Inside the computer

Classification of Systems:
- Microcomputer
- Minicomputer
- Mainframe
- Supercomputer

- Chapters 1-5 in Capron

Microcomputer

- Personal Computer / Workstation.
- Desktop machine, including portables.
- Used for small, individual tasks - such as simple desktop publishing, small business accounting, etc....
- Typical cost: £500 to £5000.
- Example: The PCs in the labs are microcomputers.

Minicomputer

- Medium sized server
- Desk to fridge sized machine.
- Used for distributed data processing and multi-user server support.
- Typical cost: £5,000 to £500,000.
- Example: Scarlet is a minicomputer.

Mainframe

- Large server / Large Business applications
- Large machines in purpose built rooms.
- Used as large servers and for intensive business applications.
- Typical cost: £500,000 to £10,000,000.
- Example: IBM ES/9000, IBM 370, IBM 390.

Supercomputer

- Scientific applications
- Large machines.
- Typically employ parallel architecture (multiple processors running together).
- Used for VERY numerically intensive jobs.
- Typical cost: £5,000,000 to £25,000,000.
- Example: Cray supercomputer
What's in a Computer System?

- The Onion Model - layers.
  - Hardware
  - BIOS
  - Software
- Where does the operating system come in?

Software

- Divided into two main areas
  - Operating system
    - Used to control the hardware and to provide an interface between the user and the hardware.
    - Manages resources in the machine, like
      - Memory
      - Disk drives
      - Applications
        - includes games, word-processors, databases, etc....

Interfaces

- CUI
  - Command Line Interface
- GUI
  - Graphical User Interface
- WIMP
  - Windows, Icons, Mouse, Pulldown menus

Hardware

- The chunky stuff!
- If you can touch it... it's probably hardware!
- The mother board.
  - If we have motherboards... surely there must be fatherboards? right?
  - What about sonboards, or daughterboards?!
- Hard disk drives
- Monitors
- Keyboards

BIOS

- Basic Input Output System
  - Directly controls hardware devices like UARTS (Universal Asynchronous Receiver-Transmitter) - Used in COM ports.
  - Stored in the ROM of the machine.
  - What's ROM? - Read only memory
  - Preserved while the computer is turned off.
    - How?
    - WHY?!!

Basics

- Peripherals
- Central Processing Unit (CPU)
- Memory
Basics

Central Processing Unit (CPU) -> Memory

Memory

- Stores the program to be executed and the data that this manipulates as bits
- Volatile - contents are lost when the computer is switched off
- Memory consists of series of cells, each of which holds one word of information
- Each cell has a unique memory address (a number) that can also be written as bits

<table>
<thead>
<tr>
<th>Memory cells</th>
<th>Memory addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101100111000010</td>
<td>00000001</td>
</tr>
<tr>
<td>1001100101111000</td>
<td>00000010</td>
</tr>
<tr>
<td>0001100001111000</td>
<td>00000011</td>
</tr>
<tr>
<td>1101101001100000</td>
<td>00000100</td>
</tr>
<tr>
<td>1111111110111110</td>
<td>00000101</td>
</tr>
<tr>
<td>1010101010101010</td>
<td>00000110</td>
</tr>
<tr>
<td>0001000100010001</td>
<td>00001000</td>
</tr>
<tr>
<td>1101101110010110</td>
<td>00001001</td>
</tr>
<tr>
<td>1000100111011010</td>
<td>00001010</td>
</tr>
</tbody>
</table>

Cell containing one word of information

Memory

Store and fetch

- Two key properties of a cell are its value and address
- Memory hardware performs to basic operations
  - store ( value ← address )
  - fetch (address → value)

Types of memory

- Random Access Memory (RAM)
- Read Only Memory (ROM)
- Erasable Programmable Read Only Memory (EPROM)

ROM

- Read Only Memory
- Contains crucial start-up information for PCs.
- Typically only 48K.
- Non-volatile - information is preserved when the power is switched off.
- Relatively slow - sometimes copied to RAM for faster execution.
- Also located on memory chips.
RAM
- Random access memory
- Volatile - information is lost when the power is switched off.
- Early computer had around 4K of RAM.
- These-days 128Mb RAM is around £30
- Read about SIMM, DIMM, SRAM, DRAM

EPROM
- Using a ROM Burner the instructions within the chips can be changed
- Used for updates
- Also seen in DVD players and videos to change/update functions

Measuring memory
- Memory not usually measured in words
- Measured instead in:
  - bytes
  - kilobytes
  - megabytes
  - gigabytes
- International word for a byte is an octet

Memory
- Every 0 or 1 in binary is called a bit
- (0 means off, 1 means on)
- 8 bits = 1 byte = one character (A,B,C,$,£,1,2)
- kilobytes (Kb) = $2^{10}$ = 1024 bytes
- megabytes (Mb) = ~1,000,000 bytes
- gigabytes (Gb) = ~1 billion bytes

Where are we ??

CPU
- Arithmetic and Logic Unit (ALU)
- General purpose register
- Special purpose register
- Memory
- 0101100111000010
• ALU is the sub-component that performs basic arithmetic and logic operations
• Registers provide very fast but expensive storage (hence there are only a few)
  – General purpose registers
  – Special purpose registers
    • IR put instructions here to execute them
    • PC memory address of the next instruction
    • MAR where addresses are placed for fetch or store
    • MBR where values are placed for fetch or store

Chapter 3 (Capron)

Program execution
• The fetch-decode-execute cycle
  – Fetch instruction at memory address PC from memory and put it into IR
  – Decode and execute the instruction in IR
  – Increment PC to point at the next instruction
• Each single instruction may involve many smaller micro-instructions, such as moving words of information between registers

The clock
• The whole cycle is driven by the CPU’s internal clock
• CPU speed is measured in Hertz (Hz)
  (500MHz computer can handle 500 million machine cycles per second)
• Chunks of information have to be moved between parts of the computer - carried along an arrangement of parallel wires called a bus

Peripherals
• Various peripherals are responsible for input, output and permanent storage
  – programs and data have to get into memory
  – results have to be displayed to users
  – permanent storage is required
• Access to peripherals is orders of magnitude slower than the speed of the CPU of the speed of memory access
• Interrupts are used to allow the CPU to get on with something else in the meantime

Peripheral Devices
• Input
  • Mouse
  • Scanner
  • Light-pen
  • Keyboard
  • Modem
    • What does modem mean?

For example, for the single instruction: "add the data at address X to the data at address Y and store the result back at Y":

- move contents of PC to MAR
- fetch
- move from MBR to IR
- Put address X into MAR
- fetch
- move MBR to general register A
- put address Y into MAR
- fetch
- move MBR to general register B
- add A to B, result in C
- move C to MBR
- store

The clock
• The whole cycle is driven by the CPU’s internal clock
• CPU speed is measured in Hertz (Hz)
  (500MHz computer can handle 500 million machine cycles per second)
• Chunks of information have to be moved between parts of the computer - carried along an arrangement of parallel wires called a bus

Peripherals
• Various peripherals are responsible for input, output and permanent storage
  – programs and data have to get into memory
  – results have to be displayed to users
  – permanent storage is required
• Access to peripherals is orders of magnitude slower than the speed of the CPU of the speed of memory access
• Interrupts are used to allow the CPU to get on with something else in the meantime
Peripheral Devices

- Output
  - Printer
  - Plotter
  - Monitor
  - Modem

Some peripherals use special dedicated I/O processors to increase performance.
Each peripheral is controlled by its own instructions:
- disk: seek, read, write …
- tape: wind, rewind, read, write ...
- printer: form feed, reset, newline …
- monitor: clear, refresh ...

Peripheral devices

- Storage
  - Hard drive
  - Floppy drive
  - CD-ROM
  - Zip disk

Summary of simple model

- Memory - words, cells, addresses, fetch and store, sizes
- CPU - registers, ALU, fetch-decode-execute, micro-programs, clock
- Peripherals - interrupts, specialised instructions

Application development software

- The development of programming languages
- Language Translators
- Software Tools

Source code

Source code:
“The written commands in a programming language, together with the comments that describe what they do.”
The development of programming languages

- Algorithms and programs revisited
- Machine-code
- Assembly language
- High-level languages
- Object oriented languages

Algorithms

- An algorithm is a logical plan for solving a task that can be understood by humans.
- There are many notations for expressing algorithms. They need to include:
  - hierarchical decomposition of tasks
  - sequences of actions
  - choice
  - iteration
  - descriptions of objects and information

Machine code 1st generation

- Computers execute binary instructions called machine code
- Typically a few simple instructions consisting of op-codes and addresses
- Very difficult for humans to work with algorithms in machine code
  - hard to read
  - easy to make but hard to correct mistakes
  - minimal program or data structures
  - not portable

Assembly languages 2nd generation

- Limited improvements over machine code
  - mnemonics for op-codes improve readability
  - more powerful addressing styles such as named locations, relative addressing and indirect addressing
- ADD A
- LOAD A + 1
- STORE (A)

High level languages 3rd generation

- Developed in the 60s and 70s to be closer to algorithms in terms of notation, control structures, data structures, and task decomposition
- Relatively portable
- Based around hierarchical task decomposition realised using functions/procedures/sub-routines

Important features of HLLs

- Readability
  - use of English words and phrases
  - mathematical notation
  - block structure
  - comments
- Powerful instructions and control structures
  - complex expressions and operators
  - choice and repetition
  - input and output
• Data structures
  – named variables and constants
  – typed variables
  – structured types (arrays and records)
  – dynamic linked types (trees, lists, queues and stacks)
• Program and system structure
  – functions
  – libraries and modules
• Abstraction and encapsulation
  – functions support procedural abstraction
  – modules support data abstraction

Object-orientated languages
• Encapsulate data and operations into objects
• Classes provide templates for objects
  – establish precise interfaces between system components before implementation
  – encourage team-work, re-use and safety

Language translators
• The role of language translators
• Interpreters and compilers
• Syntax and semantics
• Stages of interpretation

The role of language translators
• Programming languages have evolved to make programming easier for humans
• Computers still execute machine code
• Language translators are required to translate between the two
• A new translator is required for each combination of programming language and machine-code

Interpreters
• One statement is translated and then executed at a time
• Start at the beginning of the program

REPEAT
  translate next program statement
  if no translation error occurred then execute the statement
UNTIL the end of the program or an error occurs
Compilers

- Whole program is translated, then executed
- The programmer writes the source program which is translated into the object program
- Start at the beginning of the source program
  REPEAT
  translate next source program statement
  UNTIL the end of the source program
  IF translation errors occurred
  THEN report errors
  ELSE execute the object program

Comparison

- Interpretation and compilation both have advantages and disadvantages
- They are suited to different environments and stages of the development process

Languages, syntax and semantics

- Program languages are much simpler than ‘natural’ languages such as English, but do share some similarities:
  – vocabulary of defined words
  – punctuation
  – names created by the programmer (identifiers)
- Programming languages have grammar that defines it’s syntax
- Statements also have a semantic meaning

- Examples of syntax versus semantics
  – In English
    • John has be a bad boys (incorrect syntax)
    • Flying tables sleep furiously (incorrect semantics)
  – In a program
    • c = b = a; (incorrect syntax)
    • d = 0; b = c/d; (incorrect semantics)
    • hyp = opp + adj; (incorrect semantics)
- Another important difference to a natural language is ambiguity

Errors

- Errors can show up at translation time
  – (syntax errors) or run-time (semantic errors)
- Run time errors might be due to the program or its environment

Scheduling and memory management

- Scheduling
- Swapping
- Memory - memory management
- Paging
- Software development
Goals of memory management

- enable several processes to be executed (seemingly) at the same time;
- provide satisfactory level of performance for users;
- protect each process from each other;
- enable sharing of memory between processes;
- make memory addressing transparent to programmer.

Physical vs. Logical

- Computer's memory size determined by hardware. This is the physical memory size.
- Usually smaller than the address space.
- The address space is what a user program can access. This is the logical memory size (sometimes known as virtual memory size).

Running a process

- First step must be to load executable image (.EXE file in MS-DOS; a.out format file in UNIX) from secondary storage into memory.

Scheduling

- Running a program results in a process
- The basic rule is that the CPU can only be allocated to one process at a time
- Multiprogramming is the illusion that the CPU is shared between many concurrent processes

Swapping

- The CPU has to swap between processes:
  - preserve the complete state of the outgoing process (program, data and registers)
  - restore the state of the incoming process
  - hand over control of the CPU
- The state of a process is captured in its core-image, a freeze-frame of it as it runs
- At any time, several processes may have core images in memory or on disk

When to swap?

- How does the CPU know when to swap?
  - process might run until it completes
  - process might run until it requests I/O
  - the OS might regularly interrupt the CPU
- Whenever a swap needs to happen the scheduler (part of the kernel) takes over the CPU and chooses the next process to run:
  - round robin
  - dynamic priority
- This is supported by the process table
Relocation

- Program with user address space starting at 0 will not necessarily be loaded into a physical space starting at address 0.
- Relocation of addresses (but not data).
  - Load time relocation (sorted out once when program is loaded - program amended).
  - Runtime relocation via relocation registers (sorted out each time a logical address is accessed).

Memory allocation schemes

- Single process system
- Fixed partition system
- Segmented systems
- Problems

Memory allocation systems

Single process system

- Operating system
- User process
- Unused space
- Relocation register needed to indicate base address of user process.
- For protection, also need register to indicate limit address

Fixed partition system

- Several partitions each capable of holding one program. Partitions might or might not be of same size.
- Largest partition needs to be large enough to hold largest program.
- Results in internal fragmentation of memory due to unused space (holes) in each partition

Segmented systems

- Program split into two or more (variable sized) segments.
- Most common division is code and data. Also possible to have programmer-defined segments (e.g. one for each module of source code).
- Segments need not be contiguous, thereby reducing problems of allocating free space.
- Need separate pairs of registers for each segment.
- Segment attributes: sharable, writeable.

Problems

- Fragmentation. Cannot use space between segments unless there is another segment small enough to fit.
- Compaction is almost as inefficient as swapping (but at least frees up some useful space).
- Trade-off between more, smaller segments making memory fitting easier against need for more segment registers.
Paging

- Memory divided up into fixed-size chunks called pages.
- Typical page sizes: 512 bytes, 1K, 2K, 4K, ...
- Relocation problem - effectively need a relocation register for each page. However since all pages are the same size, no limit register is needed

Virtual memory

Paging (and to a lesser extent segmentation) suggests that not all pages (or segments) of a program need be in physical memory to run the program. Only the bits containing the code being executed and the data being accessed need be there. The remainder could be on secondary storage.

Page addressing

- Each process has a page table - defines mapping from logical pages to physical pages (and any page attributes).
- Each logical address is split into two parts: page number and offset within page. Page number mapped by page table, then offset added in.
- Solves most of fragmentation problem since a process can be fitted in to a number of separate holes. Some fragmentation still exists because of unused space within a page

Consequences

- More processes can be held in memory (ready to run) thereby easing scheduling problems. In extreme we could hold 1 page of each process.
- Each process can be larger than the actual physical memory available. Illusion that physical memory is as big as logical memory. With 32-bit address spaces the illusion is a necessity.
- The page table must record whether a page is in memory or on disk.
- If an address that is not in a page in memory (the resident set) is accessed, we have a slight problem. This is called a page fault (usually an interrupt) - the requisite page must be fetched from disk (demand paging).

- Efficiency is dependent on few page faults. Experiments have shown however that most processes exhibit locality of reference (i.e. they are more likely to refer to addresses close by than those far away - e.g. access to local variables within a procedure). Thus a working set of related pages will tend to generate few page faults. (It is consequential that good use of modular programming strategies will improve efficiency of program.)
- If a process's resident set is significantly smaller than its working set, rather than continue and have increasing numbers of page faults (thrashing), it might be better to swap the whole process out (thus freeing memory for other processes) and bring a larger proportion of it back later.

If at any time a page is to be brought into memory and there are no free pages, another page must be swapped out to make room for it (page replacement).
- If the page table records whether the contents of a page have changed (since it was read in from disk), then only dirty pages need to copied back out to the disk.
- Repeated swapping of pages is inefficient. Need to keep a substantial portion of the program in memory to reduce number of page faults to a minimum. This is the working set. Working set can be determined by observing which pages are accessed over a period of time. (Information to record this can be stored in the page table.) Pages not accessed for a long time are good candidates to be first to be swapped out.
Software development

- The waterfall process
- Functions
- Objects
- Classes

The waterfall process

- Requirements
- Design
- Coding
- Testing
- Maintenance

In the “waterfall model” of software development, each stage should be completed before starting the next.
However, there is no reason why different parts of a program should not progress at different rates.
Also in practice, there is often some overlap between phases, e.g. one may need to implement a small part of a proposed design in order to find out whether it is feasible or not.

NOTE: There is an old joke that says:

“In theory there no difference between theory and practice, but in practice there is.”

It applies here.

Structure in computer programs

- Software engineering suggests that you should divide programs into separate pieces so that you can understand a piece at a time
- A function is an early form of program division: A function has:
  - An interface which describes how you can use it
  - An implementation which carries out the task

- You can use a function without knowing how it is implemented….
- Consider the following examples:
  - Ask a chef to prepare a meal
  - Calculate the square root of a decimal number
  - Send an email message
Object-orientated structures

- An alternative to dividing a program into functions is to use objects
- Objects are a natural way of thinking about things
  - We are surrounded by objects (cars, television, people)
- A computer program which is written in terms of objects, closely represents the real world problem it is trying to solve

- Similar to a function, an object has…
  - An interface which describes how the object can be used
  - An implementation which represents the object’s internal state
- An object’s state dictates how it responds to requests

An example object: a CD Player
- Operations include:
  - Play
  - pause
  - seek
  - forward
  - reverse
  - eject
- The state:
  - Is there a disk in the player?
  - What track is the player currently at?

What the CD player illustrates

- An object provides a set of operations (Methods) that the user may request
- An object has an internal state, which should be hidden from the user
  - Thus an object is a black box
  - You should only use an object through its interface

A class of CD Players

- We have discussed a single CD Player
  - But there are millions of such players throughout the world
- Most different makes and models have a lot in common
  - To capture this commonality we can define a class of CD players
    - A class would describe the operations and internal state which is common to all CD players