Assembler Tutorial

This program is part of the software suite
that accompanies the book

The Elements of Computing Systems

by Noam Nisan and Shimon Schocken

MIT Press

www.idc.ac.il/tecs

This software was developed by students at the
Efi Arazi School of Computer Science at IDC

Chief Software Architect: Yaron Ukrainitz

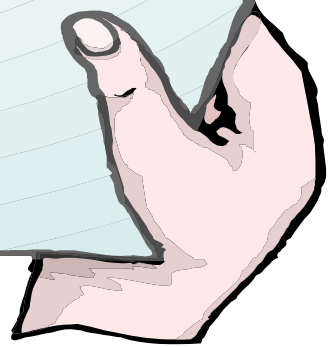
Background

"The Elements of Computing Systems" evolves around the construction of a complete computer system, done in the framework of a 1- or 2-semesters course.

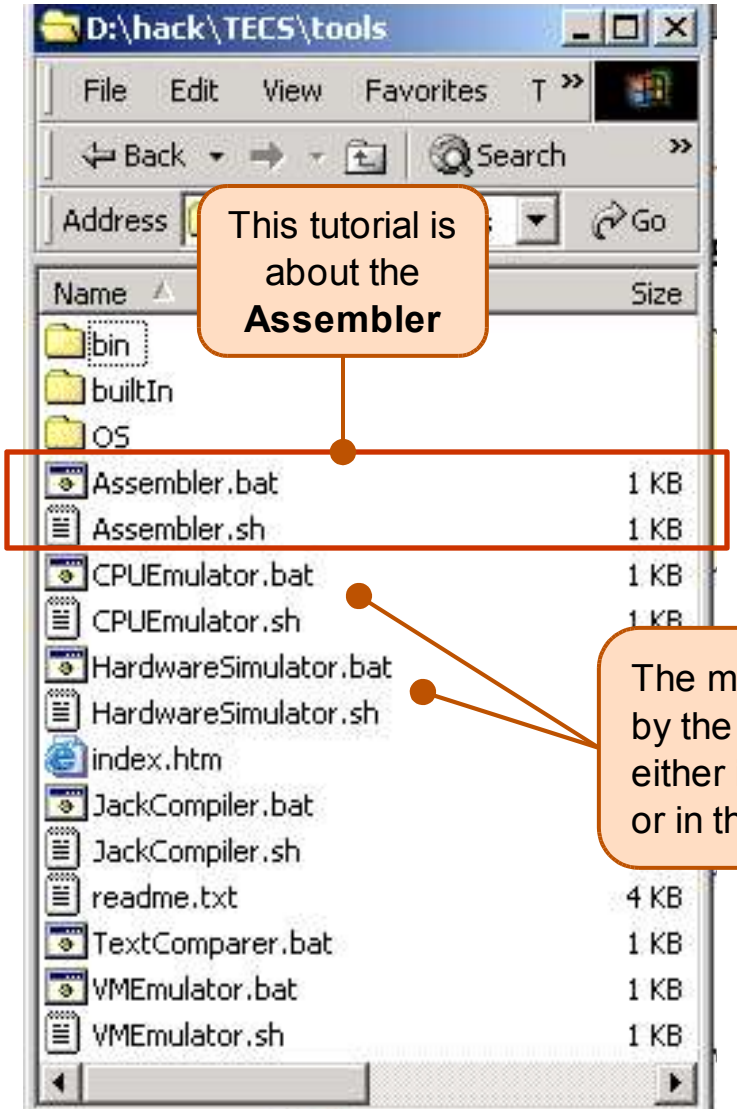
In the first part of the book, we build the hardware platform of a simple yet powerful computer, called Hack. In the second part of the book, we build the computer's software hierarchy, consisting of an assembler, a virtual machine, a simple Java-like language called Jack, a compiler for it, and a simple operating system, written in Jack.

The book is completely self-contained, requiring only programming as a pre-requisite.

The book's software suite includes some 200 test programs, test scripts, and all the software tools necessary for doing all the projects.



The book's software suite



(All the supplied tools are dual-platform: **`Xxx.bat`** starts **`Xxx`** in Windows, and **`Xxx.sh`** starts it in Unix)

Simulators

(**`HardwareSimulator`**, **`CPUEmulator`**, **`VMEmulator`**):

- Used to build hardware platforms and execute programs;
- Supplied by us.

Translators (**`Assembler`**, **`JackCompiler`**):

- Used to translate from high-level to low-level;
- Developed by the students, using the book's e solutions supplied by us.

d translators software;

- **`builtIn`**: executable versions of all the logic gates and chips mentioned in the book;
- `os`: executable version of the Jack OS;
- **`TextComparer`**: a text comparison utility.

Assembler Tutorial

I. Assembly program example

II. Command-level Assembler

III. Interactive Assembler

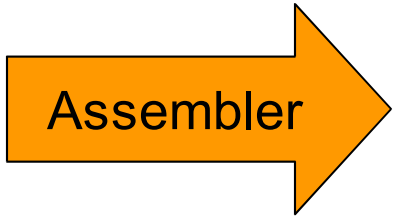
Relevant reading: Chapter 4: *Machine and Assembly Language*


Part I:
Assembly
Programming
at a Glance

Example

Sum.asm

```
// Computes sum=1+...+100.
    @i     // i=1
    M=1
    @sum   // sum=0
    M=0
(LOOP)
    @i     // if (i-100)=0 goto END
    D=M
    @100
    D=D-A
    @END
    D;JGT
    @i     // sum+=i
    D=M
    @sum
    M=D+M
    @i     // i++
    M=M+1
    @LOOP  // goto LOOP
    0;JMP
(END)      // infinite loop
    @END
    0;JMP
```



Sum.hack

```
0000000000010000
1110111111001000
0000000000010001
1110101010001000
0000000000010000
1111110000010000
0000000001100100
1110010011010000
0000000000010010
1110001100000001
0000000000010000
1111110000010000
0000000000010001
1111000010001000
0000000000010000
1111110111001000
0000000000000100
1110101010000111
```

Example

Sum.asm

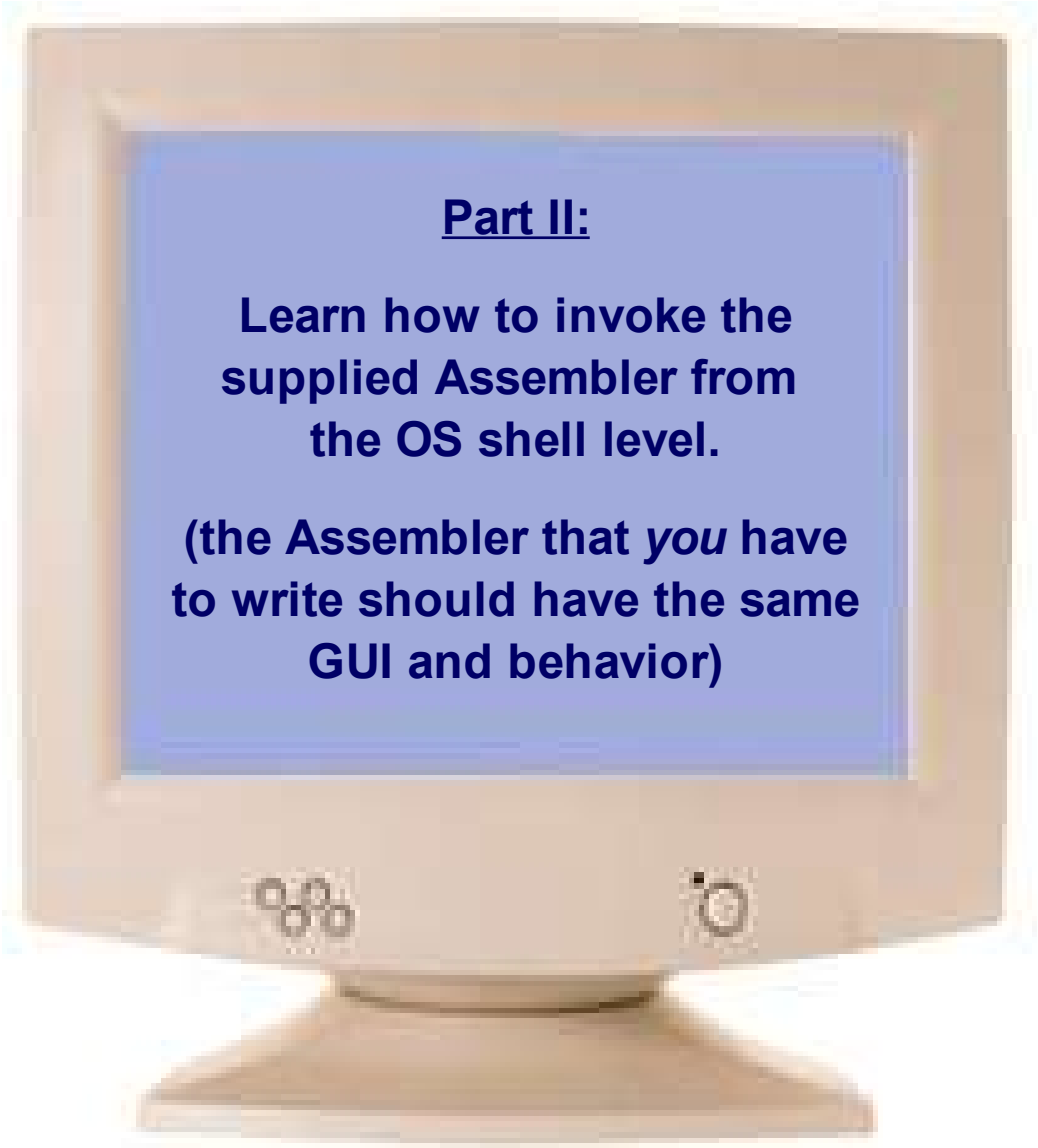
```
// Computes sum=1+...+100.
    @i     // i=1
    M=1
    @sum   // sum=0
    M=0
(LOOP)
    @i     // if (i-100)=0 goto END
    D=M
    @100
    D=D-A
    @END
    D;JGT
    @i     // sum+=i
    D=M
    @sum
    M=D+M
    @i     // i++
    M=M+1
    @LOOP  // goto LOOP
    0;JMP
(END)      // infinite loop
    @END
    0;JMP
```

The assembly program:

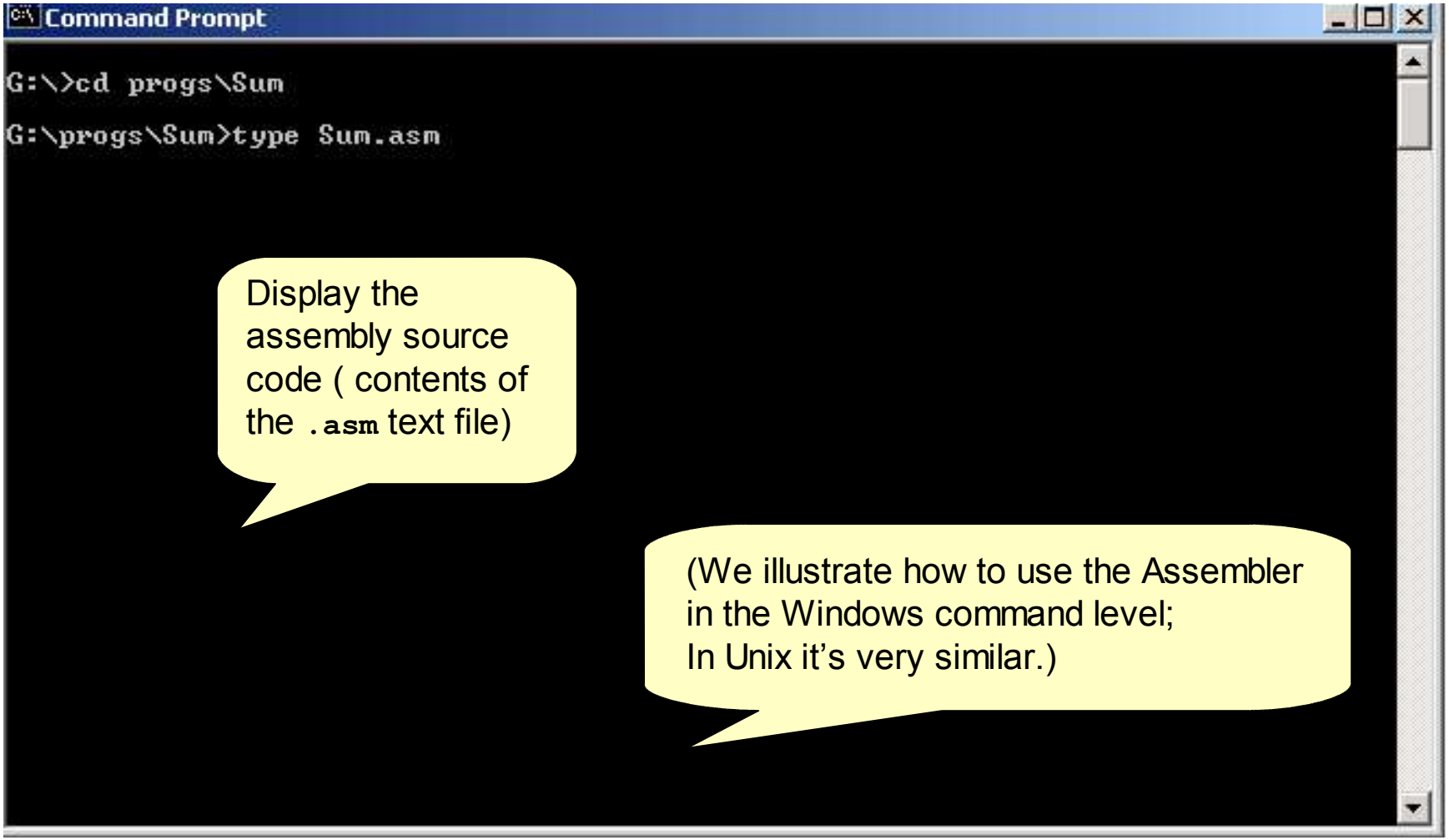
- Stored in a text file named `Prog.asm`
- Written and edited in a text editor

The assembly process:

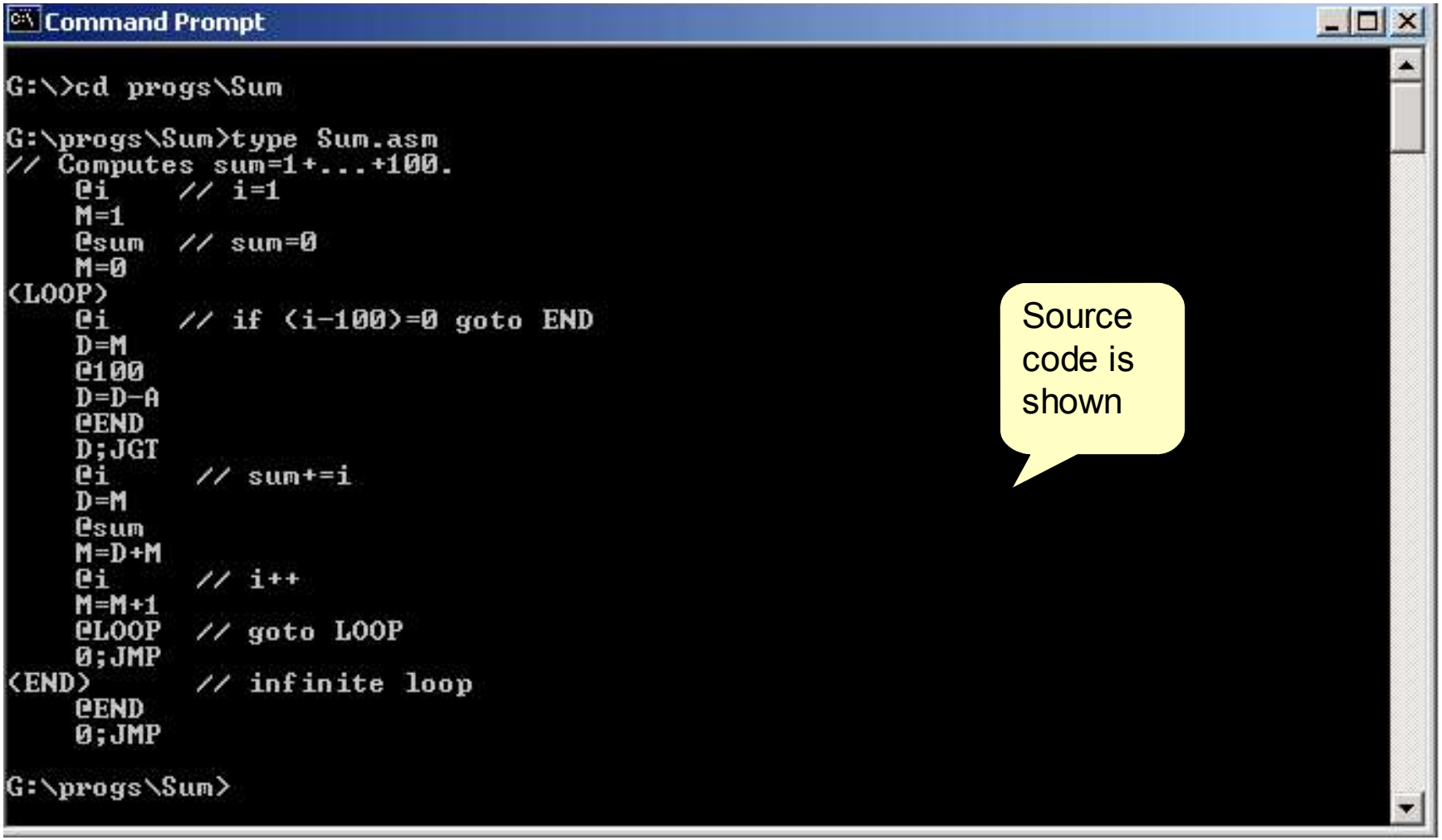
- Translates `Prog.asm` into `Prog.hack`
- Eliminates comments and white space
- Allocates variables (e.g. `i` and `sum`) to memory
- Translates each assembly command into a single 16-bit instruction written in the Hack machine language
- Treats labels (like `loop` and `end`) as pseudo commands that generate no code.


Part II:
Learn how to invoke the supplied Assembler from the OS shell level.
(the Assembler that *you* have to write should have the same GUI and behavior)

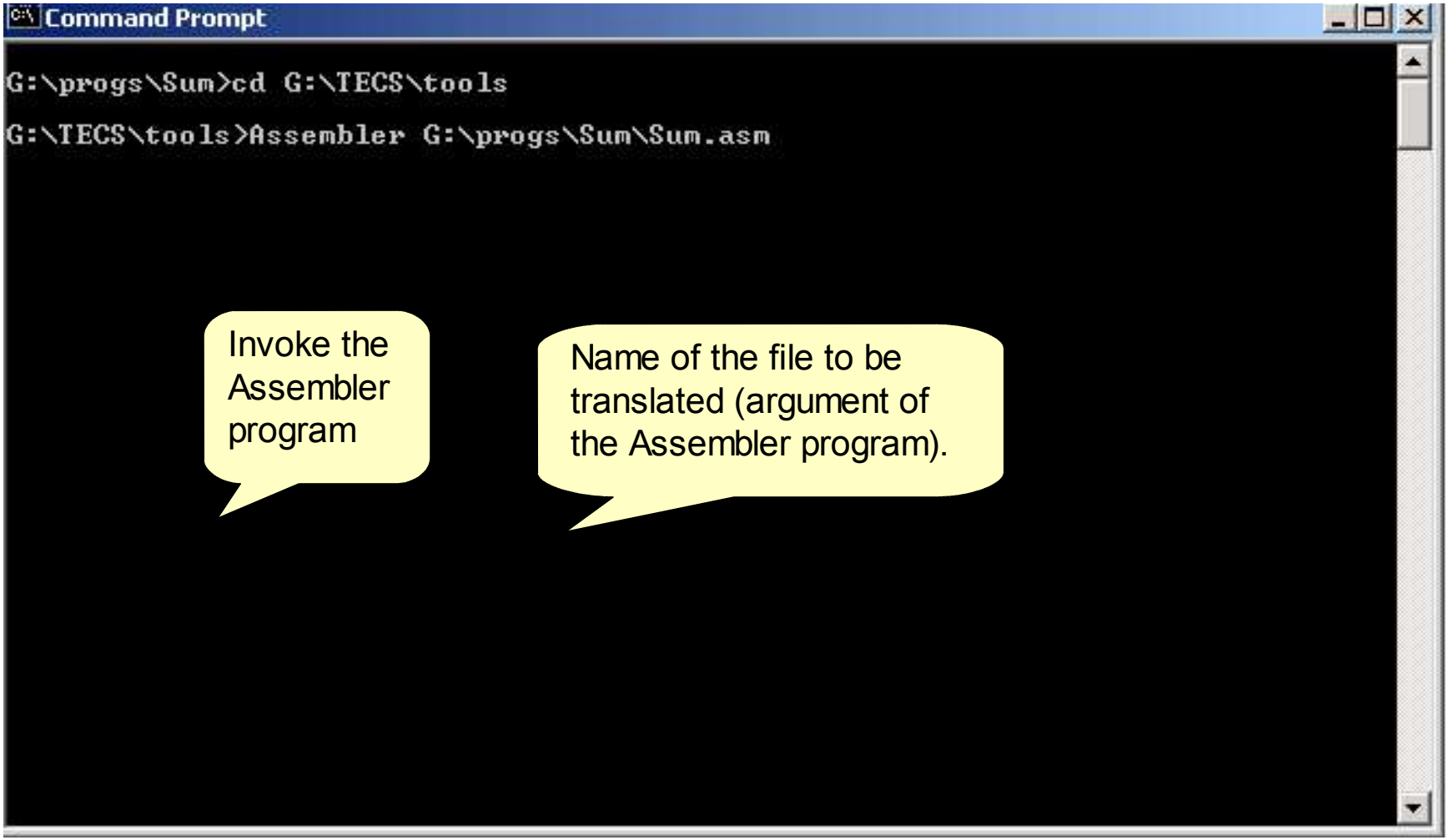
The command-level assembler



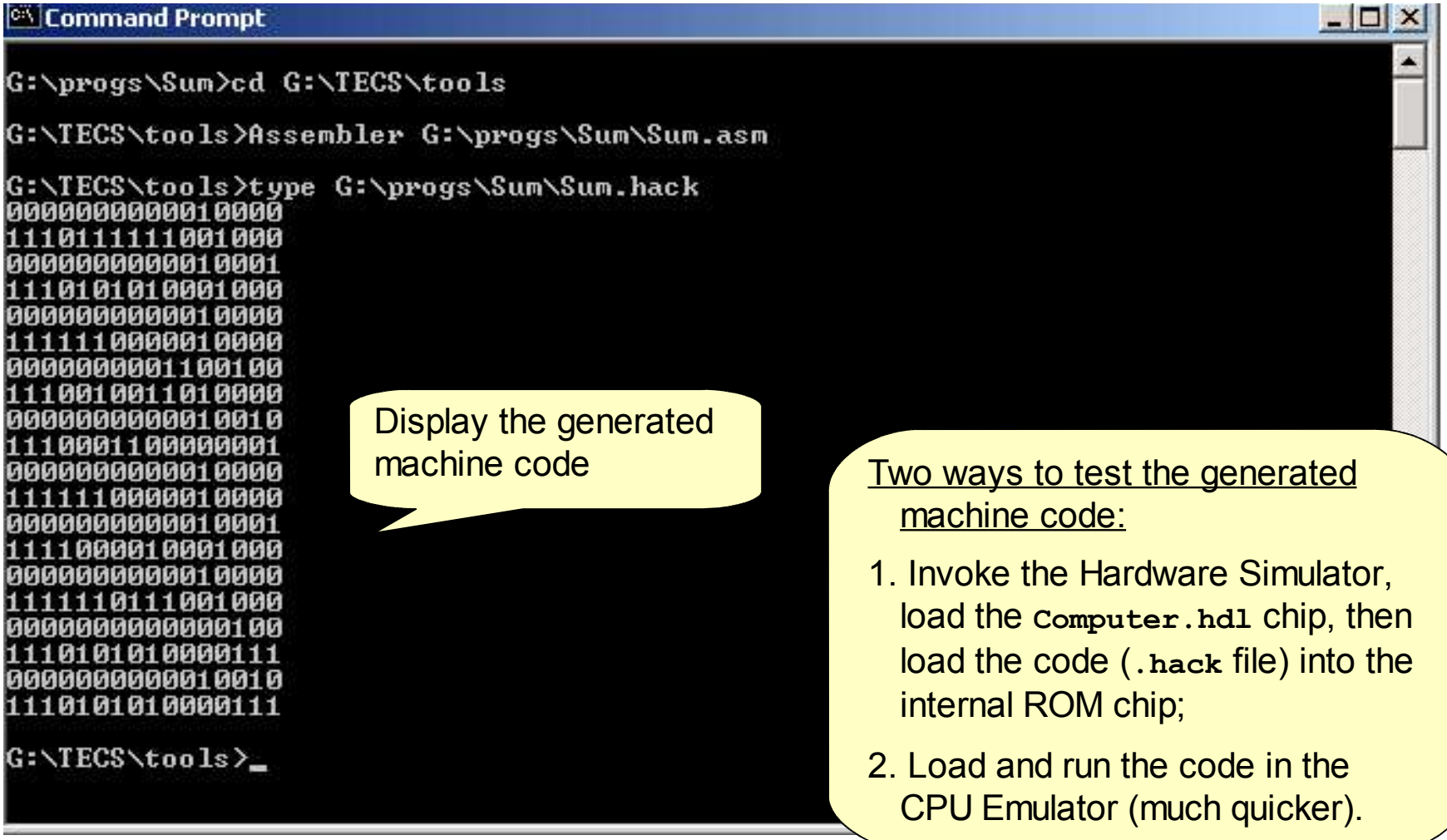
Inspecting the source file



Invoking the Assembler

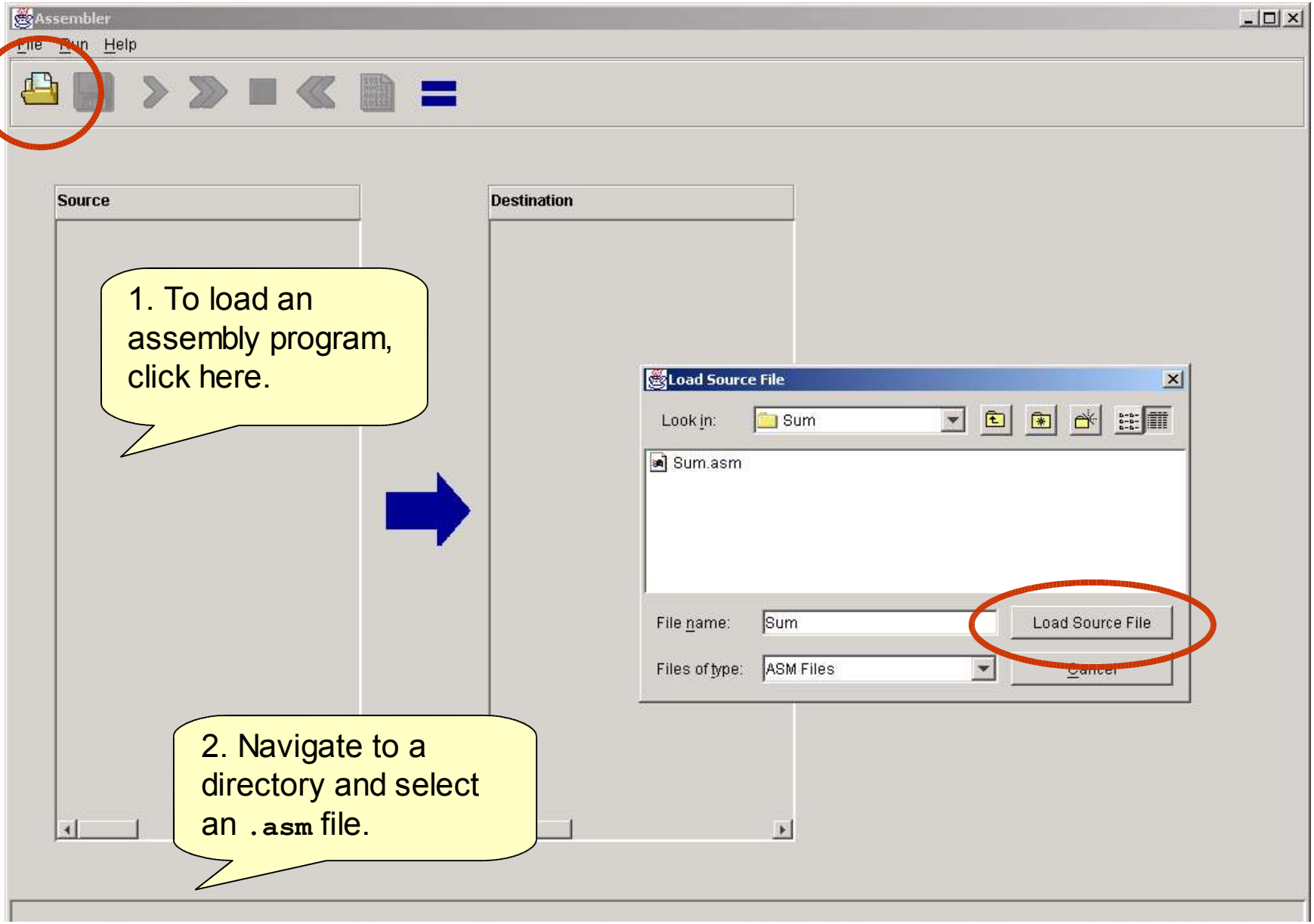


Invoking the Assembler

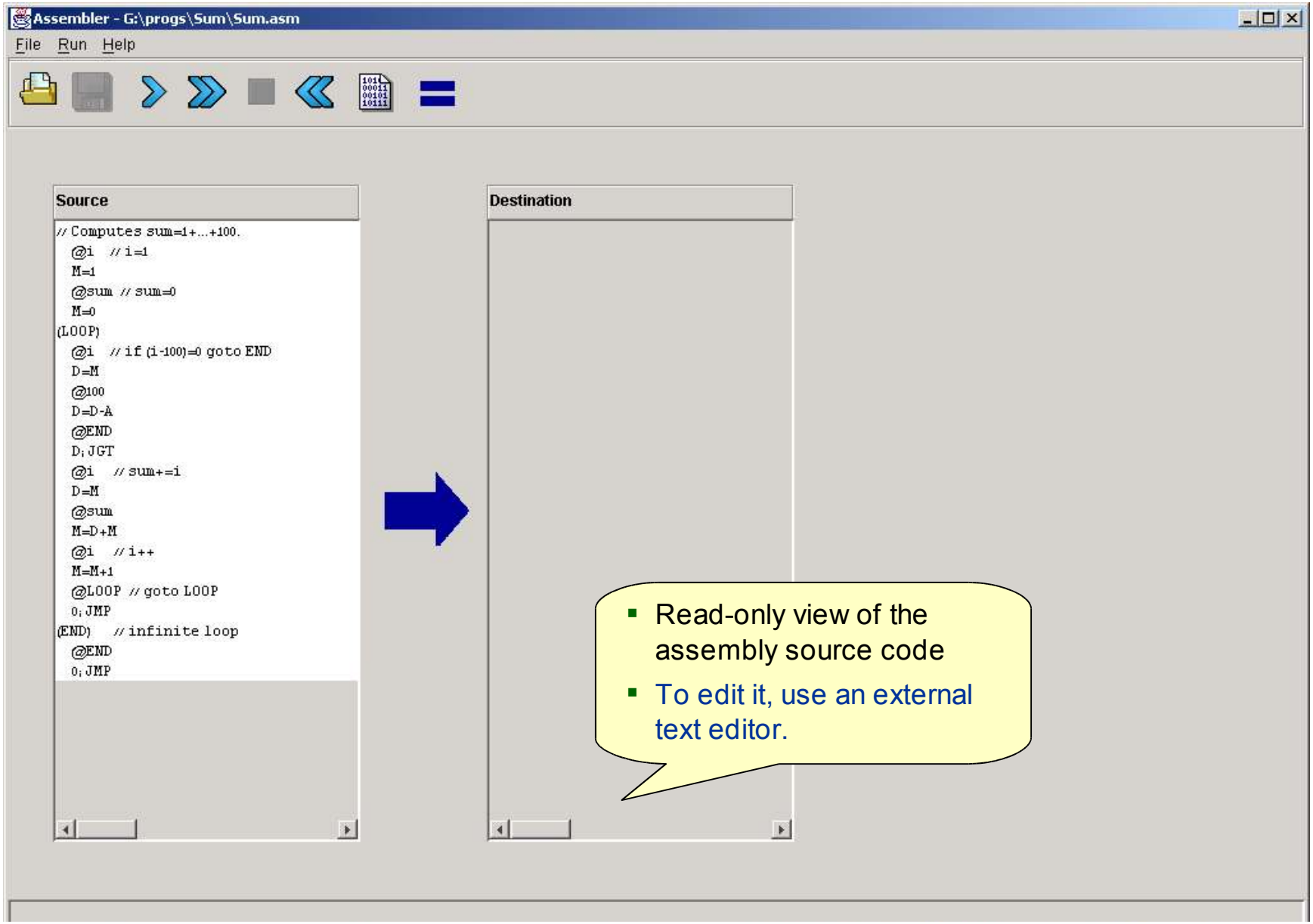



Part III:
Learn how to use
the interactive
Assembler

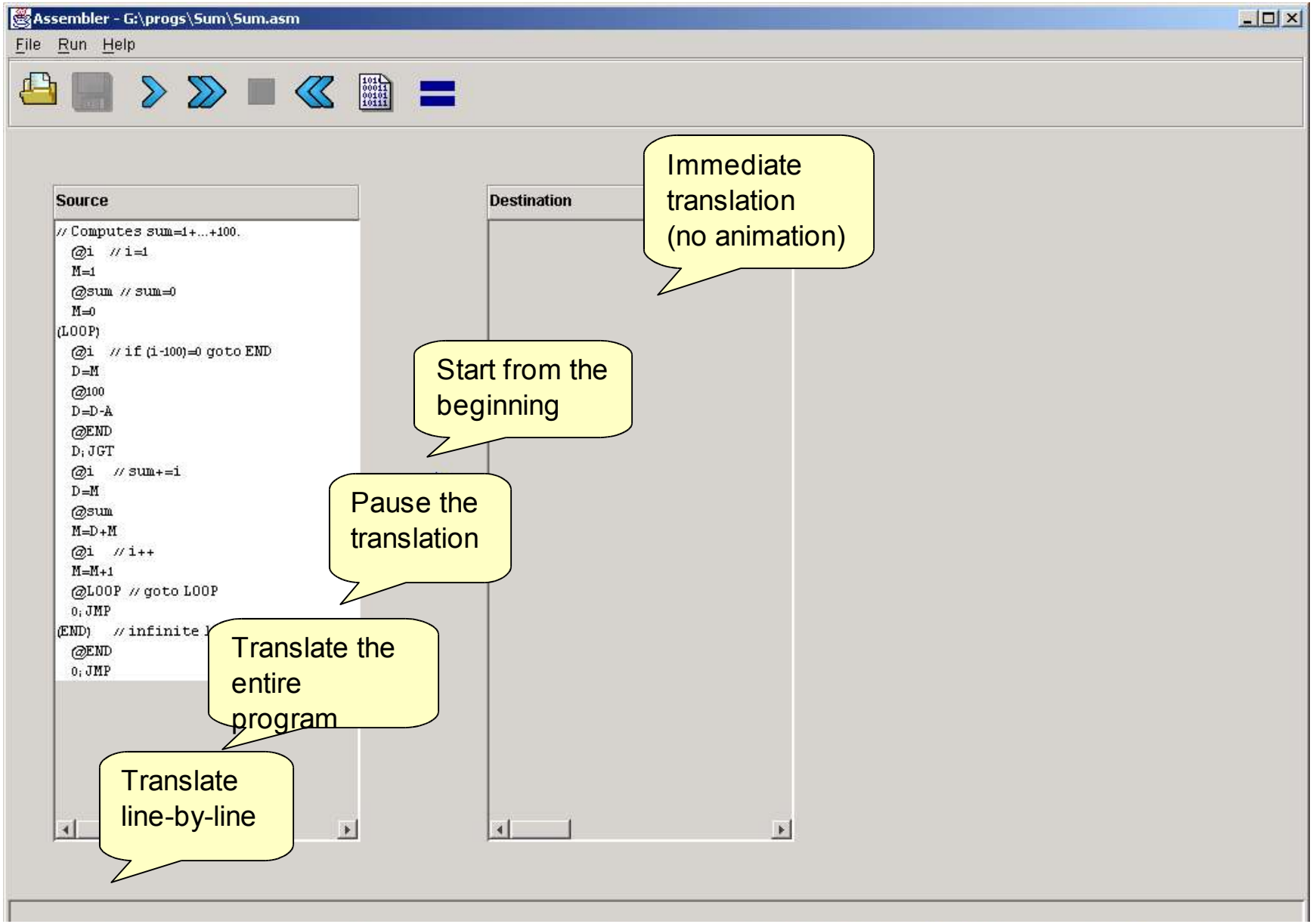
Loading an assembly program



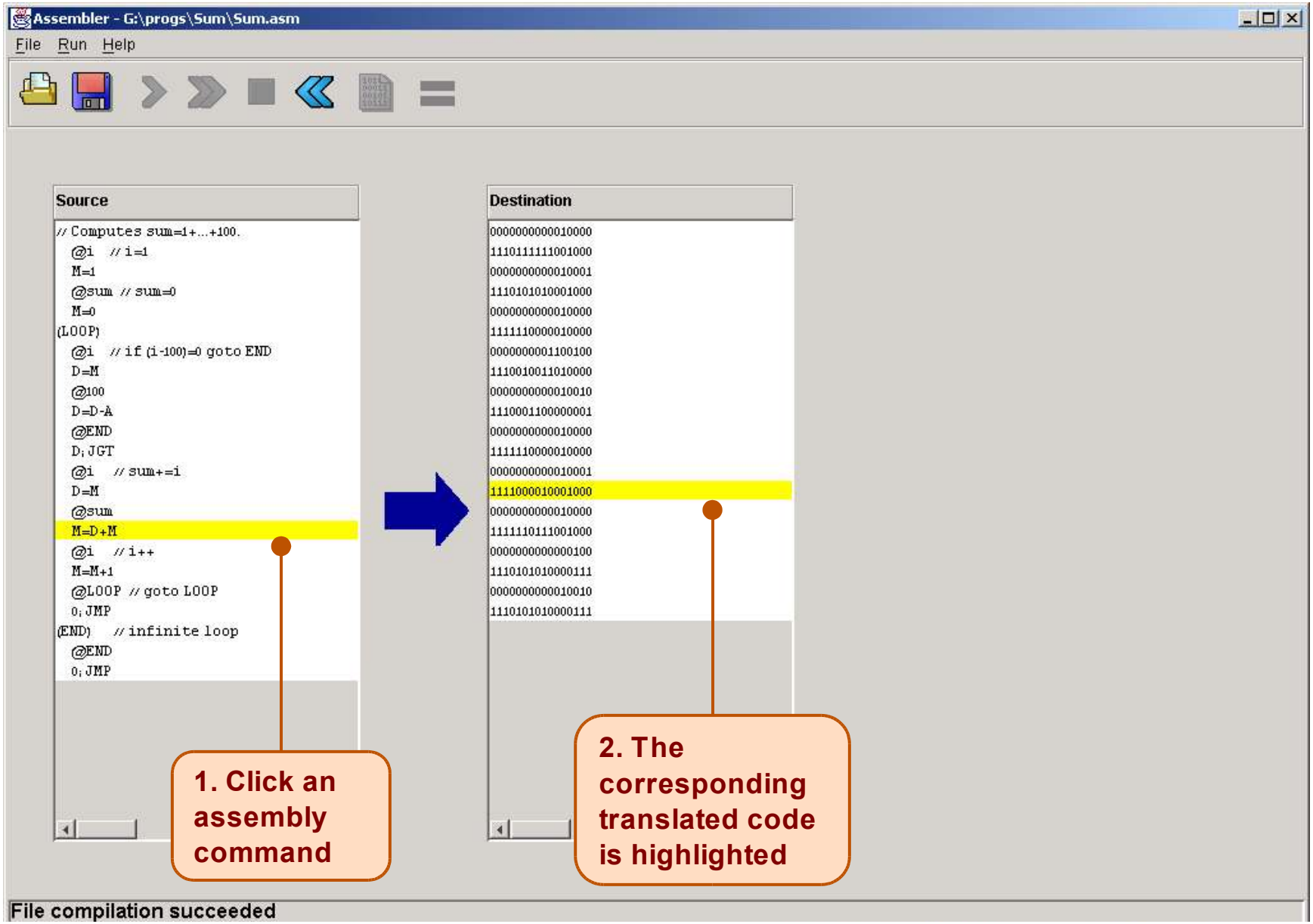
Loading an assembly program



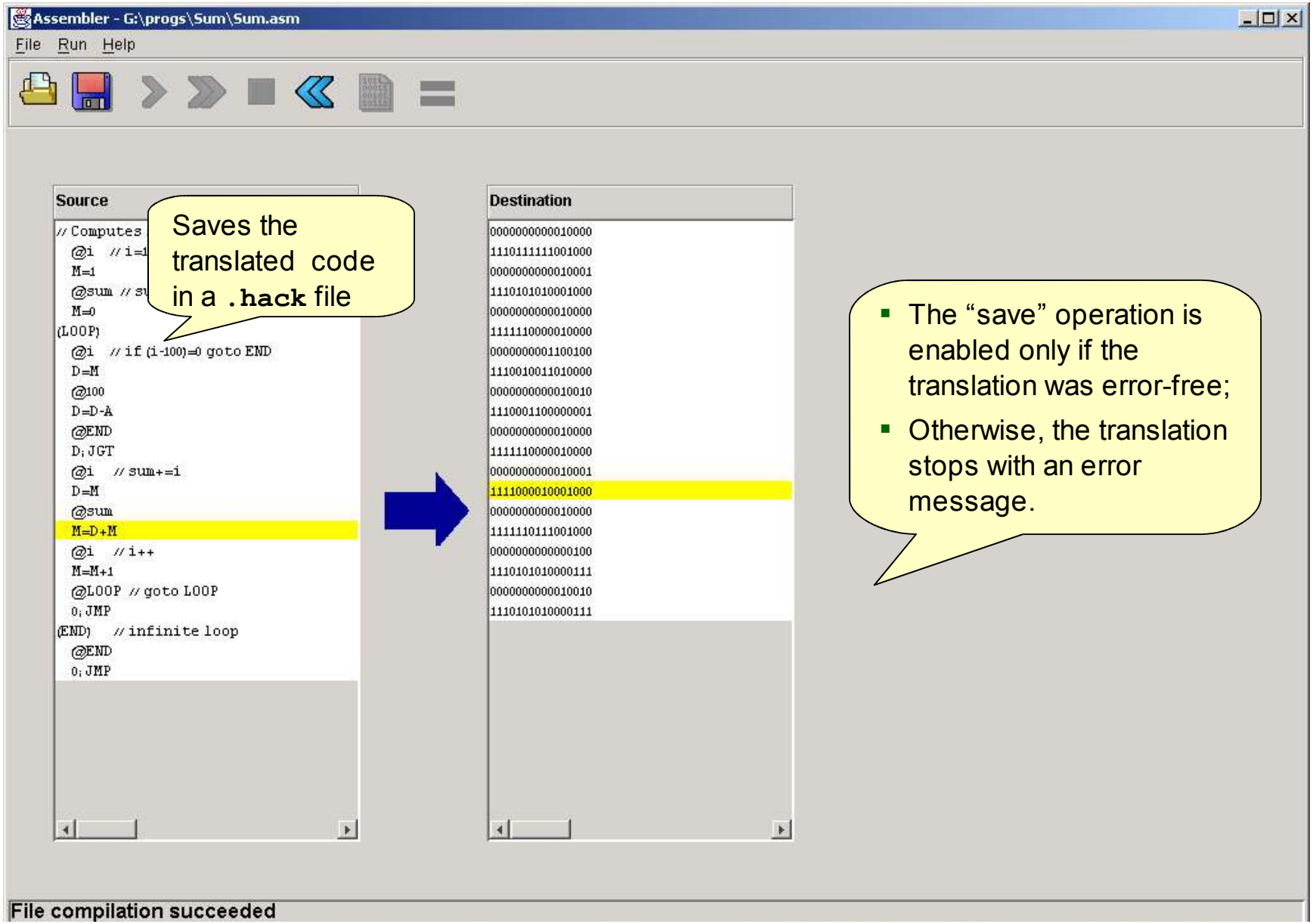
Translating a program



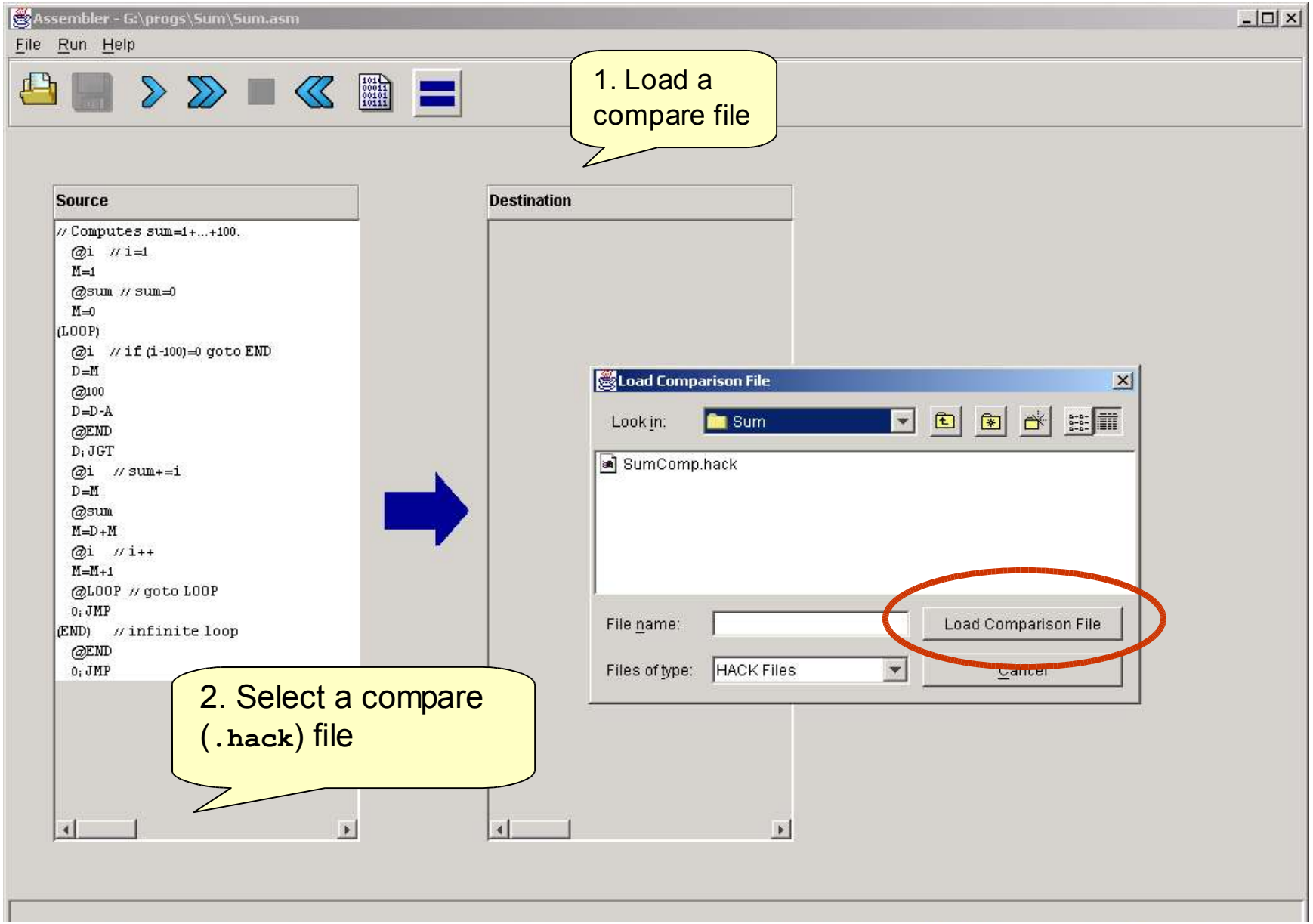
Inspecting the translation



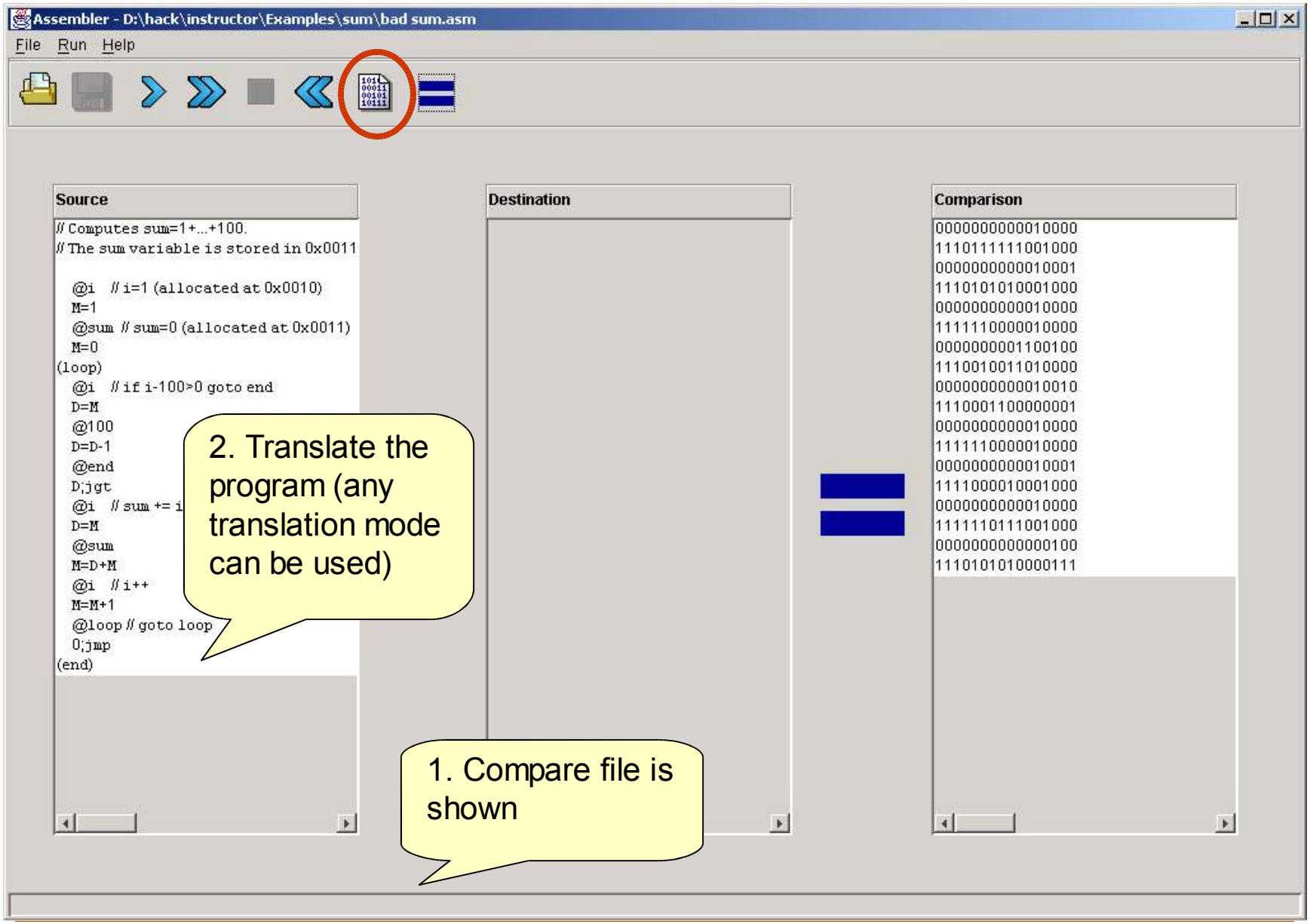
Saving the translated code



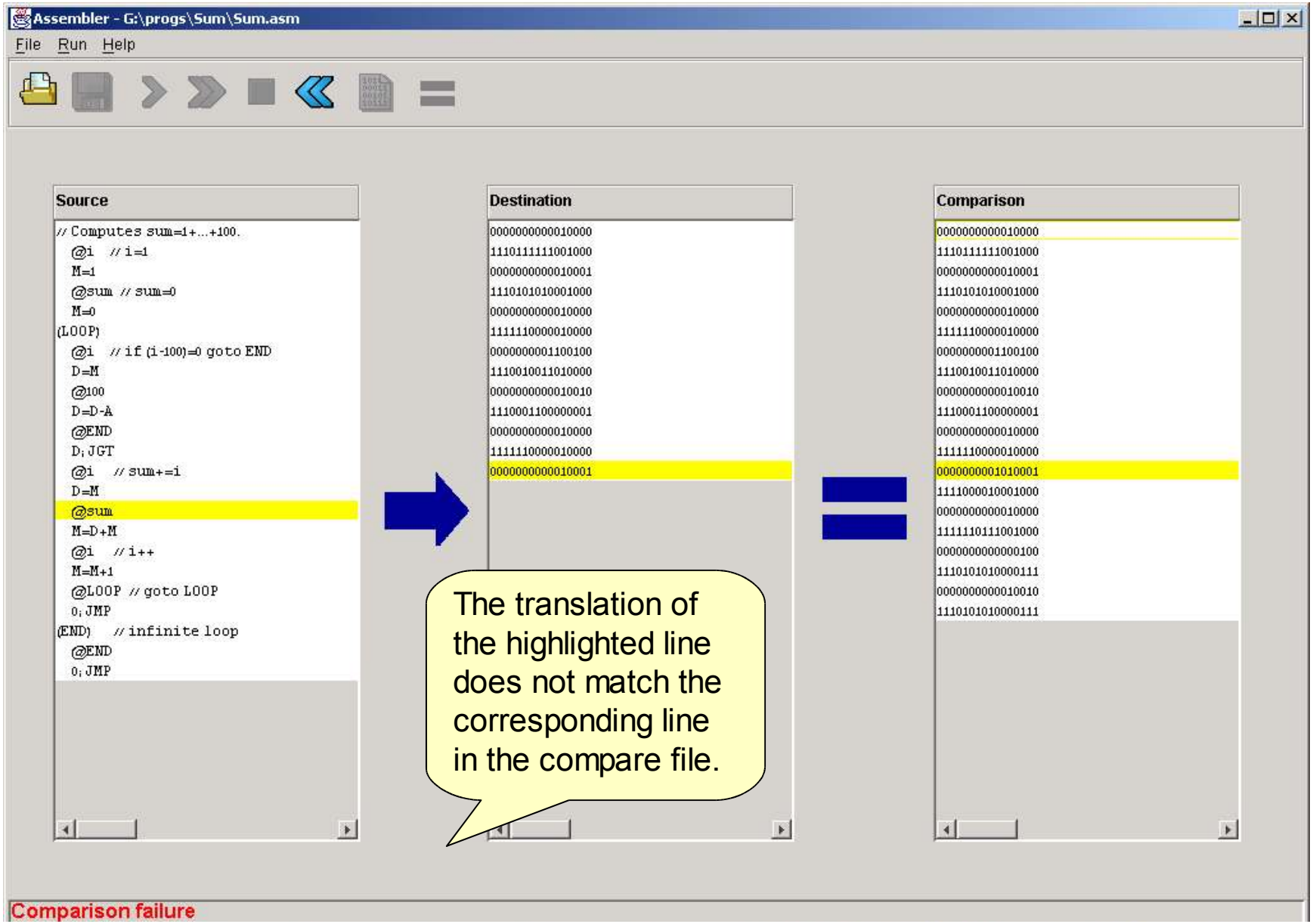
Comparing the translated code to a compare-file



Comparing the translated code to a compare-file



Comparing the translated code to a compare-file



Postscript: R. Feynman on why symbols don't matter compared to their meaning

On weekends, my father would take me for walks in the woods and he'd tell me about interesting things that were going on. "See that bird?" he says. "It's a Spencer Warbler." (I knew he didn't know the real name.) "Well, in Italian, it's Chutto Lapittida. In Portuguese, it's a Bom da Peida. In Chinese, it's a Chung-long-tah, and in Japanese, it's Katano Tekeda. You can know the name of that bird in all the languages of the world, but when you're finished, you'll know absolutely nothing whatever about the bird. You'll only know something about people in different places, and what they call the bird. So let's look at the bird and see what it is doing – that's what counts." This is how I learned very early the difference between knowing the name of something and knowing something.

Richard P. Feynman, The Making of a Scientist, 1988.