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

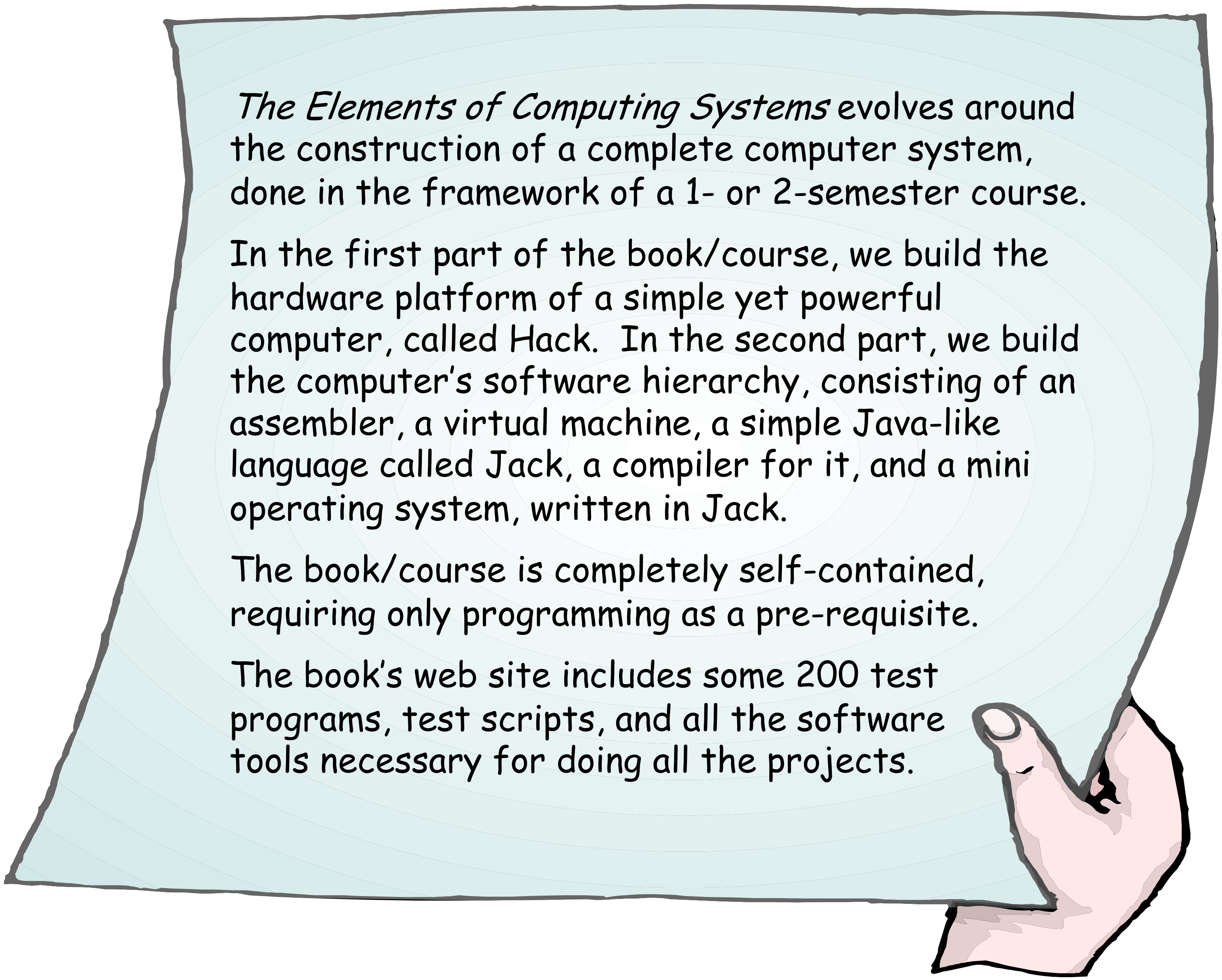
MIT Press

www.nand2tetris.org

Developed by students at the
Efi Arazi School of Computer Science at Reichman University

Chief Software Architect: Yaron Ukrainitz

Background



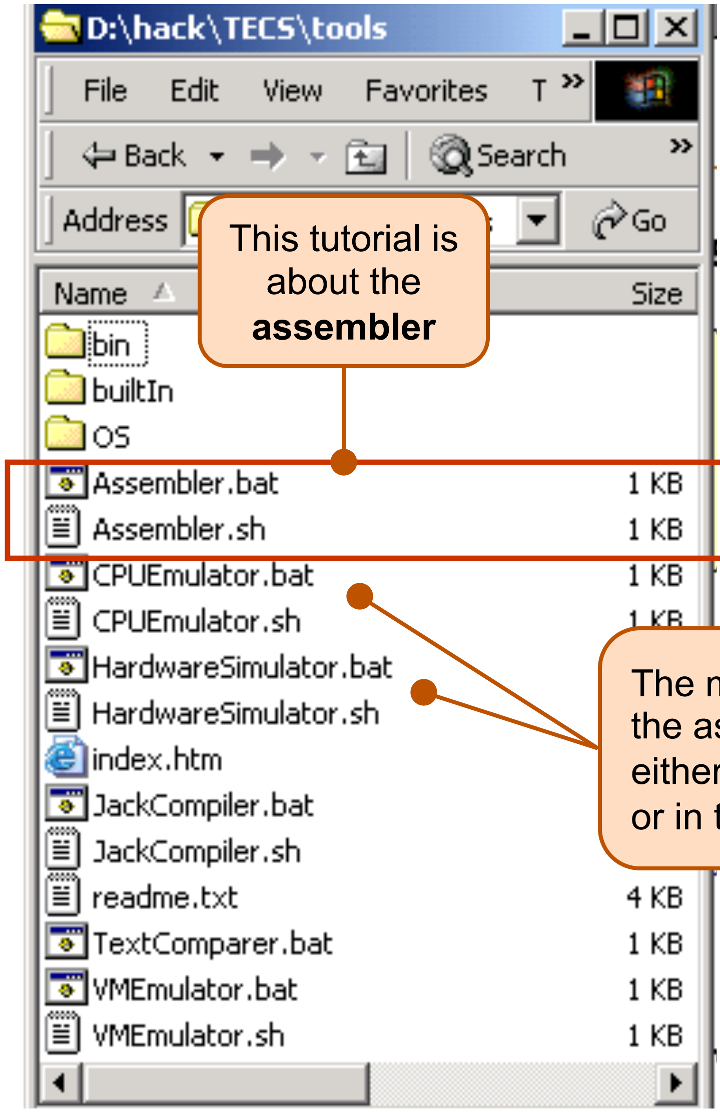
The Elements of Computing Systems evolves around the construction of a complete computer system, done in the framework of a 1- or 2-semester course.

In the first part of the book/course, we build the hardware platform of a simple yet powerful computer, called Hack. In the second part, 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 mini operating system, written in Jack.

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

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

The book’s software suite



```
Name                      Size
bin
builtIn
OS
Assembler.bat             1 KB
Assembler.sh              1 KB
CPUEmulator.bat           1 KB
CPUEmulator.sh            1 KB
HardwareSimulator.bat
HardwareSimulator.sh
index.htm
JackCompiler.bat
JackCompiler.sh
readme.txt                4 KB
TextComparer.bat          1 KB
VMEmulator.bat            1 KB
VMEmulator.sh             1 KB
```

This tutorial is about the **assembler**

(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.

The machine code generated by the assembler can be tested either in the hardware simulator or in the CPU emulator.

id 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

Assembler Tutorial

Part I:

Assembly

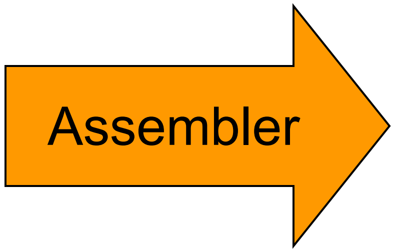
Programming

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
```

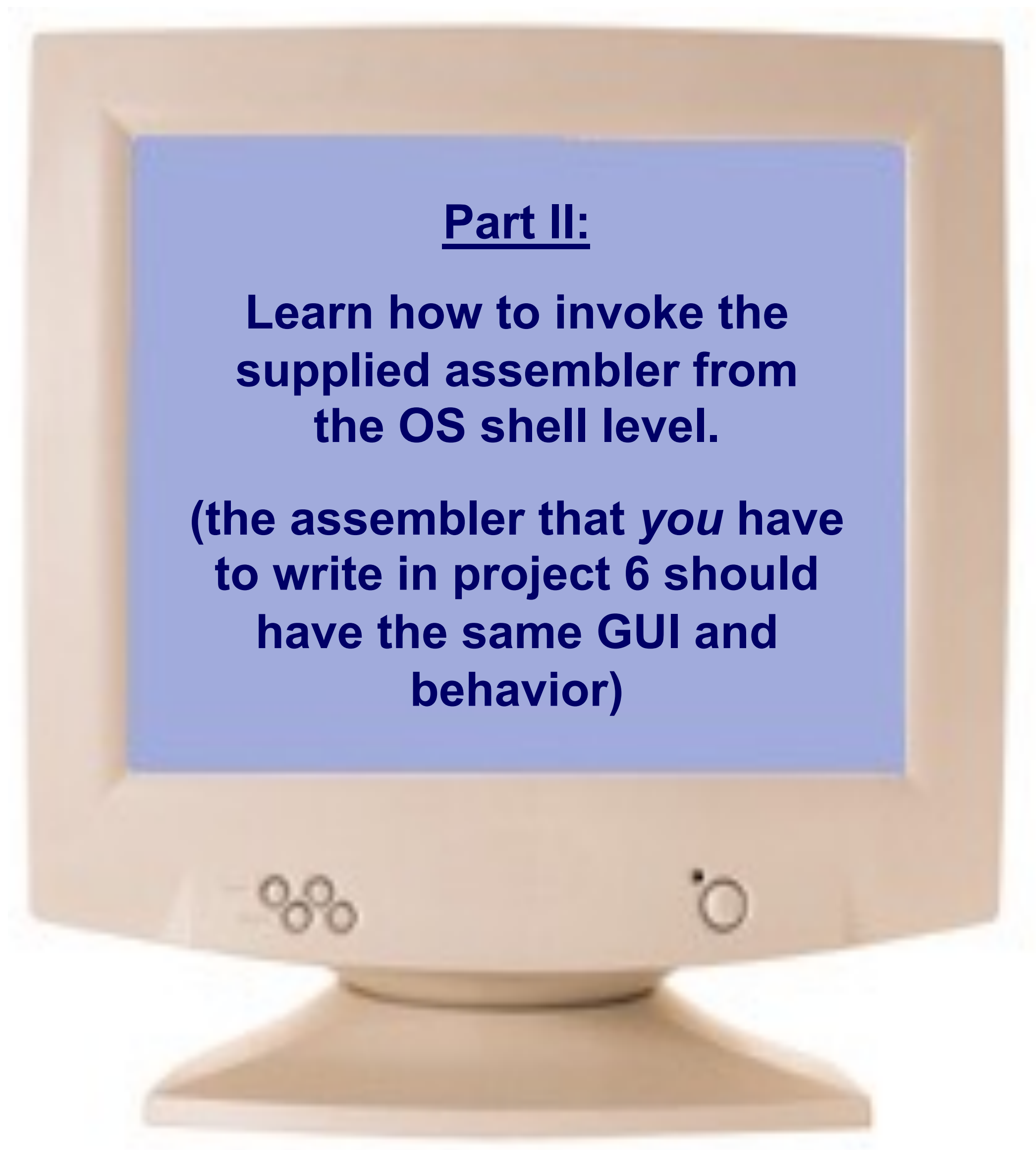
The assembly program:

- 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 label declarations like `(LOOP)` and `(END)` as pseudo commands that generate no code.

Assembler Tutorial

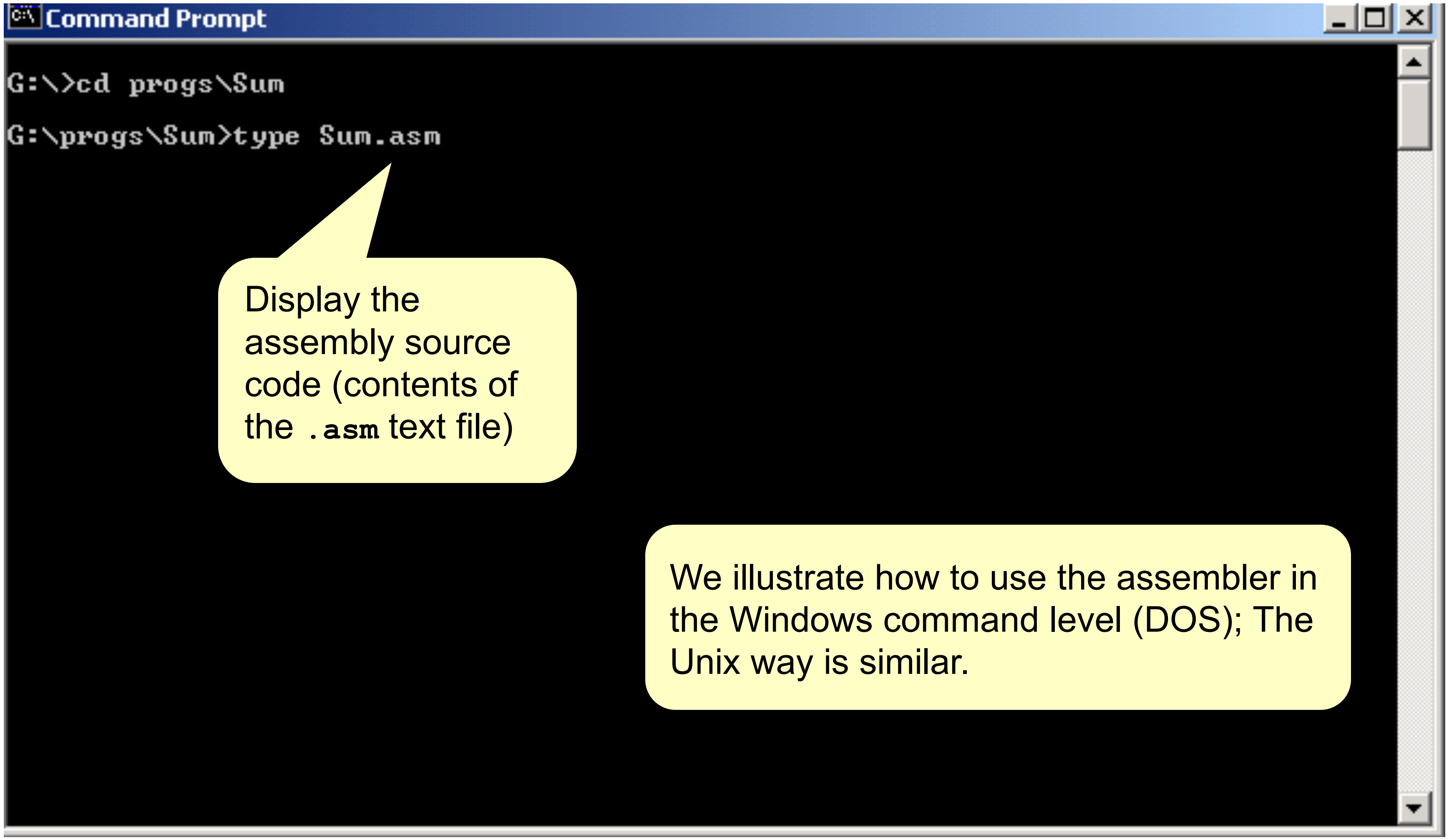


Part II:

Learn how to invoke the supplied assembler from the OS shell level.

(the assembler that *you* have to write in project 6 should have the same GUI and behavior)

The command-level assembler

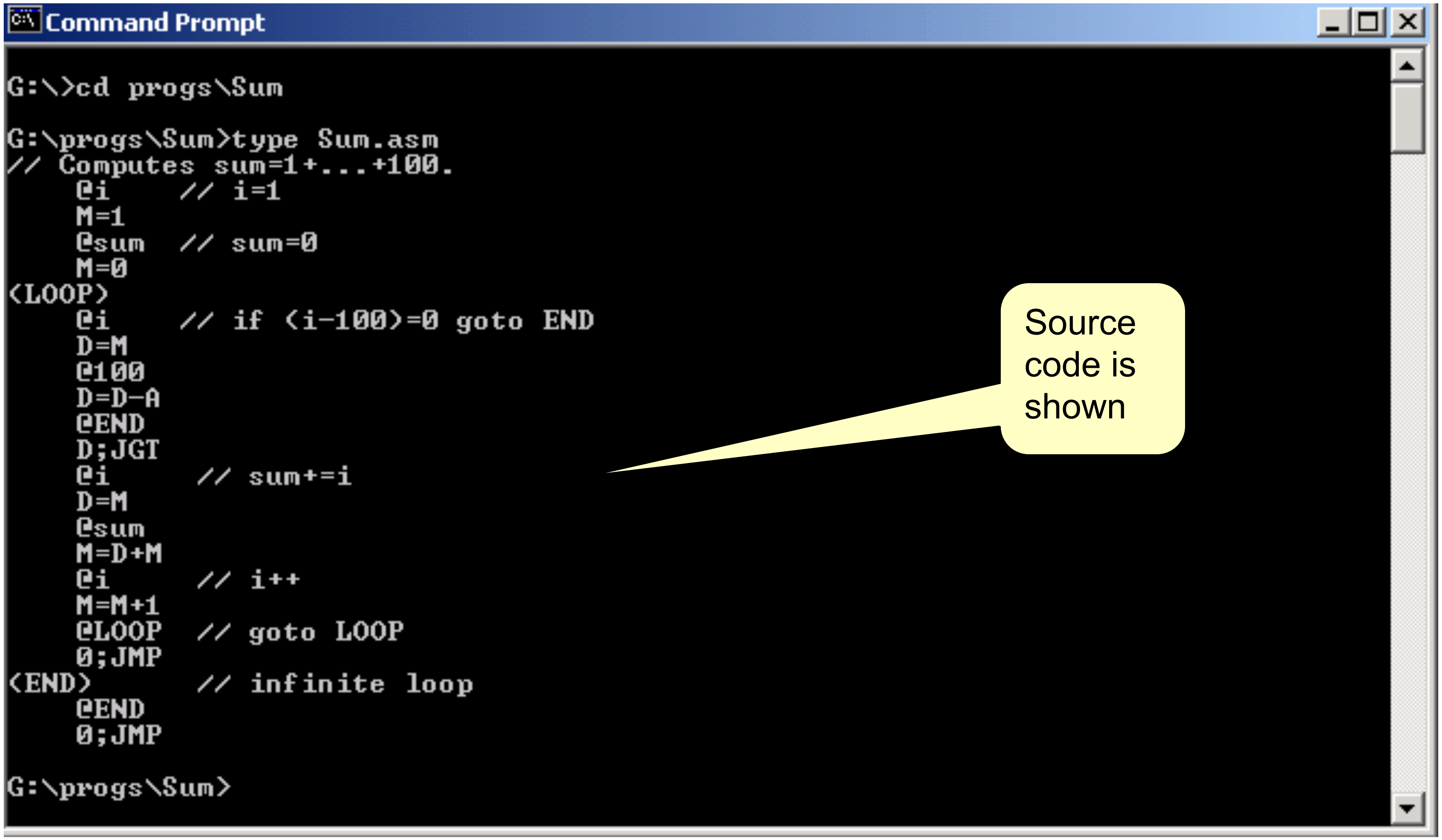


```
G:\>cd progs\Sum

G:\progs\Sum>type Sum.asm
```

Display the assembly source code (contents of the `.asm` text file)

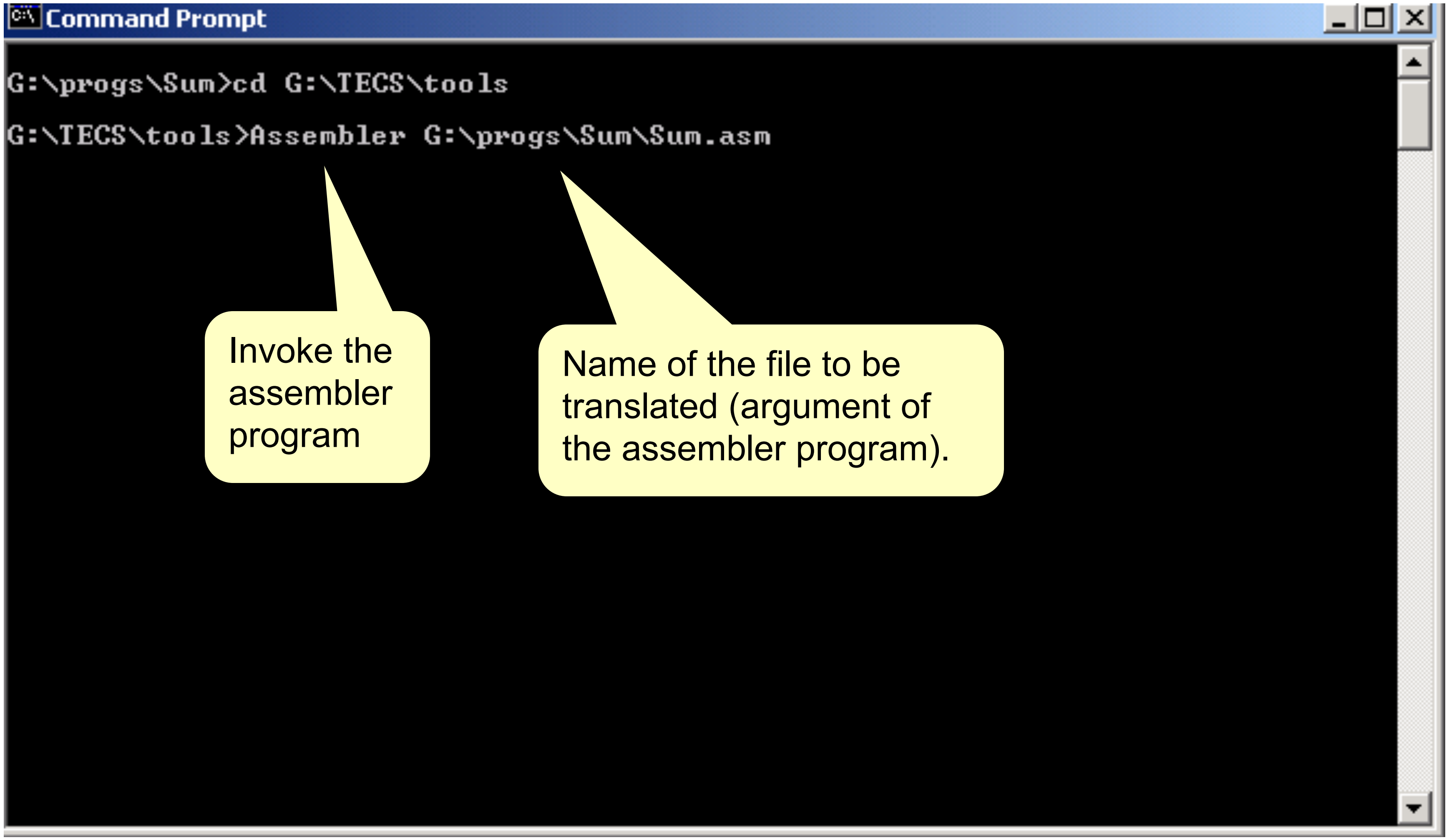
We illustrate how to use the assembler in the Windows command level (DOS); The Unix way is similar.

Inspecting the source file



```
G:\>cd progs\Sum

G:\progs\Sum>type 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

G:\progs\Sum>
```

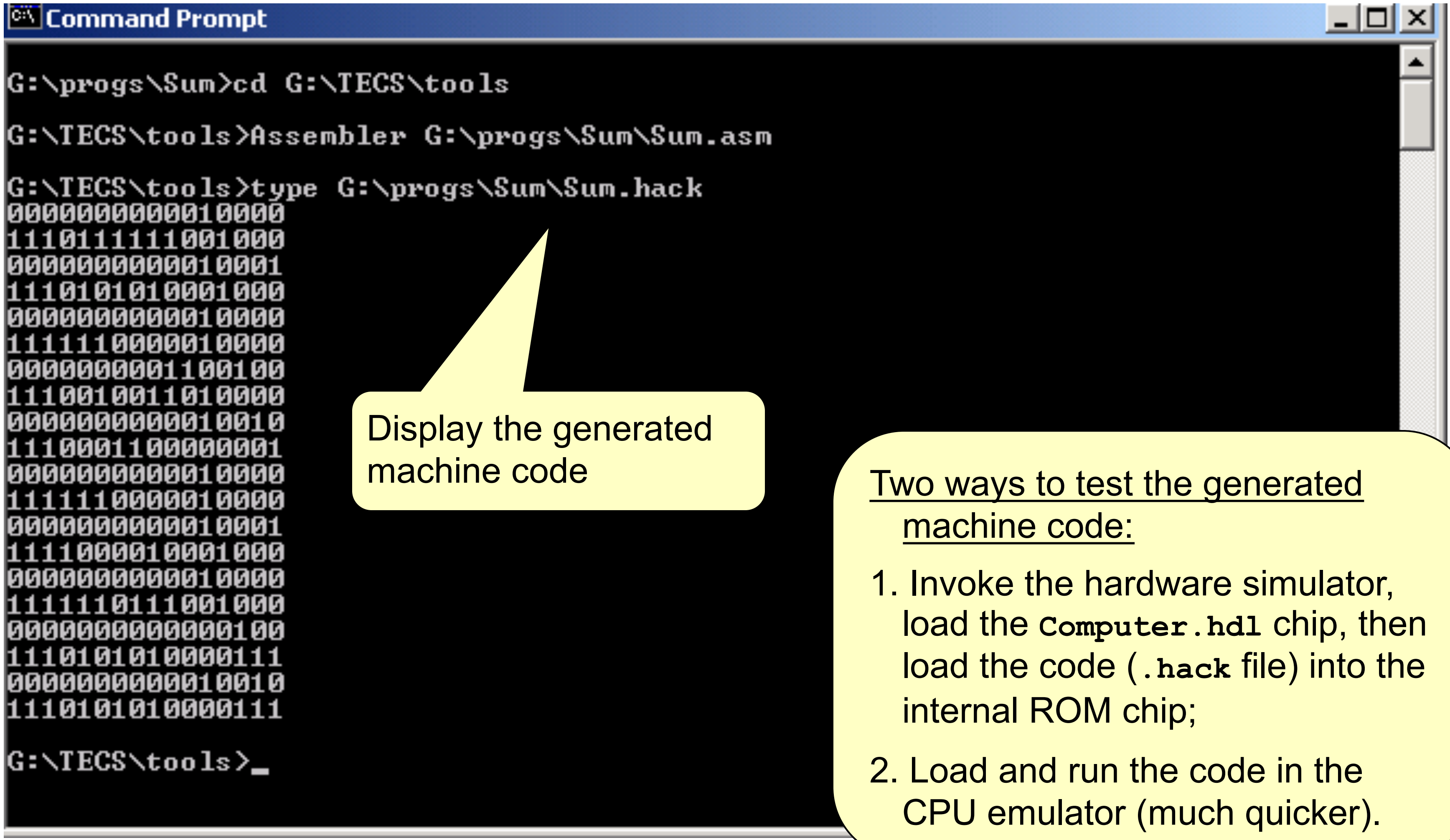
Invoking the Assembler



```
G:\progs\Sum>cd G:\TECS\tools

G:\TECS\tools>Assembler G:\progs\Sum\Sum.asm
```

Invoke the assembler program
Name of the file to be translated (argument of the assembler program).

Invoking the Assembler



```
G:\progs\Sum>cd G:\TECS\tools

G:\TECS\tools>Assembler G:\progs\Sum\Sum.asm

G:\TECS\tools>type G:\progs\Sum\Sum.hack
0000000000010000
1110111111001000
0000000000010001
1110101010001000
0000000000010000
1111110000010000
0000000001100100
1110010011010000
0000000000010010
1110001100000001
0000000000010000
1111110000010000
0000000000010001
1111000010001000
0000000000010000
1111110111001000
0000000000000100
1110101010000111
0000000000010010
1110101010000111

G:\TECS\tools>_
```

Display the generated machine code

Two ways to test the generated machine code:

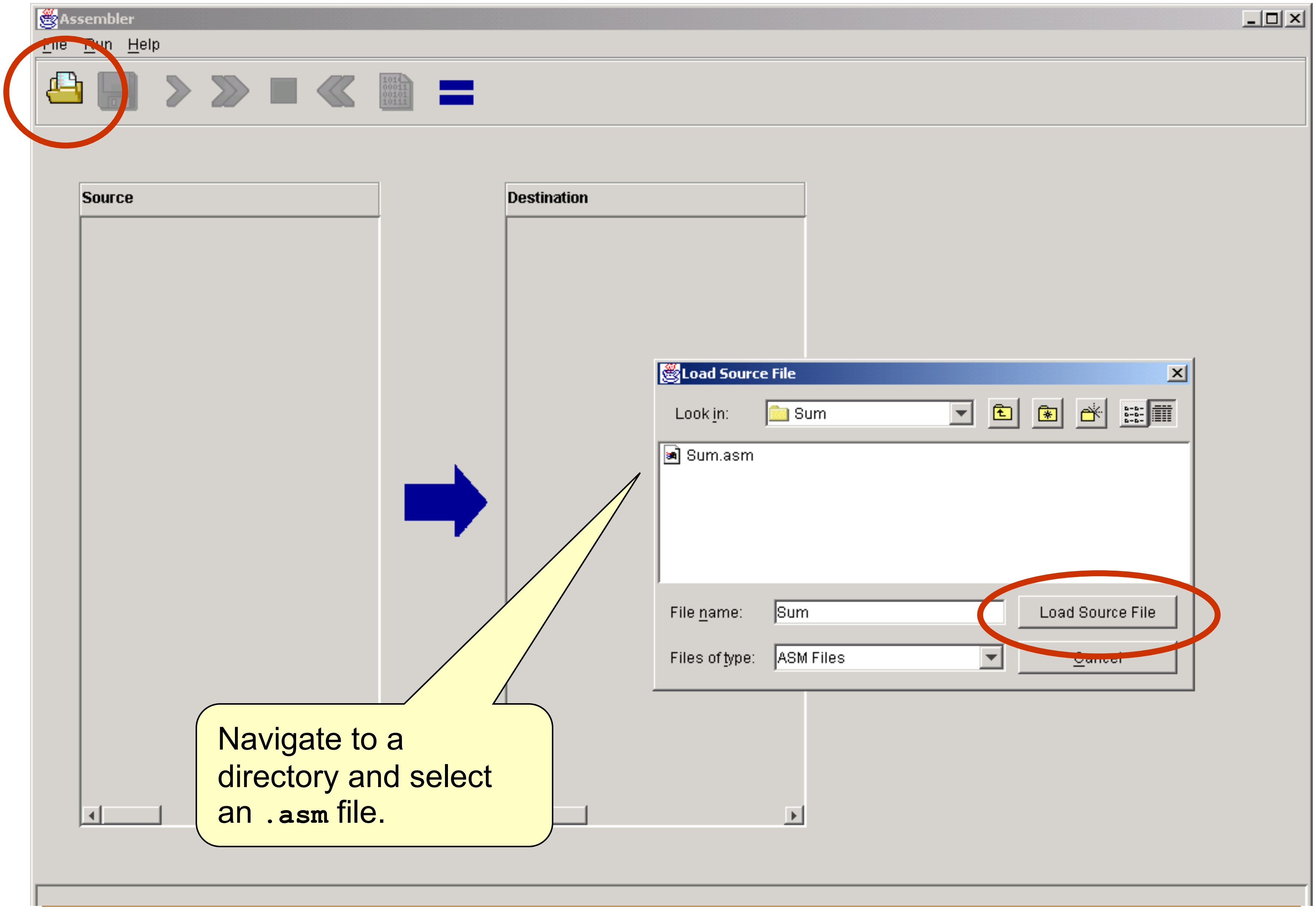
1. Invoke the hardware simulator, load the `Computer.hdl` chip, then load the code (`.hack` file) into the internal ROM chip;
2. Load and run the code in the CPU emulator (much quicker).

Hardware Simulation Tutorial

Part III:

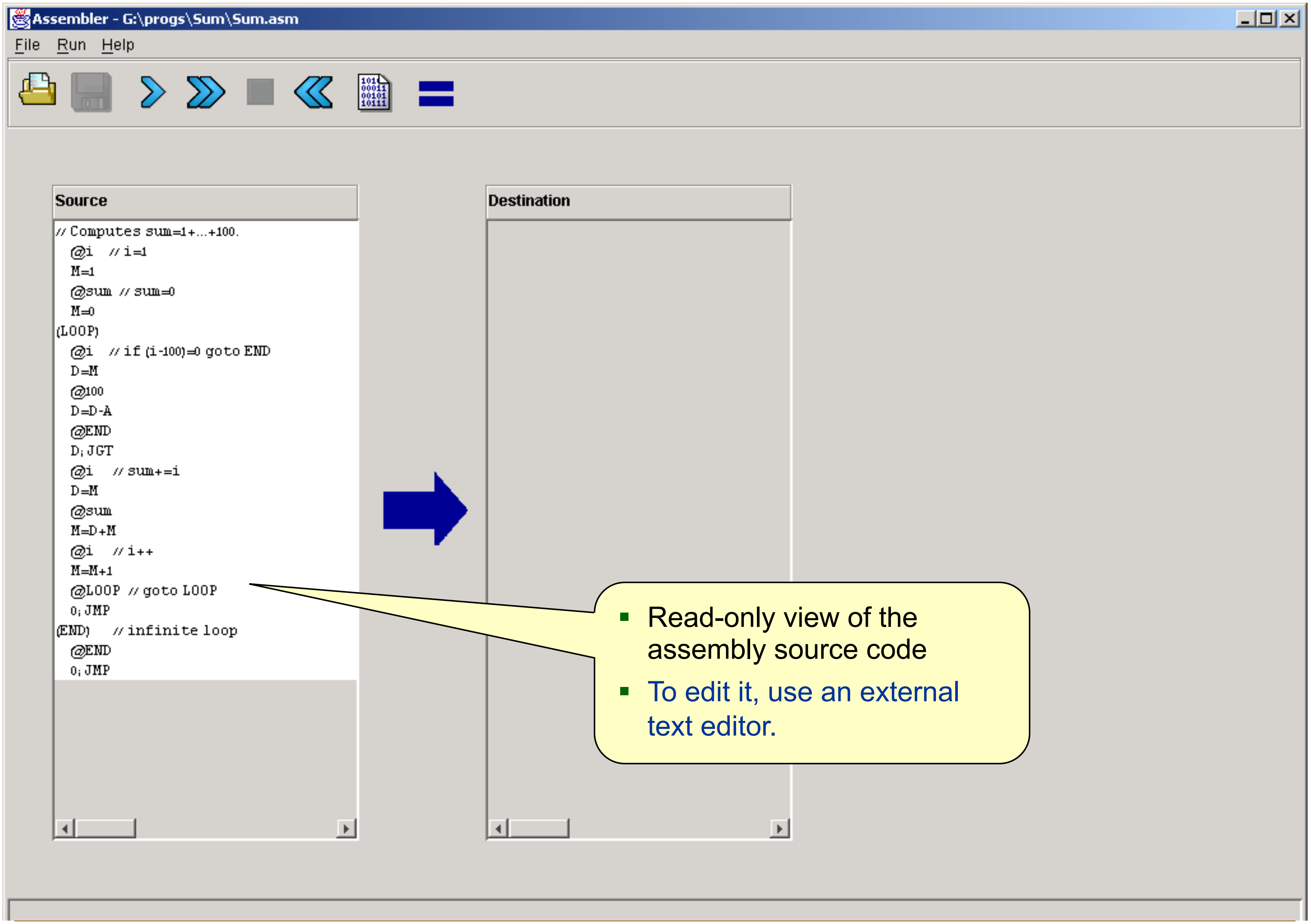
Learn how to use the interactive Assembler

Loading an assembly program



Navigate to a directory and select an **`.asm`** file.

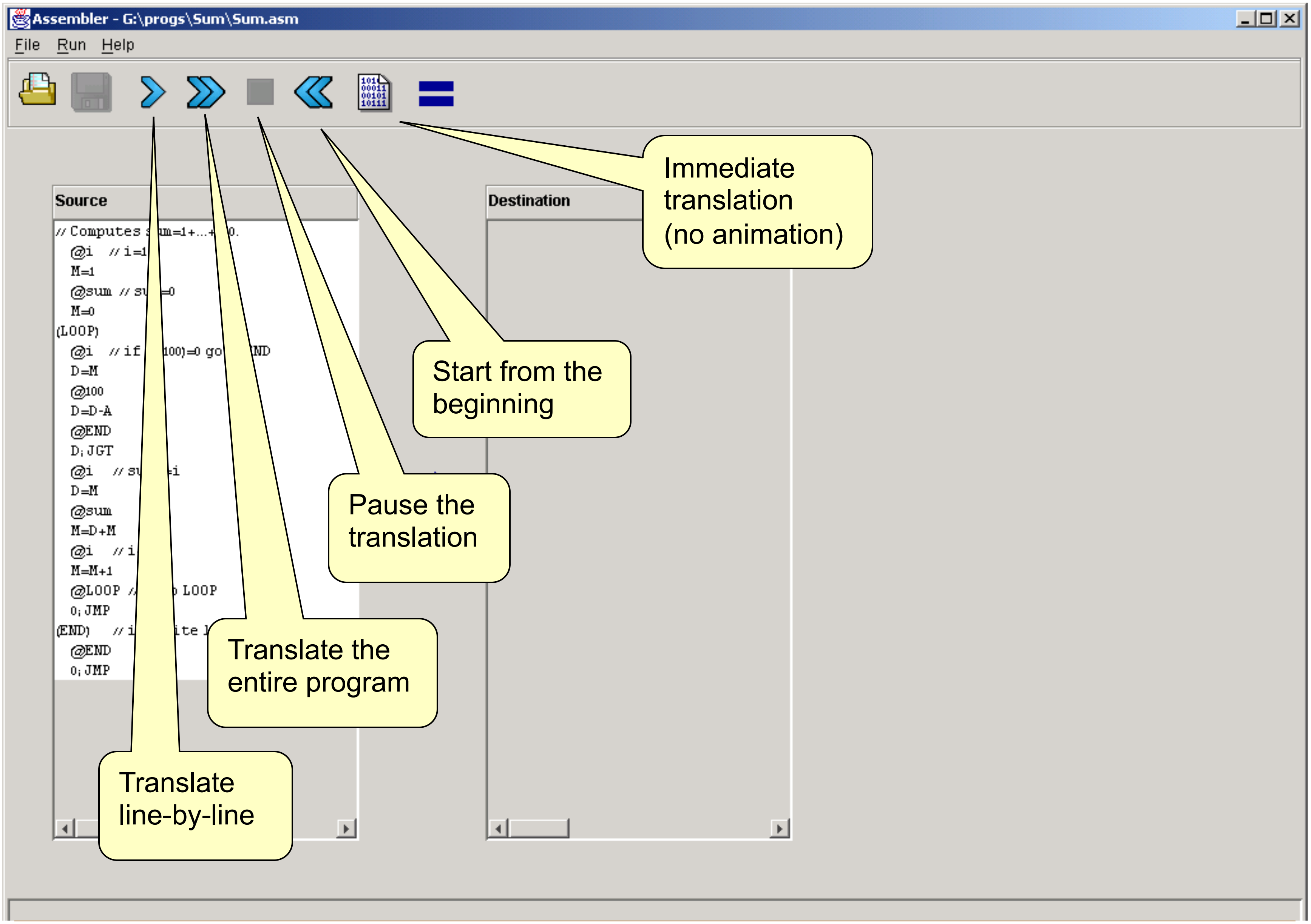
Loading an assembly program



```
// 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
```

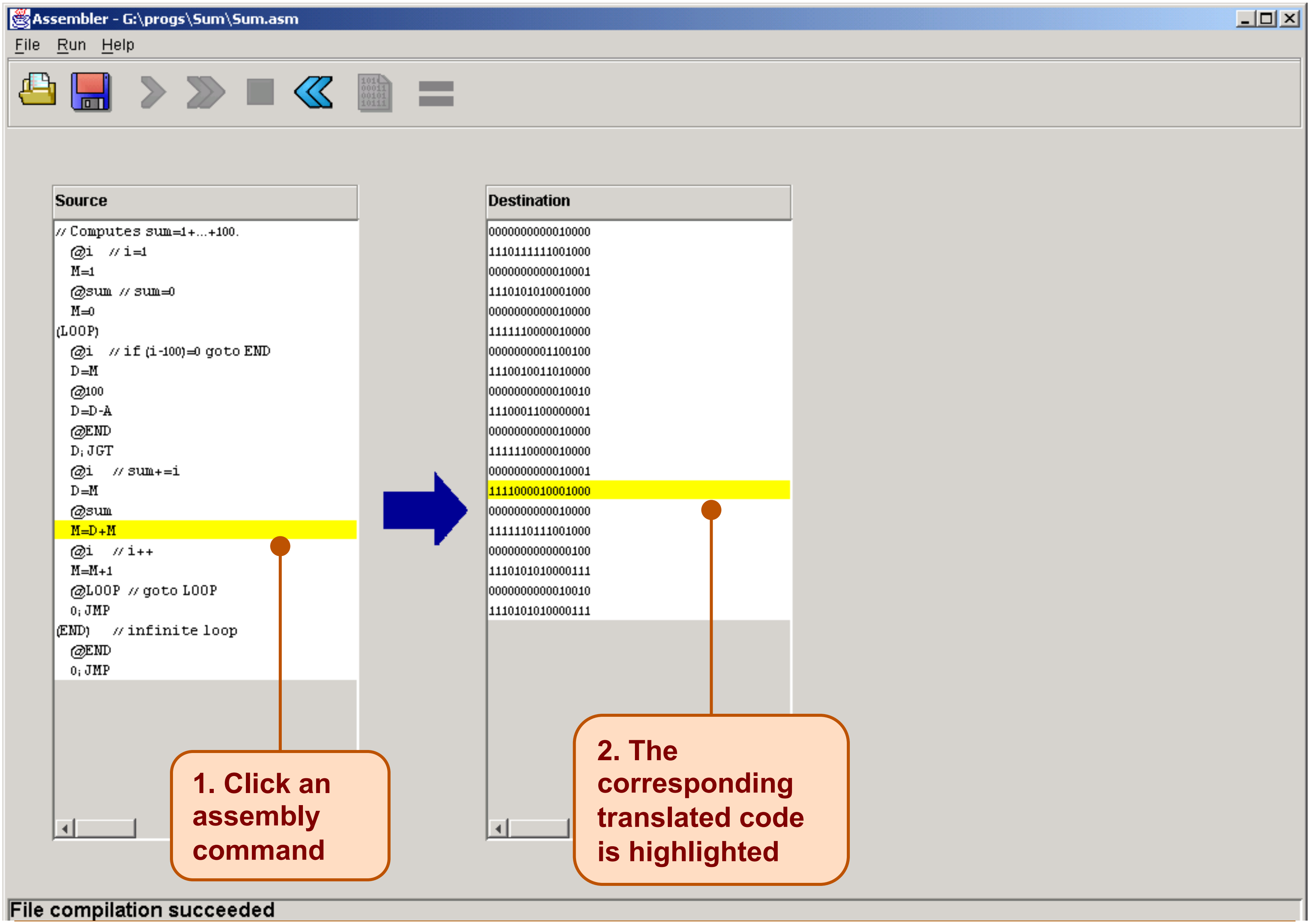
- Read-only view of the assembly source code
- To edit it, use an external text editor.

Translating a program



```
// Computes s um=1+...+ 0.
  @i  // i=1
  M=1
  @sum // su =0
  M=0
(LOOP)
  @i  // if  100)=0 go  ND
  D=M
  @100
  D=D-A
  @END
  D;JGT
  @i   // su  =i
  D=M
  @sum
  M=D+M
  @i   // i
  M=M+1
  @LOOP //    LOOP
  0;JMP
(END)   // i   ite l
  @END
  0;JMP
```

Inspecting the translation



Source

```
// 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
```

Destination

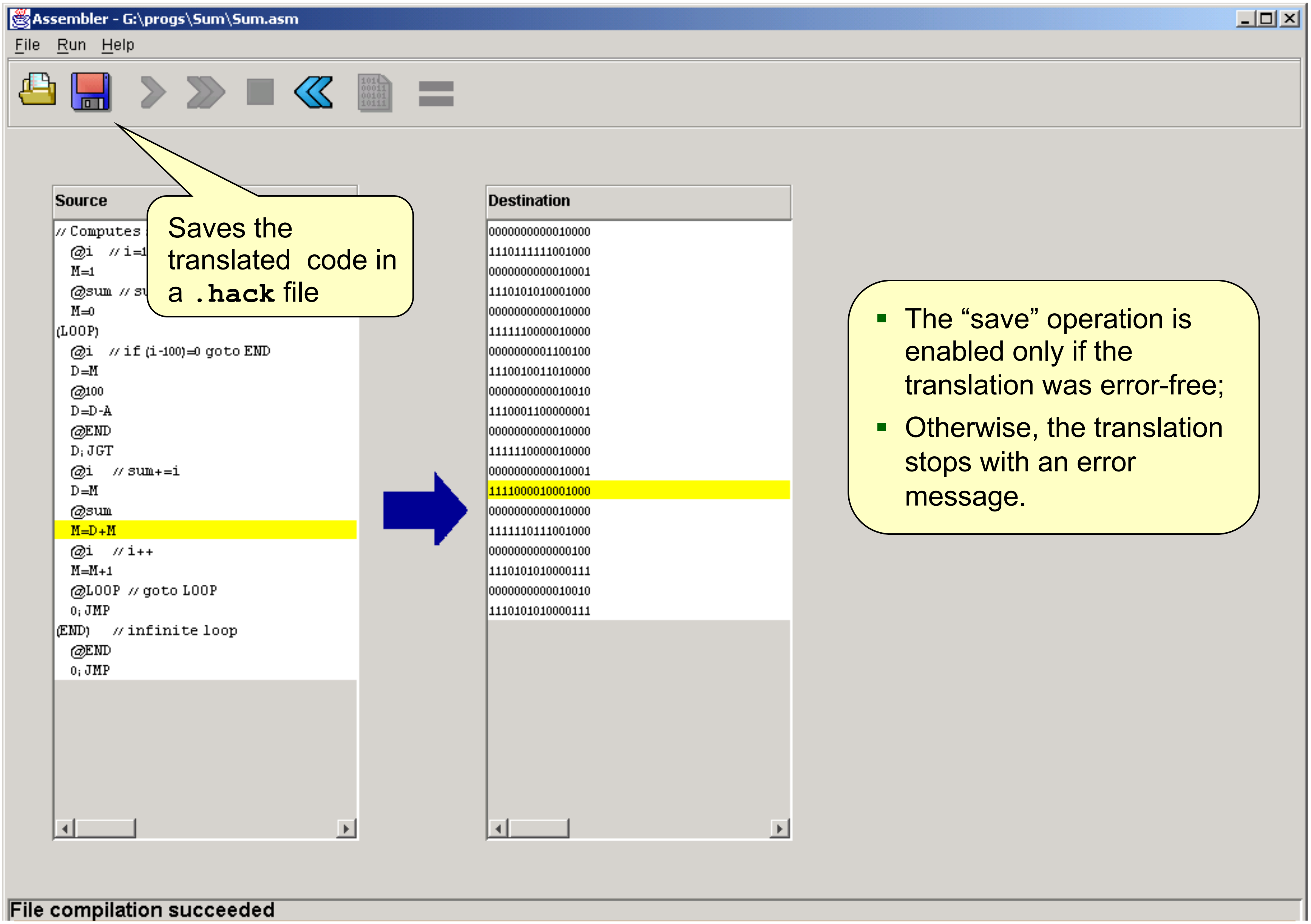
```
0000000000010000
1110111111001000
0000000000010001
1110101010001000
0000000000010000
1111110000010000
0000000001100100
1110010011010000
0000000000010010
1110001100000001
0000000000010000
1111110000010000
0000000000010001
1111000010001000
0000000000010000
1111110111001000
0000000000000100
1110101010000111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

1. Click an assembly command

2. The corresponding translated code is highlighted

File compilation succeeded

Saving the translated code



```
// Computes
  @i   // i=1
  M=1
  @sum // su
  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
```

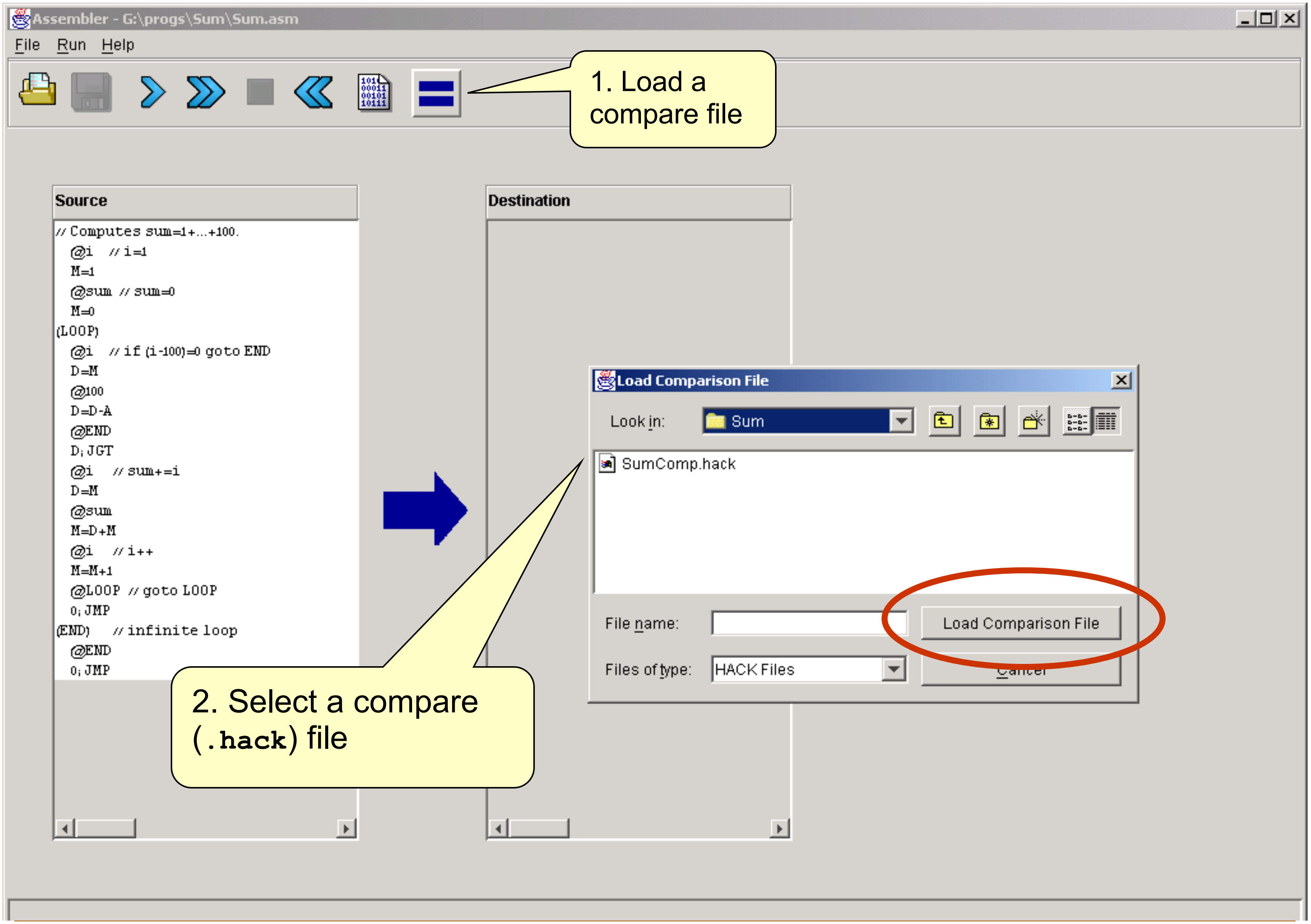
```
0000000000010000
1110111111001000
0000000000010001
1110101010001000
0000000000010000
1111110000010000
0000000001100100
1110010011010000
0000000000010010
1110001100000001
0000000000010000
1111110000010000
0000000000010001
1111000010001000
0000000000010000
1111110111001000
0000000000000100
1110101010000111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

File compilation succeeded

Saves the translated code in a **`.hack`** file

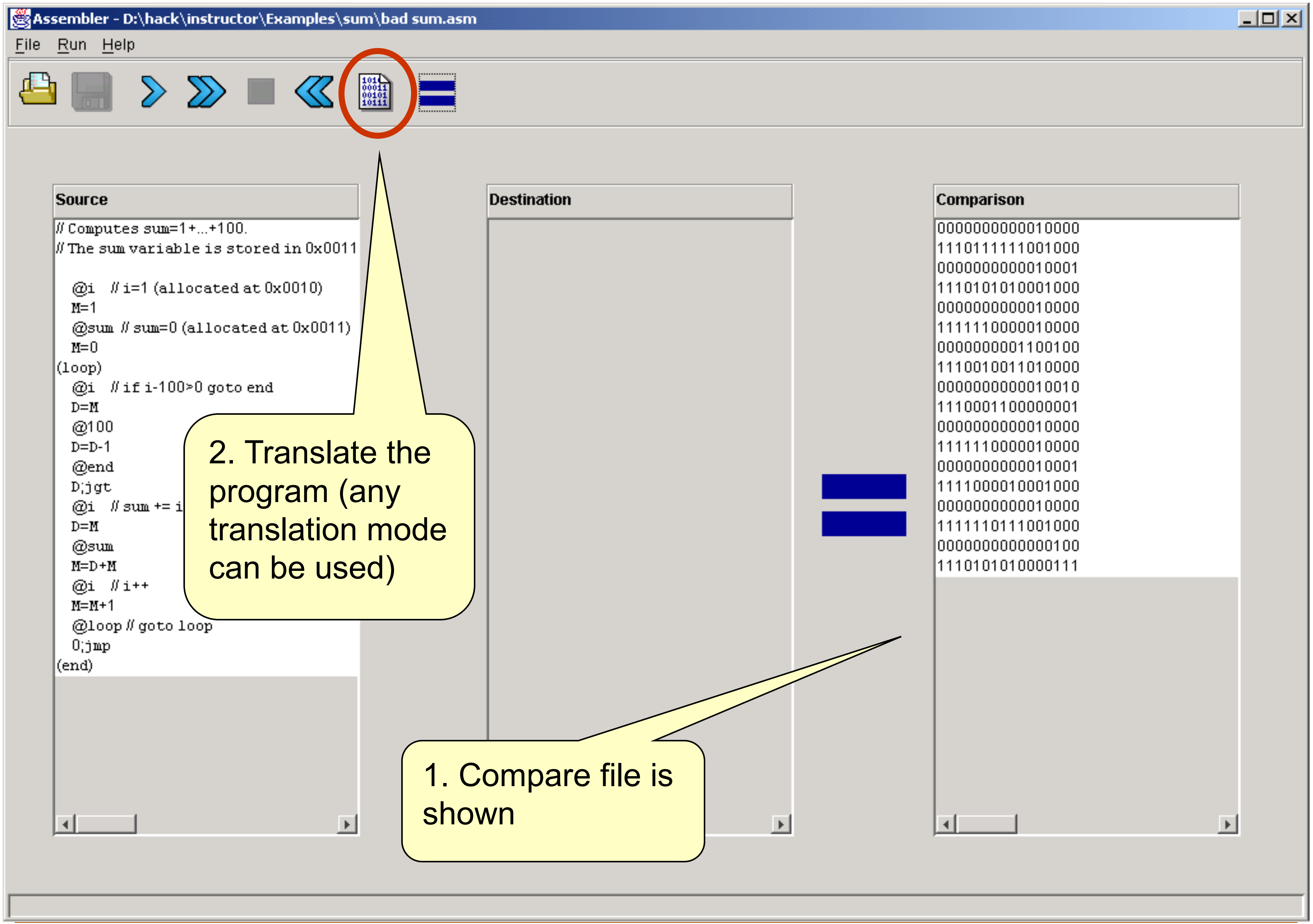
- The "save" operation is enabled only if the translation was error-free;
- Otherwise, the translation stops with an error message.

Using Compare Files



```
// 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
```

Using Compare Files



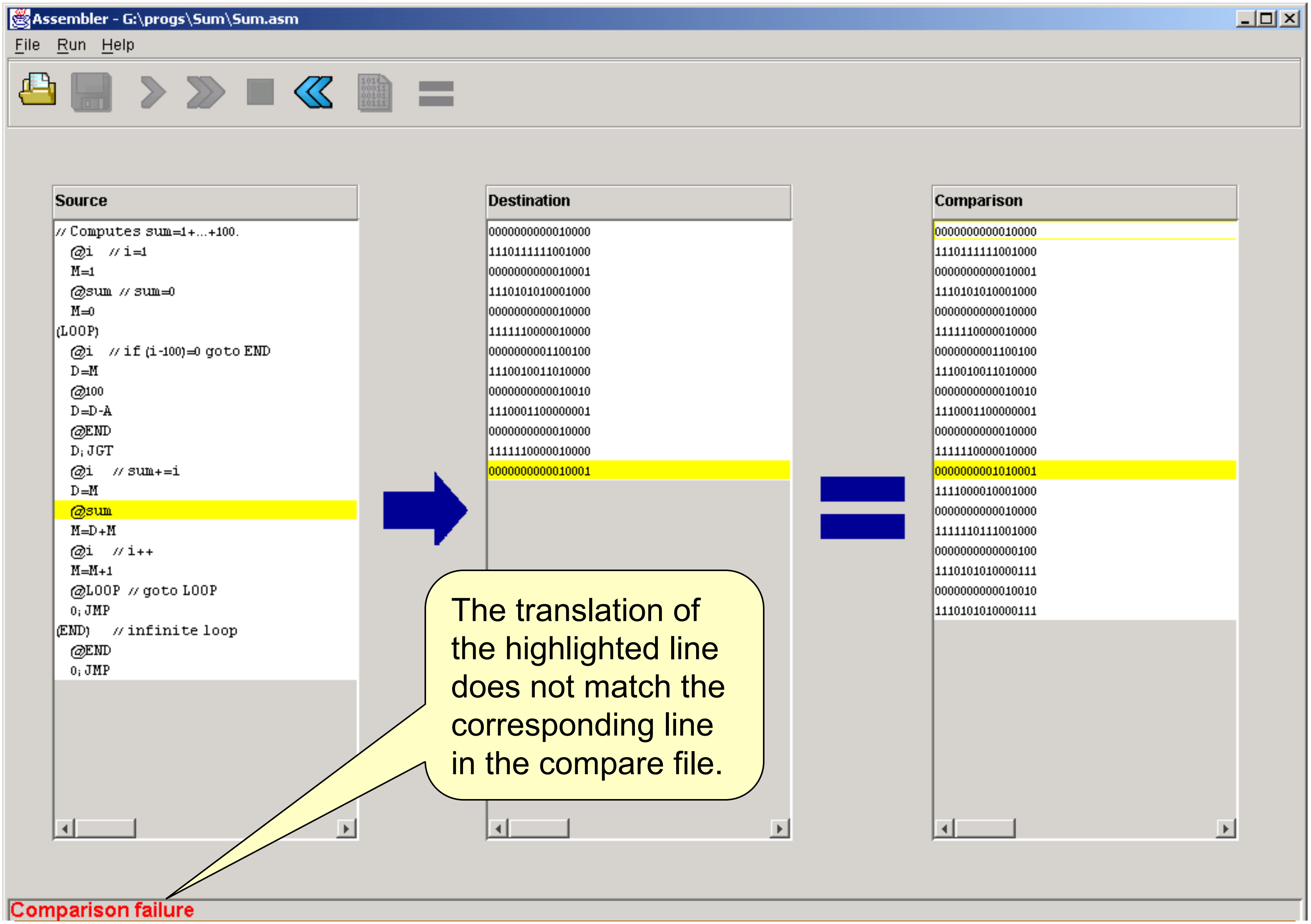
```
// Computes sum=1+...+100.
// The sum variable is stored in 0x0011

   @i   // i=1 (allocated at 0x0010)
   M=1
   @sum // sum=0 (allocated at 0x0011)
   M=0
(loop)
   @i   // if i-100>0 goto end
   D=M
   @100
   D=D-1
   @end
   D;jgt
   @i   // sum += i
   D=M
   @sum
   M=D+M
   @i   // i++
   M=M+1
   @loop // goto loop
   0;jmp
(end)
```

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

2. Translate the program (any translation mode can be used)

1. Compare file is shown

Using Compare Files



Source

```
// 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
```

Destination

```
0000000000010000
1110111111001000
0000000000010001
1110101010001000
0000000000010000
1111110000010000
0000000001100100
1110010011010000
0000000000010010
1110001100000001
0000000000010000
1111110000010000
0000000000010001
```

Comparison

```
0000000000010000
1110111111001000
0000000000010001
1110101010001000
0000000000010000
1111110000010000
0000000001100100
1110010011010000
0000000000010010
1110001100000001
0000000000010000
1111110000010000
0000000001010001
1111000010001000
0000000000010000
1111110111001000
0000000000000100
1110101010000111
0000000000010010
1110101010000111
```

Comparison failure

The translation of the highlighted line does not match the corresponding line in the compare file.