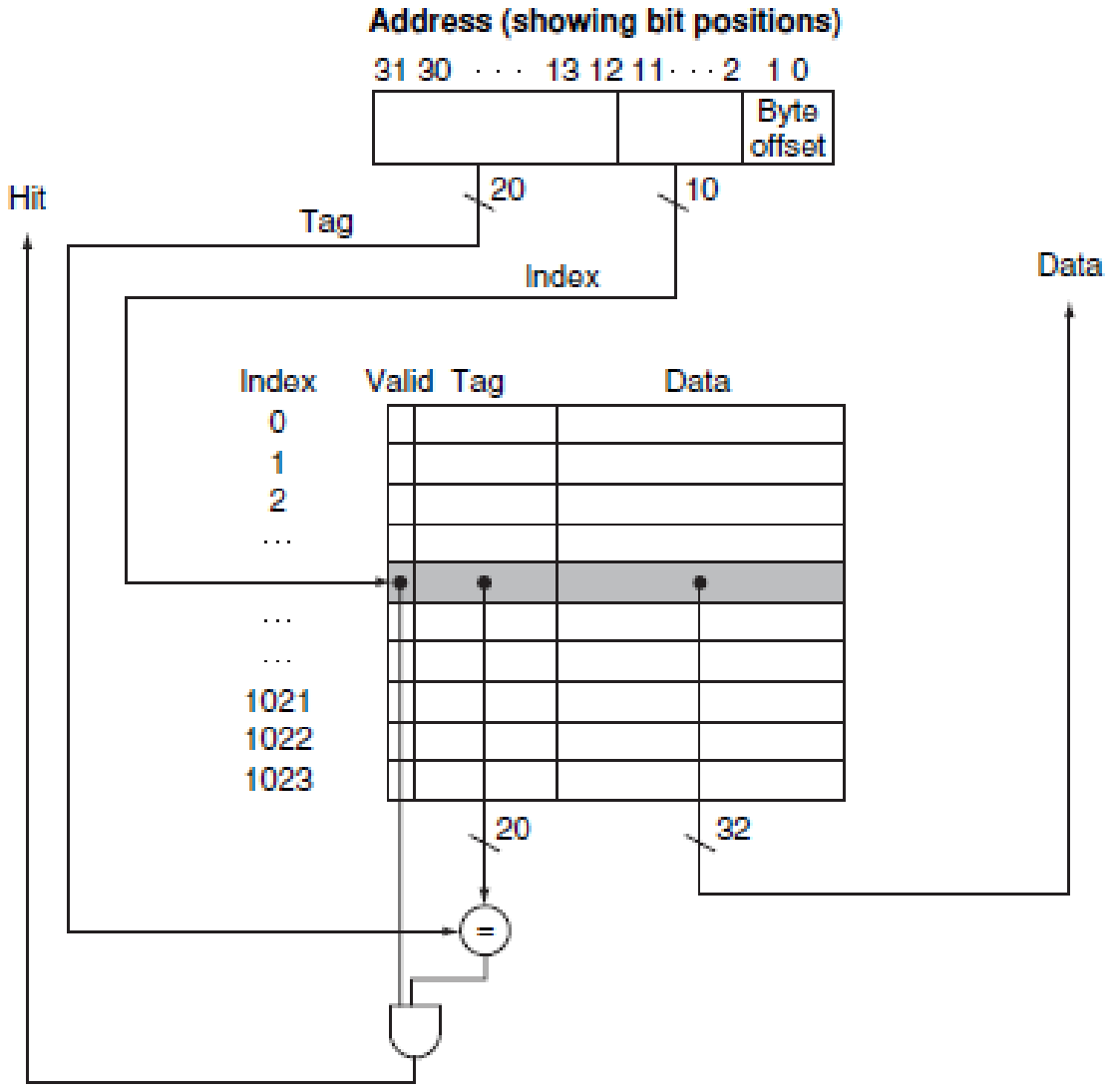
The figures show the cache contents after each miss in the sequence has been handled. When address 10010two (18) is referenced, the entry for address 11010two (26) must be replaced, and a reference to 11010two will cause a subsequent miss. The tag field will contain only the upper portion of the address.

Direct Mapped cache and cache access

Indian Institute of Information Technology Kottayam

- The tags contain the address information required to identify whether a word in the cache corresponds to the requested word.
- The tag needs only to contain the upper portion of the address, corresponding to the bits that are not used as an index into the cache.
- For example, we need only have the upper 2 of the 5 address bits in the tag, since the lower 3bit index field of the address selects the block
- valid bit -A field in the tables of a memory hierarchy that indicates that the associated block in the hierarchy contains valid data.
- 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.

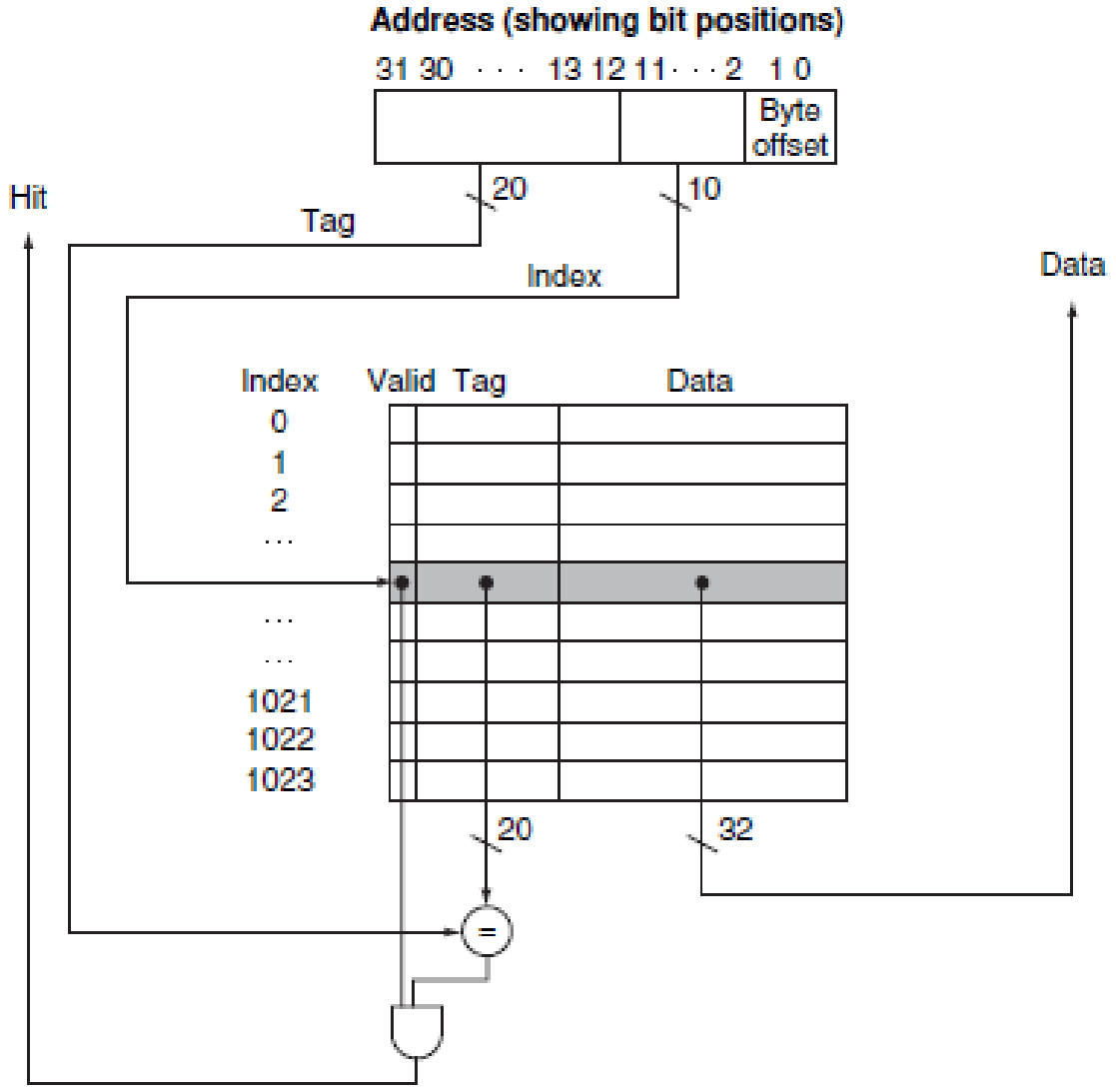
Direct Mapped cache and cache access



In this example Processor requests are each one word and the blocks also consist of a single word

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

Direct Mapped cache and cache access



Here blocks consist of a single word
For this cache, the lower portion of the address is used to select a cache entry consisting of a data word and a tag.
This cache holds 1024 words or 4 KB.
We assume 32-bit addresses.
The tag from the cache is compared against the upper portion of the address to determine whether the entry in the cache corresponds to the requested address.
Because the cache has 2^{10} (or 1024) words and a block size of one word, 10 bits are used to index the cache, leaving 32 - 10 - 2 = 20 bits to be compared against the tag.
If the tag and upper 20 bits of the address are equal and the valid bit is on, then the request hits in the cache, and the word is supplied to the processor.
Otherwise, a miss occurs

Direct Mapped cache and cache access

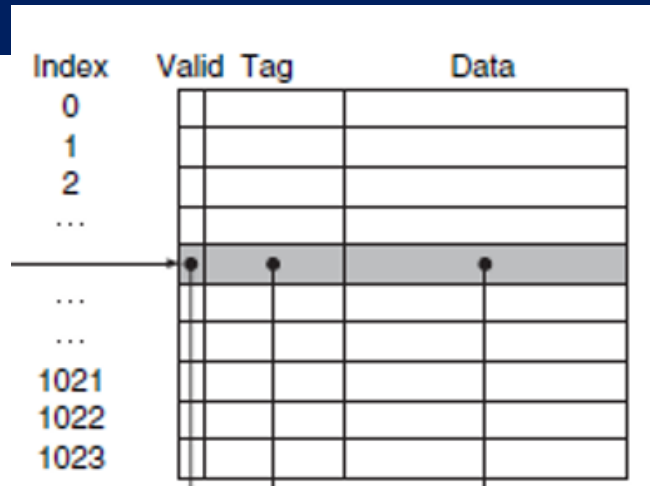
- The index of a cache block, together with the tag contents of that block, uniquely specifies the memory address of the word contained in the cache block.
- Because the index field is used as an address to reference the cache, and because an n-bit field has 2^n values, the total number of entries in a direct-mapped cache must be a power of 2.
- In the MIPS architecture, since words are aligned to multiples of four bytes, the least significant two bits of every address specify a byte within a word. Hence, the least significant two bits are ignored when selecting a word in the block.
- The total number of bits needed for a cache is a function of the cache size and the address size, because the cache includes both the storage for the data and the tags.
- The size of the block above was one word, but normally it is several.

Direct Mapped cache and cache access

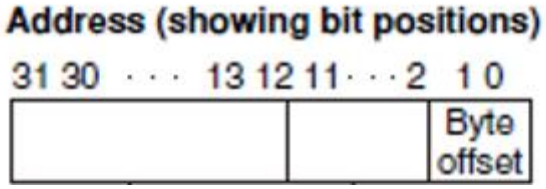
- The index of a cache block, together with the tag contents of that block, uniquely specifies the memory address of the word contained in the cache block.
- Because the index field is used as an address to reference the cache, and because an n-bit field has 2^n values, the total number of entries in a direct-mapped cache must be a power of 2.
- In the MIPS architecture, since words are aligned to multiples of four bytes, the least significant two bits of every address specify a byte within a word. Hence, the least significant two bits are ignored when selecting a word in the block.
- The total number of bits needed for a cache is a function of the cache size and the address size, because the cache includes both the storage for the data and the tags.
- The size of the block above was one word, but normally it is several.

Direct Mapped cache and cache access

Indian Institute of Information Technology Kottayam



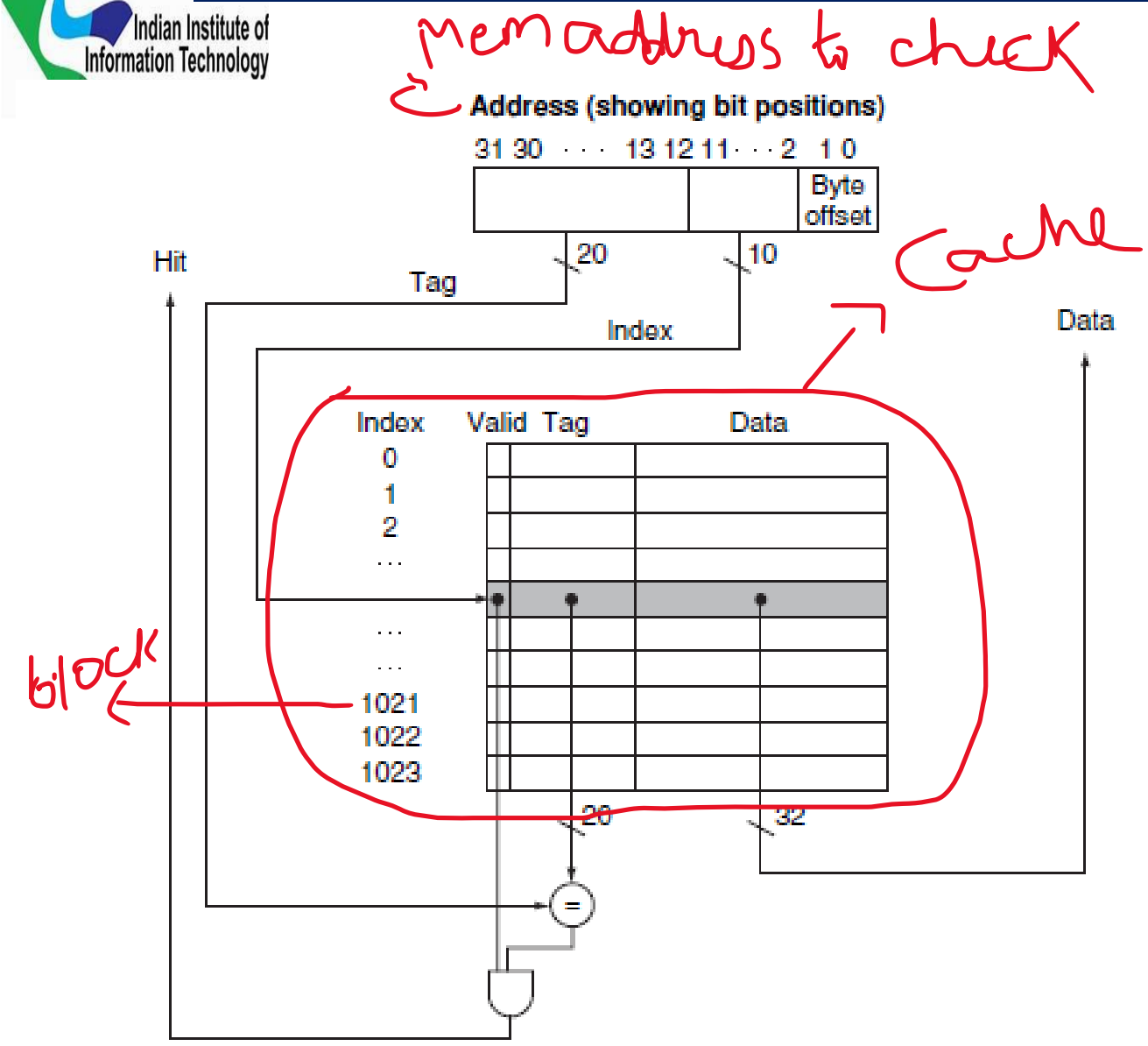
Under the following conditions



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

- the size of the tag field is 32 - (n + m + 2).
- The total number of bits in a direct-mapped cache is
- $2^n \times$ (block size + tag size + valid field size).
- Since the block size is 2^m words (2^{m+5} bits), and we need 1 bit for the valid field, the number of bits in such a cache is
- $2^n \times (2^m \times 32 + (32 - n - m - 2) + 1) = 2^n \times (2^m \times 32 + 31 - n - m)$.

Direct Mapped cache and cache access



Here blocks consist of a single word
This cache holds 1024 words or 4 KB.

Given a memory address.
To check if it exists in cache.

That memory address exists if the tag (tag of cache block entry)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

Each block contain 1 or more words (In this example only 1 word).
So word address is given by tag and index bits

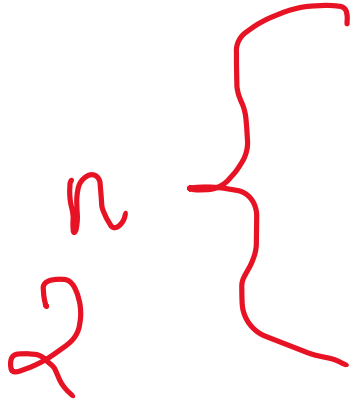
Byte offset – LOWER order 2 Bits specify bytes within a word- 1 word means 4 bytes – to distinguish Adrres of each byte from one another you need 2 bits (00-1 st byte 01-second byte 10-third byte 11- fourth byte

Direct Mapped cache and cache access

- 1kbyte= 1000 BYTE 10^3 (DECIMAL)
- 1kbYTE- 1024 BYTES= 2^{10} (IN BASE2)
- Memory size is expressed in Kbytes,Mbytes etc
- convert words into byte
- If a CACHE CAN STORE 1 WORD IN EACH BLOCK AND IF STORAGE CAPACITY IS 4 kiB memory
- 4kiB= 1024x 4 bytes
- 1 word=4 bytes
- 1024 words can be stored.
- Each block holds 1 word
- Hence 1024 block will be there- 10 bits are required for index

Problem-Bits in a cache

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



index	valid	Tag	data			

Problem-Bits in a cache

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

Direct Mapped cache and cache access

- Each block holds 4 words
- So express given mem capacity in words
- 16 kiB=16 *1024=16384 bytes
- 4bytes=1 word
- Required no:of words to be stored=16384/4byte=4096= 2^{12} words
- Block size =4 words
- No:of blocks in cache= $2^{12}/2^{2}=2^{10}$ (4096/4)
- n=10
- total cache size is 2^{n} × (block size + tag size + valid field size).
- Block size= 2^{m} words= 4 words. (m=2)
- tagsize-=(32-(n+m+2))
- Valid field size = 1
- 2^{10} × (4 × 32 + (32 - 10 - 2 - 2) + 1) = 2^{10} × 147 = 147 Kbits
- 147/8=18.4Kbytes

Direct Mapped cache and cache access

- We know that 16 KB is 4K (2^{12}) words.
- With a block size of 4 words (2^2), there are 1024 (2^{10}) blocks.
- Block size=4 words=4x32 bits; index (n) =10 as there are 2^{10} blocks
- Each block has 4 × 32 or 128 bits of data plus
- a tag, which is 32 - (n + m + 2)=32 – 10 – 2 – 2 bits, plus
- a valid bit.
- Thus, the total cache size is 2^n × (block size + tag size + valid field size).
- $2^{10} \times (4 \times 32 + (32 - 10 - 2 - 2) + 1) = 2^{10} \times 147 = 147$ Kbits
- or 18.4 KB for a 16 KB cache.

Problem-Mapping an address to multiword cache

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

Problem-Mapping an address to multiword cache

Indian Institute of Information Technology Kottayam

- Consider a cache with 64 blocks and a block size of 16 bytes. To what block number does byte address 1200 map?
- block is given by (Block address) modulo (Number of blocks in the cache)
- the address of the block is Byte address/Bytes per block
- with 16 bytes per block, byte address 1200 is block address
- 1200/16 = 75
- which maps to cache block number (75 modulo 64) = 11
- this block maps all addresses between 1200 and 1215.

1200 16 byte is block size
1215-16 byte
Byte address= 1200
[1200/bytes per block] *Bytes per block =1200

Block address is the block containing all addresses between

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

And

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

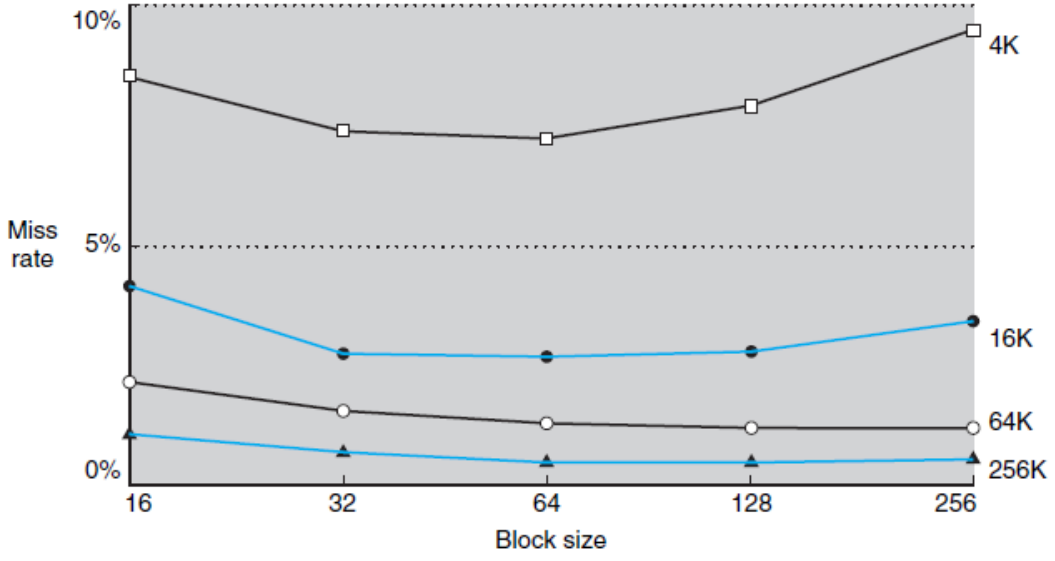
Direct Mapped cache Performance

Indian Institute of Information Technology Kottayam

- Larger blocks exploit spatial locality to lower miss rates.
- increasing the block size usually decreases the miss rate.
- The miss rate may go up eventually if the block size becomes a significant fraction of the cache size, because the number of blocks that can be held in the cache will become small, and there will be a great deal of competition for those blocks.
- As a result, a block will be bumped out of the cache before many of its words are accessed. Stated alternatively, spatial locality among the words in a block decreases with a very large block; consequently, the benefits in the miss rate become smaller.

Direct Mapped cache and cache access

- with just increasing the block size 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
- the increase in the miss penalty overwhelms the decrease in the miss rate for blocks that are too large, and cache performance thus decreases.



Handling cache miss on read

- 1. Send the original PC value (current PC – 4) to the memory.
- 2. Instruct main memory to perform a read and wait for the memory to complete its access.
- 3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.
- 4. Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.

Handling writes

- Problem upon write -cache and main memory would have different value
- cache and memory are said to be inconsistent.
- Solution
- Write-through
  - A scheme in which writes always update both the cache and the next lower level of the memory hierarchy, ensuring that data is always consistent between the two.
- Write back
  - 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.

Handling writes

- Problem upon write -cache and main memory would have different value
- Problem: Suppose on a store instruction, we wrote the data into only the data cache (without changing main memory); then, after the write into the cache, memory would have a different value from that in the cache.In such a case, the cache and memory are said to be inconsistent.
- Solution
- Write-through
  - A scheme in which writes always update both the cache and the next lower level of the memory hierarchy, ensuring that data is always consistent between the two.
- Write back
  - When a write occurs, the new value is written only to the block in the cache.
  - The modified block is written to the lower level of the hierarchy when it is replaced.

Write through scheme

- 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.
- For example, suppose 10% of the instructions are stores.
- If the CPI without cache misses was 1.0, spending 100 extra cycles on every write would lead to a
- CPI of 1.0 + 100 × 10% = 11,
- Reducing performance by more than a factor of 10.
- Hence a write miss on write through scheme reduces performance as it takes more clock cycles to access main memory on every write
- Solution
- Use of write buffer

Write buffer

- write buffer A queue that holds data while the data is waiting to be written to memory.
- write buffer stores the data while it is waiting to be written to memory. After writing the data into the cache and into the write buffer, the processor can continue execution.
- When a write to main memory completes, the entry in the write buffer is freed. If the write buffer is full when the processor reaches a write, the processor must stall until there is an empty position in the write buffer.

Write back

Indian Institute of Information Technology Kottayam

- The alternative to a write-through scheme is a scheme called write-back or copy back.
- In a write-back scheme, when a write occurs, the new value is written only to the block in the cache.
- The modified block is written to the lower level of the hierarchy when it is replaced.
- Write-back schemes can improve performance, 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

Cache performance improvement-
Use of split cache- Increasing cache bandwidth

- Combined cache-same cache for instruction and data
- Split cache-separate instruction and data cache
- Many processors use a split instruction and data cache to increase cache *bandwidth*.
- ■ Total cache size: 32 KB
- ■■ Split cache effective miss rate: 3.24%
- ■■ Combined cache miss rate: 3.18%
- The miss rate of the split cache is only slightly worse.
- The advantage of doubling the cache bandwidth, by supporting both an instruction and data access simultaneously, easily overcomes the disadvantage of a slightly increased miss rate.

Cache performance measurement

- CPU time can be divided into
- clock cycles that the CPU spends executing the program and the
  - Normally, we assume that the costs of cache accesses that are hits are part of the normal CPU execution cycles
- clock cycles that the CPU spends waiting for the memory system.
- CPU time = (CPU execution clock cycles + Memory-stall clock cycles) × Clock cycle time
- The memory-stall clock cycles come primarily from cache misses,
- Memory-stall clock cycles can be defined as the sum of the stall cycles coming from reads plus those coming from writes:
- Memory-stall clock cycles = Read-stall cycles + Write-stall cycles

Cache performance measurement

- 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:
- Read-stall cycles = (Reads /Program)× Read miss rate × Read miss penalty
- Writes are more complicated.
- For a write-through scheme, we have two sources of stalls:
  - write misses, which usually require that we fetch the block before continuing the write
  - 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:
- Write-stall cycles = (Writes /Program) × Write miss rate × Write miss penalty)
  + Write buffer stalls

In most write-through cache organizations, the read and write miss penalties are the same (the time to fetch the block from memory).

If we assume that the write buffer stalls are negligible, we can combine the reads and writes by using a single miss rate and the miss penalty:

Cache performance measurement

- Memory-stall clock cycles =(Memory access/Program) × (Misses/Instruction) × Miss penalty
- Memory-stall clock cycles = (Instructions/Program) × (Misses/Instruction) × Miss penalty

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

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

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

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

- 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:
- AMAT = Time for a hit + Miss rate × Miss penalty

Calculate Average memory access time

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

Calculate Average memory access time

- AMAT = Time for a hit + Miss rate × Miss penalty
- = 1 + 0.05 × 20
- = 2 clock cycles
- or 2 ns.

Calculate cache performance Homework

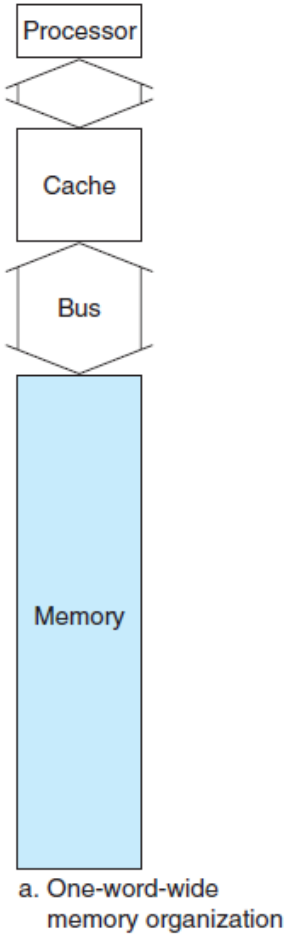
Indian Institute of Information Technology Kottayam

REVISE PERFORMANCE METRICS IN FIRST UNIT

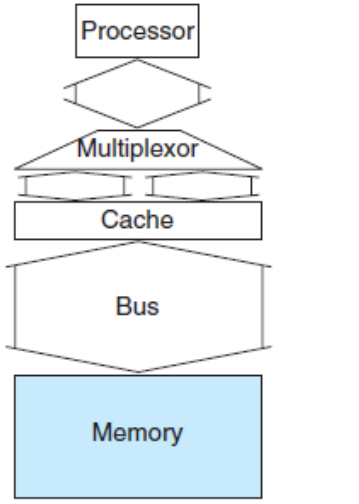
- Assume the miss rate of an instruction cache is 2% and the miss rate of the data cache is 4%. If a processor has a CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never missed. Assume the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

- Instruction miss cycles =
- Data miss cycles =
- total number of memory-stall cycles=
- Total CPI including memory stalls=
- CPU time with stalls / CPU time with perfect cache =

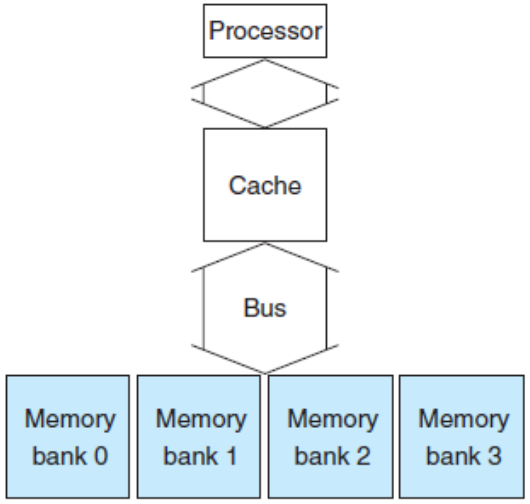
Improving performance – Increase main memory bandwidth



a. One-word-wide memory organization



b. Wider memory organization



c. Interleaved memory organization

Double Data Rate (DDR) DRAMs.
The name means data transfers on both the leading and falling edge of the clock,
thereby getting twice as much bandwidth

Improving performance –Increase main memory bandwidth

we can reduce the miss penalty if we increase the bandwidth from the memory to the cache. This reduction allows larger block sizes to be used while still maintaining a low miss penalty

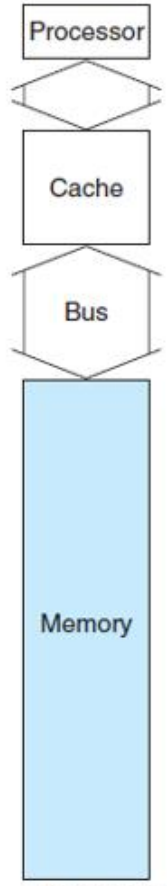
- MAKING MAIN MEMORY WIDER: Increasing the width of the memory and the bus will increase the memory bandwidth proportionally, decreasing both the access time and transfer time portions of the miss penalty.
- Interleaving-Instead of making the entire path between the memory and cache wider, the memory chips can be organized in banks to read or write multiple words in one access time rather than reading or writing a single word each time.
- Each bank could be one word wide so that the width of the bus and the cache need not change, but sending an address to several banks permits them all to read simultaneously.

Improving performance –Increase main memory bandwidth

- we can reduce the miss penalty if we increase the bandwidth from the memory to the cache. This reduction allows larger block sizes to be used while still maintaining a low miss penalty
- 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.

Improving performance –Increase main memory bandwidth

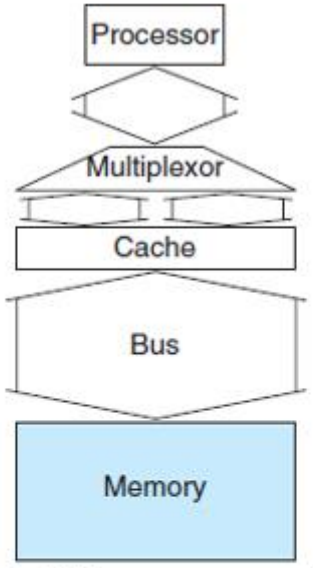
- Bandwidth (Number of bytes transferred per bus clock cycle)for 1 word wide Memory and bus
- Example memory bus clock cycle to send the address
- ■■ 15 memory bus clock cycles for each DRAM access initiated
- ■■ 1 memory bus clock cycle to send a word of data
- If we have a cache block of four words and a one-word-wide memory of DRAMs,
- the miss penalty would be 1 + (4 × 15)+ 4 × 1 = 65 memory bus clock cycles
- Number of bytes transferred per bus clock cycle for a single miss would be
- 4WORDS/65= 4 × 4/ 65
- = 0.25



a. One-word-wide memory organization

Improving performance –Increase main memory bandwidth

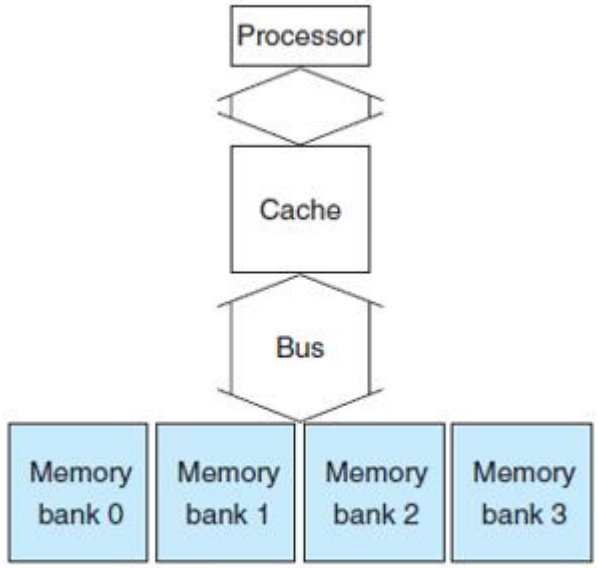
- With a wider main memory
- Increasing the width of the memory and the bus will increase the memory bandwidth proportionally, decreasing both the access time and transfer time portions of the miss penalty.
- With a main memory width of two words,
- the miss penalty drops from 65 memory bus clock cycles to
- 1 + (2 × 15) + 2 × 1 = 33 memory bus clock cycles.
- The bandwidth for a single miss is then 16/33= 0.48 (almost twice as high) bytes per bus clock cycle for a memory that is two words wide.



b. Wider memory organization

Improving performance –Increase main memory bandwidth

- With interleaving
- For example, with four banks, the time to get a four-word block would consist of
- 1 cycle to transmit the address and read request to the banks,
- 15 cycles for all four banks to access memory, and
- 4 cycles to send the four words back to the cache.
- This yields a miss penalty of 1 + (1 × 15) + 4 × 1 = 20 memory bus clock cycles.
- This is an effective bandwidth per miss of 16/20=0.80 bytes per clock, or about three times the bandwidth for the one-word-wide memory and bus.



c. Interleaved memory organization

Direct MAPPED CACHE SUMMARY

- A direct-mapped cache with a one-word block. In such a cache, both hits and misses are simple, since a word can go in exactly one location and there is a separate tag for every word.
- To keep the cache and memory consistent, a write-through scheme can be used, so that every write into the cache also causes memory to be updated.
- The alternative to write-through is a write-back scheme that copies a block back to memory when it is replaced;
- To take advantage of spatial locality, a cache must have a block size larger than one word. The use of a larger block decreases the miss rate and improves the efficiency of the cache by reducing the amount of tag storage relative to the amount of data storage in the cache.
- Although a larger block size decreases the miss rate, it can also increase the miss penalty. If the miss penalty increased linearly with the block size, larger blocks could easily lead to lower performance.
- To avoid performance loss, the bandwidth of main memory is increased to transfer cache blocks more efficiently.
- Common methods for increasing bandwidth external to the DRAM are making the memory wider and interleaving.
- DRAM designers have steadily improved the interface between the processor and memory to increase the bandwidth of burst mode transfers to reduce the cost of larger cache block sizes-DDRAM.

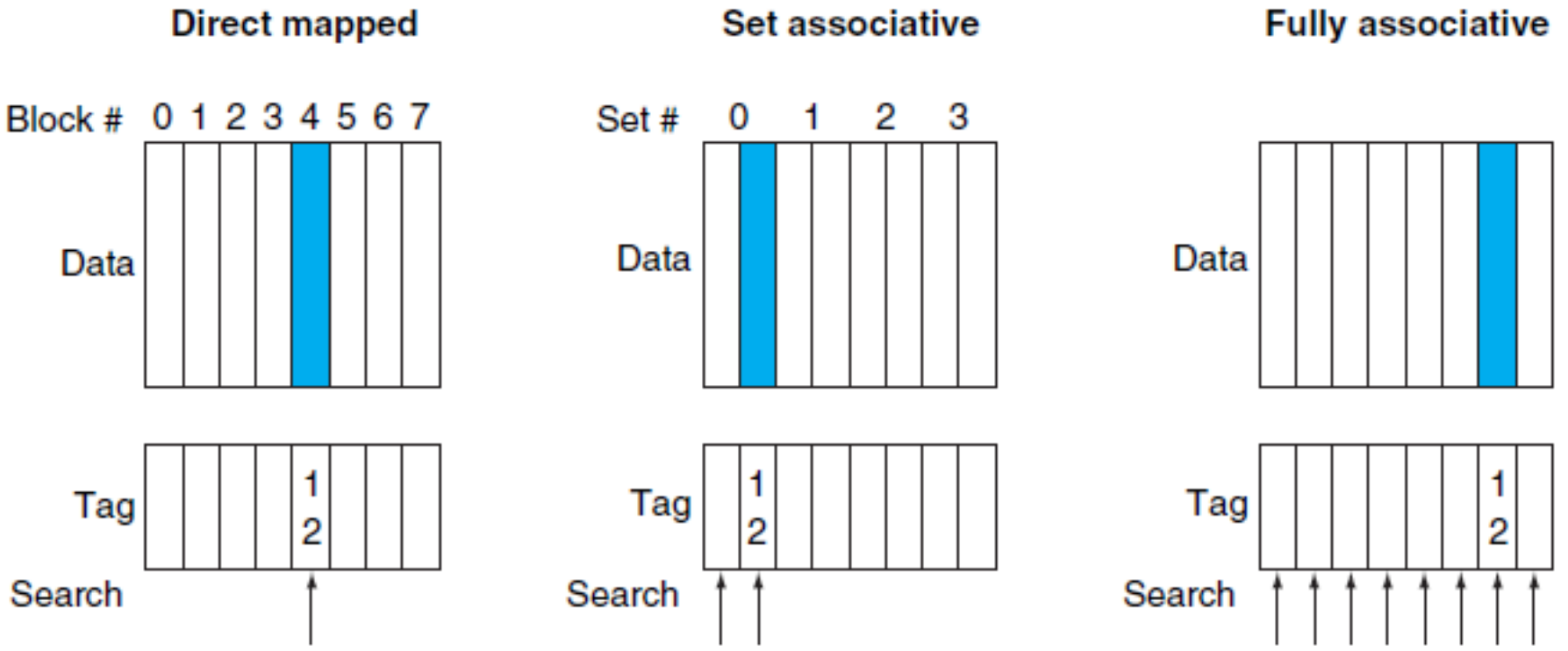
Work it out again (Done in class)

- Below is a sequence of ten memory references to an empty eight-block cache. The tag field size is 2 bits. Show the values of all the fields in direct mapped cache memory for all the ten memory references in tabular form. The sequence of memory references:
- 22, 26, 23, 26, 15, 3, 15, 17, 15, and 2.

Homework

- Consider the main memory size is 4 GB, Cache size is 1 MB, Block size is 4 KB. In a direct memory mapping, if CPU request a particular word from memory, how the bits of memory address are split into offset, cache index and tag bits, assuming a 32-bit address?
- Similar to problem done in class- How many total bits are required for a direct-mapped cache with 16 KB of data and 4-word blocks, assuming a 32-bit address?

Different mapping techniques –Placement of Blocks

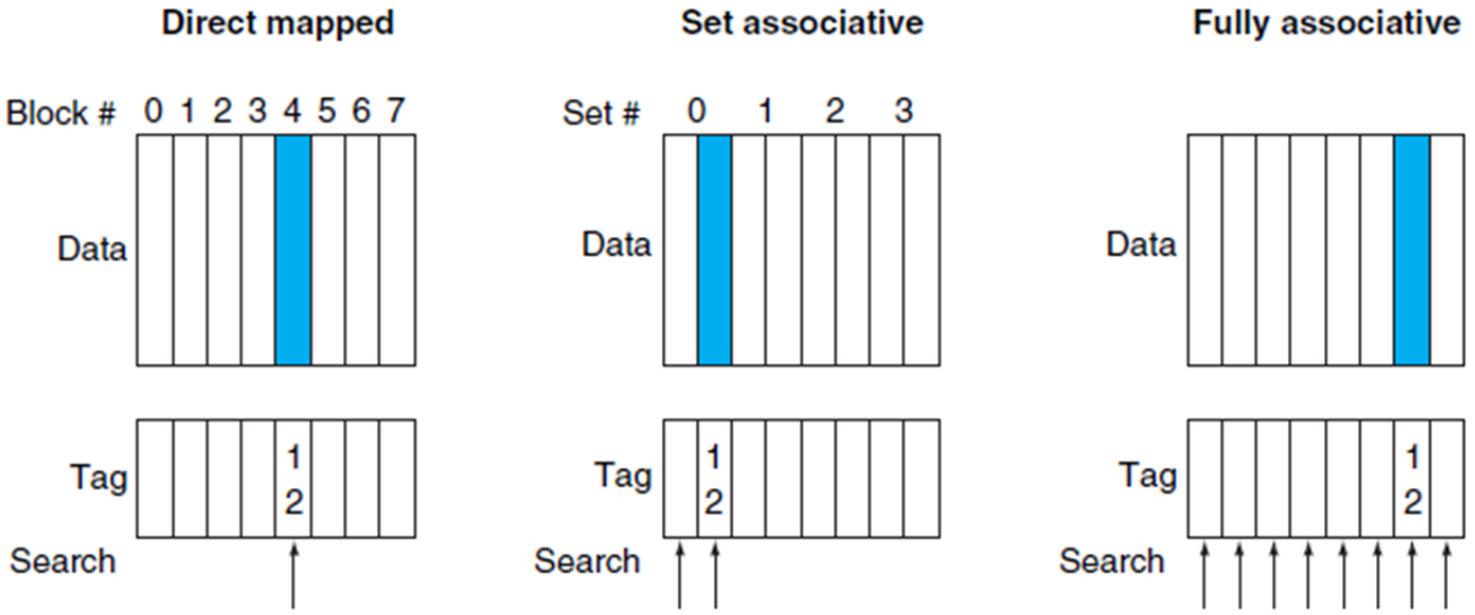


in a direct-mapped cache, the position of a memory block is given by (Block number) modulo (Number of blocks in the cache)

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

a block can be placed in any location in the cache

Different mapping techniques –Placement of Blocks



The location of a memory block whose address is 12 in a cache with eight blocks varies for directmapped, set-associative, and fully associative placement.

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.

Misses and Associativity in Caches

- 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.

Misses and Associativity in Caches

- direct-mapped cache
- 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

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.

Misses and Associativity in Caches

- Two-way set-associative
- Cache size is 4 BLOCKS
- Set-associative cache has two sets (with indices 0 and 1) with two elements per set.
- 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

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]		

Four misses

Replaces least recently used block

Misses and Associativity in Caches

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

The 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:

Different mapping techniques

- A block can go in exactly one place in the cache-it is called direct mapped because there is a direct mapping from any block address in memory to a single location in the upper level of the hierarchy
- Fully associative cache -A cache structure in which a block can be placed in any location in the cache.
- Set-associative cache- A cache that has a fixed number of locations (at least two) where each block can be placed

Set associative a compromise between Direct Mapped and Fully associative

- 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.
- To make the search practical, it 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 a compromise between Direct Mapped and Fully associative

- 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.
- To make the search practical, it 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.
- 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.

Set associative cache

- 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.

Set associative cache

- In a set associative 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.

Set associative cache

- All block placement strategies are variations of set associative

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

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

An n-way set-associative cache consists of a number of sets, each of which consists of n blocks

Set associative cache

- All block placement strategies are variations of set associative
- A direct-mapped cache is simply a one-way set-associative cache:
- each cache entry holds one block and each set has one element.
- A fully associative cache with m entries is simply an m-way set-associative cache; it has one set with m blocks, and an entry can reside in any block within that set.

Set associative cache

- 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.

Set associative cache

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								

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.

Misses and Associativity in Caches

- 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.

Misses and Associativity in Caches

- direct-mapped cache
- 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

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.

Misses and Associativity in Caches

- Cache size is 4 BLOCKS
- Set-associative cache has two sets (with indices 0 and 1) with two elements per set.
- 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

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]		

Replaces least recently used block

Misses and Associativity in Caches

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

The 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:

Set associative cache block replacement strategy-Least recently used block

- Choosing which block to replace when a miss occurs
- we have a choice of where to place the requested block, and hence a choice of which block to replace
- In a set-associative cache, we must choose among the blocks in the selected set.
- Least recently used (LRU) -A replacement scheme in which the block replaced is the one that has been unused for the longest time.

LRU

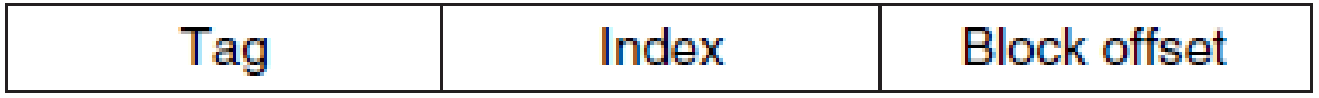
- LRU replacement is implemented by keeping track of when each element in a set was used relative to the other elements in the set.
- For a two-way set-associative cache, tracking when the two elements were used can be implemented by keeping a single bit in each set and setting the bit to indicate an element whenever that element is referenced.

Set associative cache

- The advantage of increasing the degree of associativity is that it usually decreases the miss rate.
- The main disadvantage, is a potential increase in the hit time.

Locating a block in a cache

- As 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
- The index is used to select the set,
- then the tag is used to choose the block by comparison with the blocks in the selected set.
- The block offset is the address of the desired data within the block.



Locating a block in a cache

Two-way set associative

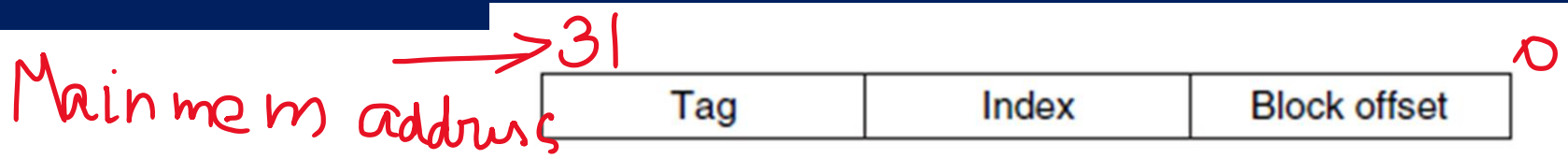
Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Each degree of associativity decreases the number of sets by a factor of 2 and thus decreases the number of bits used to index the cache by 1 and increases the number of bits in the tag by 1.

- Assuming a cache of 4K blocks, a 4-word block size, and a 32-bit address, find the total number of sets and the total number of tag bits for caches that are direct mapped, two-way and four-way set associative, and fully associative.
- Cache size=
- Block size =
- Direct-mapped cache
- number of sets =
- total number OF tag bits =

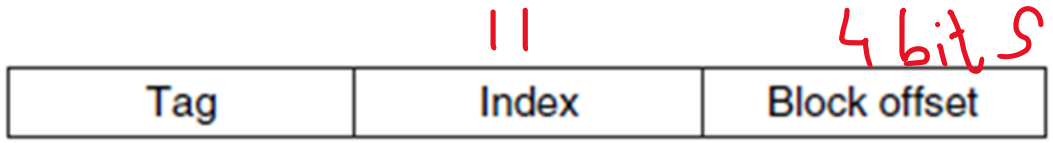


- Assuming a cache of 4K blocks, a 4-word block size, and a 32-bit address, find the total number of sets and the total number of tag bits for caches that are direct mapped, two-way and four-way set associative, and fully associative.
- Two way associative
- number of sets
- total number OF tag bits

- Assuming a cache of 4K blocks, a 4-word block size, and a 32-bit address, find the total number of sets and the total number of tag bits for caches that are direct mapped, two-way and four-way set associative, and fully associative.
- Four way associative
- number of sets =
- total number OF tag bits

- Assuming a cache of 4K blocks, a 4-word block size, and a 32-bit address, find the total number of sets and the total number of tag bits for caches that are direct mapped, two-way and four-way set associative, and fully associative.
- Fully associative

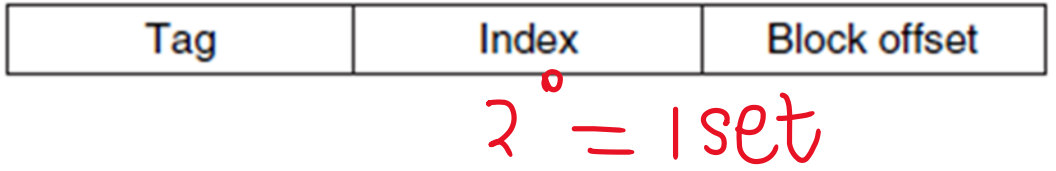
- Assuming a cache of 4K blocks, a 4-word block size, and a 32-bit address, find the total number of sets and the total number of tag bits for caches that are direct mapped, two-way and four-way set associative, and fully associative.
- Cache size= 4K blocks= 4x1024= 4096 cache blocks
- Block size =4 words= 4x32 bits=4x4BYTES=16BYTES(2^4) bytes /block
- 32-bit address yields 32 - 4 = 28 bits to be used for index and tag.
- Direct-mapped cache
- same number of sets as blocks-4096, and
- $\log_2(4096) = 12$; hence 12 bits of index
- hence, the total number is (28 - 12) × 4K = 16 × 4K = 64 K tag bits.



- Each degree of associativity decreases the number of sets by a factor of 2 and thus decreases the number of bits used to index the cache by 1 and increases the number of bits in the tag by 1.
- For a two-way set-associative cache,
- Block size =4 words= 4x32 bits=4x4BYTES=16BYTES(2^4) bytes /block
- 32 - 4 = 28 bits to be used for index and tag
- Total size of the cache in blocks=No :of sets x Associativity)
- No:of sets= 4K/2= 2K sets
- To access 2K (2x 2^{10}) sets we need 11 index bits
- the total number of tag bits is (28 -11) × Total cache blocks.
- the total number of tag bits is (28 -11) × No :of sets x Associativity
- the total number of tag bits is (28 -11) × 2K x2= 34 × 2K = 68Kbits.

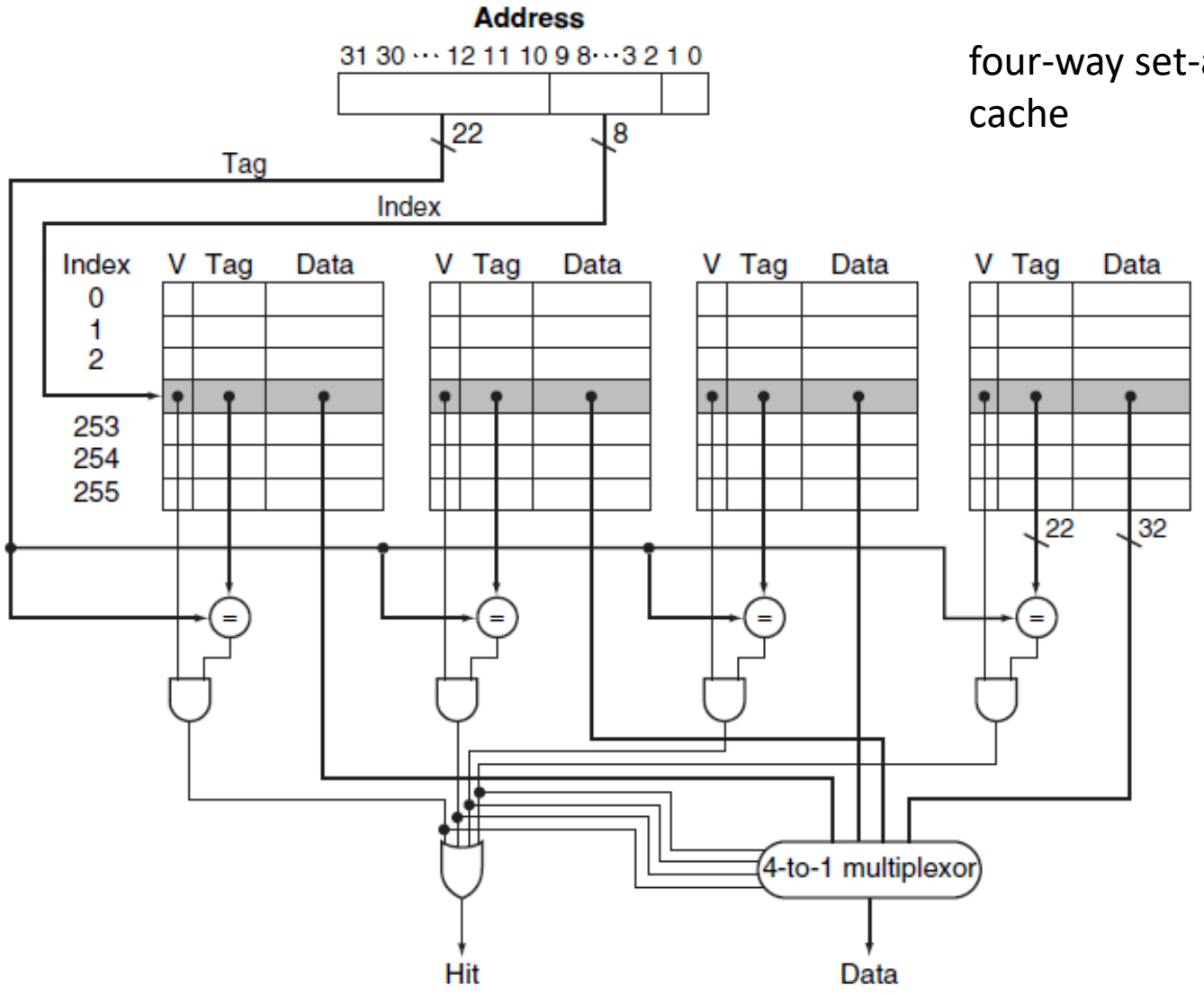
- Each degree of associativity decreases the number of sets by a factor of 2 and thus decreases the number of bits used to index the cache by 1 and increases the number of bits in the tag by 1.
- Block size =4 words= 4x32 bits=4x4BYTES=16BYTES(2^4) bytes /block
- 32 - 4 = 28 bits to be used for index and tag
- For a four-way set-associative cache,
- Total size of the cache in blocks=No :of sets x Associativity
- No:of sets in 4 way set associative cache= Total size of the cache in blocks/ Associativity
- = 4K/4=1K sets
- To access 1K (2^{10}) sets we need 10 index bits
- The total number of tag bits is (28 - 10) × Total cache blocks bits.
- The total number of tag bits is (28 - 10) × 4K = 72 K = 72 K tag bits.

- For a fully associative cache, there is only one set with 4K blocks,
- the tag is 28 bits, (32-4 bits)



- leading to Total no:no :of tag bits- 28 × 4K = 112K tag bits.
- (Block size =4 words= 4x32 bits=4x4BYTES=16BYTES(2^4) bytes /block
- 32 - 4 = 28 bits to be used for index and tag)

Locating a block in a cache



four-way set-associative cache

Multi-level cache

- 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.

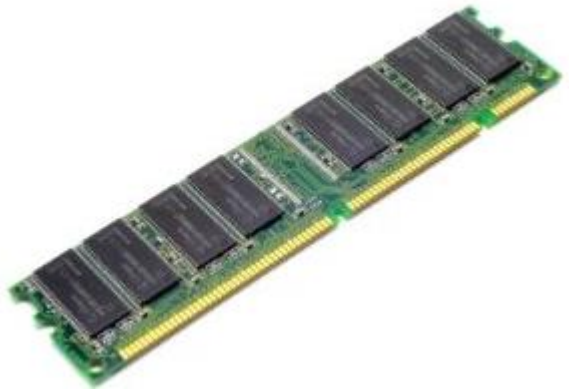
Performance Multi-level cache

- We have a processor with a base CPI of 1.0, assuming all references hit in the primary cache, and a clock rate of 4 GHz.
- Assume a main memory access time of 100 ns, including all the miss handling.
- Suppose the miss rate per instruction at the primary cache is 2%.
- How much faster will the processor be if we add a secondary cache that has a 5 ns access time for either a hit or a miss and is large enough to reduce the miss rate to main memory to 0.5%?

Performance Multi-level cache

- We have a processor with a base CPI of 1.0, assuming all references hit in the primary cache, and a clock rate of 4 GHz.
- Assume a main memory access time of 100 ns, including all the miss handling.
- Suppose the miss rate per instruction at the primary cache is 2%.
- How much faster will the processor be if we add a secondary cache that has a 5 ns access time for either a hit or a miss and is large enough to reduce the miss rate to main memory to 0.5%?

Difference between virtual memory and physical memory



RAM physical Memory (Main memory/Primary memory)

Physical memory is a volatile memory, and it has a limited size because of the RAM chip.

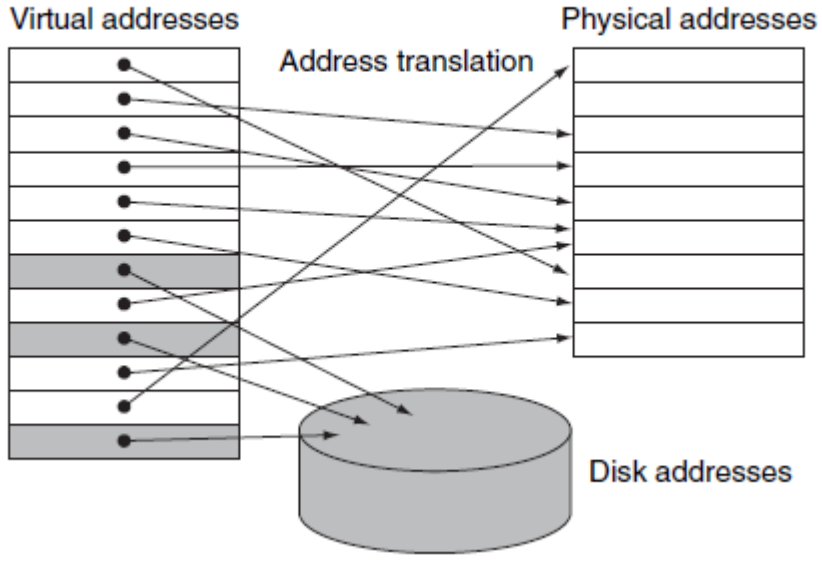
Virtual memory is a memory management strategy that creates an image of a very extensive memory for users.

This concept provides a one-of-a-kind method for apparent primary memory extension. This allows us to access secondary memory as if it were a part of the main memory.

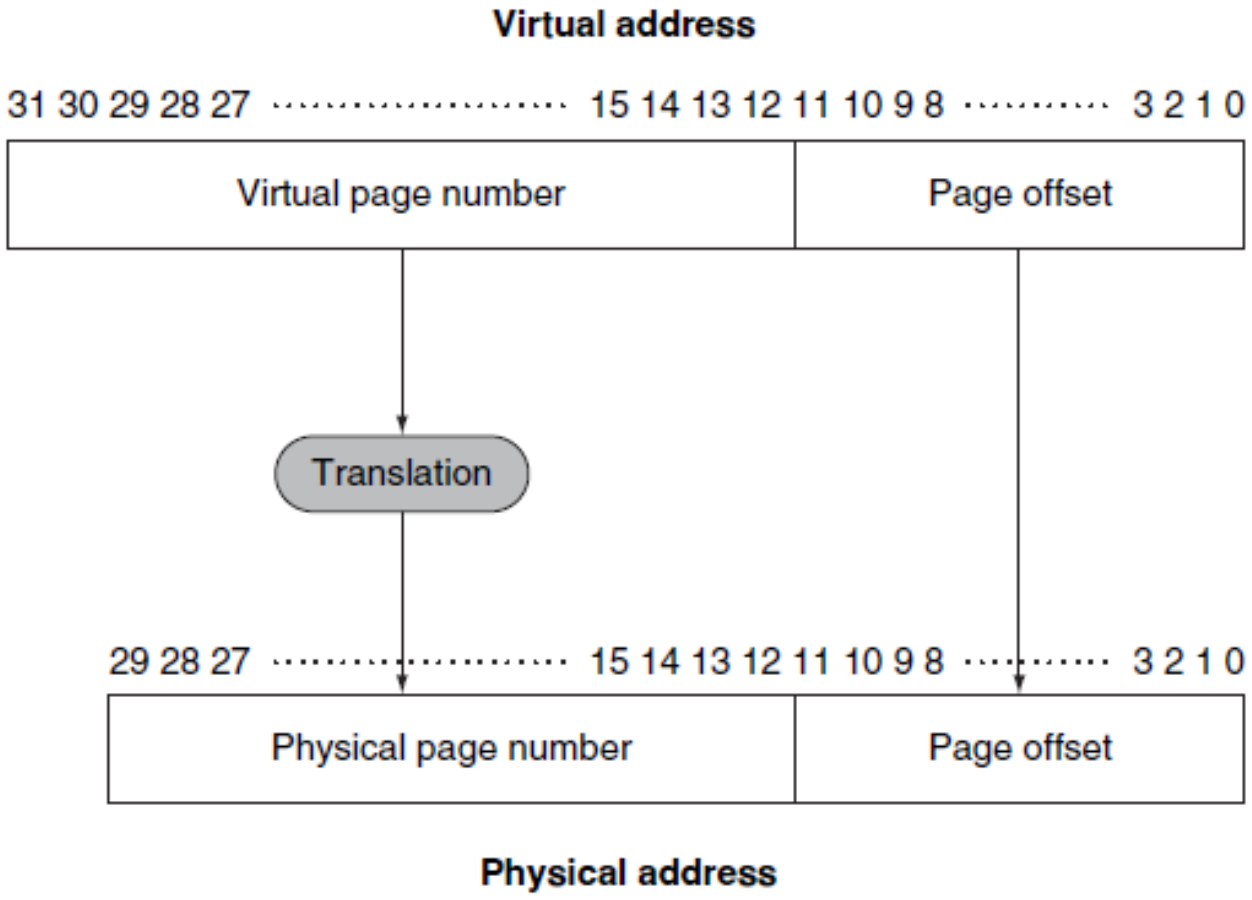
VIRTUAL Memory

- Virtual memory A technique that uses main memory as a “cache” for secondary storage.
- Virtual memory implements the translation of a program’s address space to physical addresses.
- 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. Of course, 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.

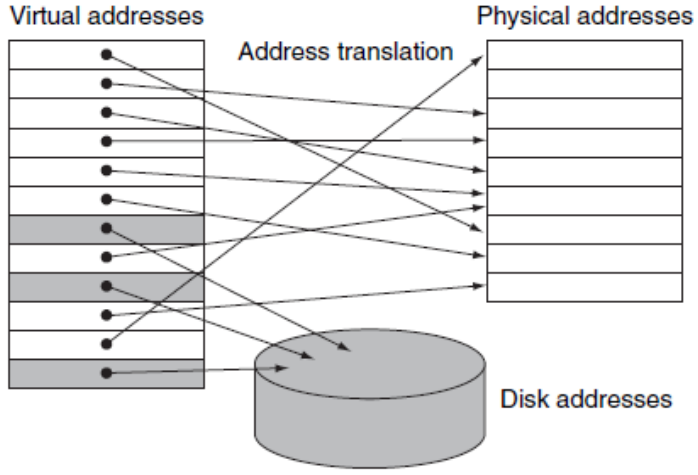


Mapping from a virtual to a physical address.



VIRTUAL Memory

- 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.

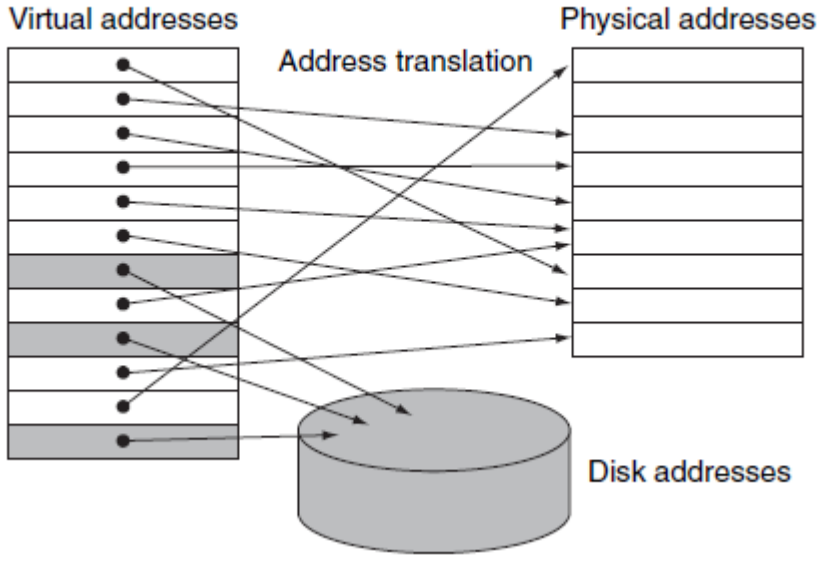


Virtual memory

- 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
- 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

VIRTUAL Memory -Terminology

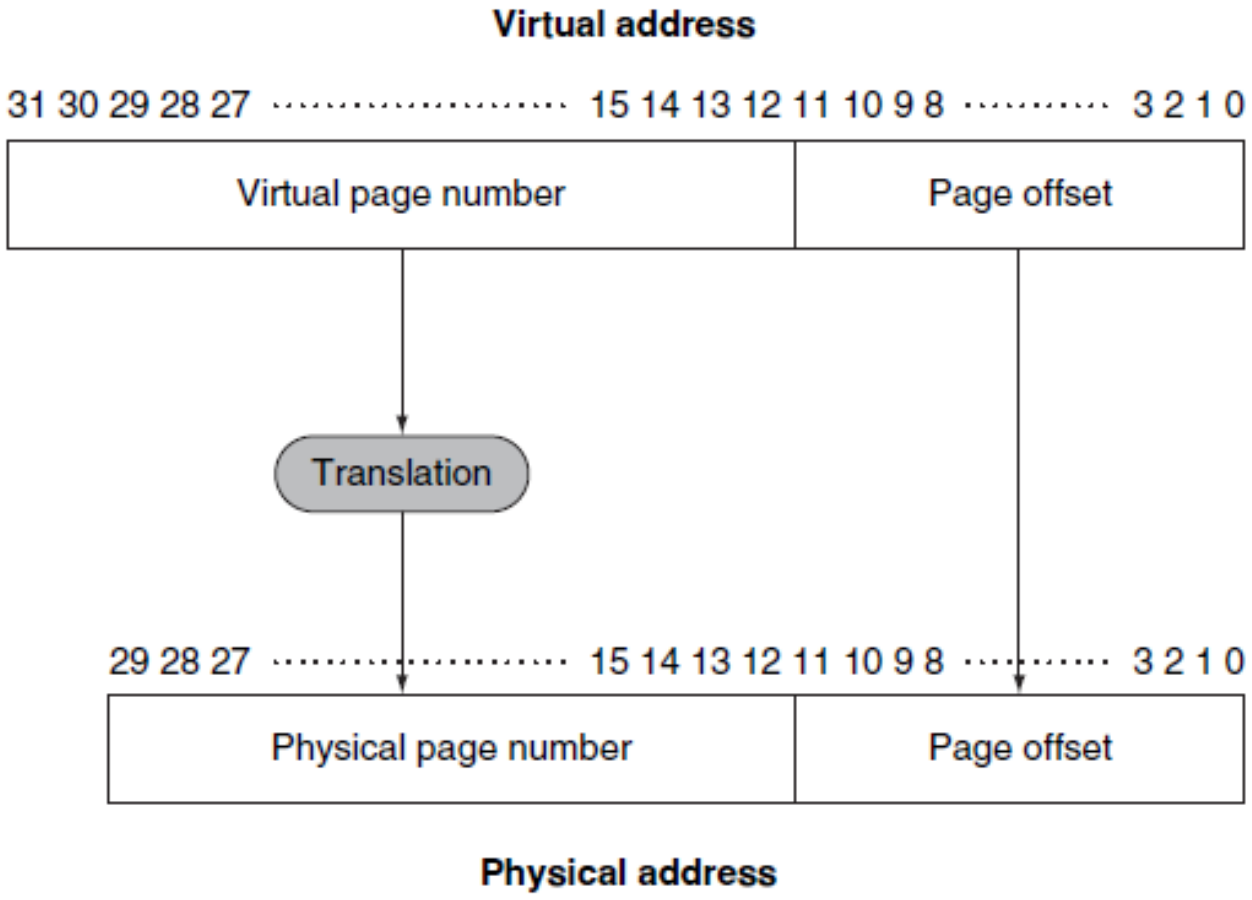
- Virtual address- An address that corresponds to a location in virtual space and is translated by address mapping to a physical address when memory is accessed.
- Address translation- Also called address mapping. The process by which a virtual address is mapped to an address used to access memory.



Page fault

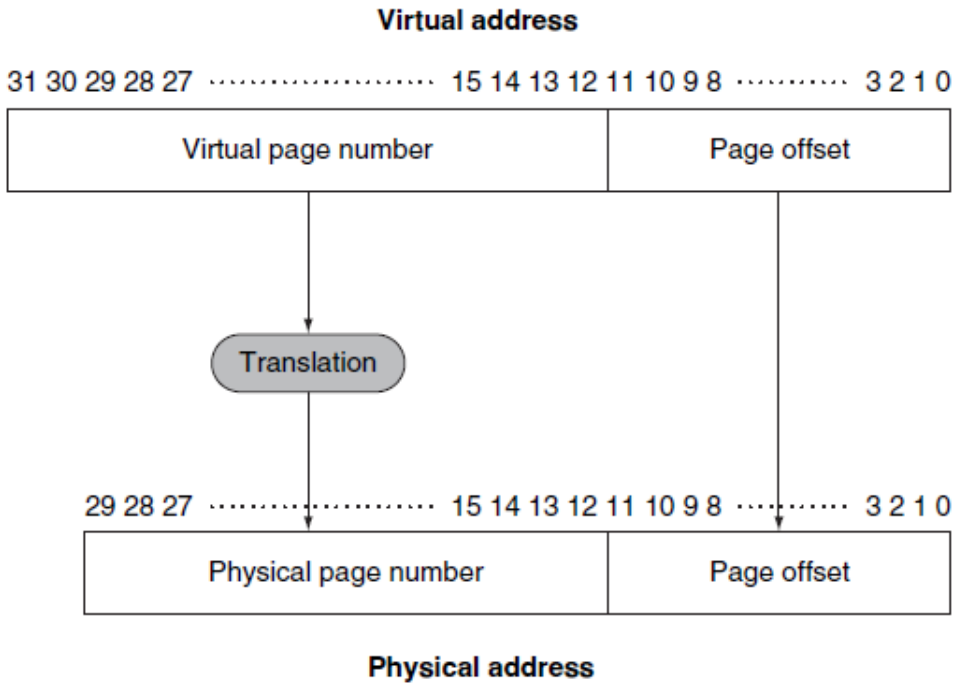
- Page fault An event that occurs when an accessed page is not present in main memory.
- A page fault will take millions of clock cycles to process.
- Key decisions in designing virtual memory systems
- Pages should be large enough.
- Reduce the page fault rate- The primary technique used here is to allow fully associative placement of pages in memory
- Page faults can be handled in software -softftware can afford to use clever algorithms for choosing how to place pages
- Write-through will not work for virtual memory, since writes take too long. virtual memory systems use write-back.

Mapping from a virtual to a physical address.



Mapping from a virtual to a physical address.

- In virtual memory, the address is broken into a virtual page number and a page offset.
- Figure shows the translation of the virtual page number to a physical page number.
- The physical page number constitutes the upper portion of the physical address, while the page 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.
- Having a larger number of virtual pages than physical pages is the basis for the illusion of an essentially unbounded amount of virtual memory.

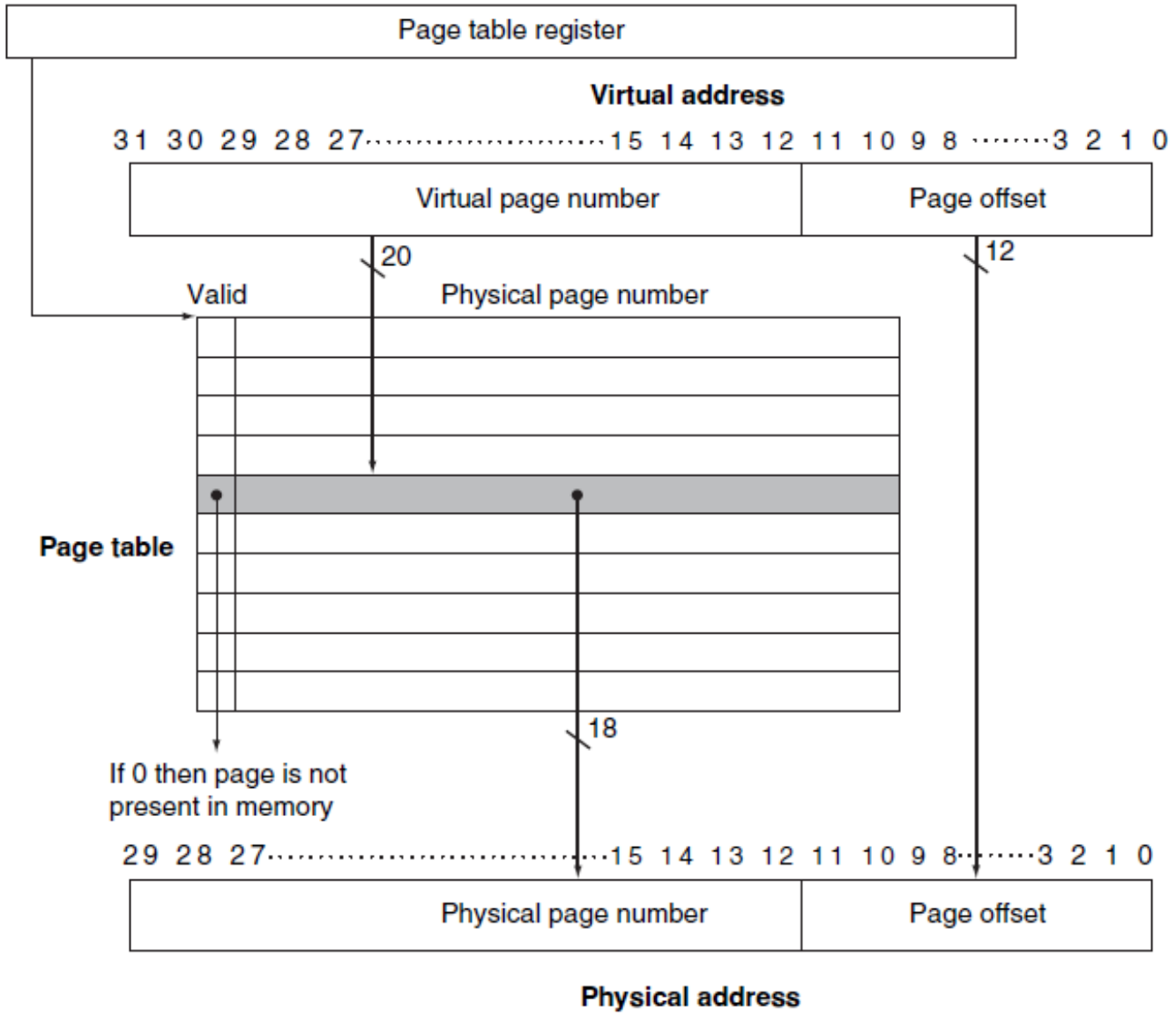


page size is 2^{12} = 4 KB.
number of physical pages allowed in memory is 2^{18}, since the physical page number has 18 bits in it. Thus,
main memory can have at most 1 GB, while the virtual address space is 4 GB.

Locating pages using page table

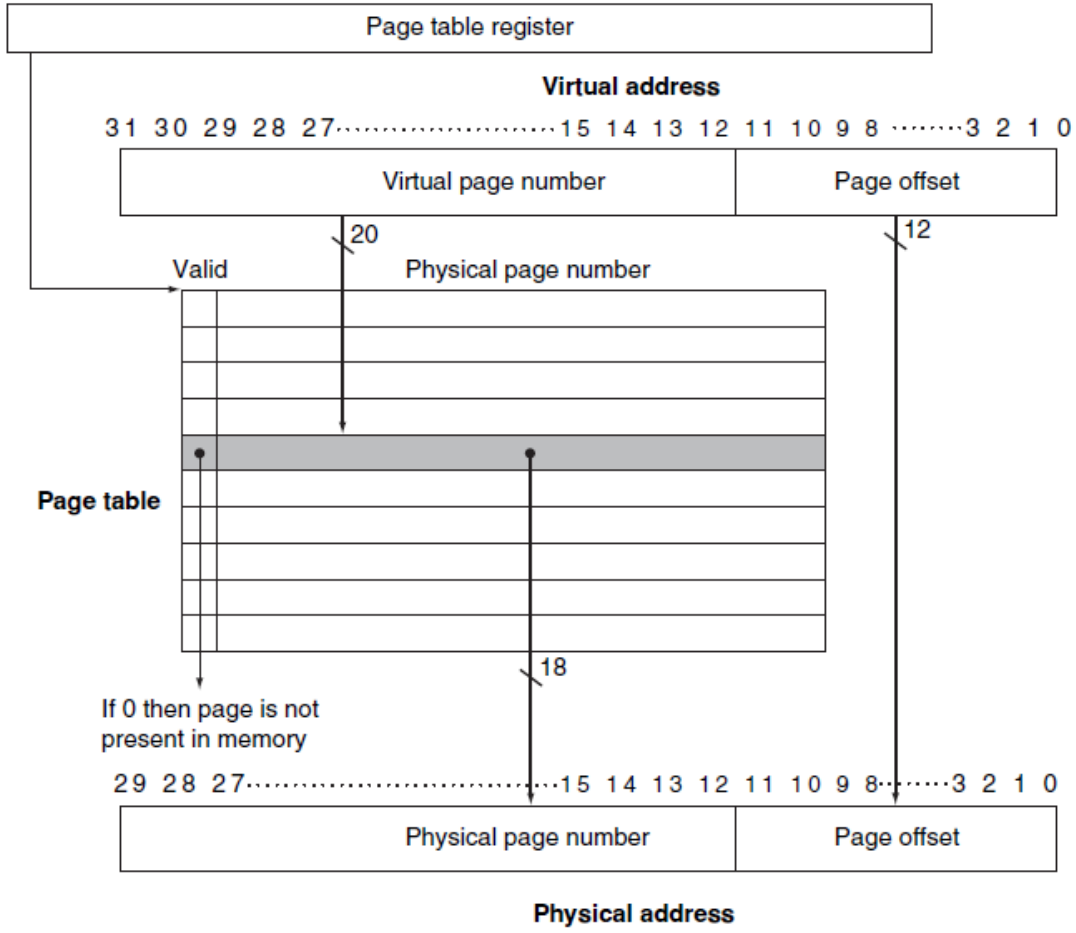
- Using fully associative placement is in locating an entry, since it can be anywhere in the upper level of the hierarchy.
- A full search is impractical.
- In virtual memory systems, we locate pages by using a table that indexes the memory; this structure is called a **page table**, and it resides in memory
- Each entry in the table contains the physical page number for that virtual page if the page is currently in 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*

Locating pages using page table



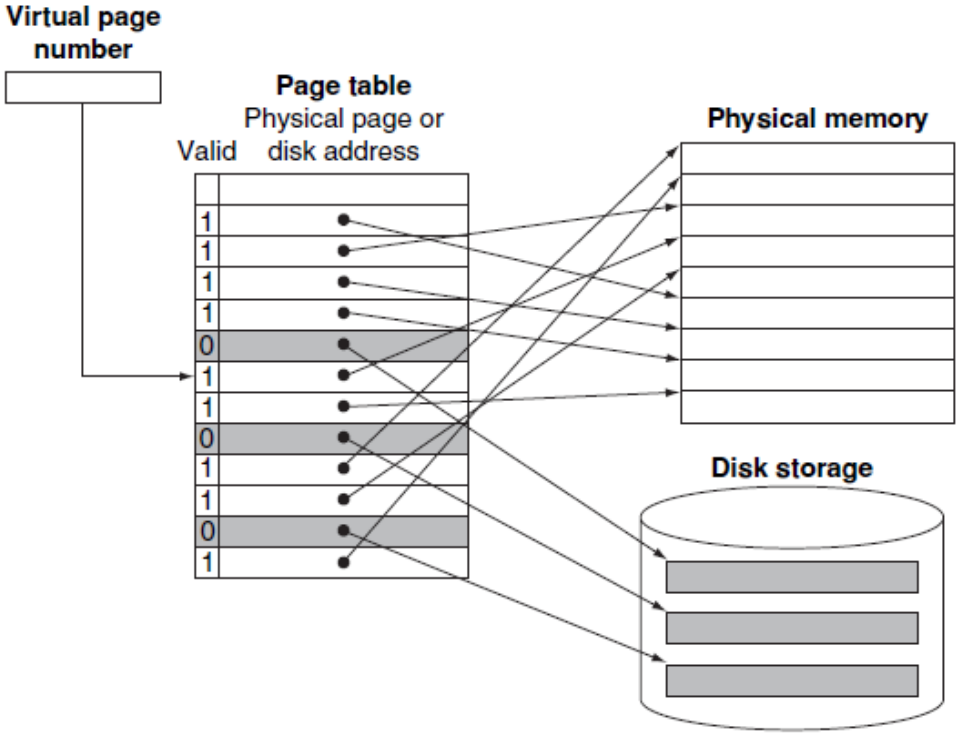
Locating pages using page table

- The page table is indexed with the virtual page number to obtain the corresponding portion of the physical address.
- We assume a 32-bit address. The starting address of the page table is given by the page table pointer. In this figure, the
- page size is 2^{12} bytes, or 4 KB. The virtual address space is 2^{32} bytes, or 4 GB, and the physical address space is 2^{30} bytes, which allows main memory of up to 1 GB.
- The number of entries in the page table is 2^{20}, or 1 million entries. The valid bit for each entry indicates whether the mapping is legal.
- If valid bit is off, then the page is not present in memory



Locating pages using page table

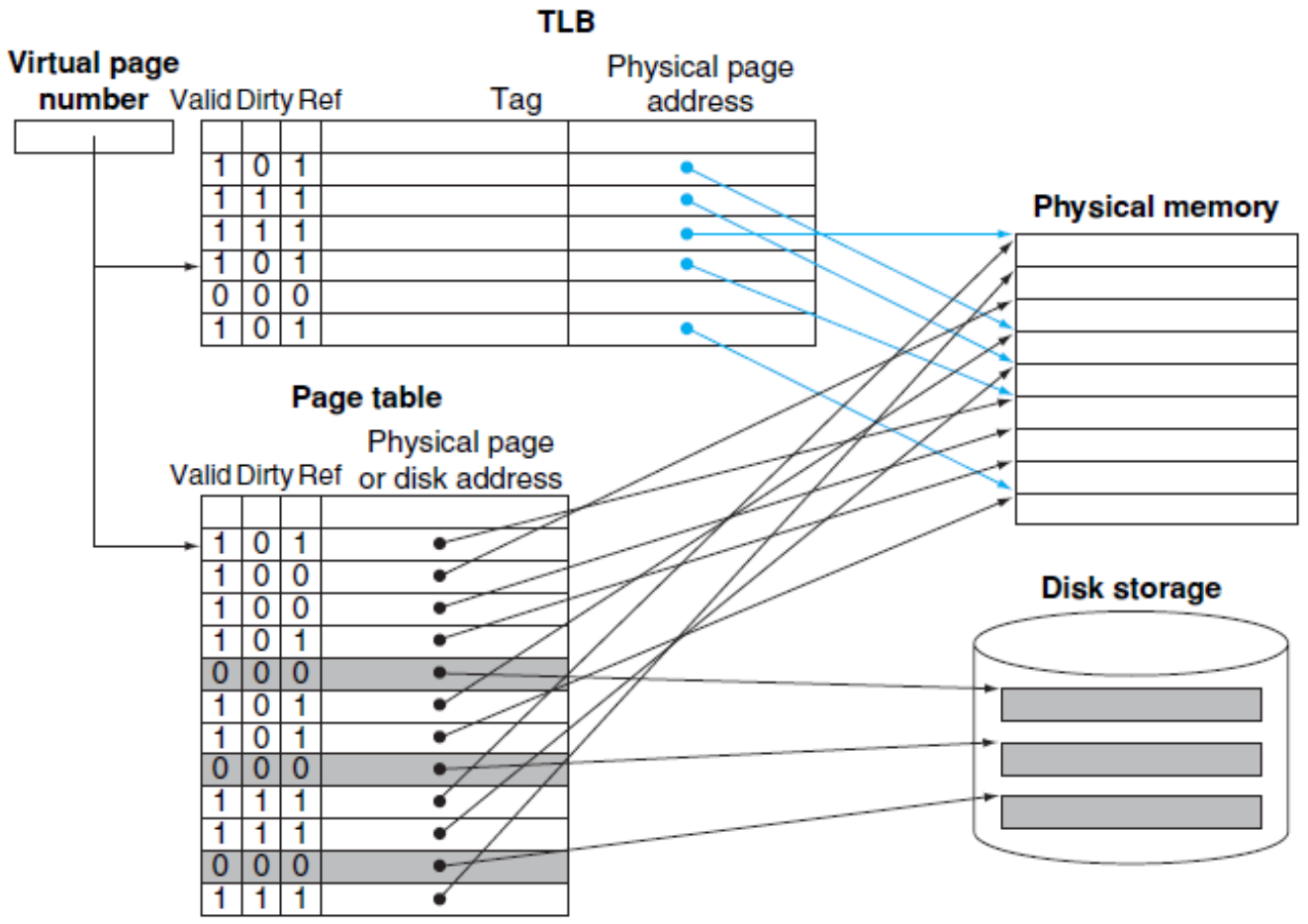
- The page table maps each page in virtual memory to either a page in main memory or a page stored on disk, which is the next level in the hierarchy.
- The virtual page number is used to index the page table. If the valid bit is on, the page table supplies the physical page number (i.e., the starting address of the page in memory) corresponding to the virtual page.
- If the valid bit is off, the page currently resides only on disk, at a specified disk address.
- In many systems, the table of physical page addresses and disk page addresses, while logically one table, is stored in two separate data structures.
- Dual tables are justified in part because we must keep the disk addresses of all the pages, even if they are currently in main memory.
- The pages in main memory and the pages on disk are the same size.



Handling writes in Virtual memory

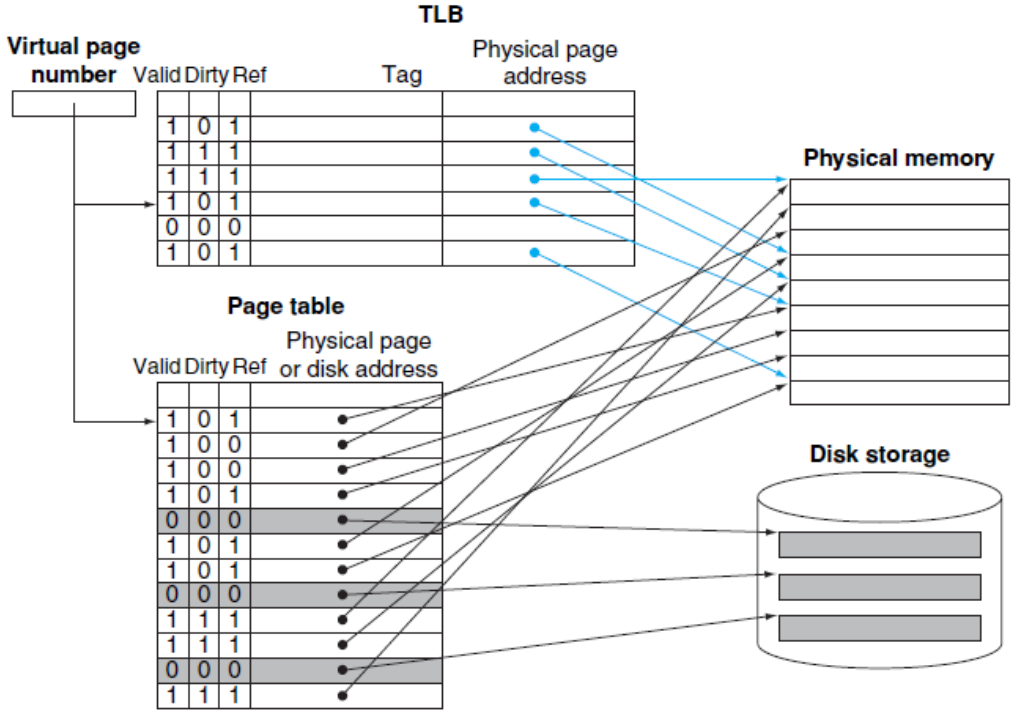
- Virtual memory systems must use write-back, performing the individual writes into the page in memory, and copying the page back to disk when it is replaced in the memory.
- To track whether a page has been written since it was read into the memory, a *dirty bit* is added to the page table. The **dirty 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.

Making address translation faster using TLB



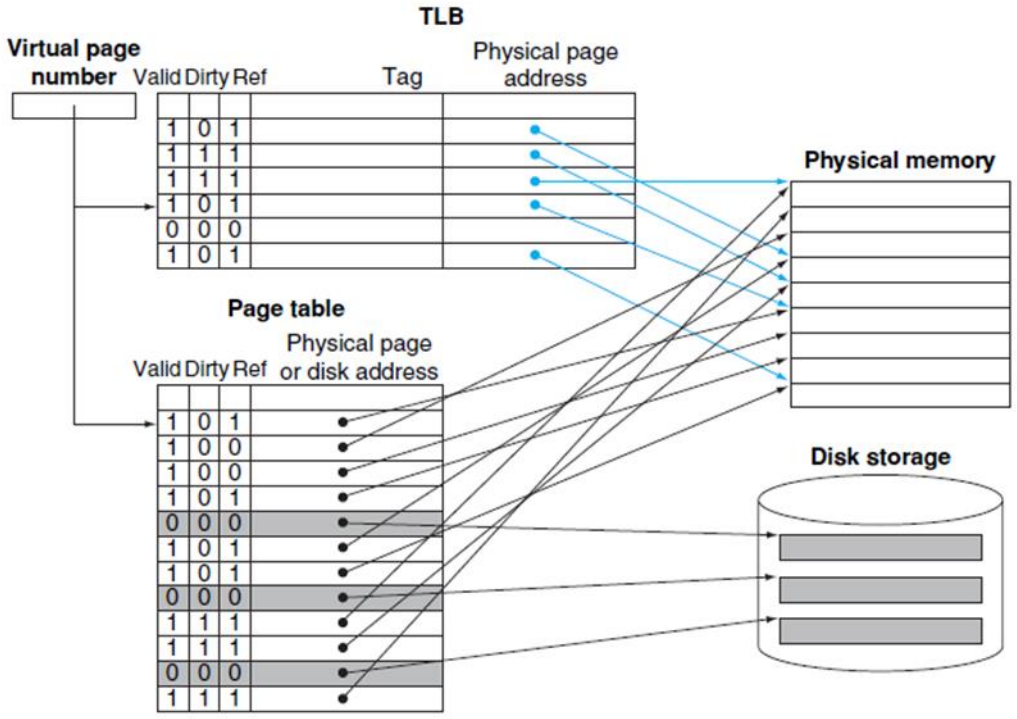
Making address translation faster using TLB

- Translation-lookaside buffer (TLB) A cache that keeps track of recently used address mappings to try to avoid an access to the page table.
- The TLB acts as a cache of the page table for the entries that map to physical pages only.



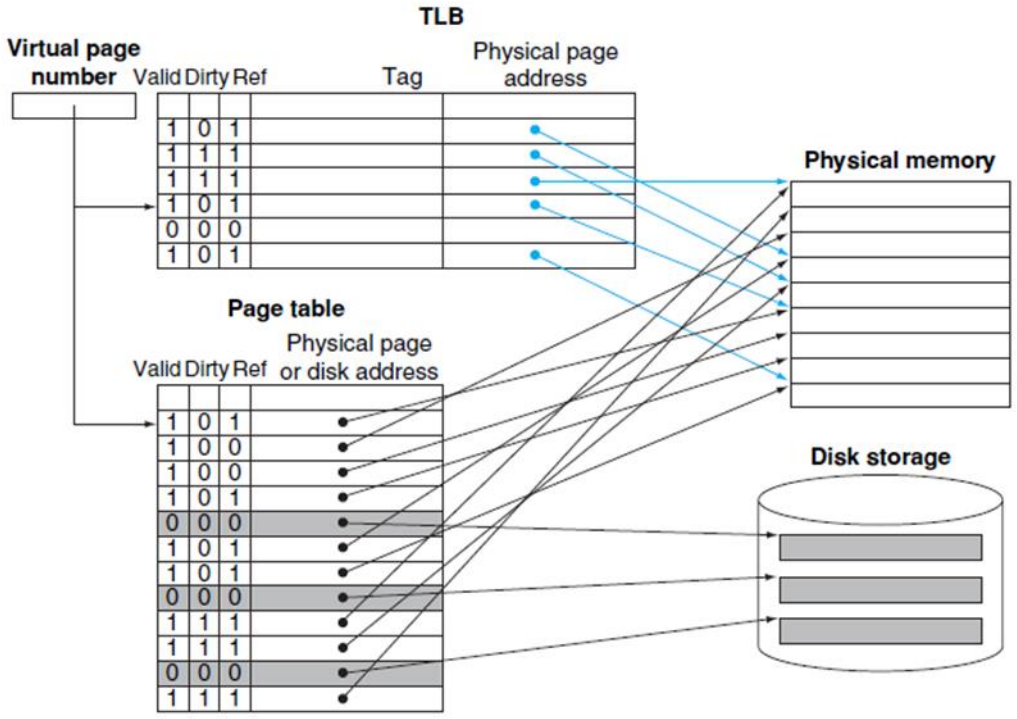
Making address translation faster using TLB

- 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 (which can then be used to build a TLB entry) or indicates that the page resides on disk, in which case a page fault occurs.
- Since the page table has an entry for every virtual page, no tag field is needed; in other words, unlike a TLB, a page table is not a cache.



Making address translation faster using TLB

- 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,



Handling TLB MISS

- 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.

Handling TLB miss

- After a TLB miss occurs and the missing translation has been retrieved from the page table, we will need to select a TLB entry to replace.
- Because the reference and dirty bits are contained in the TLB entry, we need to copy these bits back to the page table entry when we replace an entry.
- These bits are the only portion of the TLB entry that can be changed.
- Using write-back—that is, copying these entries back at miss time rather than when they are written—is very efficient, since we expect the TLB miss rate to be small.
- Some systems use small, fully associative TLBs because a fully associative mapping has a lower miss rate;

Handling page fault- Brief

- Page fault An event that occurs when an accessed page is not present in main memory.
- If the valid bit for a virtual page is off, a page fault occurs.
- The operating system must be given control. This transfer is done with the exception mechanism
- 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.
- operating system usually creates the "swap space" on disk for all the pages. The space on the disk reserved for the full virtual memory space of a process.

Handling page fault

- When a page fault occurs, if all the pages in main memory are in use, the operating system must choose a page to replace.
- Because we want to minimize the number of page faults, most operating systems try to choose a page that they hypothesize will not be needed in the near future.
- Using the past to predict the future, operating systems follow the least recently used (LRU) replacement scheme, to choose a page to replace.
- To help the operating system estimate the LRU pages, some computers provide a reference bit or use bit, which is set whenever a page is accessed.
- With this usage information, the operating system can select a page that is among the least recently referenced (detected by having its reference bit off).

Handling Page fault- Steps summarized

Indian Institute of Information Technology Kottayam

- If the page table entry indicates the page is not in memory, this time it will get a page fault exception.
- To restart the instruction after the page fault is handled, the program counter of the instruction that caused the page fault must be saved. the exception program counter (EPC) is used to hold this value
- 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.

Implementing Protection –Write access bit in TLB

- 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. The write access bit forms part of the protection mechanism.

Implementing Protection with virtual memory

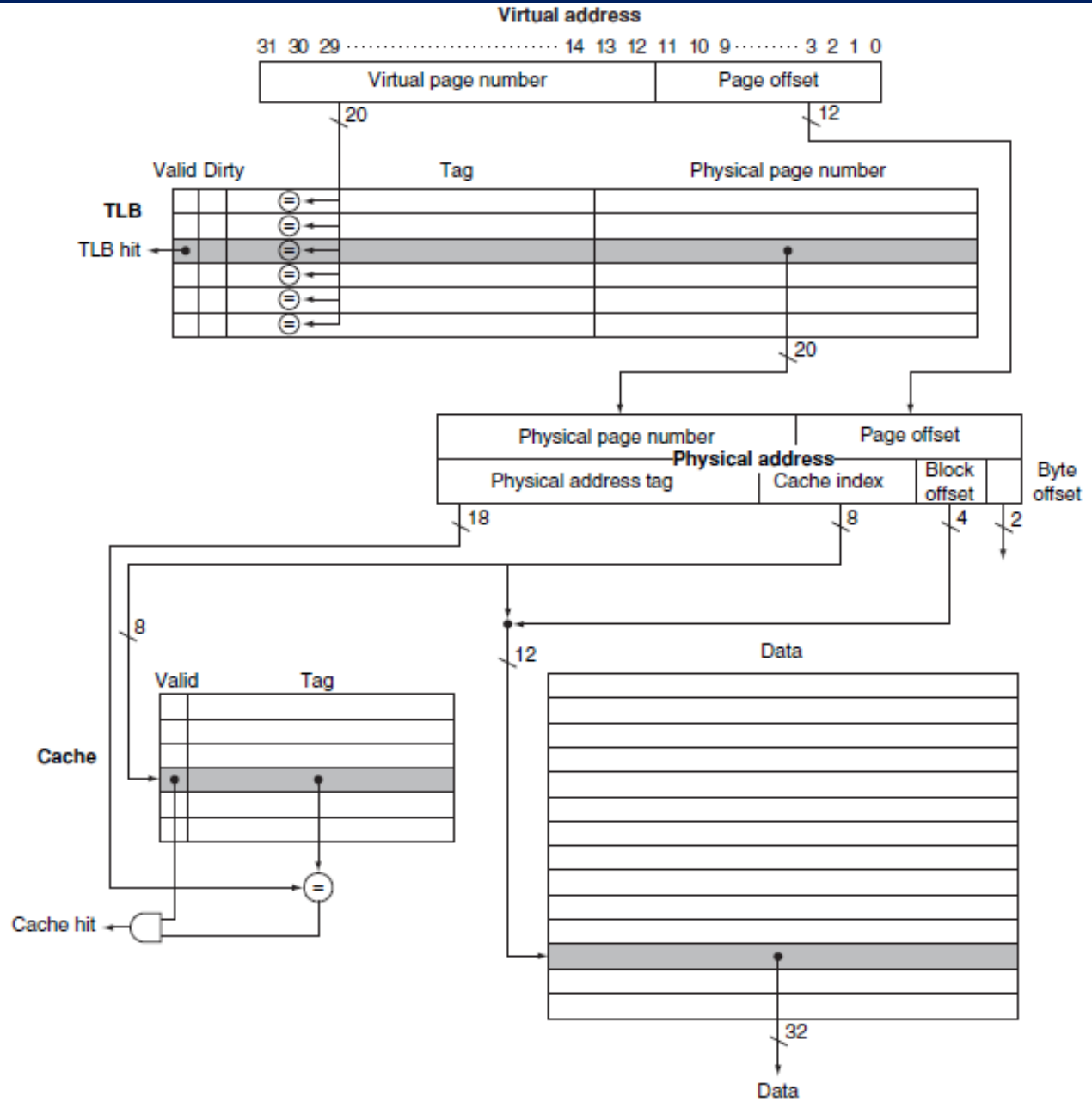
Indian Institute of Information Technology Kottayam

- 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 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
- supervisor mode -Also called kernel mode. A mode indicating that a running process is an operating system process.
- Provide a portion of the processor state that a user process can read but not write.
- system call: A special instruction that transfers control from user mode to a dedicated location in supervisor code space, invoking the exception Mechanism in the process

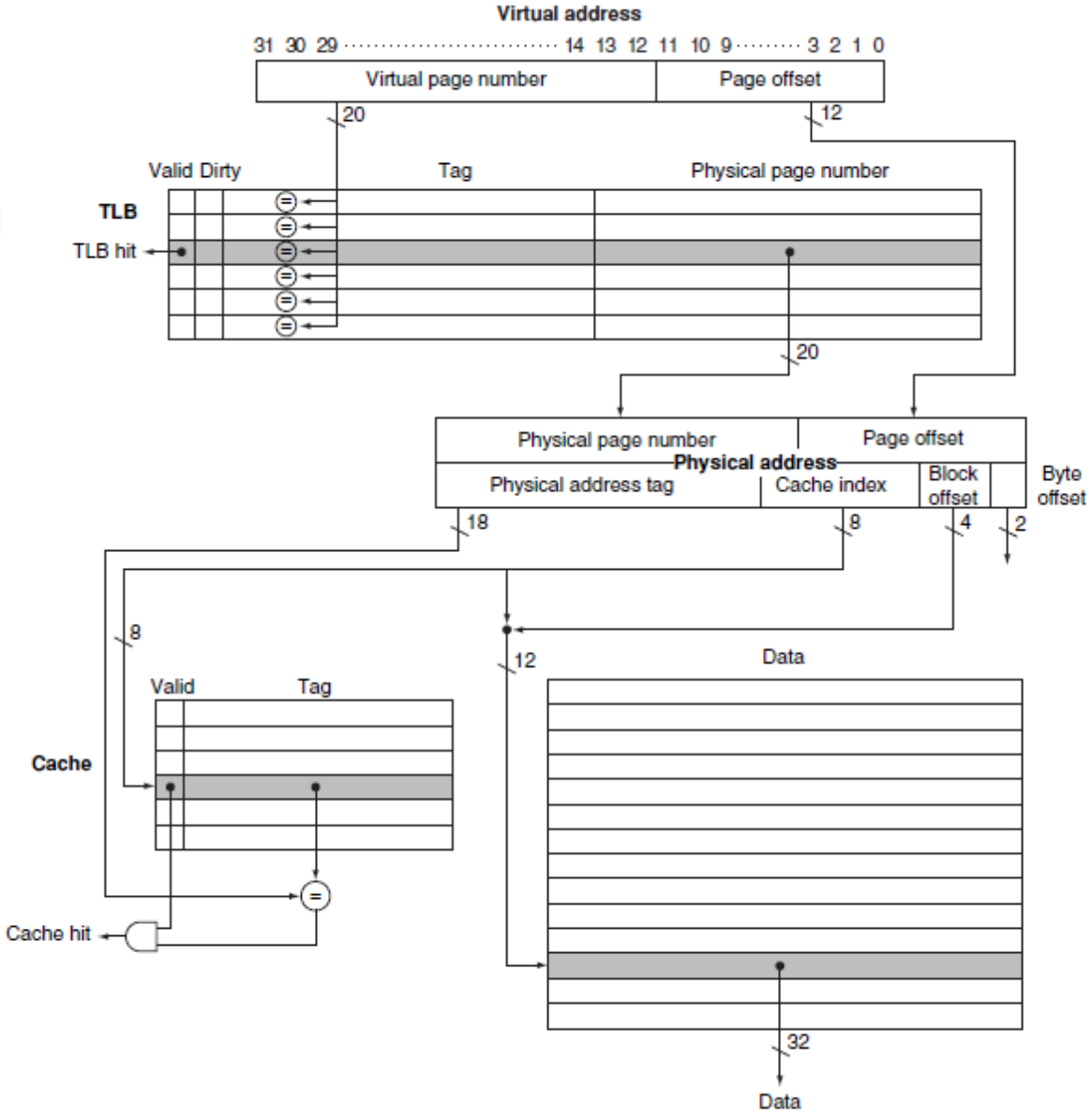
Overall operation of memory hierarchy

Handling Read



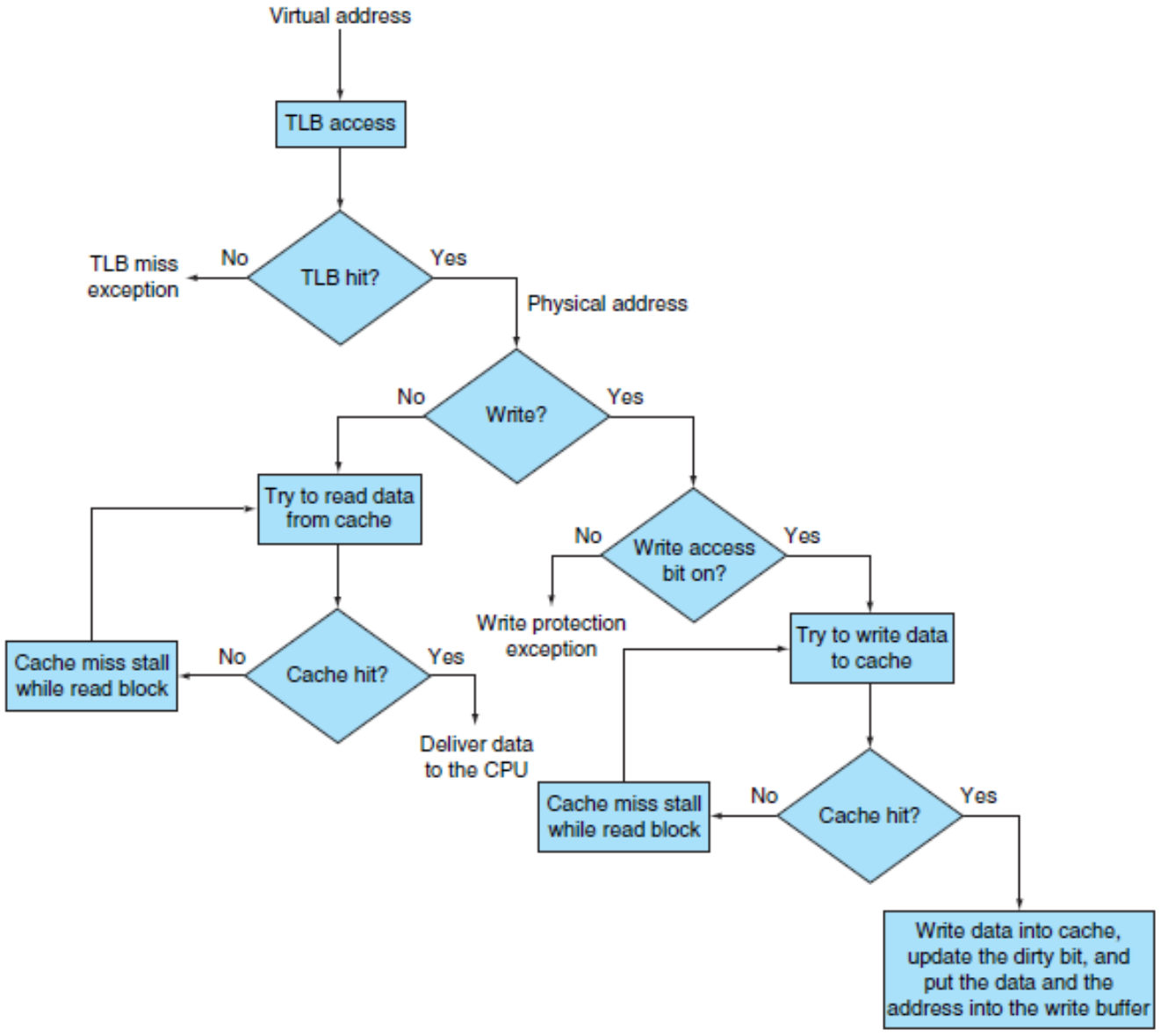
Overall operation of memory hierarchy

Handling Read



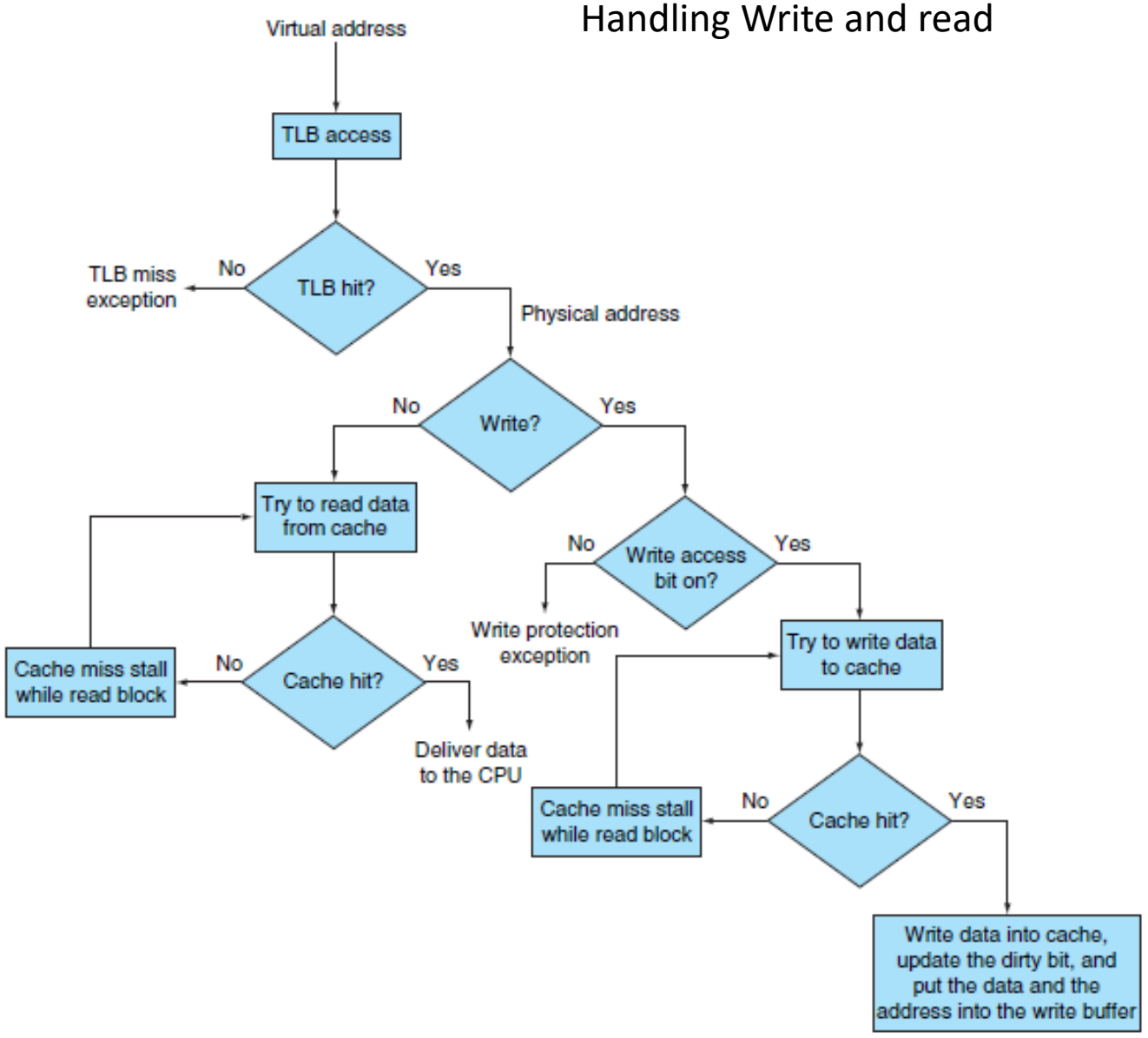
The TLB and cache implement the process of going from a virtual address to a data .
Cache is direct mapped.
TLB is fully associative.
Implementing a fully associative TLB requires that every TLB tag be compared against the virtual page number, since the entry of interest can be anywhere in the TLB.
If the valid bit of the matching entry is on, the access is a TLB hit, and bits from the physical page number together with bits from the page offset form the index that is used to access the cache.

Overall operation of memory hierarchy



Overall operation of memory hierarchy

Handling Write and read



If the TLB generates a hit, the cache can be accessed with the resulting physical address.

For a read, the cache generates a hit or miss and supplies the data or causes a stall while the data is brought from memory.

If the operation is a write, a portion of the cache entry is overwritten for a hit and the data is sent to the write buffer if we assume write-through.

A write miss is just like a read miss except that the block is modified after it is read from memory.

Write-back requires writes to set a dirty bit for the cache block, and a write buffer is loaded with the whole block only on a read miss or write miss if the block to be replaced is dirty. Notice that a TLB hit and a cache hit are independent events, but a cache hit can only occur after a TLB hit occurs, which means that the data must be present in memory.

Overall operation of memory hierarchy

- A memory reference can encounter three different types of misses:
- a TLB miss due to missing page table entry,
- a page fault, and
- a cache miss

Indian Institute of Information Technology Kottayam

Possible combinations of events in the TLB, virtual memory system, and cache.

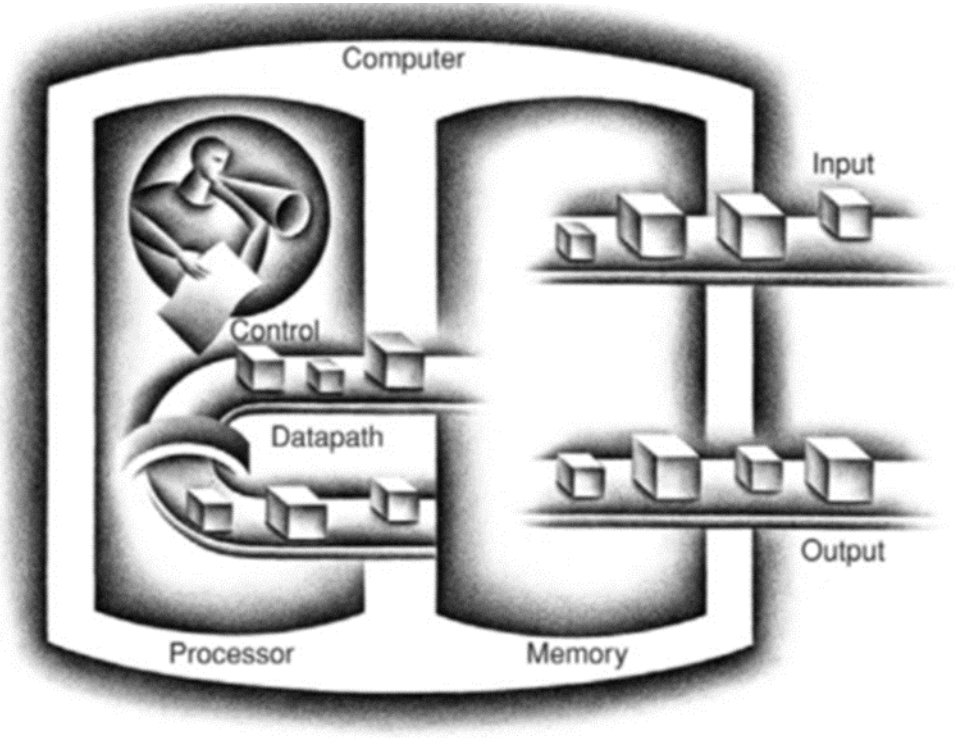
TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after 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.

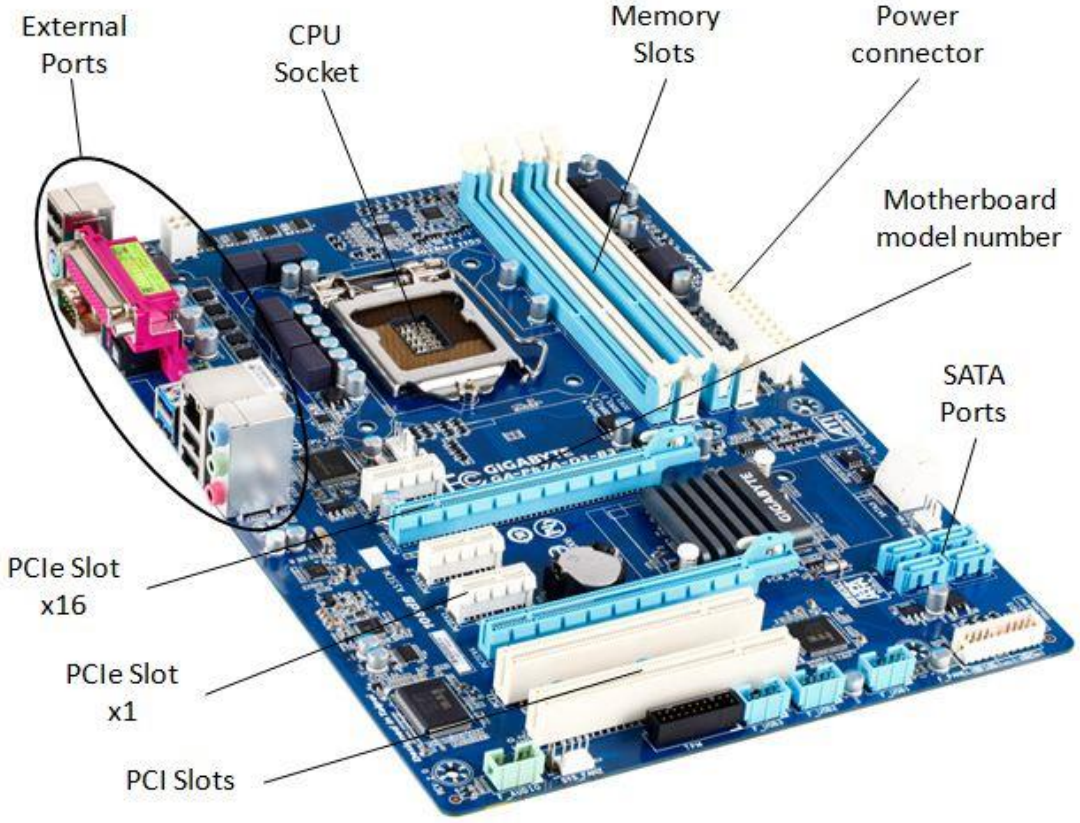
Common framework for memory

- Consolidate whatever you have studied in Memory
- The advantage of increasing the degree of associativity is that it usually decreases miss rate
- How is a block found?
- Which block to be replaced on cache miss
- What happens on write.
-

Scheme name	Number of sets	Blocks per set
Direct mapped	Number of blocks in cache	1
Set associative	$\frac{\text{Number of blocks in the cache}}{\text{Associativity}}$	Associativity (typically 2–16)
Fully associative	1	Number of blocks in the cache

Input output devices

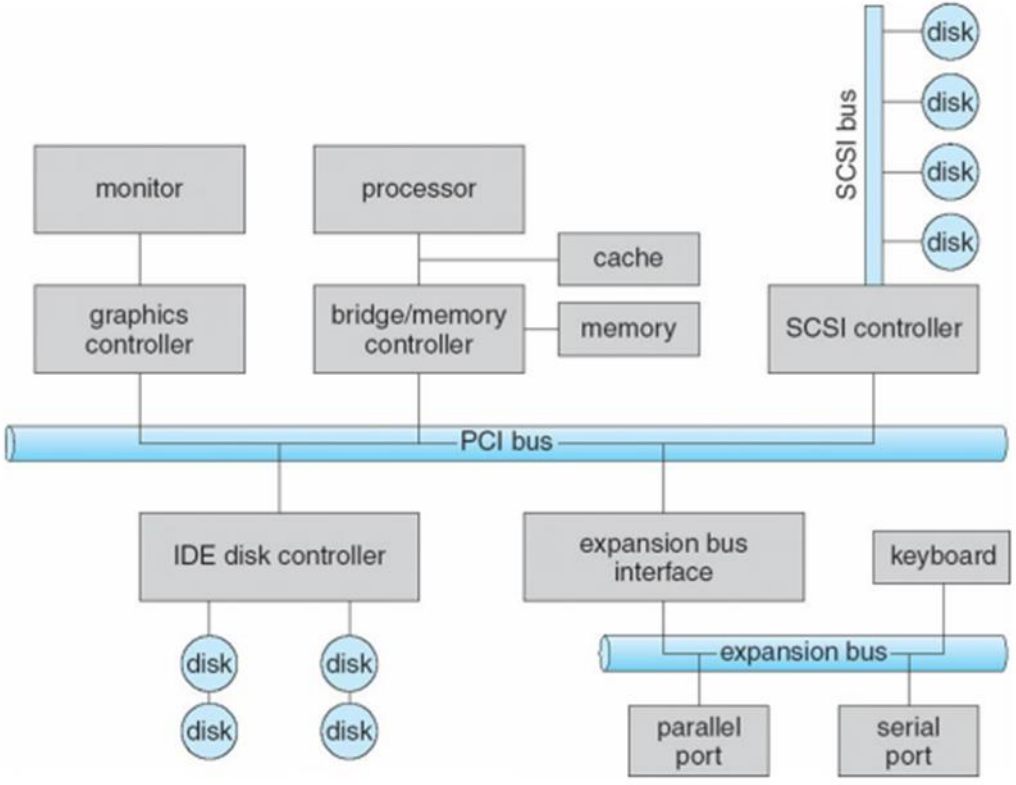




Typical PCI cards used in PCs include: network cards, sound cards, modems, extra ports such as Universal Serial Bus (USB) or serial, TV tuner cards and hard disk drive host adapters.

Input output devices

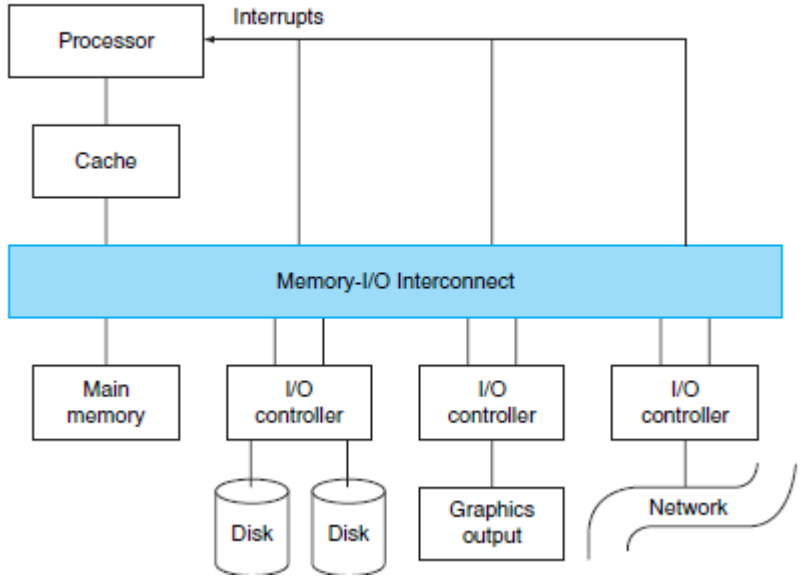
- signals from I/O devices interface with computer
- Port – connection point for device
- Bus - daisy chain or shared direct access
- PCI bus common in PCs and servers, PCI Express (PCIe)
- Expansion bus connects relatively slow devices
- Controller (host adapter) – electronics that operate on port, bus, devices.



SCSI (Small Computer System Interface) and IDE (Integrated Drive Electronics) are two interface types used to connect storage devices to computers.

Input output devices

- The connections between the I/O devices, processor, and memory are historically called buses, although the term means shared parallel wires and most I/O connections today are closer to dedicated serial lines.
- There're three types of buses required for I/O communication: address bus, data bus, and control bus. We assign an address to each I/O device for the CPU to communicate to that device using its address.
- Communication among the devices and the processor uses both interrupts and protocols
- Data rate: The peak rate at which data can be transferred between the I/O device and the main memory or processor.
- It is useful to know the maximum demand the device may generate when designing an I/O system.



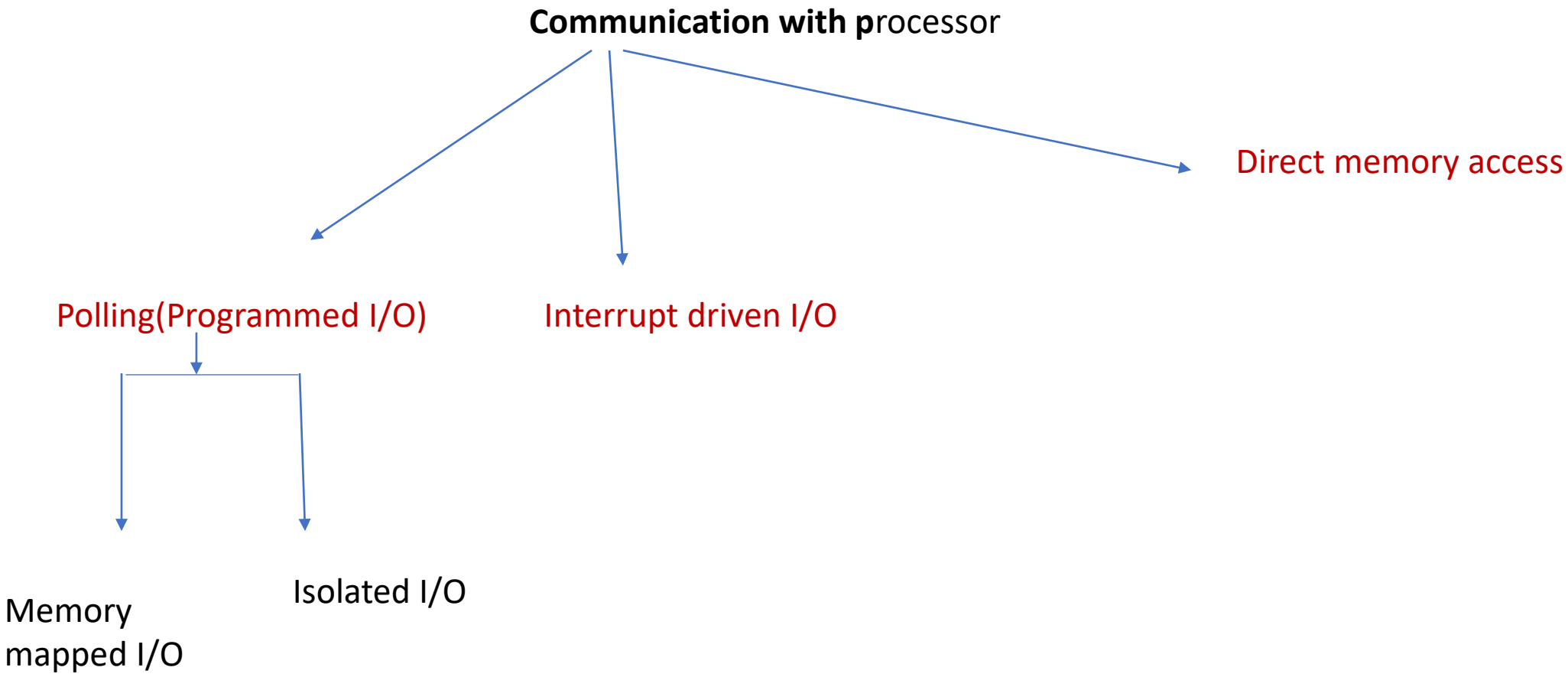
Input output devices

- If the I/O requests are extremely large, response time will depend heavily on bandwidth.

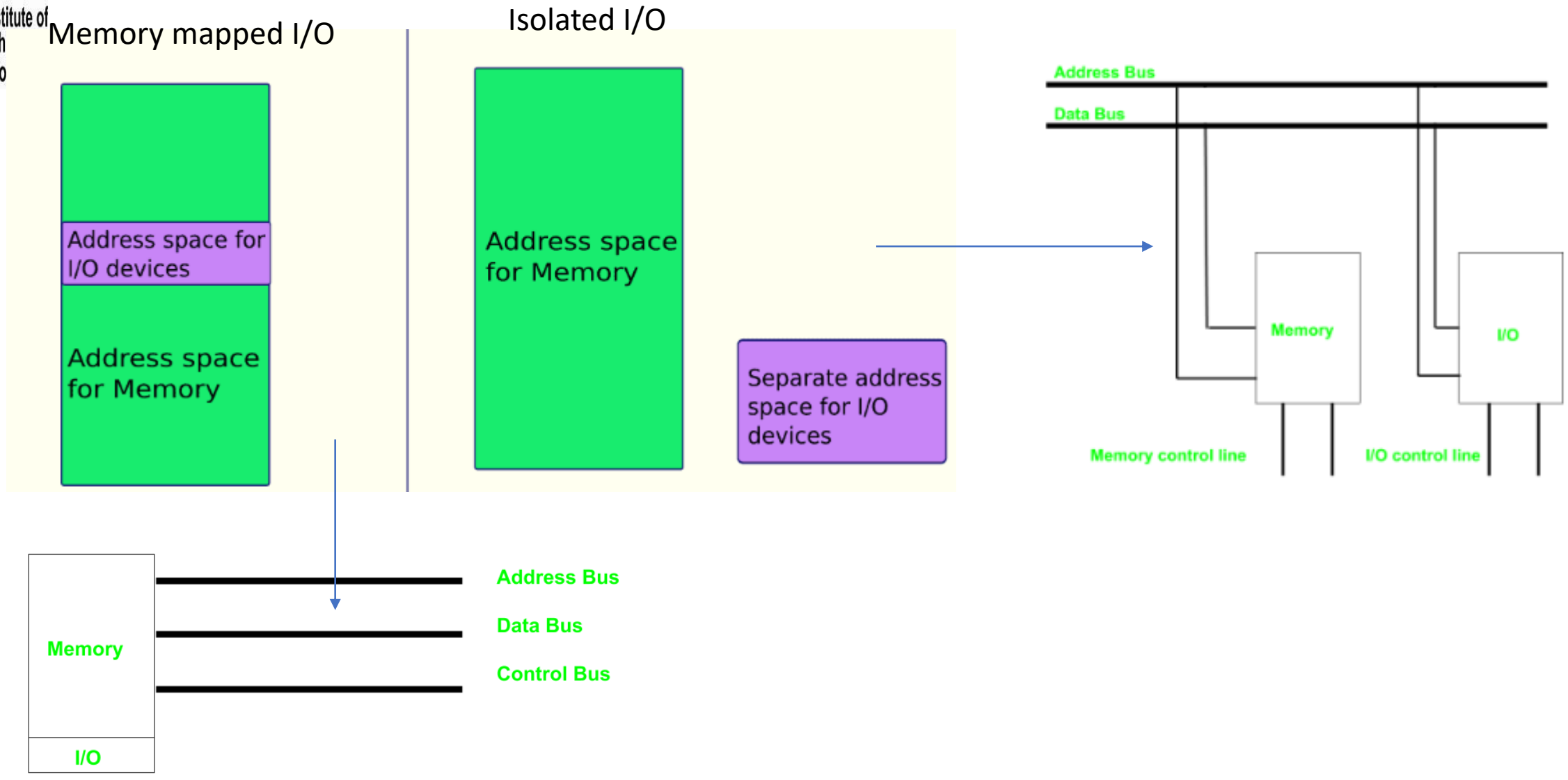
Input output devices

- I/O instructions control devices
- Devices usually have registers where device driver places
  - commands,
  - addresses, and
  - data to write, or read data from registers after command execution
- Data-in register, data-out register, status register, control register
- Typically 1-4 bytes, or FIFO buffer

Communicating with processor-I/O operations



Difference between Memory mapped I/O and isolated I/O



Difference between Memory mapped I/O and isolated I/O

Indian Institute of Information Technology Kottayam

- In memory-mapped I/O, both memory and I/O devices use the same address space. We assign some of the memory addresses to I/O devices. The CPU treats I/O devices like computer memory. The CPU either communicates with computer memory or some I/O devices depending on the address. Therefore, we reserve a part of the address space for I/O devices, which is not available for computer memory.
- In the case of memory-mapped I/O, all the buses are the same for both memory and I/O devices. Therefore, building a CPU that uses memory-mapped I/O is easier and cheaper. Additionally, such CPUs consume less power due to reduced complexity. One advantage of memory-mapped I/O is that we don't need separate instruction sets for accessing I/O devices. Instructions used for accessing memory can be easily used for accessing I/O devices.

Difference between Memory mapped I/O and isolated I/O

- In the case of isolated I/O, we provide a separate address space other than a memory address space to I/O devices. The addresses of I/O devices are also referred to as ports. I/O devices and memory use the same address and data bus. However, the control bus is different for data and memory:
- Therefore, isolated I/O becomes costlier compared to memory-mapped I/O. The isolated I/O technique has its own dedicated instruction set for accessing I/O devices. The CPUs that use isolated I/O are bigger and more complex to build.

Difference between Memory mapped I/O and isolated I/O

Indian Institute of Information Technology Kottayam

Isolated I/O	Memory-mapped I/O
Different address spaces are used for computer memory and I/O devices. I/O devices have dedicated address space.	Same address space is used for memory and I/O devices.
Separate control unit and control instructions are used in case of I/O devices.	Control units and instructions are same for memory and I/O devices.
More complex and costlier than memory -mapped I/O as more bus are used.	Easier to build and cheap as it's less complex.
Entire address space can be used by memory as I/O devices have separate address space.	Some part of the address space of computer memory is consumed by I/O devices as address space is shared.
Computer memory and I/O devices use different control instructions for read write.	Computer memory and I/O devices can both use same set of read and write instructions.
Separate control bus is used for computer memory and I/O devices. Though same address and data bus are used.	Address, data and control bus are same for memory and I/O devices.

need separate instruction sets for accessing I/O devices

separate instruction sets for accessing I/O devices

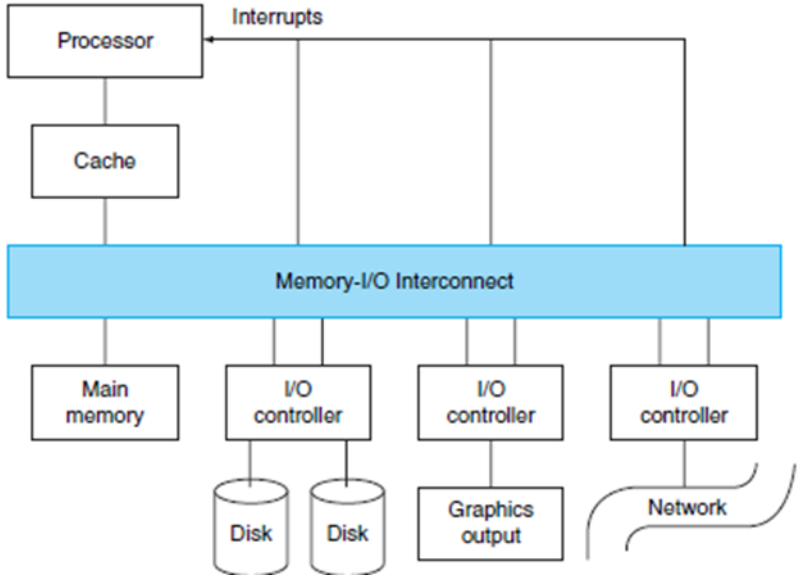
Communicating with processor

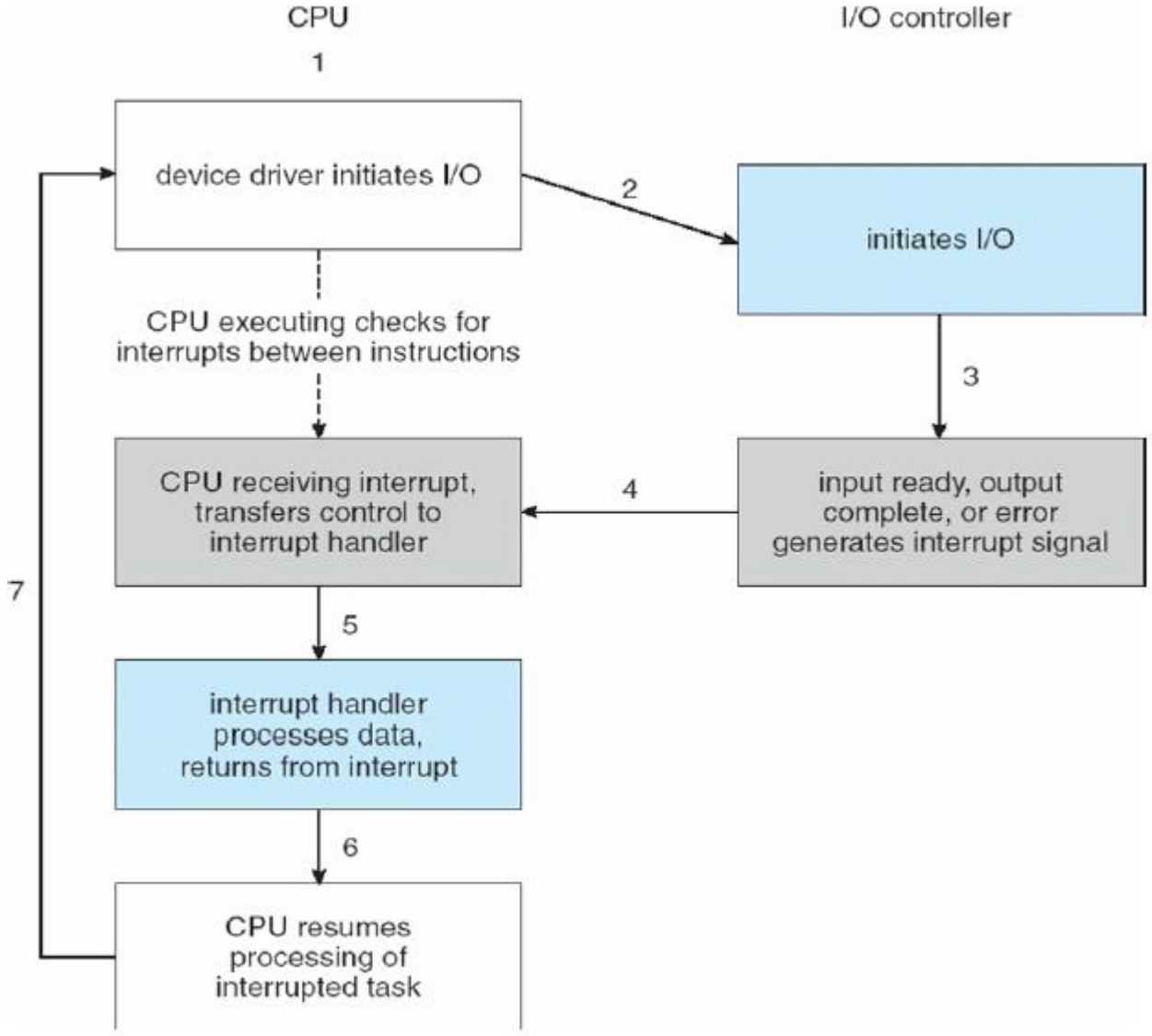
- Three techniques are possible for data transfer between processor and I/O device
- Polling (Programmed I/O)
  - process of periodically checking the status of an I/O device to determine the need to service the device.
  - If there isn't, it keeps executing its normal workflow. If there is, it handles the I/O request instead.
  - When the processor issues a command to the I/O module, it must wait until the I/O operation is complete. If the processor is faster than the I/O module, this is wasteful of processor time.
- Interrupt driven I/O
  - An I/O scheme that employs interrupts to indicate to the processor that an I/O device needs attention.
  - With interrupt-driven I/O, the processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work.
- Direct Memory access
  - In this mode, the I/O module and main memory exchange data directly, without processor involvement.
  - Mechanism that provides a device controller with the ability to transfer data directly to or from the memory without involving the processor.

Communicating with processor

- Both polling and interrupt techniques work best with lower-bandwidth devices
- For high-bandwidth devices like hard disks, the transfers consist primarily of relatively large blocks of data (hundreds to thousands of bytes) DMA is used

Interrupt driven I/O



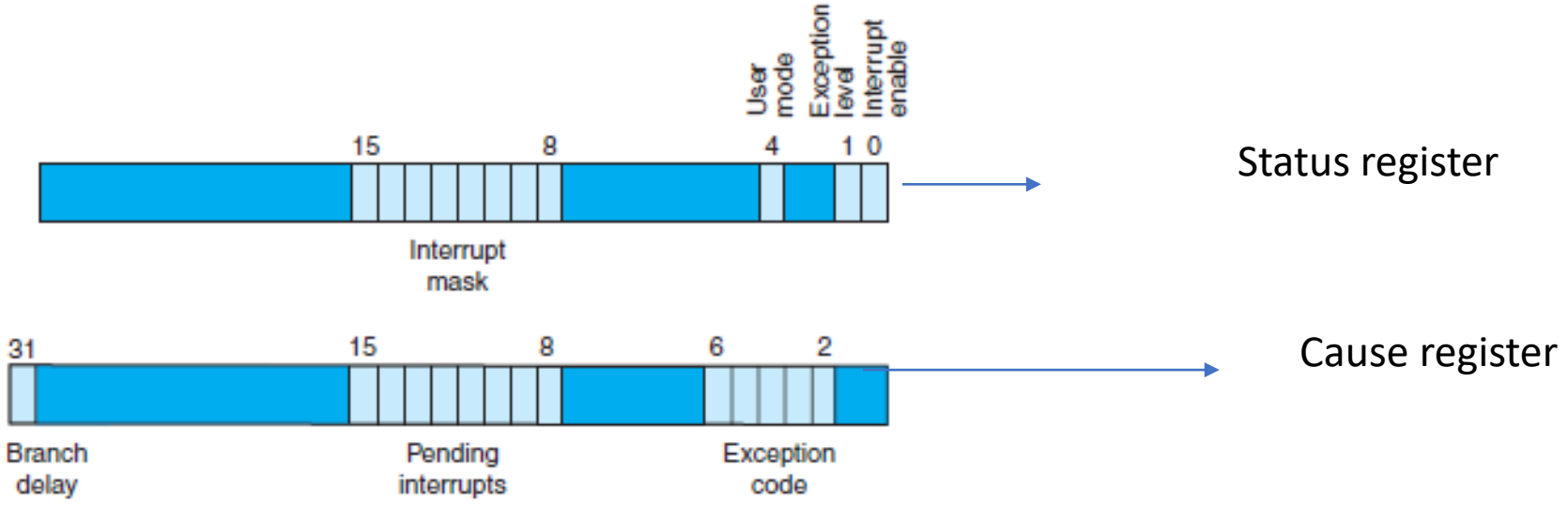


Interrupt driven I/O

- Our control unit need only check for a pending I/O interrupt at the time it starts a new instruction.
- To communicate information to the processor, such as the identity of the device raising the interrupt, a system can use either vectored interrupts or an exception Cause register.
- When the processor recognizes the interrupt, the device can send either the vector address or a status field to place in the Cause register.
- As a result, when the OS gets control, it knows the identity of the device that caused the interrupt and can immediately interrogate the device. An interrupt mechanism eliminates the need for the processor to poll the device and instead allows the processor to focus on executing programs.

Interrupt driven I/O

Cause and Status registers



The Status register determines who can interrupt the computer.
If the interrupt enable bit is 0, then none can interrupt. A more refined blocking of interrupts is available in the interrupt mask field.
There is a bit in the mask corresponding to each bit in the pending interrupt field of the Cause register.
To enable the corresponding interrupt, there must be a 1 in the mask field at that bit position.
Once an interrupt occurs, the operating system can find the reason in the exception code field of the Status register: 0 means an interrupt occurred.

Steps to process interrupt

Indian Institute of Information Technology Kottayam

1.Logically AND the pending interrupt field and the interrupt mask field to see which enabled interrupts could be the culprit

2. Select the higher priority of these interrupts. The software convention is that the leftmost is the highest priority.

Save the interrupt mask field of the Status register.

4. Change the interrupt mask field to disable all interrupts of equal or lower priority.

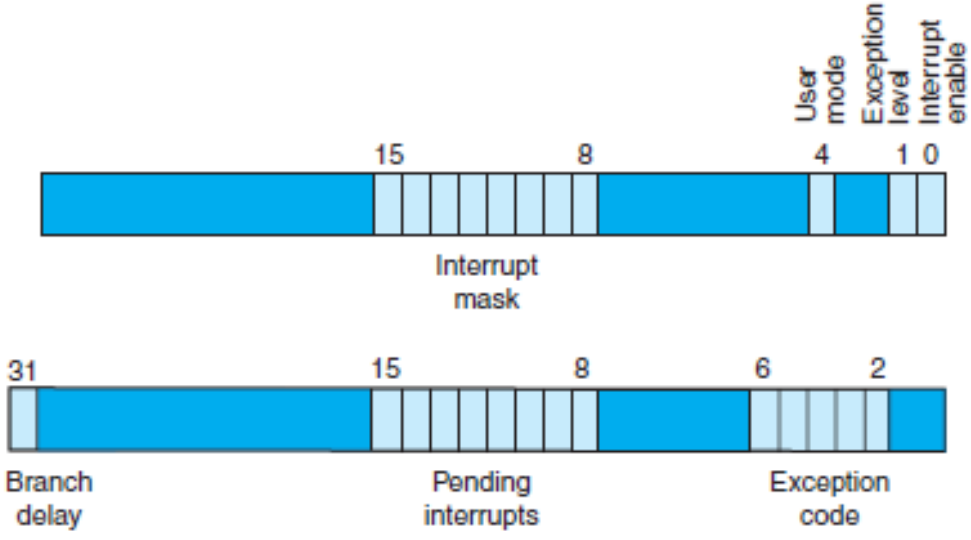
5. Save the processor state needed to handle the interrupt.

6. To allow higher-priority interrupts, set the interrupt enable bit to 1.

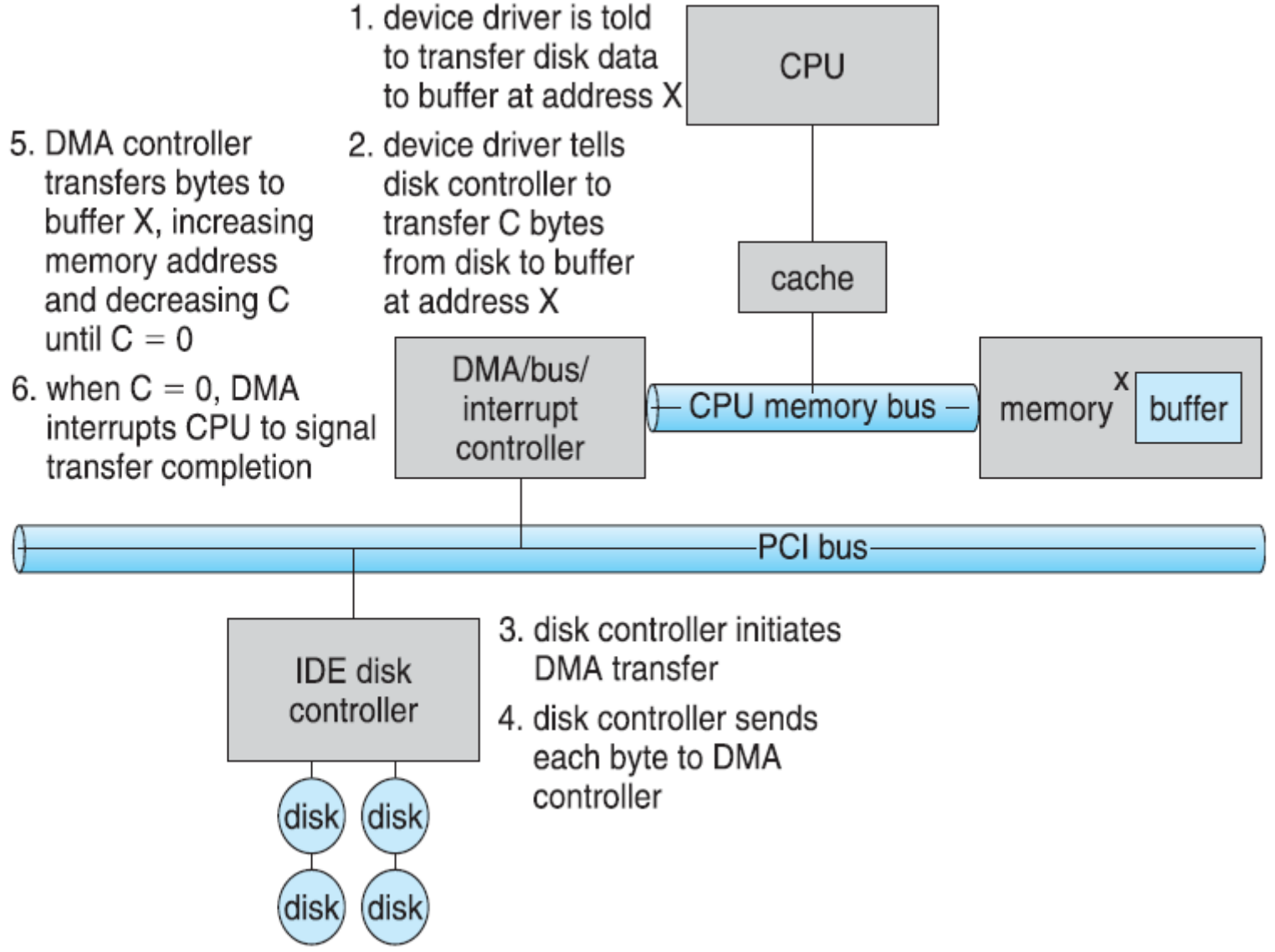
7. Call the appropriate interrupt routine.

8. Before restoring state, set the interrupt enable bit to 0.

This allows you to restore the interrupt mask field.



DMA


1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion
CPU
cache
DMA/bus/ interrupt controller
CPU memory bus
memory
x
buffer
PCI bus
IDE disk controller
disk
disk
disk
disk

DMA

Indian Institute of Information Technology Kottayam

- Used to avoid programmed I/O (one byte at a time) for large data movement
- Requires DMA controller
- Bypasses CPU to transfer data directly between I/O device and
- memory
- OS writes DMA command block into memory
- Source and destination addresses
- Read or write mode
- Count of bytes
- Writes location of command block to DMA controller
- Bus mastering of DMA controller – grabs bus from CPU
- Cycle stealing from CPU but still much more efficient
- When done, interrupts to signal completion

DMA

Buses

Indian Institute of Information Technology Kottayam

- Input/output: I/O performance measures, types and characteristics of I/O devices, buses, interfaces in I/O devices, design of an I/O system, parallelism and I/O. Introduction to multicores and multiprocessors

A typical PC bus structure-I/O interface