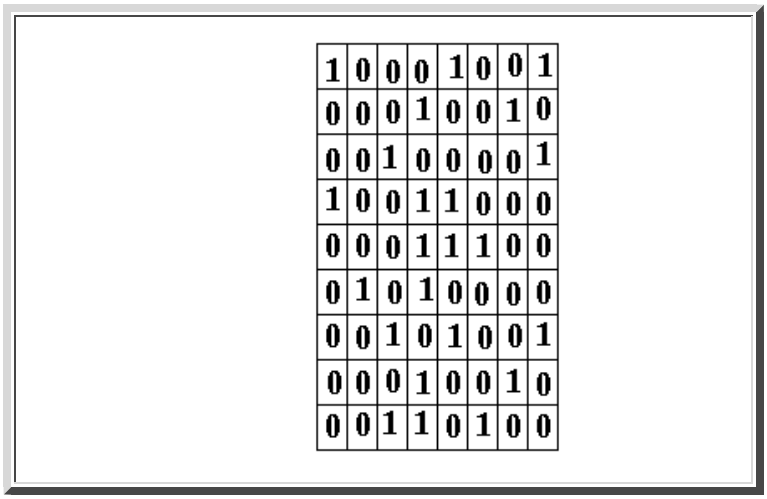
The program *translation* process

- **Machine instructions**
  - **Recall** that the **computer memory** contains a **bunch** of **binary digits (= bits)**:



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

  - **Fact:**
    - ***Some*** of these **bits** represent **instructions** of the **computer program**
      - We **call** them: **machine instructions**
    - ***Some*** of these **bits** are **variables** used in the **computer program**
  - **Important thing** to **remember** for this course:
    - ***Everything*** stored inside the **computer (memory)** must be ***represented*** by:
      - **binary numbers** (they are formed by **binary digits, or bits**)
- **The Java program translation process**
  - A **programmer (= you)** writes a **(Java) program**:

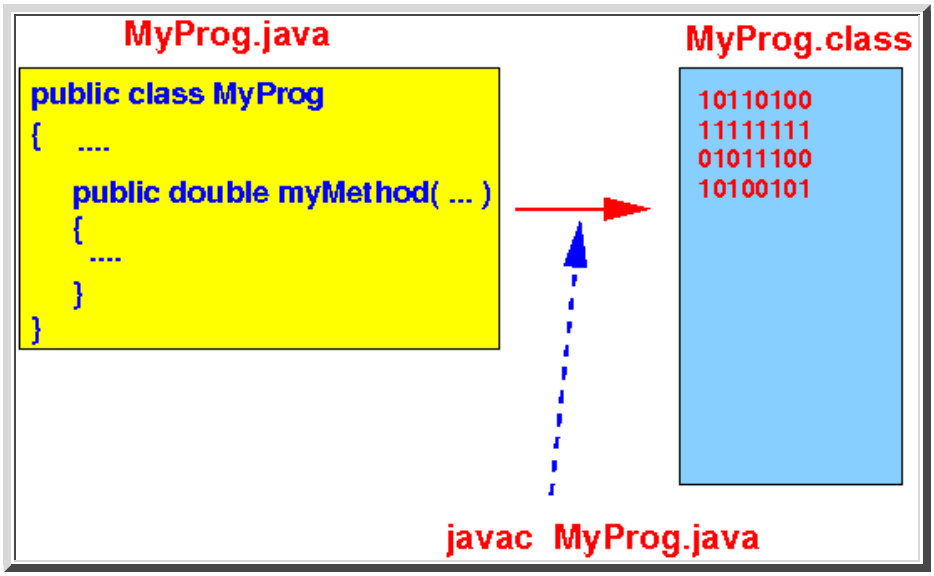
```
public class MyProg
{
    public  int  x;

    public  double myMethod( ... )
    {
       ...
    }
```

```
        ....
    }
```

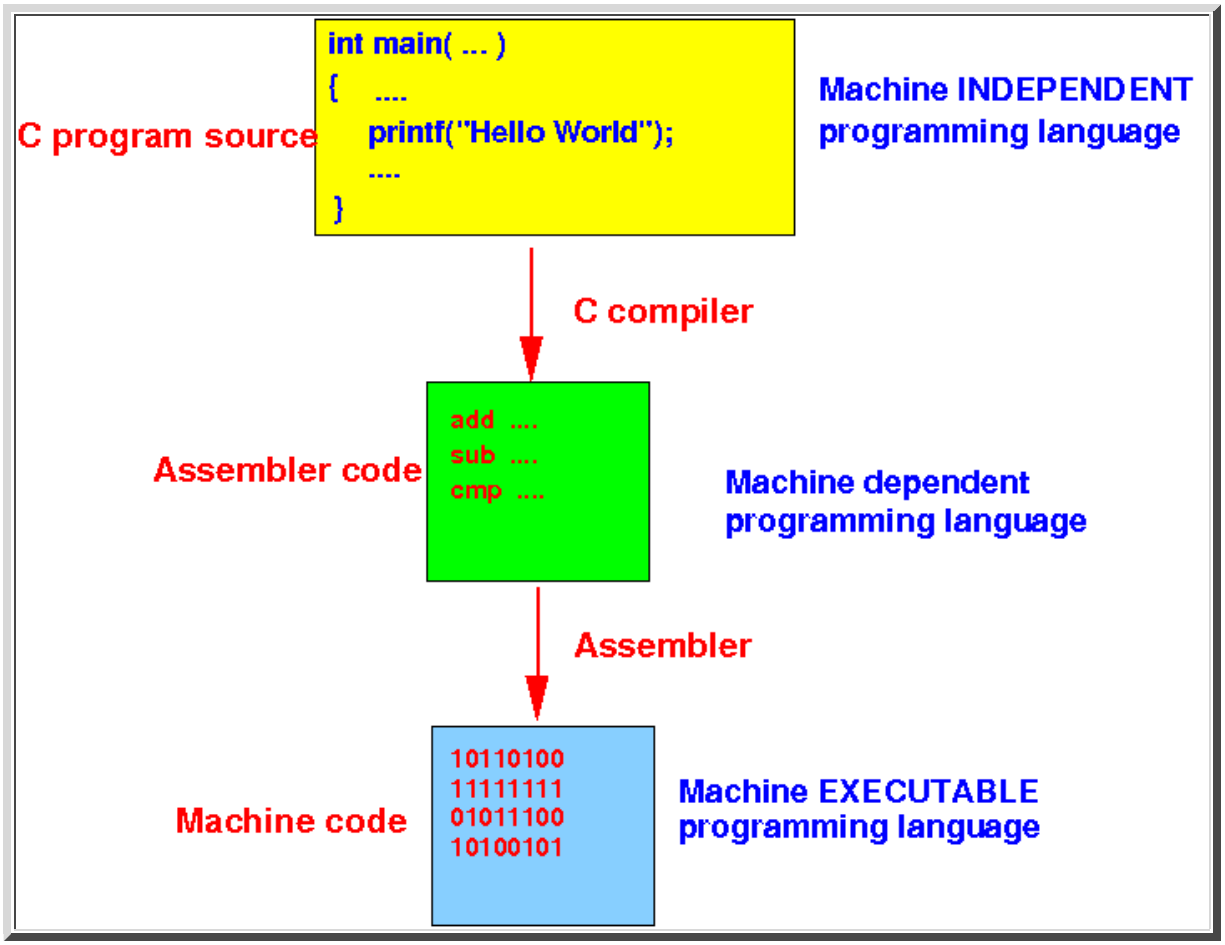
Fact:

- The **computer** **cannot** execute the **statements** in the **(Java) program**

- A **translator program** (`javac`) must **first translate** the **statements (commands)** into a **sequence** of (equivalent) **machine instructions**:



- **The translation process in general**
  - The **program translation process** for ***most*** **programming languages** is as **follows**:



- **Example Program:** *Example*

  - Check out the **C programs** in `/home/cs255000/demo/c-asm`

  - **Compile** the `main.c` program with this **command**:

    ```
    gcc  -S  main.c
    ```

    The **C compiler** `gcc` will produce the **assembler program** `main.s`

  - You can then **compile** the **assembler code** `main.s` with:

    ```
    gcc   -c   main.s
    ```

    The **C compiler** will invoke the **assembler** to produce the **machine (object) program code** `main.o`

- **Assembler programming language**

  - **Assembler language**:

- **Assembler programming language** = a **machine dependent** **programming language** that is ***closely*** **associated** with the **machine language** of a **computer**

  In fact:

  - **Each instruction** in an **assembler language** **corresponds** (uniquely) to ***one*** **instruction** in the **machine language** !!!

- **Assembler program:**

  - **Assembler program** = a **program** that is **written** in an **assembler language**

- **Why we learn programming in assembler in CS255**

  - **Main reason:**

    - **Because** the **assembler language** is ***closely*** **associated** with **machine instructions**, we will **understand** ***what*** **is going on** inside the **computer** when it **execute** a **computer program** !!!
