Computer Systems
Lecture 20

Overview

- Memory parameters
- Address width
- Memory modules
- Memory mapping
- Memory address decoding
- Registers
- Cache memory
- Memory hierarchy

Back to memory

- Any memory location in main memory has its own **address**.
- It follows then the more memory the larger addresses are needed.

Maximal memory length depends on **address width**

Address width

- Address width is determined by:
  - The number of bits in the CPU address registers such as IP, MAR.
  - The number of lines in the address bus.

Address width and memory length

Address width			Maximal memory length	
16			**64**	**Kbytes**
20			**1**	**Mbytes**
24			**16**	**Mbytes**
32	**Pentium**		**4**	**Gbytes**
64	**64-bit architectures**	>	**17 billion Gbytes**	

Memory parameters

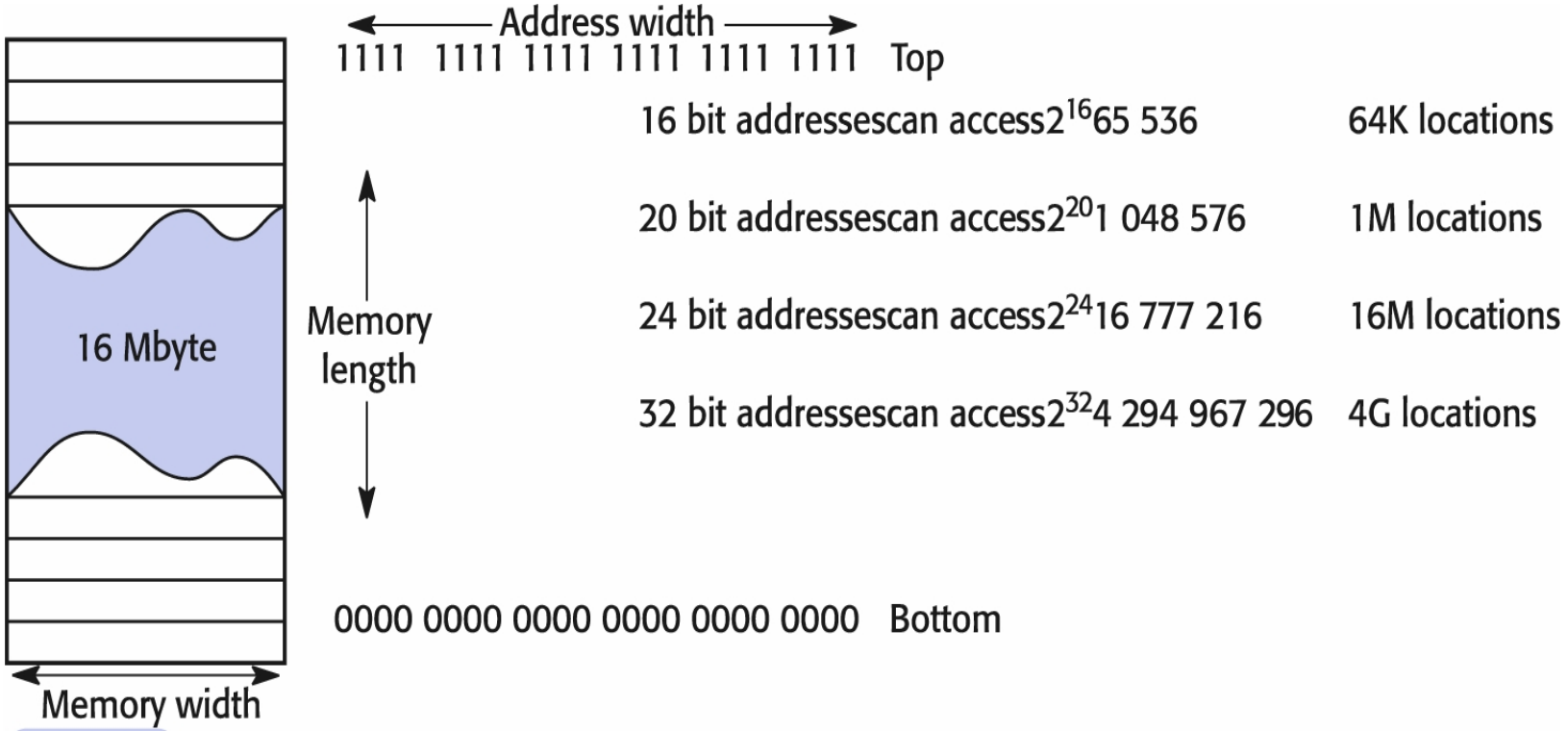


Fig. 3.17 Address width and memory length.

Ideal configuration

- With 32-bit address width the maximum memory addressing space is 4 Gbytes.
- Ideal configuration would be to have a single memory chip and to send an address via 32 address lines:

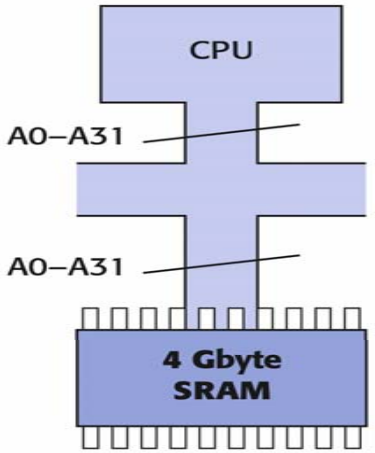
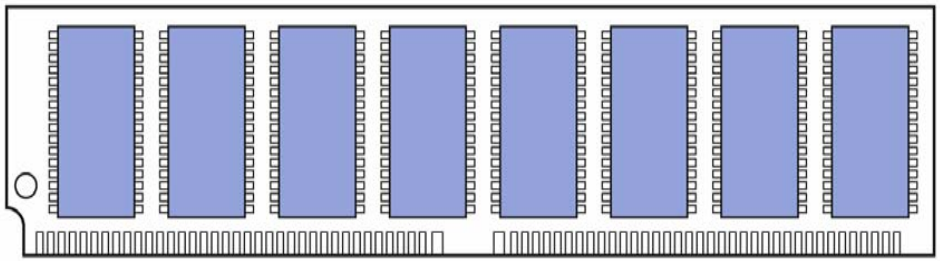


Fig. 6.12 Ideal configuration for a memory map.

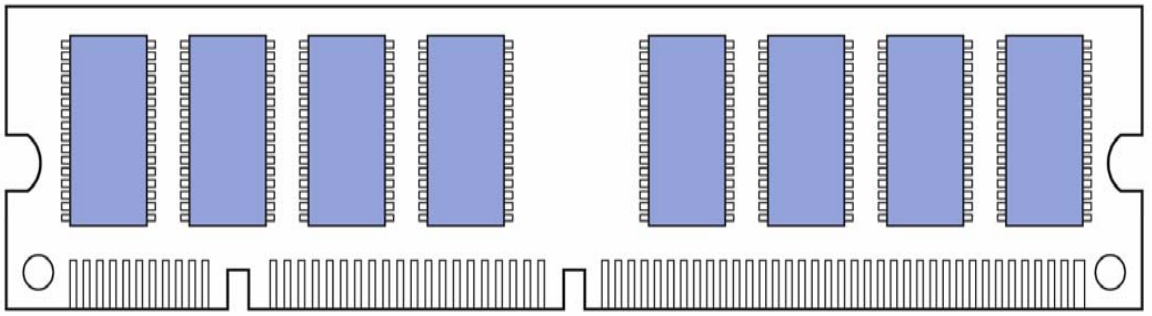
Not so ideal in practice

- 4Gbyte is still a lot of space and many systems do not use it all.
- Memory chips (DRAM) have a typical size 128 Mbits – 256 Mbits – 1 Gbits and they are combined into the memory modules of 8 – 16 chips.
- For example, the module of 8 of 256 Mbit chips has a capacity of 8 * 256 Mbits = 256 Mbytes.
- To address memory, we need to select memory chips as well as locations within the chips.

Memory modules



16 Mbyte, 50 ns access, 32 bit, 72 pin SIMM card



64 Mbyte, 100 MHz clock, 64 bit, 168 pin DIMM card

Fig. 6.11 72 pin SIMM and 168 pin DIMM DRAM modules.

Memory mapping

- Given an address, how does the system decide on which of the chips the memory location to be found?
- When the CPU sends out an address:
  - A part of the address locates the correct chip.
  - Another part specifies an address within the correct chip.
- How actually the addresses are mapped to the memory locations is defined by **memory maps**.

Memory map for a small system

Device	Size	Pins	32 bit address bus	Address range
PROM1	1 Mbyte	20	0000 0000 xxxx ++++ ++++ ++++ ++++ ++++	0000 0000 – 000F FFFF
RAM1	16 Mbyte	24	0000 0001 ++++ ++++ ++++ ++++ ++++ ++++	0100 0000 – 01FF FFFF
RAM2	16 Mbyte	24	0000 0010 ++++ ++++ ++++ ++++ ++++ ++++	0200 0000 – 02FF FFFF
RAM3	16 Mbyte	24	0000 0011 ++++ ++++ ++++ ++++ ++++ ++++	0300 0000 – 03FF FFFF
RAM4	16 Mbyte	24	0000 0100 ++++ ++++ ++++ ++++ ++++ ++++	0400 0000 – 04FF FFFF

\+ address line used directly for internal selection
x line ignored, indicates partial (degenerate) addressing
0 must be 0 for chip selection
1 must be 1 for chip selection

Fig. 6.13 Memory map for a small computer system.

Memory address decoding

- Memory chips are not normally matched to the width of the address bus. For example:
  - CPU may send 32-bit address.
  - RAM may receive directly 24-bit address.
- Special **Memory Address Decoding** circuit implements necessary decoding (see Fig 6.14 in the text book).

More detailed look at the memory levels

- Registers.
- Cache memory (Level 1 and Level 2).
- Main memory.
- Mass storage.

Why we need memory levels?

- To keep up with the demands of faster CPUs.
- To limit system cost.
- To cope with ever-expanding software systems.

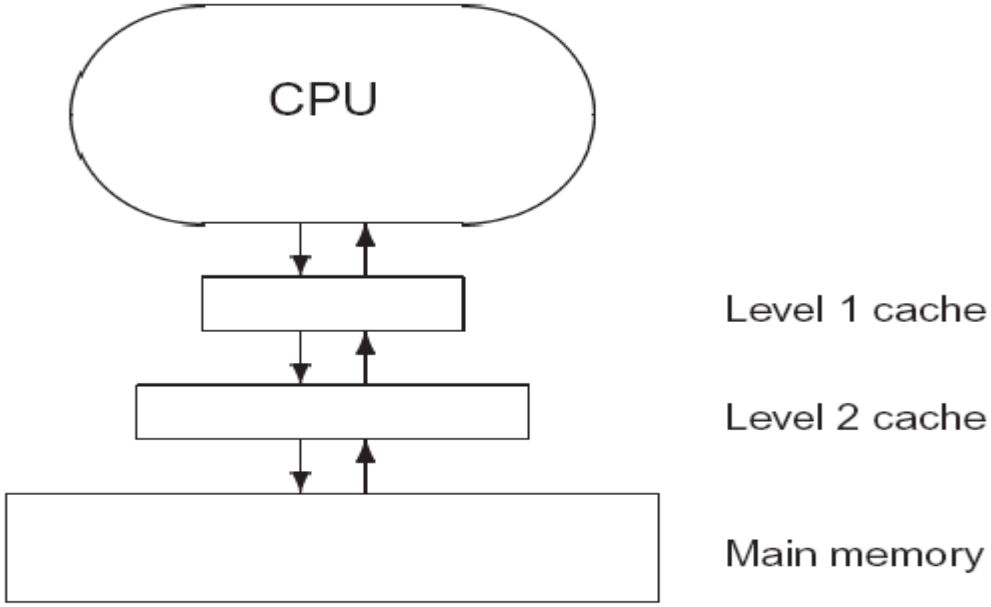
Registers

- Registers are the memory cells which are core part of the processor itself.
- It has very fast access (a few nanoseconds).
- Not that much memory: tens of 32-, 64-, 80-bit registers (typically).

Cache memory

- A memory (more expensive, but faster SRAM) placed between CPU and main memory.
- Contains a copy of the portion of main memory.
- The aim is to maintain in fast cache the currently active sections of code and data.
- Processor when needs some information first checks cache.
- If not found in cache, the block of memory containing the needed information is moved into the cache.

Levels of cache



- Typically Level 1 cache has the size 8-64 KB.
- Typically Level 2 cache has the size 128-512KB

Localisation of access

- The idea of cache memory exploits **Localisation of Memory Access** principle:
  - Computers tend to spend periods of time accessing the same locality of memory.
- Therefore:
  - A portion of code or data which require access needs to be loaded into the fastest memory nearest to CPU.
  - Other sections of the program and data can be held in readiness lower down the memory hierarchy.

Why localisation of access works?

- Partly due to the programmer clustering related data items together in arrays or records.
- Partly due to the repeating patterns in a program (i.e. loops)
- Partly due to the compiler attempting to organise the code in an efficient manner.

Cache memory and cache control unit

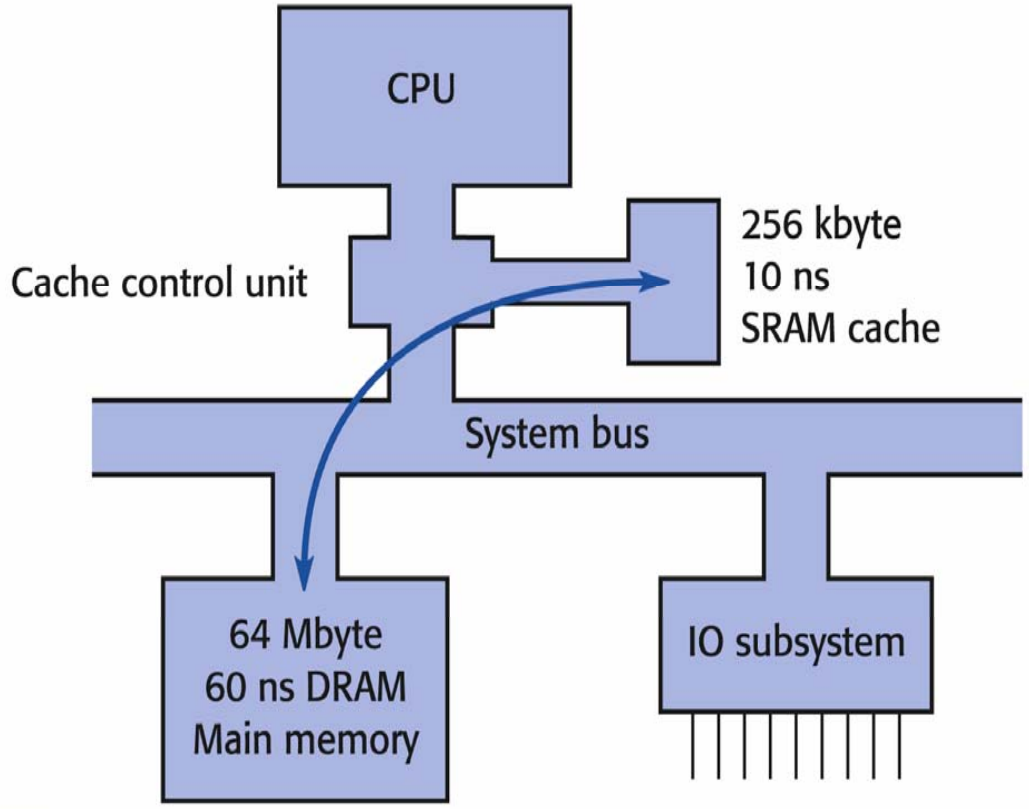


Fig. 12.9 Cache memory and controller unit.

Memory hierarchy

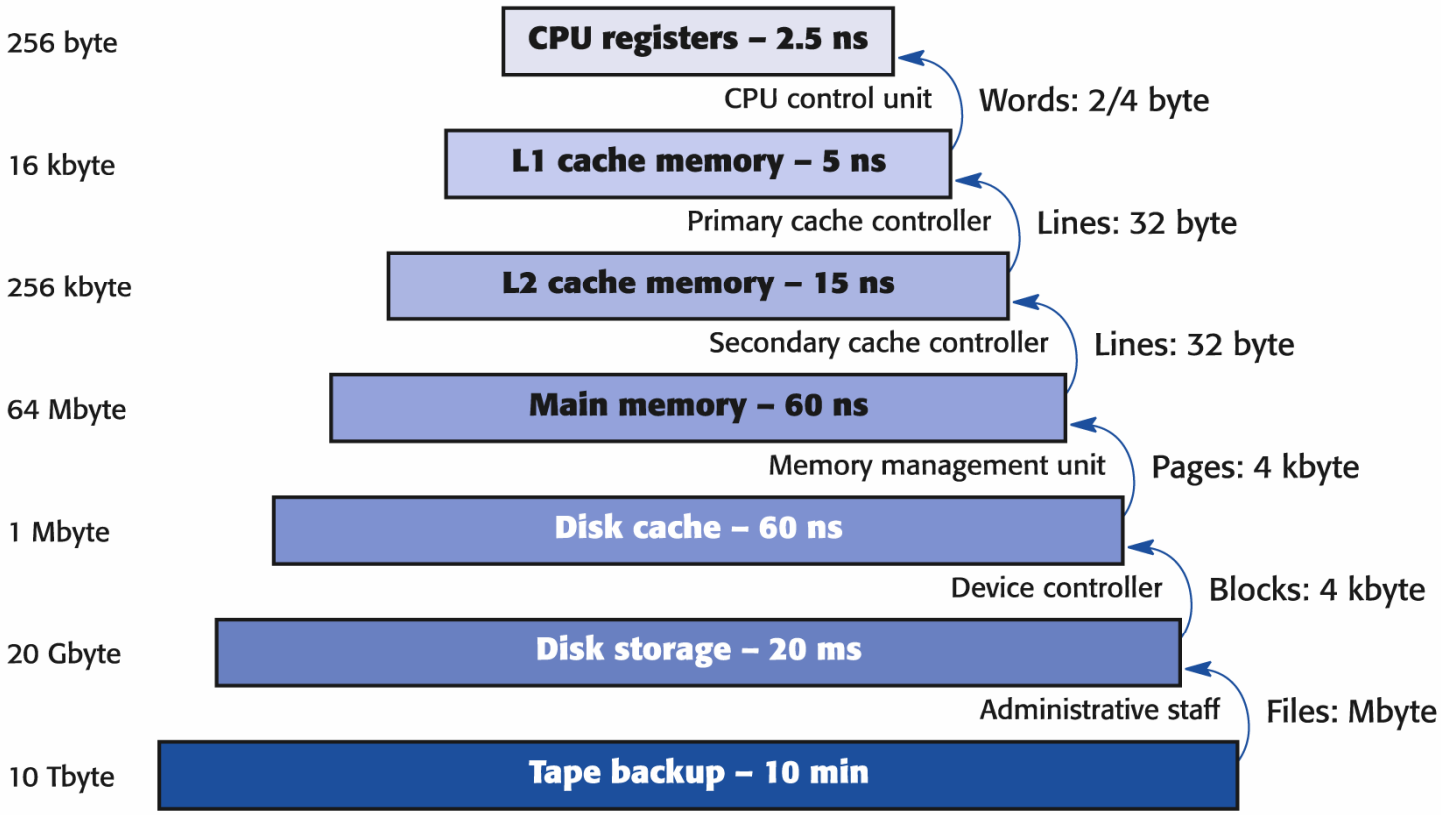


Fig. 12.2 Memory performance and storage hierarchy.

Memory hierarchy

- Going down the hierarchy:
  - Increased capacity.
  - Increased access time.
  - Decreased frequency of access of the memory by the processor.
  - Decreased cost per bit.

Exercise

Identify locality using the following program code:

```
int myarray[5]; // declaration of an array of integers
myarray[0] = 1; //
myarray[1] = 3; //
myarray[2] = 5; // initialise the array
myarray[3] = 7; //
myarray[4] = 9; //
_asm {
lea ebx,myarray //address of the array (its 0th element) is saved in ebx
mov ecx,5 // size of the array is saved in the counter
mov eax,0 // eax will be used to store the sum, initialise to 0
Loop1: add eax,[ebx] //read an element of the array at the address stored in ebx
add ebx,4 //int occupies 4 bytes (32 bits),
//so to read next element increase an address in ebx by 4
loop Loop1 // the end of the loop
...

}
```

Exercise

Identify locality using the following program code:

```
/* Handling arrays with Register Indirect with Displacement addressing mode */
// declaration of an array of integers and initialisation are as above
...
_asm {
mov ecx,5 // size of the array is saved in the counter
mov eax,0 // eax will be used to store the sum, initialise to 0
mov ebx,0
Loop2: add eax,myarray[ebx] // read an element of the array at the address myarray+ebx
add ebx,4 // int occupies 4 bytes (32 bits), so to read next element increase an address in ebx
// by 4
loop Loop2 // the end of the loop
...
}
```

Q&A

- Q. Which of the following has fastest access?
- A. register
- B. cache
- C. disk
- D. tape

Q&A

- Q. which of the following is most expensive in terms of cost?
- A. register
- B. cache
- C. disk
- D. tape

Q&A

- Q. Given the following components, form a reasonable memory hierarchy to ensure good cost-speed tradeoff.
- Cache, disk, RAM

Q&A

- Q. Locality can be temporal, which means data that are recently used tend to be accessed more likely than the others. (T or F)

Readings

- [Wil06] section 3.6 for memory address and address width.
- [Wil06] section 6.7 for ideal configuration, addressing and mapping.
- [Wil06] section 6.2 - 6.4 for chips, modules, SRAM and DRAM.
- [Wil06] section 12.1 for memory levels.
- [Wil06] section 12.3 for cache.
- [Wil06] section 12.2 for localisation of access.