Structure of computer main memory

- **Review of CS170/CS171**
  - **Structure** of **computer memory**:

    - **Computer memory** consists of **sequence bytes**
    - Each **bytes** consists of 8 **bits**
    - A **bit** = the **most elementary electrical memory element**

      - A **byte** can **"remember" (store)** either the **value 0** ***or*** the **value 1**

        (a **byte** is an **electrical switch**: it can be off or on)

      Therefore:

      - A **byte** (= **8 bits**) can contain $2^8 = 256$ different values

        (A **byte** can **remember only** ***one*** specify value **at a time**.

        A **byte cannot remeber** ***all*** **256 values at the** ***same*** **time**)

    - **Each byte** is **identified** by an **address**

  - **Usage** of a **byte**:

    - **Byte** = a **memory element** used to **store values** from a ***small*** **representation set**

      Example:

      - **Character set**:

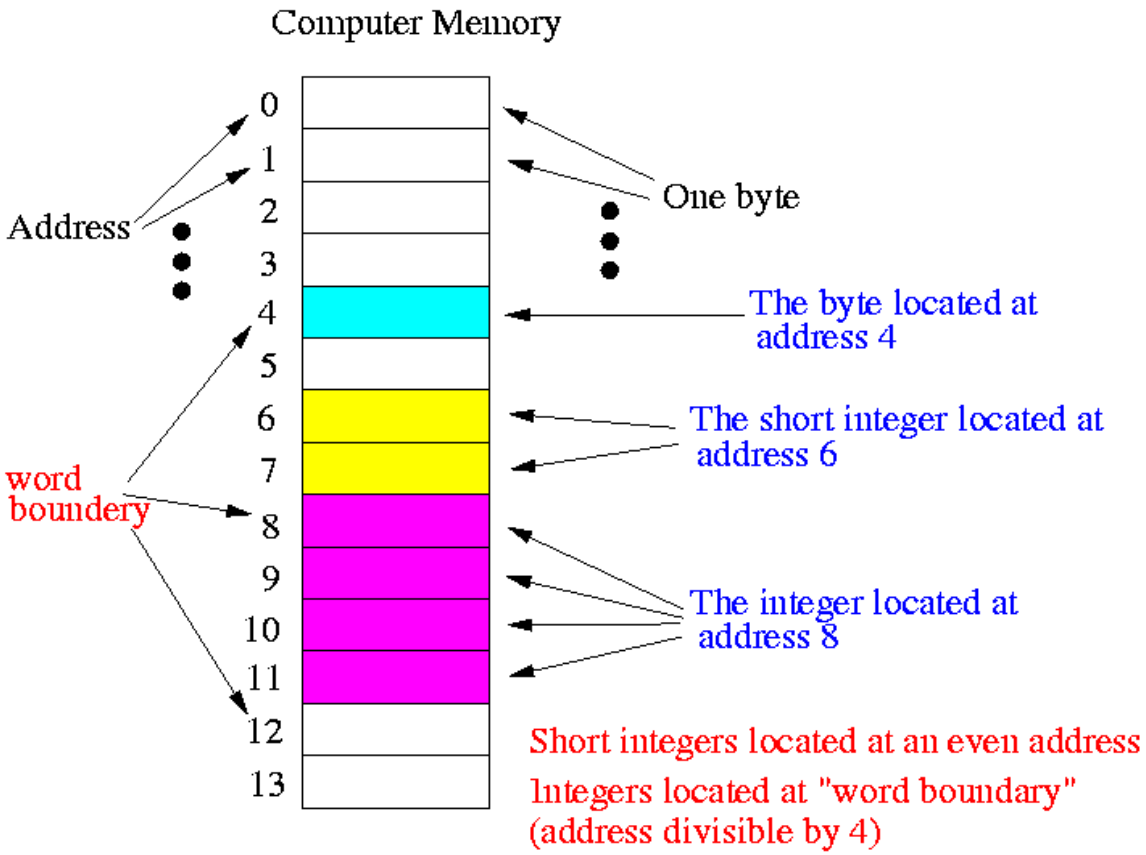
        ```
        a, b, c, b, e, .....
        A, B, C, D, ....
        0, 1, 2, ....,
        !, @, #, $, %, ....

        (Total approximately 128 different characters)
        ```

- **Combining *adjacent* memory cells**
  - **Larger** memory cells are needed for larger value sets, like integers, floating point numbers.

- 2 bytes are used to hold values of "**short**" integers (the `short` type in Java)
- 4 bytes are used to hold values of "**ordinary integer**" integers (the `int` type in Java)
- 8 bytes are used to hold values of "**long integer**" integers (the `long` type in Java)

- To make "larger" memory elements, consecutive bytes in main memory are used together:



- **Alignment constraint**

  - Due to **architectural constraints**, the **computer manufacturer** will **always** imposes a **constraint** on:

    - ***where*** the ***different*** **types of variables** can be **placed (= stored)** in memory.

  - **Alignment constraits** on **variables**:

    - A `byte` **(typed) variable** can be **placed (located)** ***anywhere*** in memory (= **divisible by 1**)
    - A `short` **(2 consecutive bytes) variable must** be **placed (located)** at an ***even*** **address** (= **divisible by 2**)
    - A `int` **or** `float` **(4 consecutive bytes) variable must** be **placed (located)** at an **address** that is **divisible by 4**
    - A `long` **or** `double` **(8 consecutive bytes) variable must** be **placed (located)** at an **address** that is **divisible by 8**

- **Note:**

  - When ***you*** write a **Java program**:

    - the **Java compiler (`javac`)** will **take care** of **allocating the program variables** at **memory locations** that **comply** with the **allocation constraint**

  - When ***you*** write an ***assembler* program** in **CS255**:

    - ***You*** will be **responsible for (= *must* !!!)** **taking care** of **allocating the program variables** at **memory locations** that **comply** with the **allocation constraint**

- **Important observation/question about memory**

  - An **important Fact** and a subtle question:

    - **Knowing** that the computer memory consists of bytes and each byte consists of 8 bits and each bit remembers one of the value 0 or 1, we can conclude that:

      - The computer memory contains **ONLY** a bunch of 0 and 1...

1	0	0	0	1	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	1
1	0	0	1	1	0	0	0
0	0	0	1	1	1	0	0
0	1	0	1	0	0	0	0
0	0	1	0	1	0	0	1
0	0	0	1	0	0	1	0
0	0	1	1	0	1	0	0

    - How can the computer know where a specific variable **begins** and **ends** in memory ???

      There are **no markers** in the memory to denotate where the address item are located and how big they are (how many bytes are used to hold the value of the item - short: 2 bytes, int 4 bytes, etc) !!!

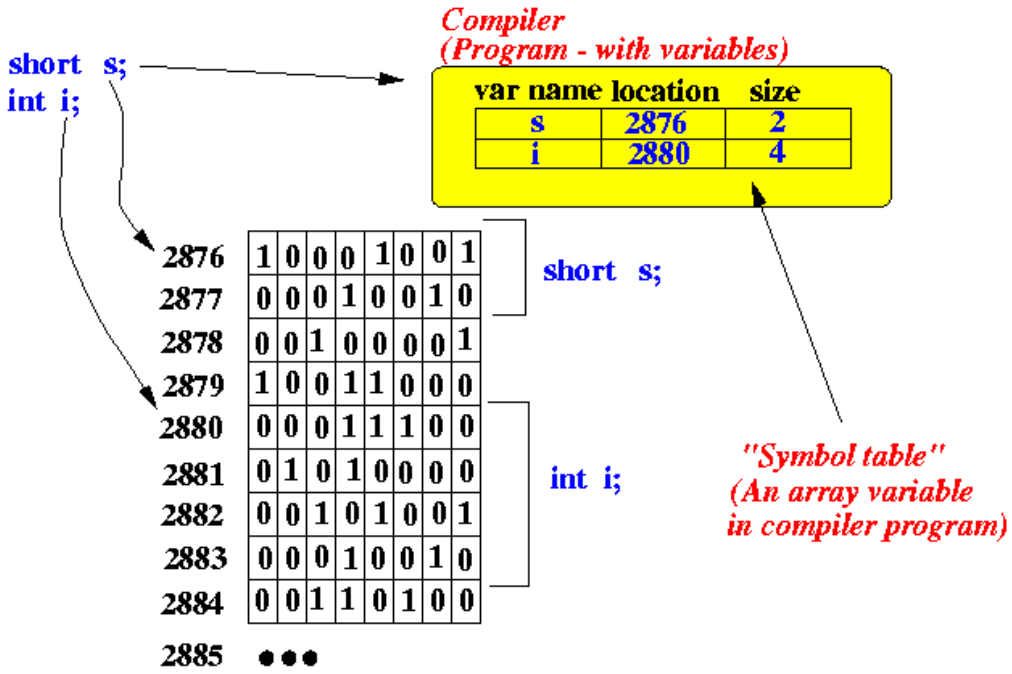
  - The answer to this subtle questions is found in the **compiler** and **programming language**:

    1. When a variable of a given type is first **defined** in a program, the variable stored in the computer memory.

       The compiler will find an **unused** portion of consecute memory bytes to store the variable.

       In addition, the **starting location** (= address) of the allocated memory and the **size** (number of bytes) are **remembered** by the ***compiler***.

       Schematically:

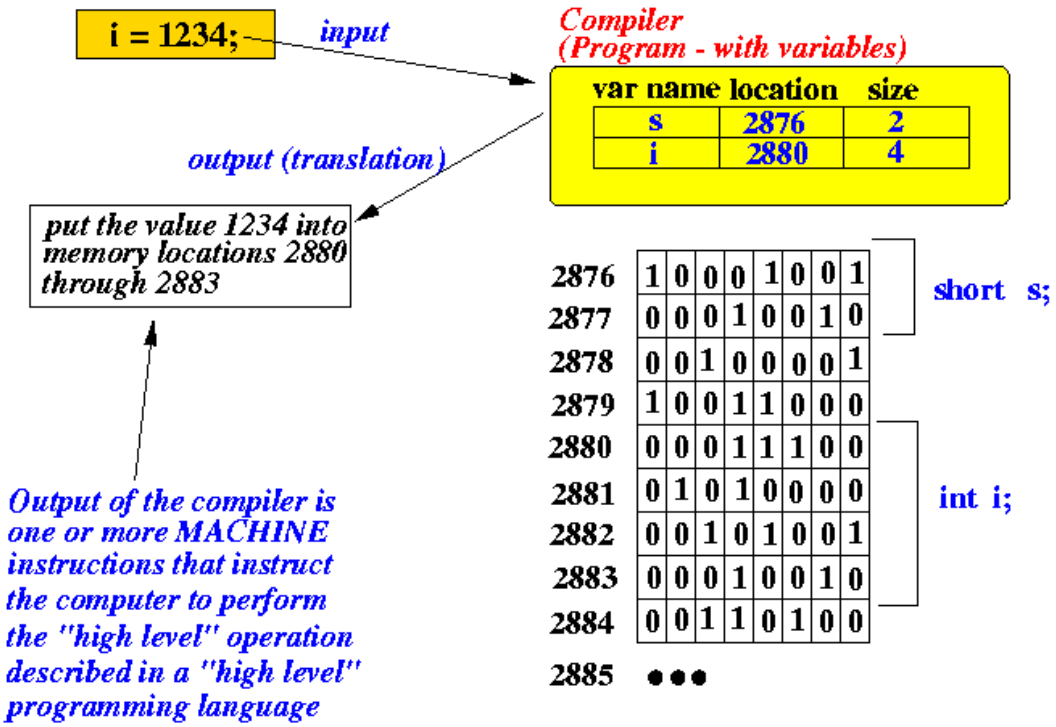


- When the **compiler** processes a **variable definition** clause in a program, it assigns an **unused portion** of the memory of the proper size (depending on the type of the variable.
- After assigning the memory, the compiler **record**
  - The variable name
  - The starting location of the variable in memory
  - The size of the variable

  in its **symbol table** variable

  (Remember, the compiler is a program and a program can have variables... The symbol table is one of the many variables used to write a compiler...)

2. When the variable is **referenced** (used) later in a program statement, the **compiler** will generate **native computer instructions** to access the memory used to store the indicated variable:



- When variable **i** is referenced in the program, the ***compiler*** consults its **symbol table** and determines that variable **i** is stored in the 4 bytes starting at address 2880
- It will generate the **appropriate computer instructions** to manipulate these 4 memory locations
- Note: You will soon see that you need to specify the **SIZE** of the memory data in **EVERY assembler instructions** !!!

- (Now you may understand why most programming languages **require** that you ***first*** **define** a variable before you **use** it....)

  The compiler needs to **find** the variable !!!!
