Why assembler programming ?

- High level programming languages are designed to help programmers construct ***correct*** **algorithms**
- To achieve this goal, high level programming languages provide:
    - Many structures or constructs, e.g., if, if-else, while, for, etc
    - Protections, e.g., private variables
    - Comforts, e.g., automatic conversion from short to int, etc.
- An assembler instruction is a **direct** command to the computer. Assembler programming does not have the above "comforts". It has **raw power**:
    - When you program a computer through a high level language, you are instructing the computer **indirectly** through an "interpreter". That "interpreter" is the compiler. Many compilers have built-in safety measures that prevent programmers to do something harmful, e.g., accessing private variables in a class.
    - When you program a computer through assembler code, you are instructing the computer **directly**. You can do **anything** you want (granted, you have to know what you're doing first !)
- Advantages of assembler programming over high level language programming:
    - Human made assembler programs can be made highly optimal, can run faster and is much smaller than programs generated by a compiler - important in embedded applications like "smart card"
    - Functions that manages computer resources - such as registers - must be written in assembler. Two of such functions are: (1) saving a "process context" and (2) restoring a "process context".

      A program that is running in the compiler can be **paused**, and made to resume later by (1) saving the process context before stopping the program and (2) restoring the process context before resuming the program
- General practice:
    - Use high level programming language as much as possible (programs are easier to write and maintain)
    - Functions that (1) **cannot** be coded using high level programming languages or (2) must be **highly optimized**, are written in assembler
    - Methods/functions written in high level programming language can interact (can invoke one another and pass parameters) with functions written assembler programs - we will see this later in the course