CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science & Technology

www.city.ac.uk

Systems Architecture

IN1006

The Memory Hierarchy

—Dr Asad Haziful

Where are we?

- Components of computers
- Data representation
- Logic Dates – computer circuits
- Simple computer, assembly programming – MARIE
- **Memory hierarchy**
- Pipelining and parallelism
- System Software

Contents

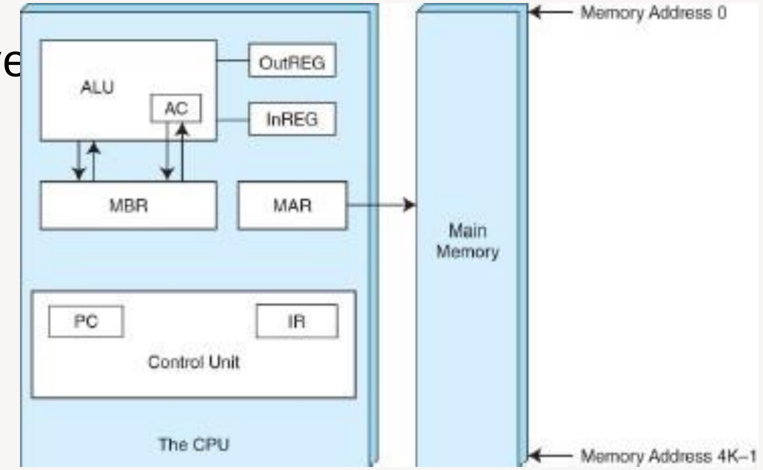
- Types of Memory (Storage)
- The Memory Hierarchy
- Caching basics
- Caching - Direct Mapped Cache
- Performance - Effective Access Time
- Introduction to Virtual memory

Question

- What do you think why do we have several types of memories (RAM, Registers, ROM, secondary memory, Cache) in a computer?

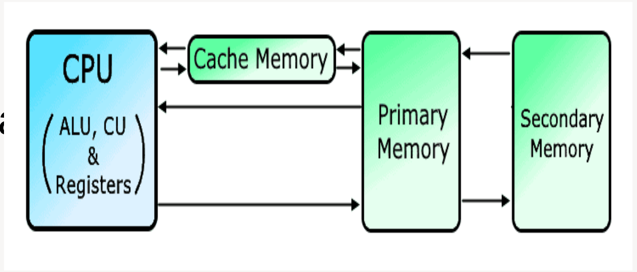
Memory in a simple computer

- Data and instructions must be transferred from memory to CPU registers to execute programs
- But, CPU can access one memory leve
- Different kinds of storage have
  - Different speeds
  - Different capacities
  - Different durations
  - Different costs



Solution: think and optimise the average case

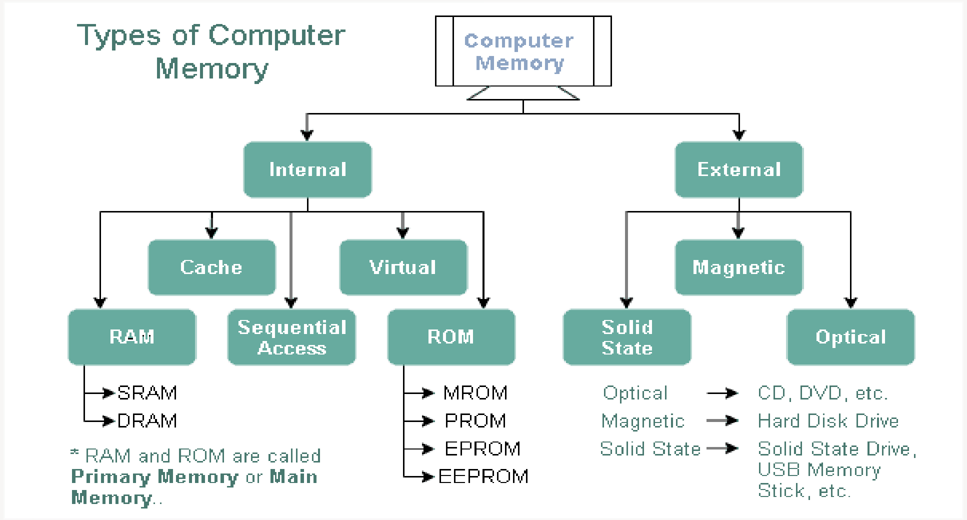
- The goal:
  - To create a memory system that is fast, large & cheap
    - Large memories are slow
    - Fast memories are small (and expensive)
- The solution
  - To organise memory so that **typical/average a**
  - **MEMORY HIERARCHY**



https://www.electricaltechnology.org/2020/05/types-computer-memory.html

Types of Memory

- Cache
- Main memory
- Secondary memory
- Virtual memory



https://www.tutorialsmate.com/2020/04/types-of-computer-memory.html

CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

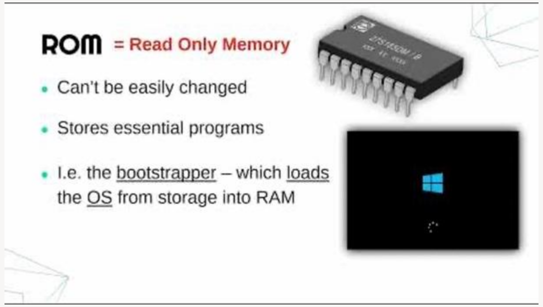
IN1006

The Memory Hierarchy: Main/Secondary Memories

—Dr H. Asad

Types of Main Memory

- There are two kinds of main memory:
  - *Random access memory, **RAM*** (read-write memory)
  - There are two types of RAM:
    - dynamic RAM (DRAM)
    - static RAM (SRAM)
- *Read-only-memory (**ROM)***



ROM = Read Only Memory

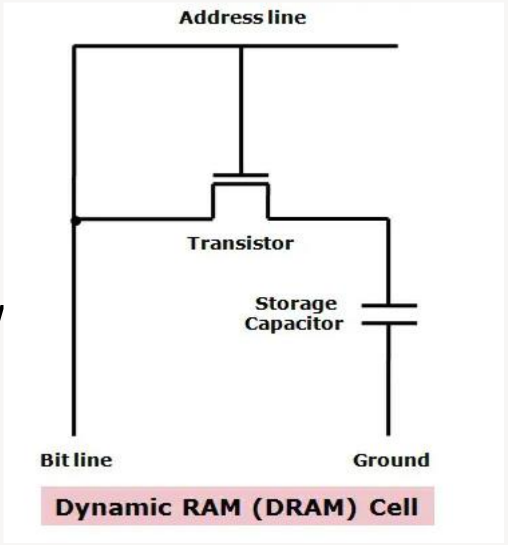
- Can't be easily changed
- Stores essential programs
- I.e. the bootstrapper – which loads the OS from storage into RAM

Dynamic RAM (DRAM)

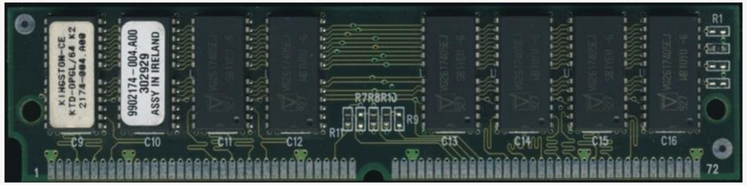
- A bit is stored in a capacitor
- DRAM consists of **capacitors** that **slowly leak** their charge over time.

- Thus, they **must be refreshed** every few data loss.

  - Low power consumption
  - Small area
    - 1 transistor + 1 capacitor
  - Low costs, **cheap memory**

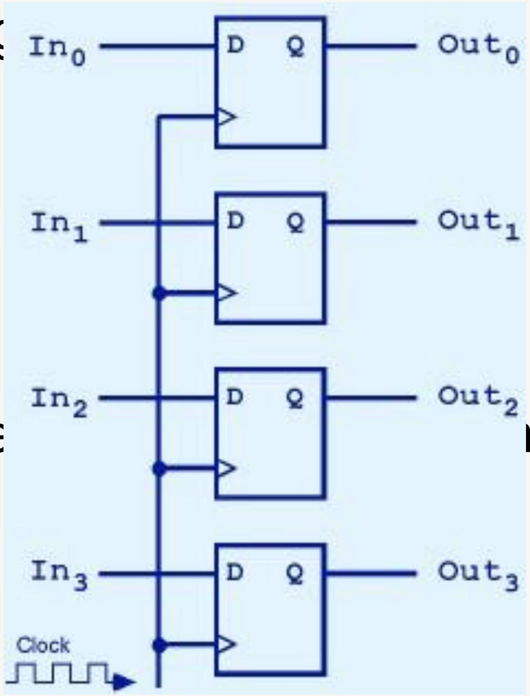


https://microcontrollerslab.com/difference-of-sram-and-dram/



Static RAM (SRAM)

- A bit is stored by six transistors working
  - Fast access
  - High power
  - Large area
    - 6 transistors / cell
    - 2 lines: bit and negated bit line
  - High cost
  - SRAM is very fast memory and it doe ned
  - It is used to build **cache memory**.



DRAM vs SRAM

- DRAM is used for *main memory*
- SRAM is used for *cache*

Read Only Memory (ROM)

- Data stored in these chips is **non volatile**
  - it is not lost when power is removed.

- ROM is used to store permanent, or semi-permanent data that while the system is turned off
  - E.g., program necessary to boot the computer.

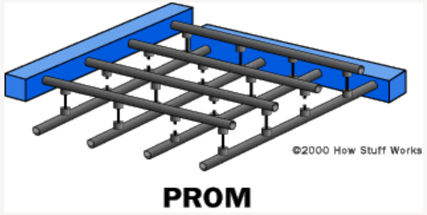
- Data stored in these chips is either unchangeable or requires a special operation to change

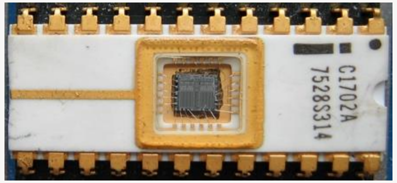
- ROM is often programmed as part of the manufacturing process
- Used in Toys, embedded systems, doors, washing machines etc.

Types of ROM

Five basic types of ROM:

- ROM
- **Programmable ROM (PROM)**
- **Erasable PROM (EPROM)**
- **Electronically Erasable Programmable ROM (EEPROM)**
- FLASH





EPROM

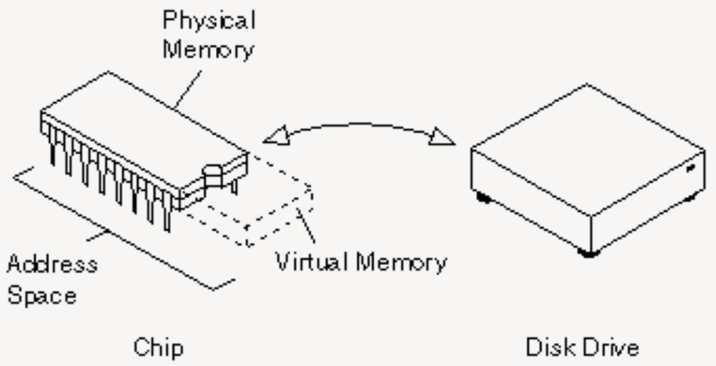
FLASH

EEPROM

http://upload.wikimedia.org/wikipedia/commons/c/cf/Viking_Modular_MO-297_SATA_SSD.jpg

What is virtual memory?

- Virtual memory is a section of **volatile** memory created temporarily on the **storage** drive. It is created when a computer is running many processes at once and **RAM** is running low.
- The **operating system** makes part of the storage drive available to use as RAM.
- Virtual memory is much slower than main memory because processing power is being taken up by moving data around, rather than just executing instructions.



CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science & Technology

www.city.ac.uk

Systems Architecture

IN1006

The Memory Hierarchy

—Dr H. Asad

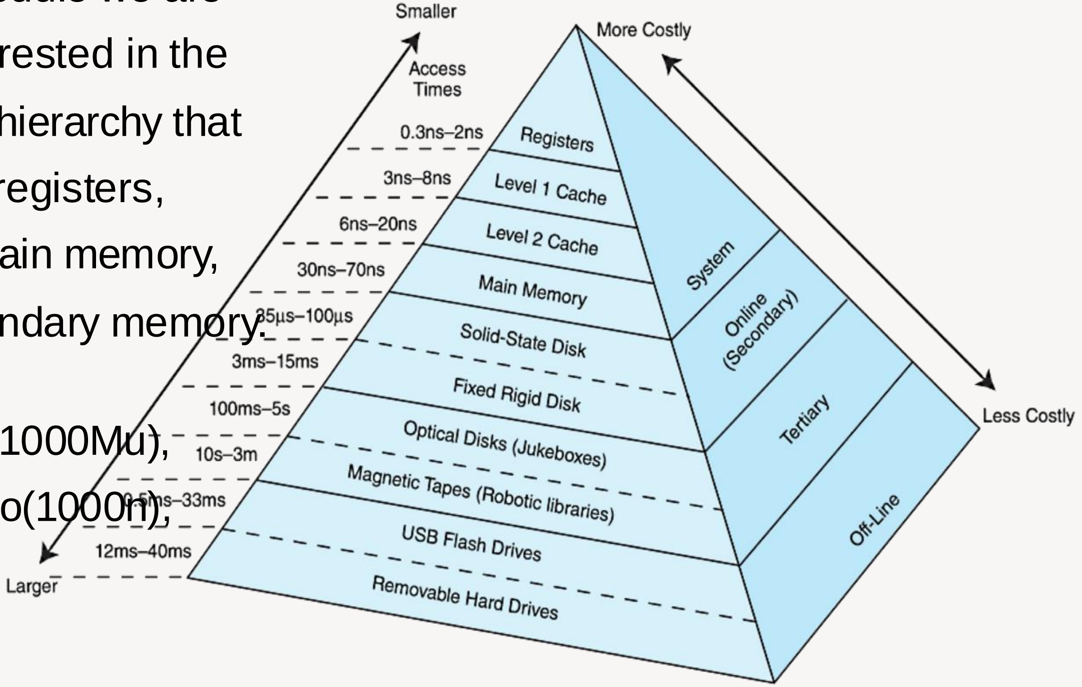
The Memory Hierarchy

- What is a memory hierarchy?
  - A **way of organising memory** in a computer
  - Such that it meets **speed, capacity** and **cost requirements**
- Why do we need a memory hierarchy?
  - We can’t afford (**$£ and power)** to use as much of the highest performance memory as we’d like
- How do we construct a memory hierarchy?
  - Using a variety of devices: registers, SRAM, DRAM, disks, ...

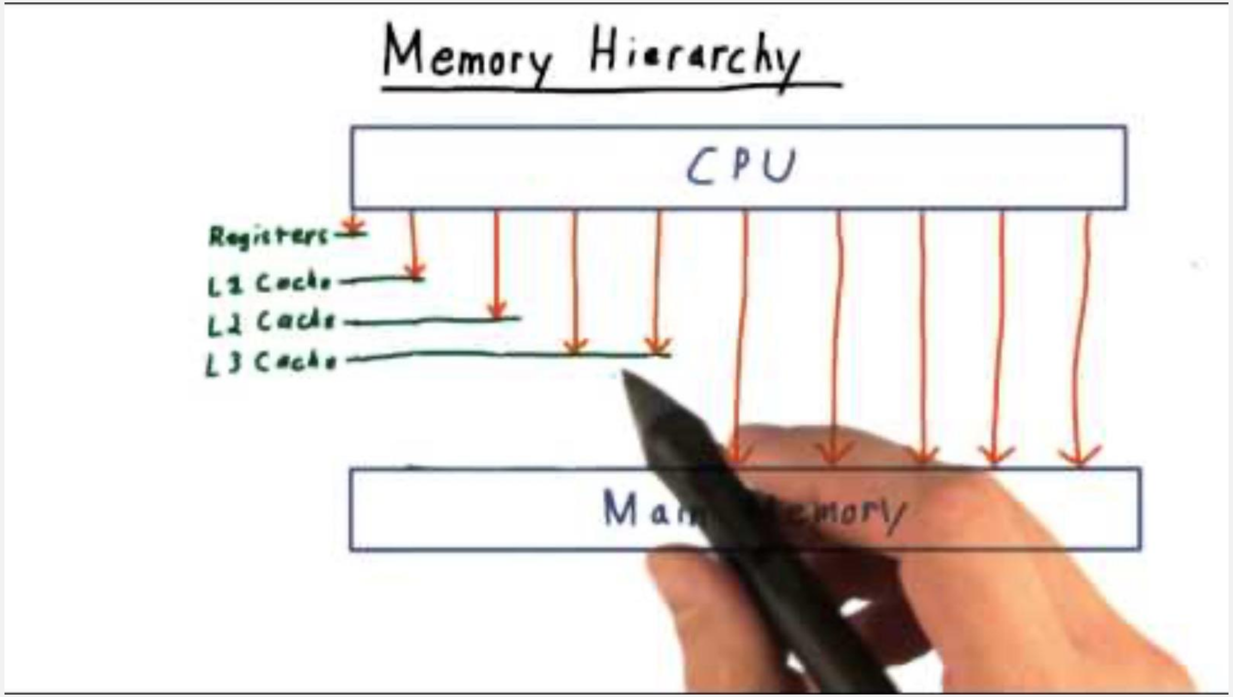
The Memory Hierarchy as a Pyramid

- In this module we are most interested in the memory hierarchy that involves registers, cache, main memory, and secondary memory.

  *m - milli(1000Mu),
  mu - micro(1000n),
  n - nano

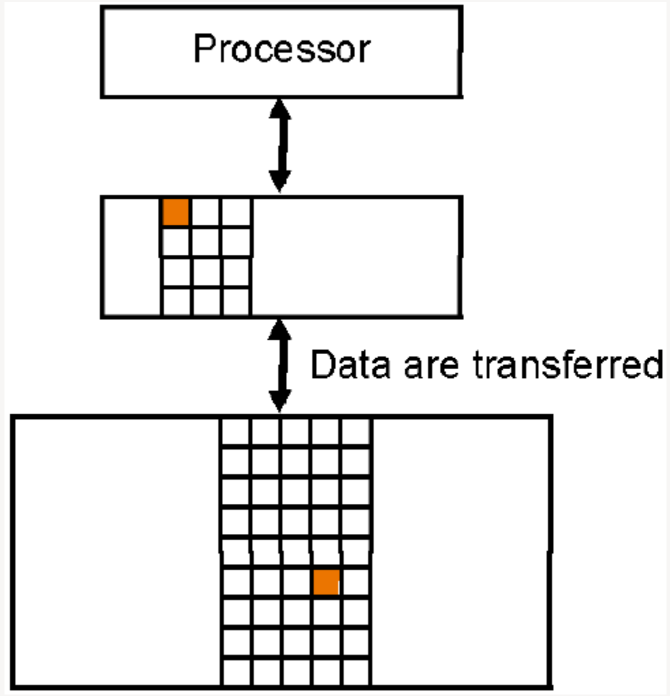


Memory Hierarchy contd…



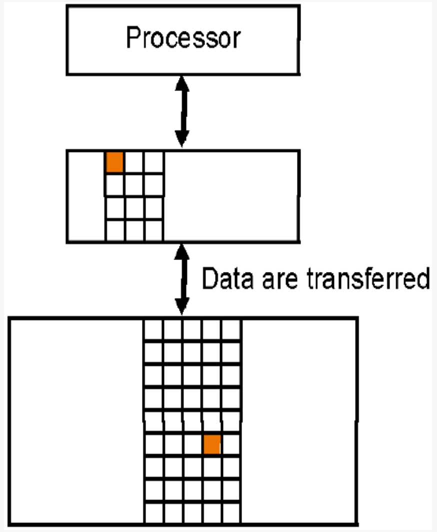
Communication Between Memory Levels

- Communication is between pairs of levels.
- There are clear protocols on how the levels interact.
- Within each level, the unit of information transfer is a **block.**



Three Steps to Locate and Fetch Data

1. To access a particular piece of data, the CPU first sends a request to its nearest memory level (apart from cash memory) ➔ main memory.
2. If the data is not in main memory (This is a miss), then the request goes to the secondary memory (i.e., the “disk”)
3. Once the data is located, then the data is moved into memory.



Hit and Miss

- **Hit**: data is locally available ➔ no penalty to access
- **Miss**: data is on a lower level ➔ penalty from **1-30 clock cycles to access**
- **Hit rate**: is the percentage of time data is found at a given memory level.
- **Miss rate**: is the percentage of time it is not.
- **Miss rate = 1 - hit rate**.
- **Hit time**: is the time required to access data at a given memory level.
- **Miss penalty**: is the time required to process a miss, including the time that it takes to replace a block of memory plus the time it takes to deliver the data to the processor

School of Science & Technology

www.city.ac.uk

Systems Architecture

IN1006

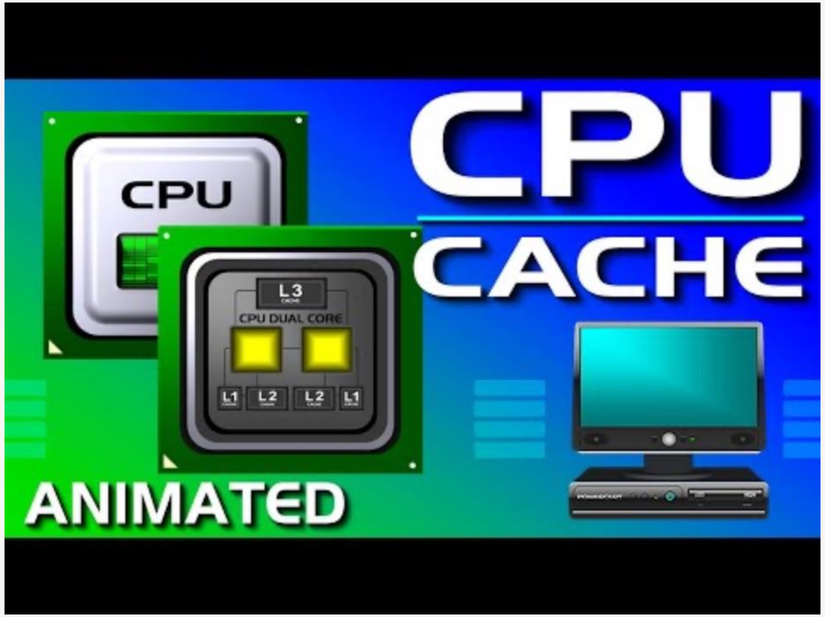
The Memory Hierarchy: Cache

—Dr H. Asad

Cache - A Fundamental Concept

- A **Cache** is a ”small local” storage medium
- What information should go into a cache?
  - Things that you expect to use **often** or **soon**.
  - BUT – **How do you predict which ones?**
    - We look at access patterns and
    - We use the principles of **Locality of Reference observation (sequence of memory addresses access)**

Steps to Locate and Fetch Data from Cache



Let’s look at memory access times:

Read from register:	**1 clock cycle (2 ns)**
Read from 1st level (L1) cache:	**1 to 2 cycles (2-4 ns**),8KByte – 64KB(per core)
Read from 2nd level (L2) cache:	**4 to 5 cycles (8-10 ns),** 256KB – 512KB
Read from main memory:	**8 to 30 cycles** (16-60 ns)

- When accessing from L1 or L2 cache we can get data faster

Basic Cache Questions

1. Is a data item in the cache?
2. If it is, how do we find the data item in the cache?

- Let’s first look at how the CPU requests data
  - The CPU first generates a **main memory address**
  - If the data is in cache, then we need **a way to “convert” the main memory address into a cache location**
  - The process of converting main memory addresses to cache locations is called **cache mapping**

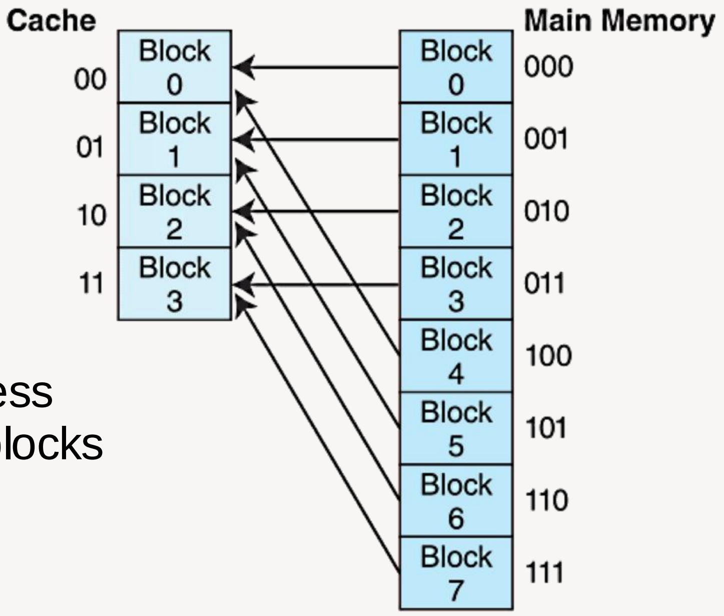
Cache Mapping methods

- There are three basic methods used for mapping of information fetched from the main memory to the cache memory:
  - direct mapping
  - associative mapping
  - set-associative mapping

We look into the **direct mapped cache** mapping.

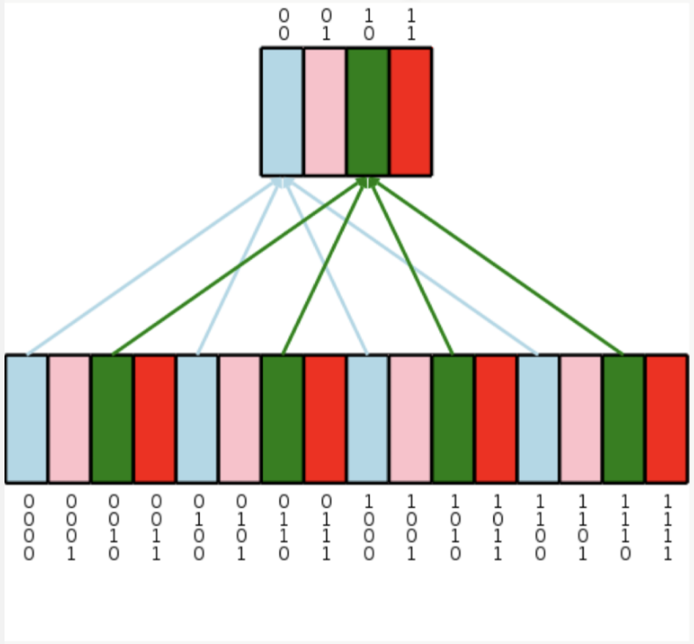
A Simple Approach: Direct Mapped Cache

- This is the case when Block= 1 word
- 8 block locations in main memory
  - Memory block address = 3 bits
- Cache with 4 blocks
  - Cache block address (2 bits)
    - Lower 2 bits of memory block address
    - (main memory address) *modulo* 4 blocks

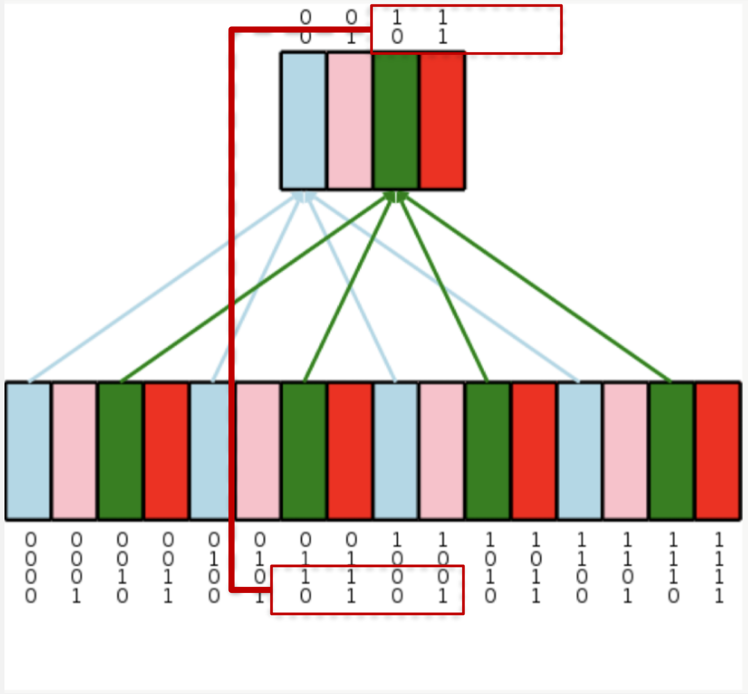


A Simple Approach: Direct Mapped Cache (cont’d)

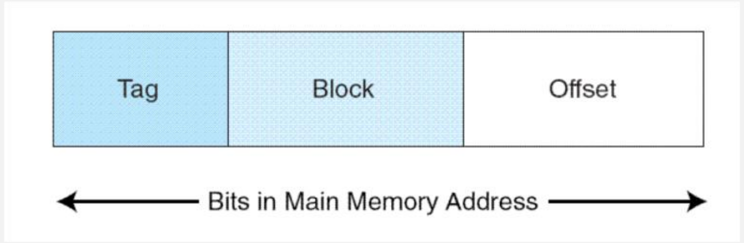
Here: This is the case when Block= 1 word



A Simple Approach: Direct Mapped Cache (cont’d)



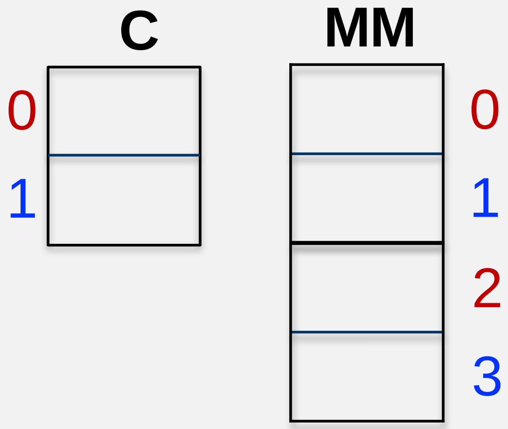
Partitioning of Memory Address Fields



- Usually Block size= Multiple words (may be byte or bytes)
- To perform direct mapping, the binary main memory address is partitioned into the *fields* shown below.
  - The ***block*** field selects a unique group of blocks of cache
  - The ***tag*** field selects the exact block in the group
  - The ***offset*** field uniquely identifies an address within a specific block.
- The sizes of these fields are determined by characteristics of both memory and cache.

Cache and Bits, Example 6.1 in Book

- Consider a byte-addressable main memory consisting of
  - 4 blocks
  - a cache with 2 blocks
  - each block is 4 bytes.
- This means Block 0 and 2 of main memory map to Block 0 of cache, and Blocks 1 and 3 of main memory map to Block 1 of cache.
- Using the tag, block, and offset fields, we can see how main memory maps to cache as follows: (next slide)

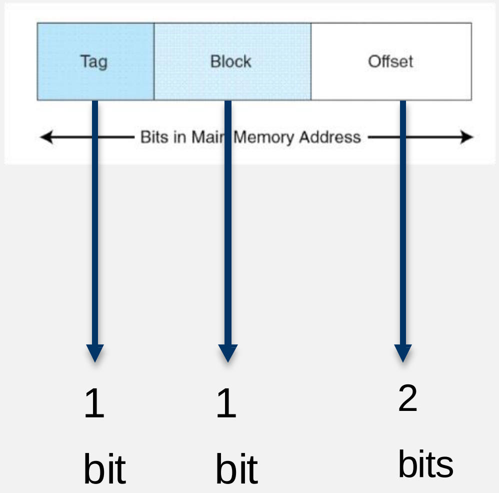


Cache and Bits, Example 6.1 in Book

- First, we need to determine the address format for mapping.
- Each block is 4 bytes and is byte-addressable

  offset field must contain 2 bits (00, 01, 10, 11)

- There are 2 blocks in cache, so the **block** field must contain 1 bit (0 for 1st block, 1 for 2nd block);
- The **memory (physical) address** requires 4 bits because there are a total of 16 words (4 blocks x 4 words each) and $16=2^4$).
- Tag is 1 bit

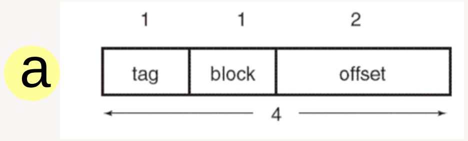


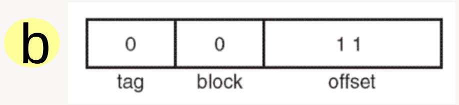
Tag bits = Physical address bits– (Block+offset)

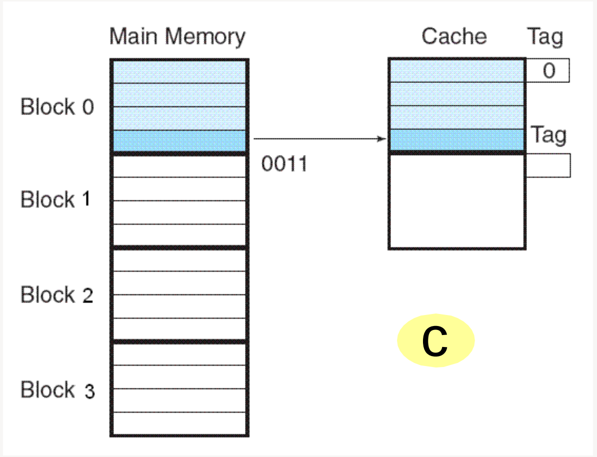
= 4 – (2+1) = 1

Cache and Bits, Example 6.1 in Book (cont'd)

- Suppose we need to access main memory address 0x0011 in binary. If we partition 0x0011 using the address format from Figure a, we get Figure b.
- Thus, the main memory address 0x0011 maps to cache block 0.
- Figure c shows this mapping, along with the tag that is also stored with the data







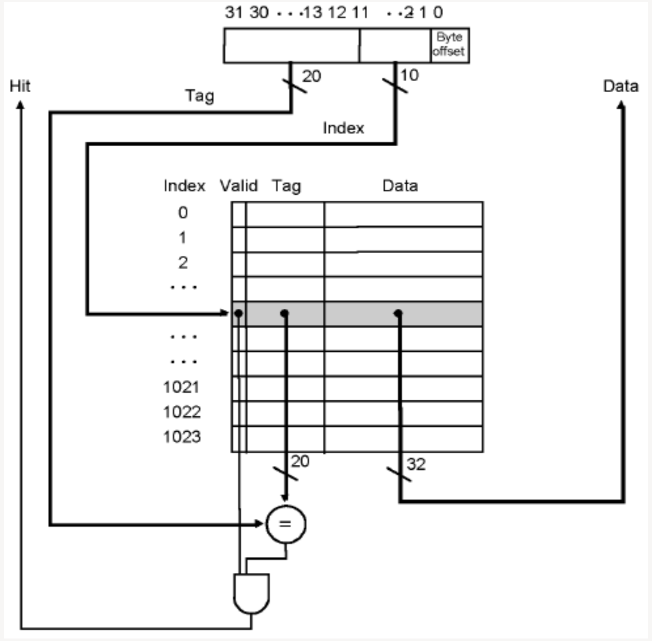
Direct Mapped Cache

- Memory locations are mapped to unique cache positions
  - Cache location = Block Address modulo Number of Cache Blocks
- QUESTION: In this way, we have two blocks mapping to the same cache location. How do we know that we are accessing the right one?
- SOLUTION: Extra data stored in cache:
  - **Tag Bits**
    - Give high order bits of address
    - It is stored into the cache with the corresponding block
  - **Valid Bit**

  Indicates whether cache value = memory value

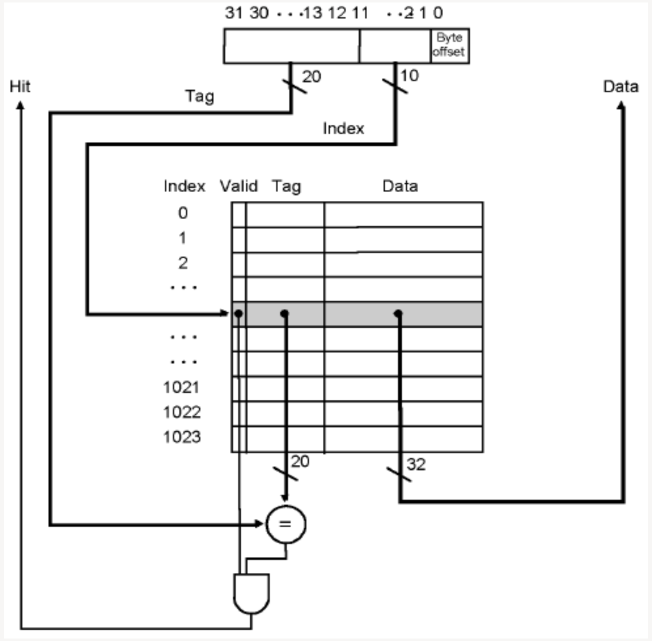
A More Realistic Cache

- Cache Organisation
- Data width: 32 bits
- Address width: 32 bits
- Cache size: 1024 blocks
  - Cache index: 10 bits (1024 = 2^{10})
- Block size: 1 word (4 bytes)
  - Byte offset: 2 bits
- Tag size: 32 – 10 - 2 = 20 bits
- Valid bit



A More Realistic Cache – Data Access

- Data Access
  1. Find cache index with address bits 11..2
  2. Compare cache tag with address bits 31..12
  3. Check valid bit to ensure data is loaded
  4. Signal a hit to the CPU
  5. Transfer data



CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science & Technology

www.city.ac.uk

Systems Architecture

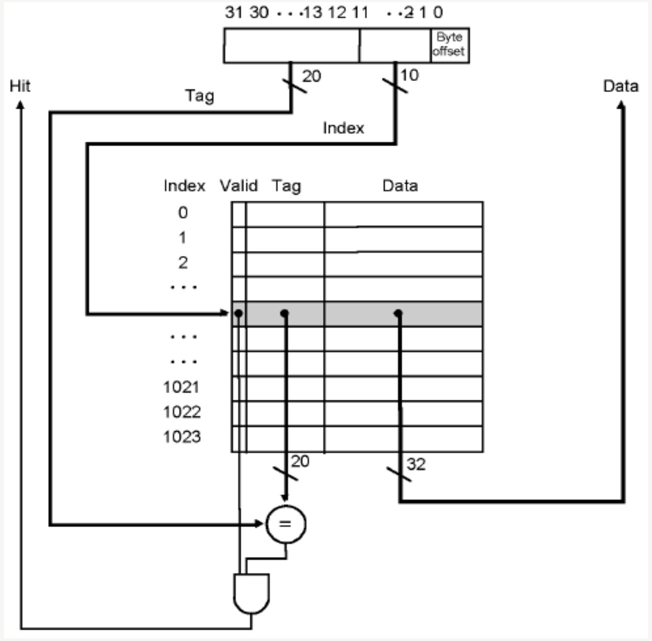
IN1006

The Memory Hierarchy: Cache Efficiency

—Dr H. Asad

Cache Memory Efficiency

- Memory Efficiency
  - Cache memory size (data)
    - 1024 x 32 bits = 32K bits
  - Tag memory size
    - 1024 x 20 bits = 20K bits
  - Valid bit
    - 1024 01 bit = 1K bits
  - Efficiency
    - 32 bits / 53 bits = 60.4%



Handling Data Cache Misses

- **What do we have to do if the data we need are not in the Cache?**
- Perform a Cache Miss process
- Stall the processor
  - Freeze contents of registers
- Activate memory controller
  - Separate controller handles memory fetch
- Request data item from next lower level of hierarchy
- Load data item into cache
  - Write data, store upper bits as tag, set valid bit
- Resume the processor

Effective Access Time (EAT)

- The performance of hierarchical memory is measured by its **effective access time (EAT).**
- EAT is a weighted average that takes into account the hit ratio and relative access times of successive levels of memory.
- The EAT for a two-level memory is given by:
  - $\mathbf{EAT = H \times Access_C + (1-H) \times Access_{MM}}$

  where H is the cache hit rate and $Access_C$ and $Access_{MM}$ are the access times for cache and main memory, respectively.
- For example: $Access_C$=10ns, $Access_{MM}$=200ns, H=0.99
  - EAT = 0.99(10ns) + 0.01(200ns) = 9.9ns + 2ns = 11.9ns.

Split vs Combined Caches

- **Split caches** for data and instructions (Harvard Architecture)
  - Higher miss rate due to their smaller size
  - Higher bandwidth due to separate data paths
  - Data and instructions can be cached simultaneously
- **Combined caches**
  - Lower miss rate due to their size
  - Lower bandwidth due to sharing of data paths
  - Data and instructions cannot be cached simultaneously

Cache Write policies

- **Write through**
  - Update cache and memory at the same time
  - Requires a buffer because the memory cannot accept data as fast as the processor can generate writes
  - Processor must be stalled only if buffer is full

- **Write back**
  - Keep data in cache and write back when it is being replaced
  - Requires more sophisticated cache contents replacement unit, potentially increasing the cost of a cache miss

How can we avoid cache misses?

- Instead of transferring a byte at-a-time, how can we optimise

the memory access time?

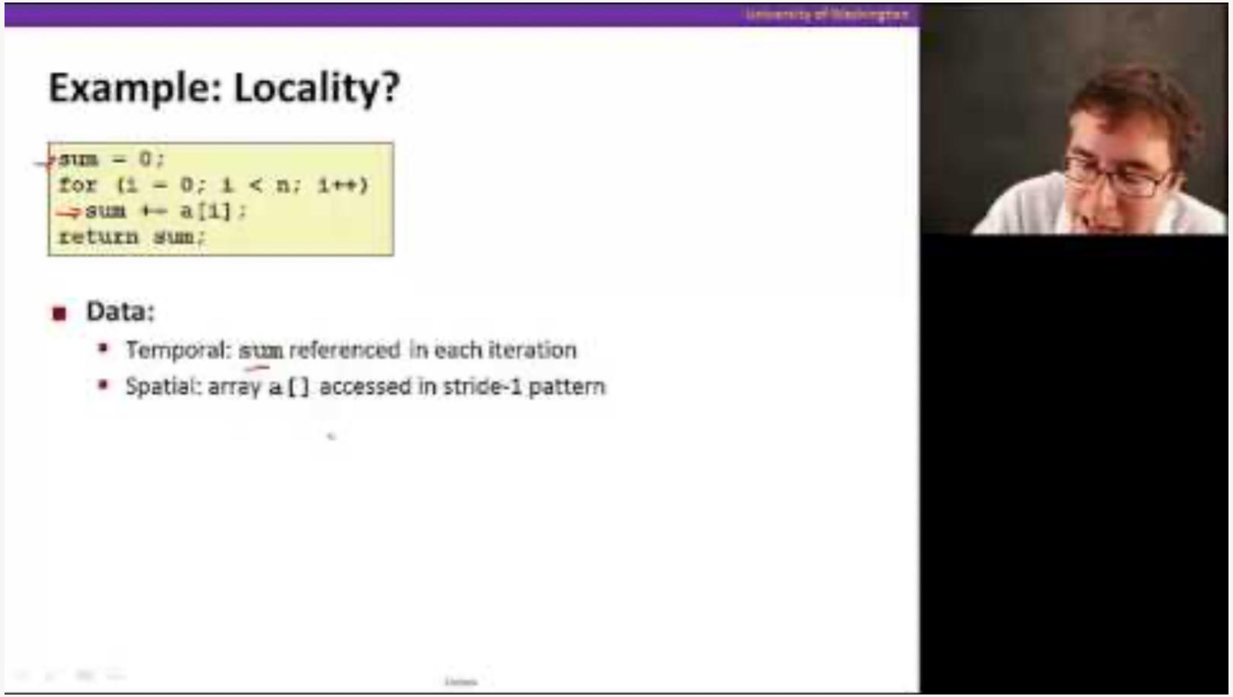
- An entire block of data is copied after a hit because the **principle of locality** tells us that once a byte is accessed, it is likely that a nearby data element will be needed soon.

- We call this observation **Locality of Reference**

- There are three basic forms of locality;

1) **Temporal Locality** ➔Items tend to get used more than once.
2) **Spatial Locality** ➔ Items stored together get used together.
3) **Sequential Locality** ➔ Instructions tend to be accessed sequentially.

Locality of Reference



University of Washington

Example: Locality?

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- **Data:**
  - Temporal: sum referenced in each iteration
  - Spatial: array a[] accessed in stride-1 pattern

Taking Advantage of Spatial Locality - Question

- Q: What data would you store in a block to exploit spatial locality?

  Think of arrays, Java classes, etc …

  - A: These objects can be stored in one block to take advantage of spatial locality. For example, in a Java class it is very likely once we use one field of the class that we will use another field soon.

Optimal Block Size

Small block size

- High miss rate
- Short block loading time
- Reduced benefits of spatial locality

Large block size

- Low miss rate
- Long block loading time
- Miss requires multiple words to be loaded

Summary

- Types of Memory (Storage)
- The Memory Hierarchy
- Caching basics
- Caching - Direct Mapped Cache
- Performance - Effective Access Time
- Introduction to Virtual memory

School of Science & Technology
City, University of London
Northampton Square
London
EC1V 0HB
United Kingdom

T: +44 (0)20 7040 5060
E: SST-ug@city.ac.uk
www.city.ac.uk/department

Acknowledgement:
Prof George Spanoudakis