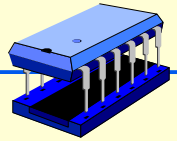


Main Memory Organisation

Eddie Edwards
eedwards@doc.ic.ac.uk

http://www.doc.ic.ac.uk/~eedwards/compsys

Main Memory Organisation – Learning Objectives

- At the end of this lecture you will:
  - Understand the main concepts of memory organisation
  - Know the different memory hardware options
    - Registers, cache, RAM, Disk
  - Understand the relationship between cost and speed of access
  - Understand byte ordering in words
  - Understand the concept of addressing
  - Understand word and byte addressing
  - Understand how RAM memory is organised into chips and modules
  - Comprehend high order and low order interleaving and their uses

Question

- Do you have a laptop or internet device here?

A yes **B** no **C** what's a laptop **D** where is here?

E none of the above

Memories

- Memories hold binary values. These can be:

  Data (e.g. Integers, Reals, Characters)

  CPU Instructions (i.e. Computer Programs)

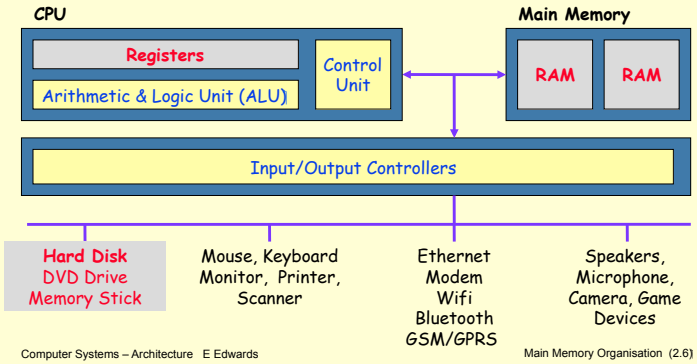
  Memory Addresses ("Pointers" to data or instructions)

- The **contents** of a memory remain unchanged unless overwritten with a new binary value. For some memories the contents are "lost" when power to the memory is turned off.

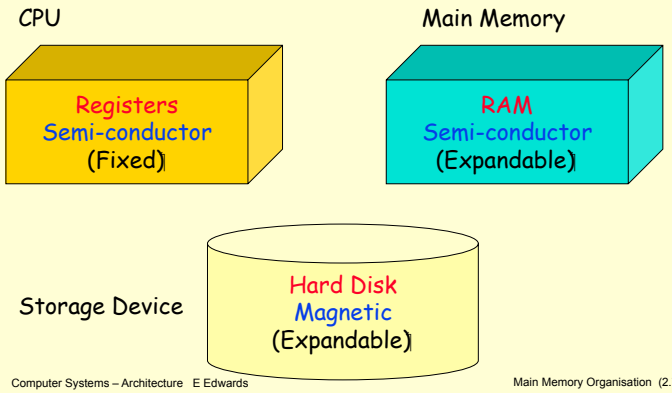
Examples

- **CPU**
  Registers
  Caches – L1, L2 [L3]
- **Motherboard**
  RAM (Random Access Memory)
  ROM (Read Only Memory)
  Caches
  I/O Registers & Buffers
  Videocard Memory
- **Storage Devices**
  Hard Disks, CDs, DVDs, Tapes,
  Memory Sticks, Flashcards

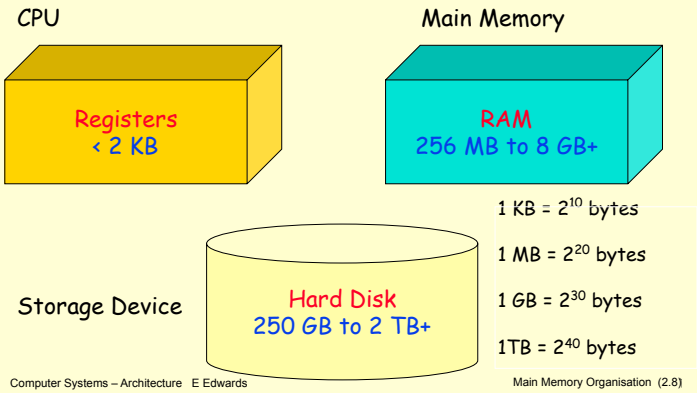
CPU Organisation



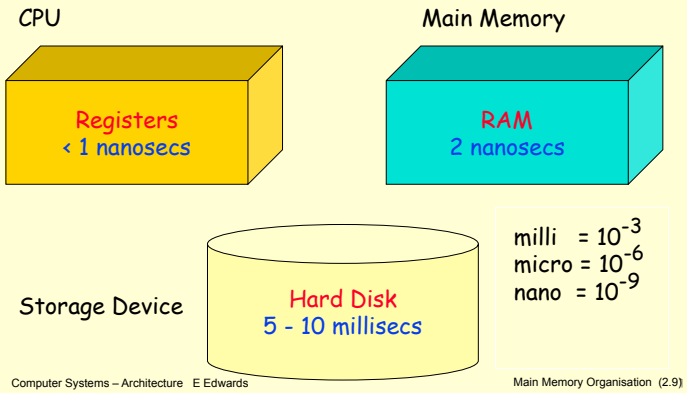
3 Types of Memory



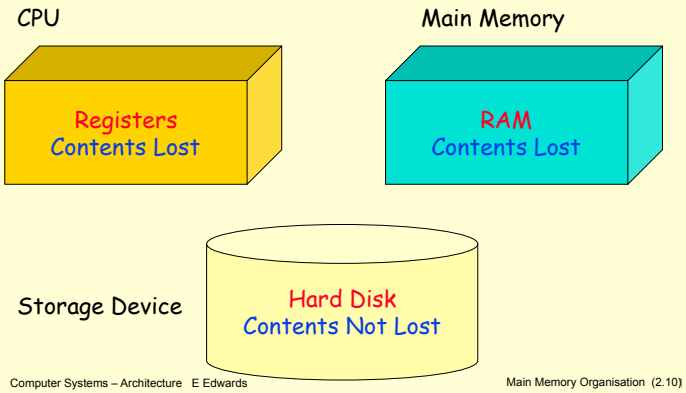
Capacity



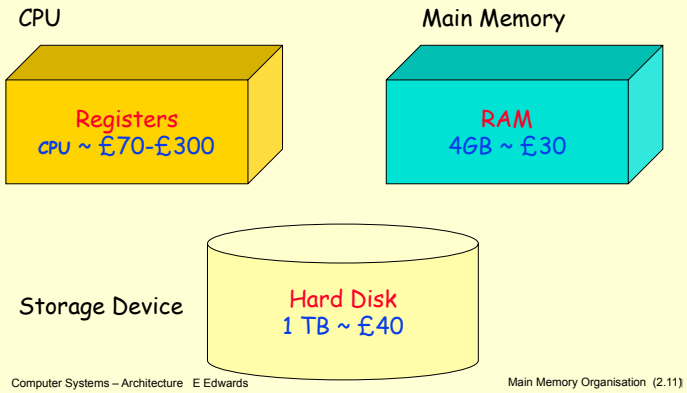
Speed (Access Time)



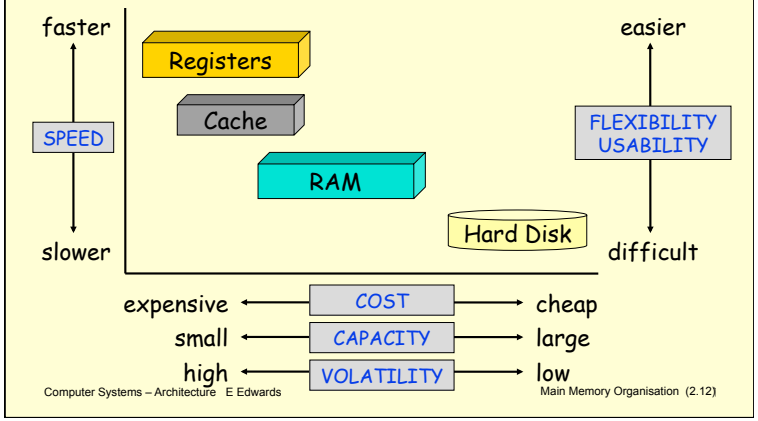
Volatility



Cost



Summary



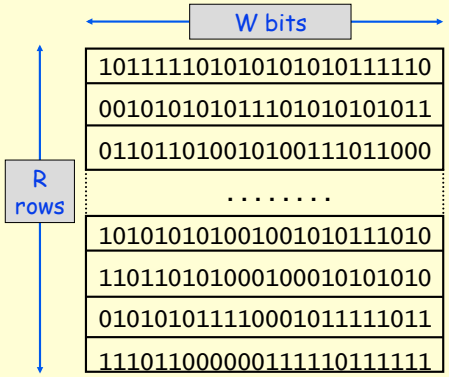
How many registers are there in a X86-64 processor?

A 2 **B** 4 **C** 16 **D** 64 **E** 256

How much memory is given to registers in a X86-64 processor?

A 16B **B** 128B **C** 256B **D** 32B **E** 1024B

Main Memory (RAM)



- **Memory Location** (Memory Word) = W bits. Normally a byte-multiple, e.g. 16-bits, 32-bits.
- **Memory Size** R x W *bits*
- **Access**: *** In our model we will always Read/Write a whole Memory Word at a time ***

Addressing

Main Memory

0110	1101	1010	1101
0000	0000	0000	0011
0000	0000	0000	0000
1111	1111	1111	1111
0000	0000	0000	0000
1001	1010	1010	0010
0000	0000	0000	0000
1111	1111	1111	1110

- Where in memory is the 16-bit two's Complement value 3?
- We need a scheme for uniquely identifying every memory location
- **ADDRESSING** Identify memory locations with a positive number called the (memory) **address**

Memory Word Addressing

Main Memory				Address	Address (binary)
0110	1101	1010	1101	← 0	0000
0000	0000	0000	0100	← 1	0001
0000	0000	0000	0000	← 2	0010
1111	1111	1111	1111	← 3	0011
0000	0000	0000	0000	← 4	0100
1001	1010	1010	0010	← 5	0101
0000	0000	0000	0000	← 6	0110
1111	1111	1111	1110	← 7	0111

Memory[1] = ?

Memory[?] = -1

Memory[?] = 0

Mem[Mem[1]]= ?

Byte Addressing

Main Memory				Word Address
0110	1101	1010	1101	← 0
0000	0000	0000	0011	← 2
0000	0000	0000	0000	← 4
1111	1111	1111	1111	← 6
0000	0000	0000	0000	← 8
1001	1010	1010	0010	← 10
0000	0000	0000	0000	← 12
1111	1111	1111	1110	← 14

- With byte addressing, every byte in main memory has an address
- In this example which is byte 0 and which is byte 1?

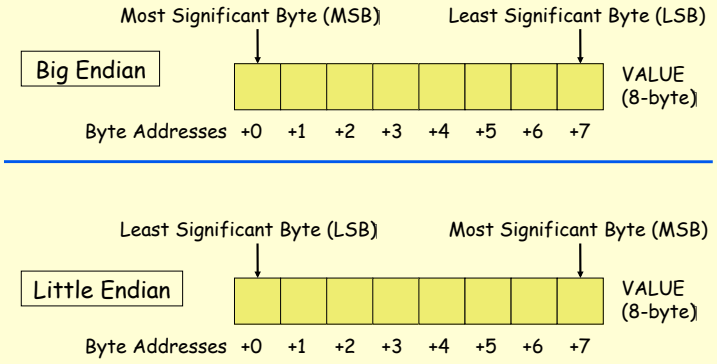
Byte Ordering (Big Endian)

Byte Address	Main Memory				Byte Address
0 →	0110	1101	1010	1101	← 1
2 →	0000	0000	0000	0011	← 3
4 →	0000	0000	0000	0000	← 5
6 →	1111	1111	1111	1111	← 7
8 →	0000	0000	0000	0000	← 9
10 →	1001	1010	1010	0010	← 11
12 →	0000	0000	0000	0000	← 13
14 →	1111	1111	1111	1110	← 15

Byte Ordering (Little Endian)

Byte Address	Main Memory				Byte Address
1 →	0110	1101	1010	1101	← 0
3 →	0000	0000	0000	0011	← 2
5 →	0000	0000	0000	0000	← 4
7 →	1111	1111	1111	1111	← 6
9 →	0000	0000	0000	0000	← 8
11 →	1001	1010	1010	0010	← 10
13 →	0000	0000	0000	0000	← 12
15 →	1111	1111	1111	1110	← 14

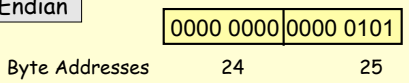
Byte Ordering of **Multibyte** Data Items



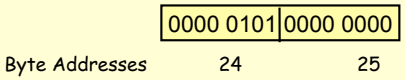
Example 1: 16-bit integer (View 1)

- 16-bit (2's Complement) integer 5 stored at memory address 24.

Big Endian



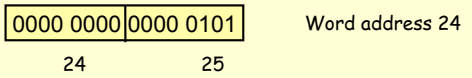
Little Endian



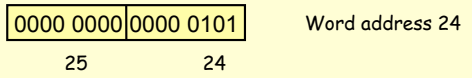
Example 1: 16-bit integer (View 2)

- 16-bit (2's Complement) integer 5 stored at memory address 24.

Big Endian



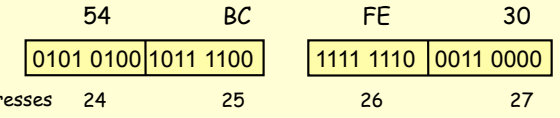
Little Endian



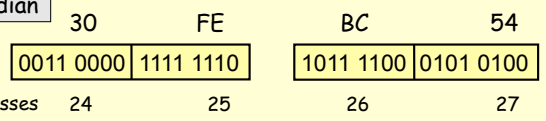
Example 2: 32-bit value (View 1)

- 32-bit hex value 54 BC FE 30 stored at memory address 24.

Big Endian



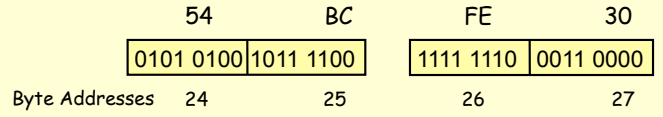
Little Endian



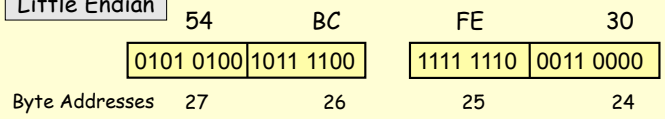
Example 2: 32-bit value (View 2)

- 32-bit hex value 54 BC FE 30 stored at memory address 24.

Big Endian



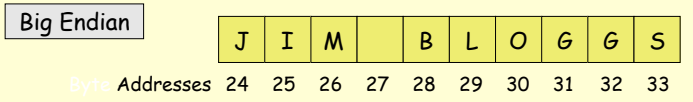
Little Endian



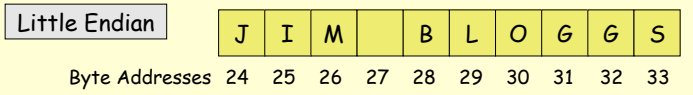
Example 3: ASCII string (View 1)

- String "JIM BLOGGS" stored at memory address 24
- Treat a string as an array of (ASCII) bytes. Each byte is considered individually.

Big Endian



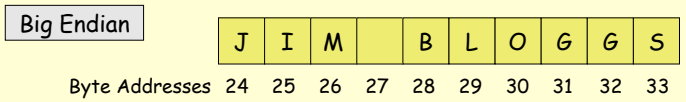
Little Endian



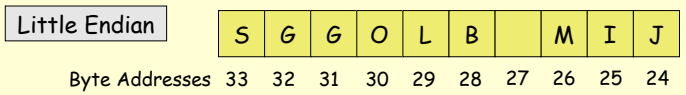
Example 3: ASCII string (View 2)

- String "JIM BLOGGS" stored at memory address 24
- Treat a string as an array of (ASCII) bytes. Each byte is considered individually.

Big Endian



Little Endian



A Problem

- How do we **transfer a multi-byte value** (e.g. a 32-bit two's complement integer) from a Big-Endian memory to a Little-Endian memory?
- How do we transfer an ASCII **string** value (e.g. "JIM BLOGGS") from a Big-Endian memory to a Little-Endian memory?
- How do we transfer an **object** which holds both types of values above?

What is the maximum amount of memory we can have in a 32bit machine with byte addressing?

A 4GB **B** 800MB **C** 16GB **D** 16MB **E** 1GB

(Un) Aligned Memory Accesses (1)

Main Memory (Big Endian)

0	0110	1101	1010	1101
2	**1010**	**1001**	**1010**	**0001**
4	0000	0000	0000	0000
6	1111	1111	**1111**	**0000**
8	**0010**	**0001**	0000	0000
10	1001	1010	1010	0010
12	0000	0000	0000	0000
14	1111	1111	1111	1110

- The 16-bit hex value (A9A1) at address 2 is memory word **aligned**.
- The 16-bit hex value (F021) at address 7 is **unaligned**.
- Some architectures prohibit unaligned accesses. Why?

(Un) Aligned Memory Accesses (2)

Main Memory (Big Endian)

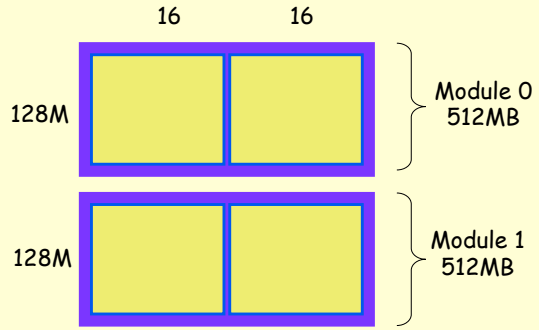
0	0110	1101	1010	1101
2	1010	1001	**1010**	**0001**
4	**0000**	**0000**	**0000**	**0000**
6	**1111**	**1111**	**1111**	**0000**
8	**0010**	**0001**	**0000**	**0000**
10	**1001**	**1010**	1010	0010
12	0000	0000	0000	0000
14	1111	1111	1111	1110

- How many memory accesses are required to read a 64-bit value from memory address 3?
- How many memory accesses are required to write a 64-bit value to memory address 3

Memory Modules and Chips

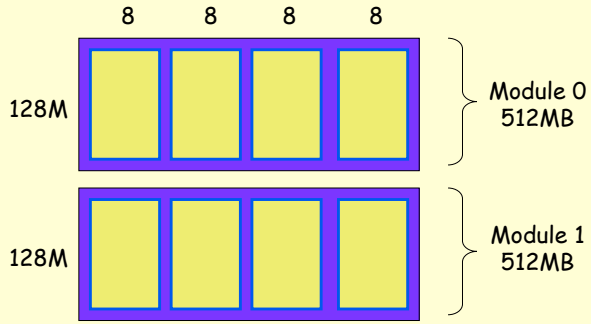
1GB (256Mx32bit) Memory

- Two 512MB memory modules. Each module has two 128M x 16-bit RAM Chips



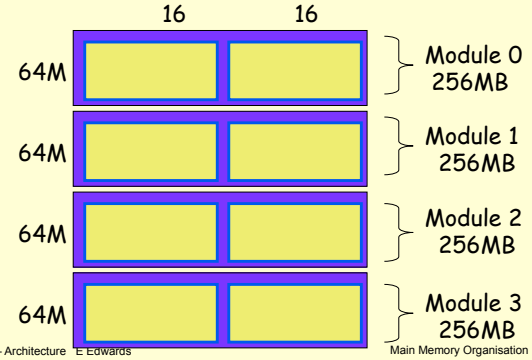
1GB (256Mx32bit) Memory

- Two 512MB memory modules. Each module has four 128M x 8-bit RAM Chips



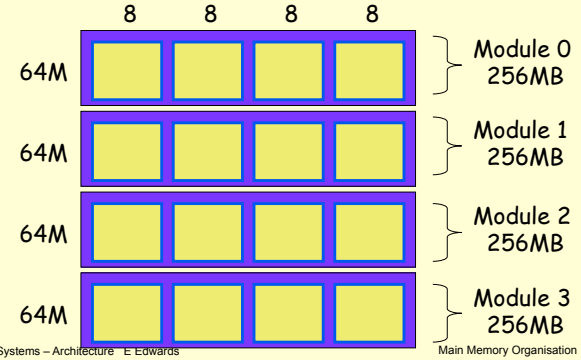
1GB (256Mx32bit) Memory

- Four 256MB memory modules. Each module has two 64M x 16-bit RAM Chips



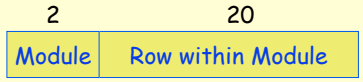
1GB (256Mx32bit) Memory

- Four 256MB memory modules. Each module has four 64M x 8-bit RAM Chips

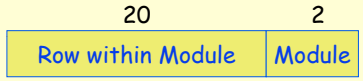


Memory Interleaving

- **Example**. Memory=4M words. Word Addressed. Each word = 32-bits. Built with 4 x 1Mx32-bit memory modules.
- For 4M words we need 22 bits for an address.
- 22 bits = 2 bits (to select Modules) + 20 bits (to select row within Module)



High-Order Interleave



Low-Order Interleave

High-Order Interleave

Address Decimal	Address Binary							
0	00	0000	0000	0000	0000	0000	Module=0	Row=0
1	00	0000	0000	0000	0000	0001	Module=0	Row=1
2	00	0000	0000	0000	0000	0010	Module=0	Row=2
3	00	0000	0000	0000	0000	0011	Module=0	Row=3
4	00	0000	0000	0000	0000	0100	Module=0	Row=4
5	00	0000	0000	0000	0000	0101	Module=0	Row=5
...								
2^20-1	00	1111	1111	1111	1111	1111	Module=0	Row=2^20-1
2^20	01	0000	0000	0000	0000	0000	Module=1	Row=0
2^20+1	01	0000	0000	0000	0000	0001	Module=1	Row=1

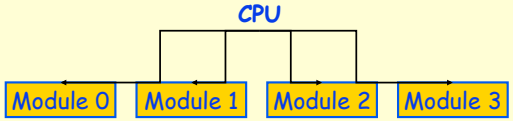
Low-Order Interleave

Address Decimal	Address Binary							
0	00	0000	0000	0000	0000	0000	Module=0	Row=0
1	00	0000	0000	0000	0000	0001	Module=1	Row=0
2	00	0000	0000	0000	0000	0010	Module=2	Row=0
3	00	0000	0000	0000	0000	0011	Module=3	Row=0
4	00	0000	0000	0000	0000	0100	Module=0	Row=1
5	00	0000	0000	0000	0000	0101	Module=1	Row=1
...								
2^20-1	00	1111	1111	1111	1111	1111	Module=3	Row=2^18-1
2^20	01	0000	0000	0000	0000	0000	Module=0	Row=2^18
2^20+1	01	0000	0000	0000	0000	0001	Module=1	Row=2^18+1

...

Low-Order Interleave

- Good if the CPU (or other unit) can request multiple adjacent memory locations.



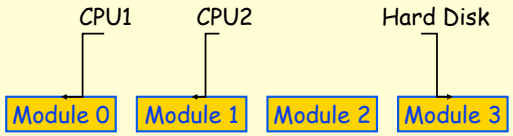
- Since adjacent memory locations lie in different Modules an "advanced" memory system can perform the accesses in parallel. Such adjacent accesses often occur in practice, e.g.

  i) Elements in an array, e..g Array[N], Array[N+1], Array[N+2], ...
  ii) Instructions in a Programs, InstructionN, InstructionN+1,...

- In the above situations, an "advanced" CPU can pre-fetch the adjacent memory locations => higher performance.

High-Order Interleave

- Good if Modules can be accessed independently by different units, e.g. by the CPU and a Hard Disk (or a second CPU) AND the units use different Modules
- => Parallel operation => Higher Performance



Think About

- Characteristics of registers, RAM and hard disks.
- Main Memory Addressing: word addressing & byte addressing.
- Byte Ordering: big-endian and little-endian
- (Un) Aligned accesses & interleaved memory
- NEXT Topic

Number Representation

This lecture - feedback

- The pace of the lecture was:

A. much too fast B. too fast C. about right D. too slow E. much too slow

- The learning objectives were met:

A. Fully B. Mostly C. Partially D. Slightly E. Not at all