MEC302: Embedded Computer Systems

Theme II: Design of Embedded Computer Systems
Lecture 7 – Memory technologies, hierarchy and models

Dr. Timur Saifutdinov

Assistant Professor at EEE, SAT

Email: Timur.Saifutdinov@xjtlu.edu.cn

Outline

- Memory technologies in microprocessors:
  - Physical realization and distinctive characteristics;
- Memory hierarchy
  - Register files, scratchpads, caches;
- Memory Models
  - Address space, data types, alignment and allocation of data;
  - Memory model of C.

Memory technologies

Memory – device or system that is used to store information for immediate use in a computer or other related hardware.

Memory vs **storage**:

- **Memory** (some times called **primary storage**) is directly accessed by the processor for immediate use via data buses;
- **Storage** (i.e., **secondary storage**) is accessed indirectly via input/output operations (it is larger but slower).

Memory technologies define physical realizations and characteristics, e.g.:

- Volatile or Non-volatile:
  - **Volatile memory** requires power to store data, while non-volatile does not.

- Random access or Non-random access:
  - In **Random Access Memory (RAM)**, data can be accessed (i.e., read or changed) in any order;
  - Data in **RAM** is read or written in the same amount of time irrespective of the physical location of data inside the memory.
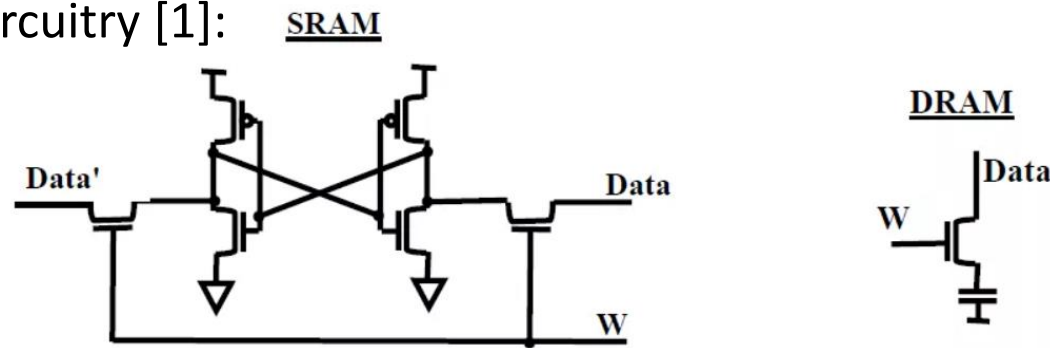
Memory technologies

Random Access Memory (RAM) is a **volatile,** computer memory that can be accessed (i.e., read or changed) in any order.

There are two main types of **RAM**:

- **Static RAM (SRAM)** – keeps data permanently (at the same location) while powered;

- **Dynamic RAM (DRAM)** – data decays in time (seconds); needs to be periodically refreshed.

One bit memory circuitry [1]:

Comparison (**SRAM** vs **DRAM**):

SRAM	DRAM
Stores data till the power is supplied	Stores data for few ms, requires refreshing
Requires 6 times more transistors	Die size is ~6 times smaller (more memory per chip)
Consumes more power	Data access is slower
Cost per bit is higher	

[1] Embedded Projects & Embedded Ideas, https://mavink.com/post/F07C551A69F4EF1223560B631189952462AMCE66B8
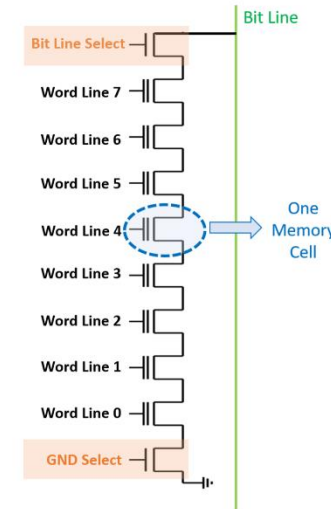
Memory technologies

Non-volatile memory – keeps data even without power supply.

- **Read-Only Memory (ROM)** – hard-wired integrated circuit whose content is fixed at a chip factory (i.e., **firmware**);

- **Electrically-Erasable Programmable ROM (EEPROM)** – a variant of ROM that can be (re)programmed in the field:

  - **Flash memory** – commonly used in embedded applications to store firmware. Shortfalls: limited number of writes; block access to data; long access time.

Simple four byte diode **ROM**:
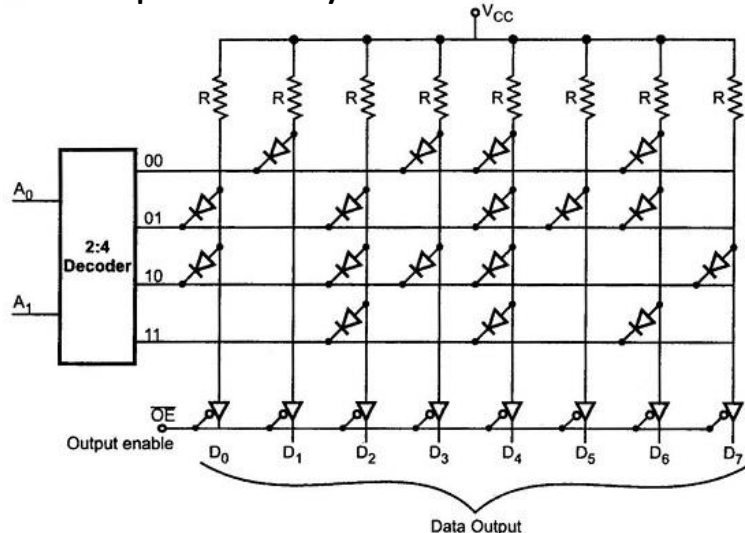
NAND **Flash memory** topology:

Memory hierarchy

Memory hierarchy is used by processors to combine different memory technologies to:
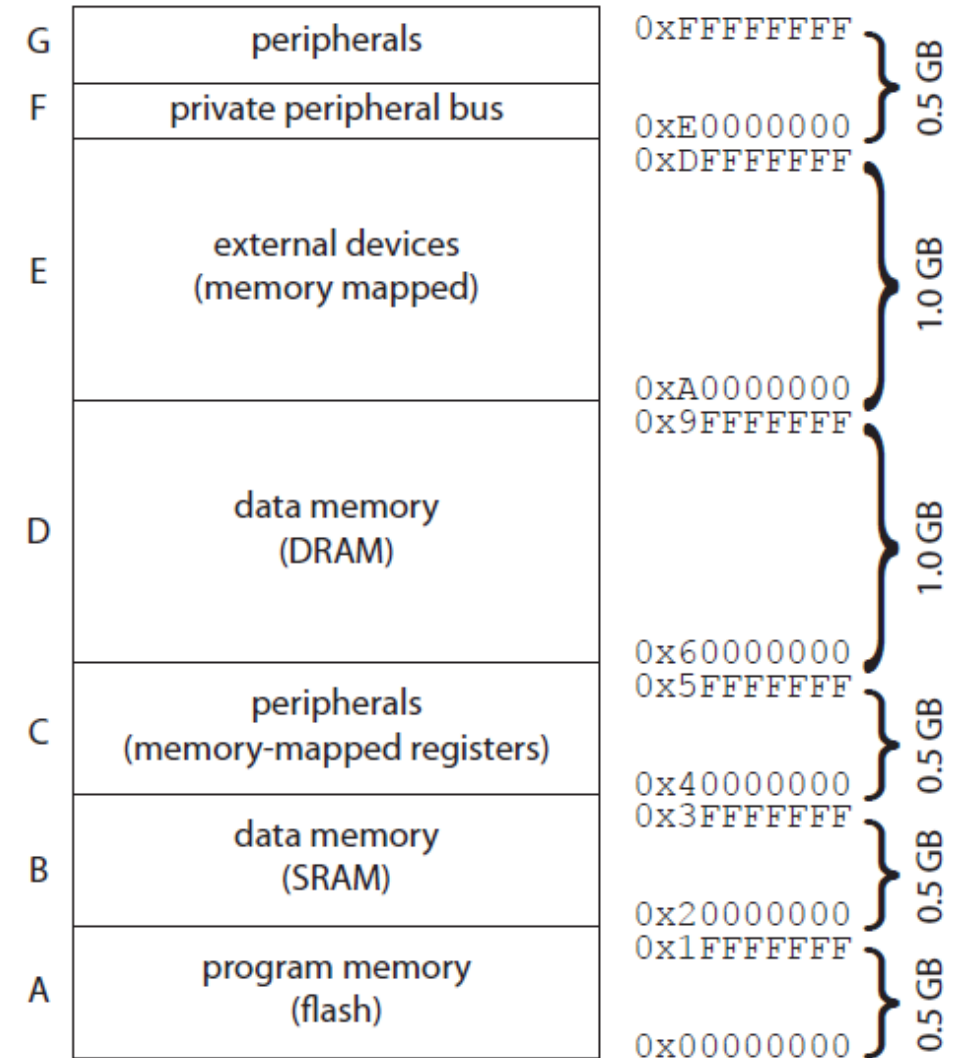
- Increase the overall memory capacity;

- Optimize cost, latency, energy consumption.

The operating system and/or hardware provide **address translation** from logical address to a physical location:

- **Virtual memory** makes the diverse technologies look to the programmer like a single memory with a contiguous address space.

- **Translation lookaside buffer (TLB)** is specialized piece of hardware used to speed up address translation.

A **memory map** for a processor defines how logical addresses are mapped to locations in hardware.

#ARM Cortex - M3 architecture
memory map:

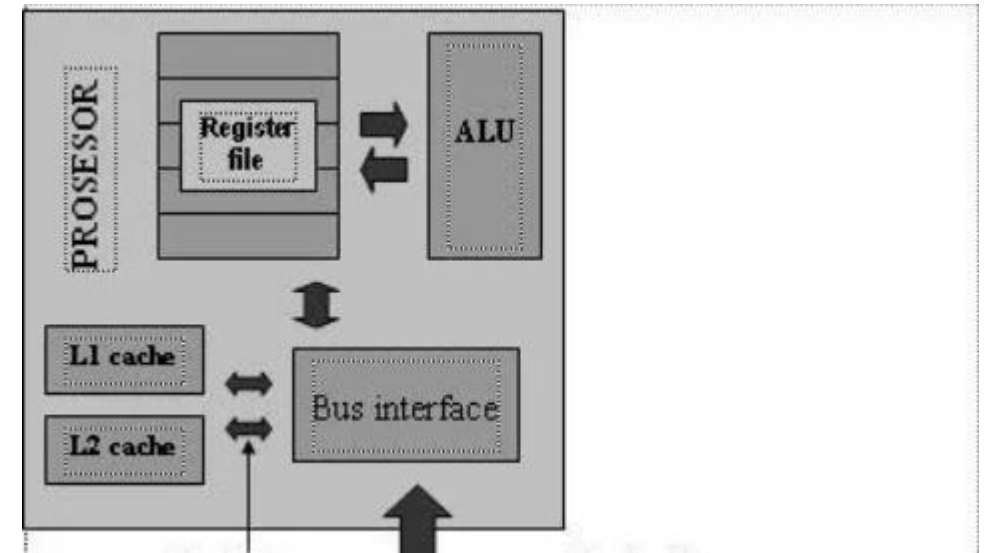|		
G	peripherals	0xFFFFFFFF
F	private peripheral bus	0xE0000000
		0xDFFFFFFF
E	external devices (memory mapped)	0xA0000000
		0x9FFFFFFF
D	data memory (DRAM)	0x60000000
		0x5FFFFFFF
C	peripherals (memory-mapped registers)	0x40000000
		0x3FFFFFFF
B	data memory (SRAM)	0x20000000
		0x1FFFFFFF
A	program memory (flash)	0x00000000

0.5 GB, 1.0 GB, 1.0 GB, 0.5 GB, 0.5 GB, 0.5 GB

Memory hierarchy

Register file is an array of processor registers in a CPU – the most tightly integrated memory. Implemented on a processor circuitry with SRAM technology.

Register file provides:

• A word of data (eg, four bytes for 32-bit arch.)

• Fast access to program data (e.g., program variables);

• Stores program auxiliary data (e.g., stack pointer, program counter (PC));

• Other purposes (i.e., general purpose).

#32-bit ARMV7 register denotation[3]:

R0-R3 – temporary program values or variables;

R4-R12 – general purpose registers;

R13 – stack pointer;

R14 – link register;

R15 – program counter (PC).

[2] Cache memory, https://jejecms.wordpress.com/2014/03/28/cache-memory/
[3] The ARM Register File, https://www.allaboutcircuits.com/technical-articles/what-is-a-microarchitecture-processor-register-files-ARM-core/

Memory hierarchy

The next level in memory hierarchy are **scratchpads** and **caches**:

- **Scratchpads** store temporary program data (like registers) to moves it in/out of the distant memory (w/o duplicating it);

- **Caches** are used to duplicate data to/from other layers of the main memory (e.g., slower DRAM) for CPU.
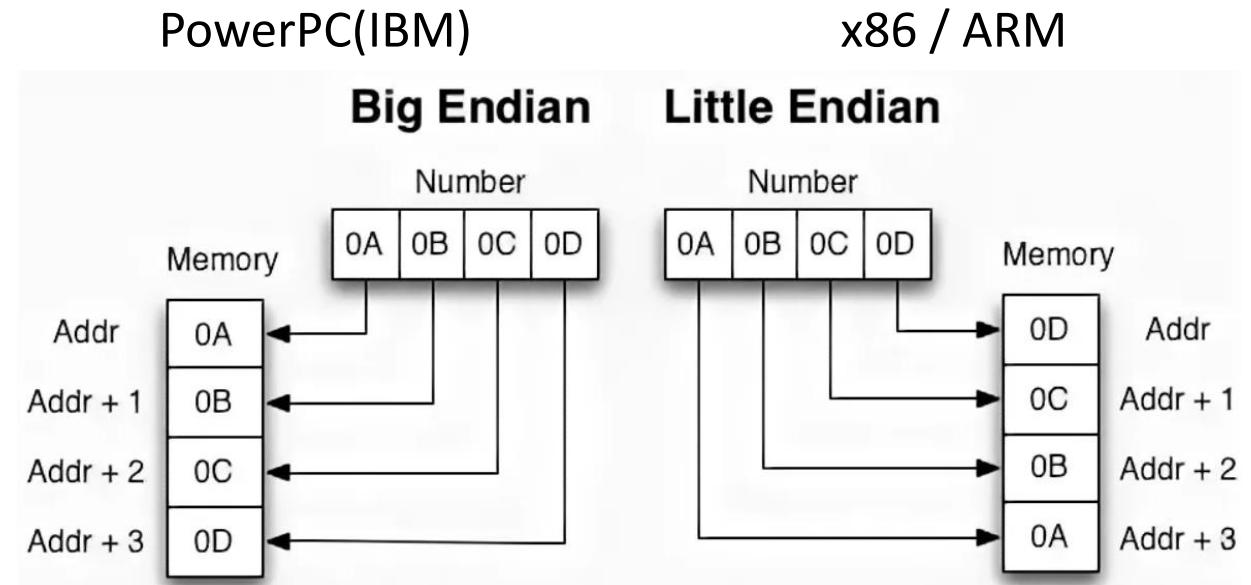
CPU organization and links with memory [2]:
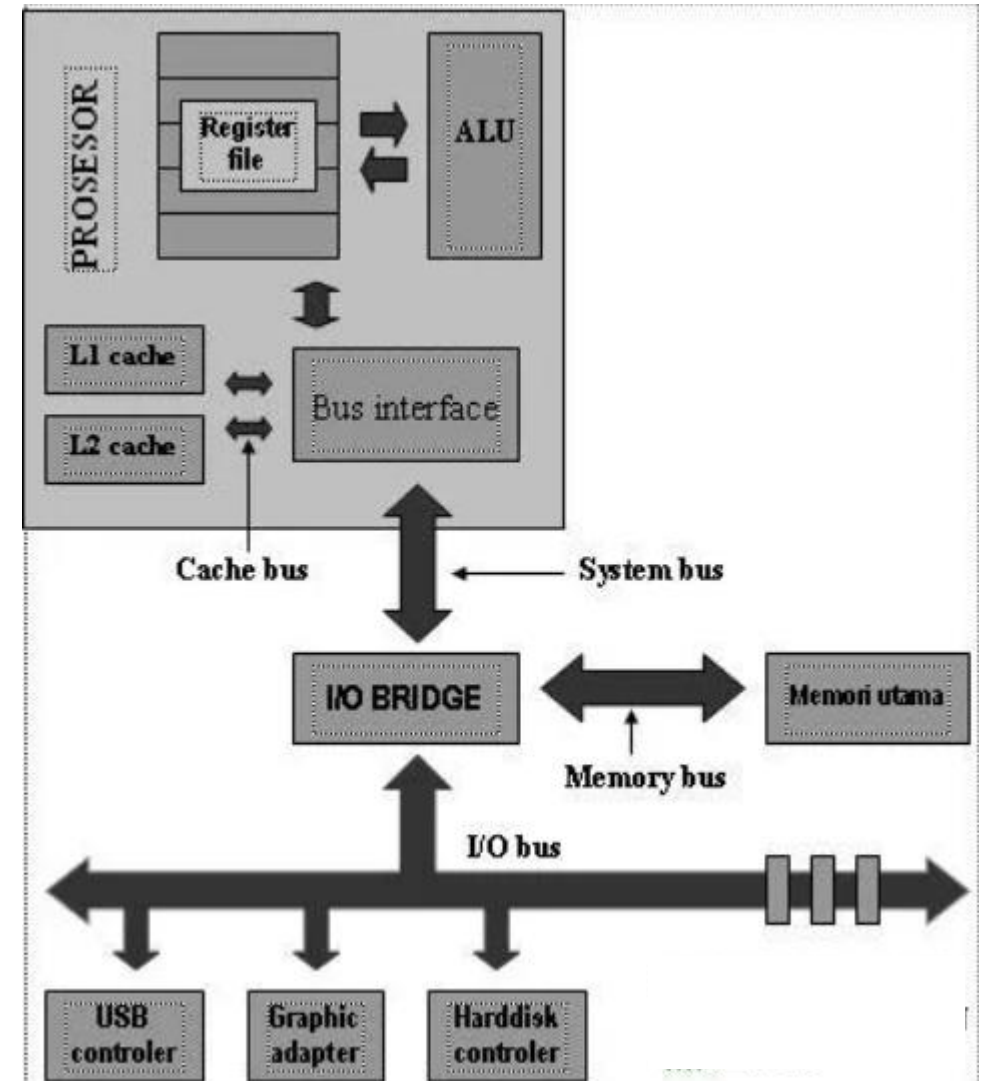
Memory models

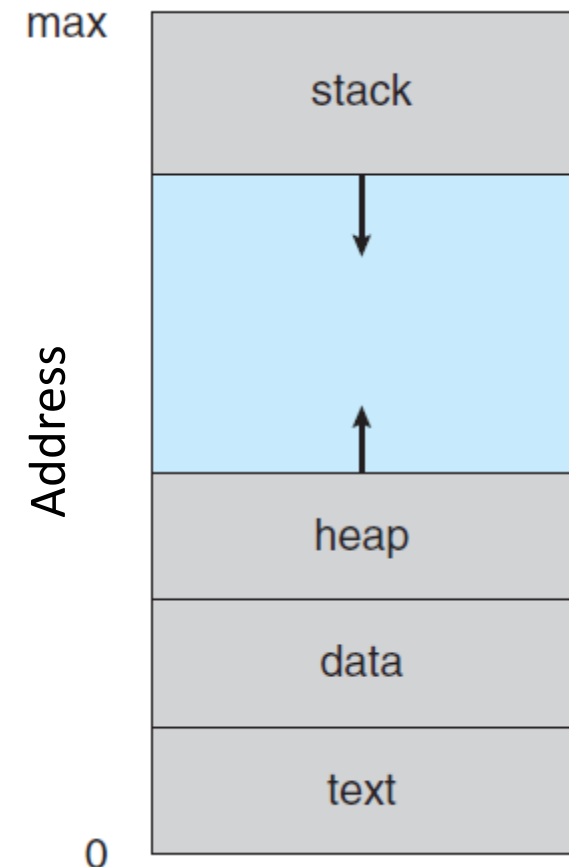A **memory model** defines how memory is used by programs, including:

- **Memory addresses** accessible to the program:
  - **Pointers** in C.

- **Data types**, e.g.:
  - char (1 byte);
  - int (2 or 4 bytes);
  - double (4 or 8 bytes).

- **Alignment** (byte order):
  - **Little Endian** (from LSByte to MSByte);
  - **Big Endian** (from MSByte to LSByte).
-

PowerPC(IBM)　　　　　　　x86 / ARM

Memory models

- **Memory allocation** (i.e. size and breakdown in space):
    - **Text** is a set of machine instructions (read-only).
    - **Data** is a statically allocated data (e.g., for global variables);
    - **Heap** is a dynamically allocated storage to store process variables and/or input data. The address for each data item is provided individually (e.g., *malloc*-function in C).
    - **Stack** store temporary data (usually, to call procedures) in a Last-In-First-Out (LIFO) manner. **Stack pointer** (remember **R13** from **register files**) keeps the memory address of the top of the stack.

#Process memory layout [4]:

max

stack

heap

data

text

0

Address

[4] Chapter 3: Processes, https://blog.csdn.net/iris_cyy/article/details/101033402
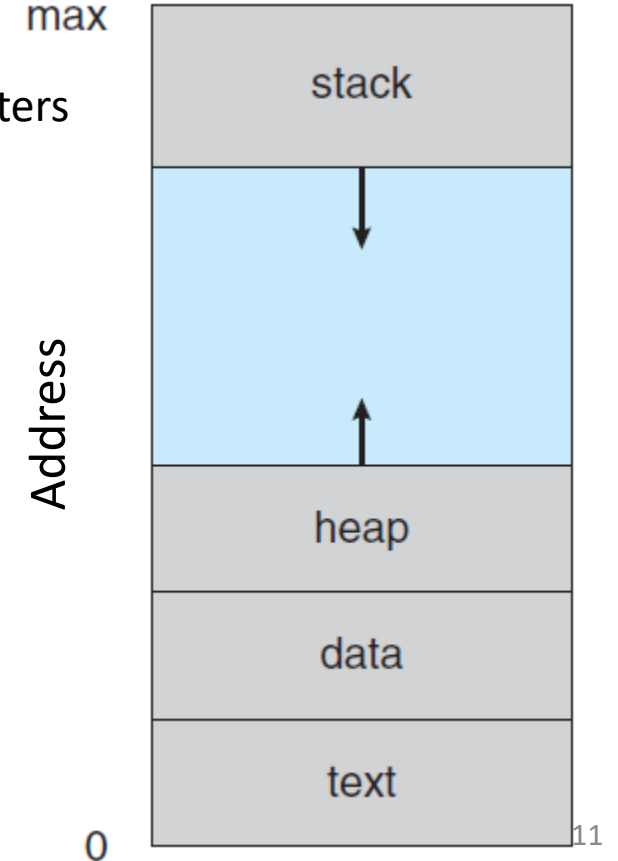
Memory model of C

C programs store data on the **stack**, on the **heap**, and in memory locations fixed by the compiler (i.e., **text** and **data**).

#Consider the following C program:

Process memory layout [4]:

```
1    int a = 2;              // Global variable stored in data
2    void foo(int b, int* c) {   // b and c are procedure parameters
3        ...                  // allocated on the stack when
4    }                        // the procedure is called
5    int main(void) {
6        int d;               // d and *e are local variable
7        int* e;              // allocated on the stack
8        d = ...;
9        e = malloc(sizeInBytes);  // Allocate memory for e
10       *e = ...;            // on the heap
11       foo(d, e);           // d is passed by value, while
12       ...                  // e is passed by reference
13   }
```

max

stack

Address

heap

data

text

0

[4] Chapter 3: Processes, https://blog.csdn.net/iris_cyy/article/details/101033402

To sum up

- **Memory technologies** are determined by their physical realization and characteristics:
  - Volatility (i.e., reliance on power supply);
  - Access type (i.e., random or in sequence).
- **Memory hierarchy** combines different memory technologies to:
  - Increase the overall memory capacity;
  - Optimize cost, latency, and energy consumption.
- Processor memory includes:
  - **Register files** – stores data for immediate access by CPU;
  - **Scratchpads** – move data to/from remote (external) memory;
  - **Caches** – duplicate data to/from remote (external) memory.
- **Memory models** determine how processes interact with memory
  - **Address space, data types, alignment and allocation of data**.

The end!

See you next time – April 17.

Do not forget about Assignment 1 deadline – **Thursday, April 13, 23:59!**

FAQs about assignments:

1. In Assignment 1 (Modelling), questions within a problem are interlinked in the alphabetic order starting from (a) if not explicitly stated otherwise. For example, for Problem 1 question (f), you are required to implement PI feedback control for a system obtained after answering questions (a), (b), (c), (d) and (e).

2. The above does not mean that the score for the latter questions depend on the answers for previous questions.

3. In Assignment 2 (Programming), the requirements for Programming task have been demonstrated on the Lab session, which can be accessed from the Lab session capture on BBB:

- Programming task 1 – starting from 1:41:50;

- Programming task 2 – starting from 3:08:30.