



# Assembler Tutorial

This program is part of the software suite  
that accompanies

## ***The Elements of Computing Systems***

by Noam Nisan and Shimon Schocken

MIT Press

[www.nand2tetris.org](http://www.nand2tetris.org)

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

Chief Software Architect: Yaron Ukrainitz

# Background

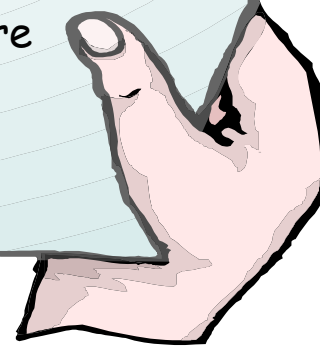
---

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

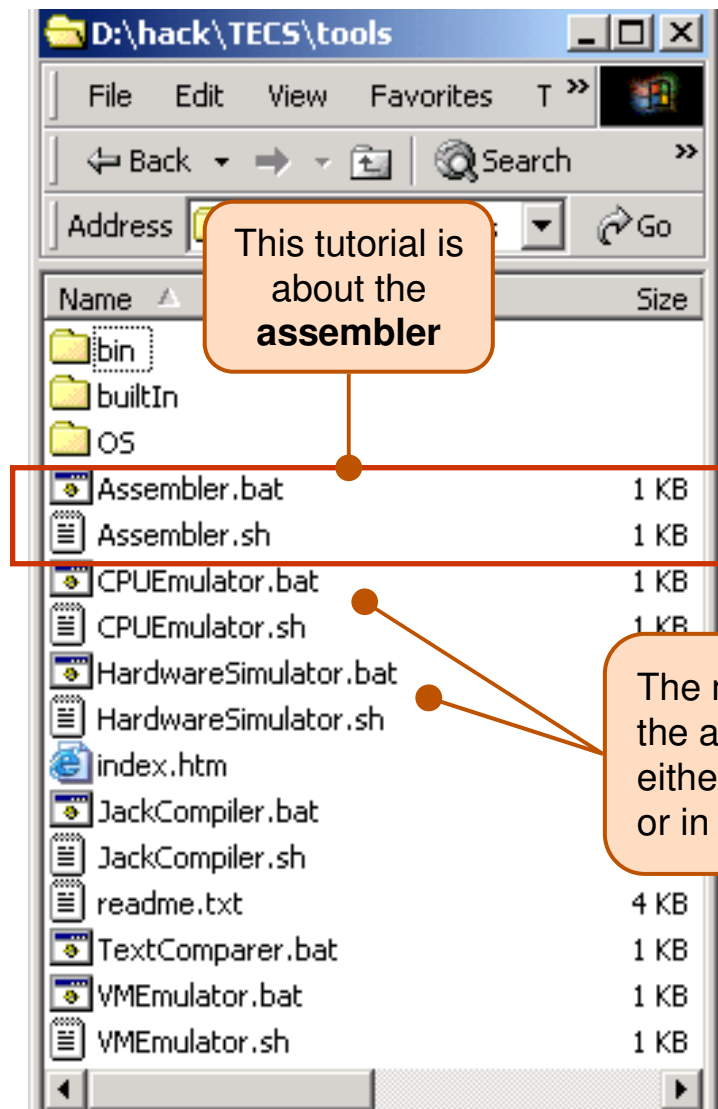
In the first part of the book/course, we build the hardware platform of a simple yet powerful computer, called Hack. In the second part, we build the computer's software hierarchy, consisting of an assembler, a virtual machine, a simple Java-like language called Jack, a compiler for it, and a mini operating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



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

## Simulators

(**HardwareSimulator**, **CPUEmulator**, **VMEulator**):

- 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 solutions supplied by us.

and 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*

# Assembler Tutorial

---



# Example

## Sum.asm

```
// Computes sum=1+...+100.
    @i      // i=1
    M=1
    @sum    // sum=0
    M=0
(LLOOP)
    @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
    @LLOOP  // goto LOOP
    0;JMP
(END)      // infinite loop
    @END
    0;JMP
```



Assembler

## 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
(LLOOP)
    @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
    @LLOOP  // goto LOOP
    0;JMP
(END)      // infinite loop
    @END
    0;JMP
```

### The assembly program:

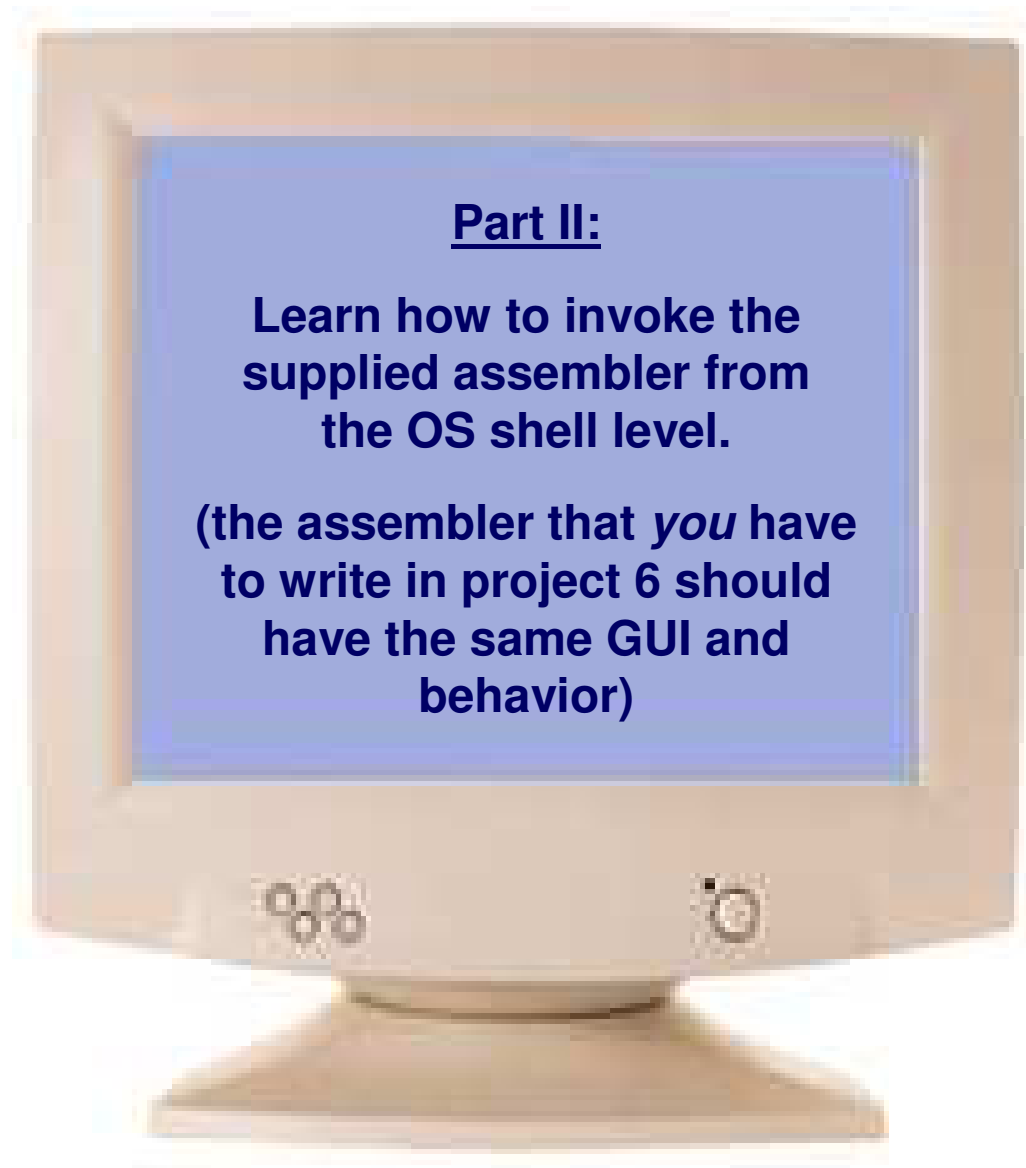
- Stored in a text file named **Prog.asm**
- Written and edited in a text editor

### 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

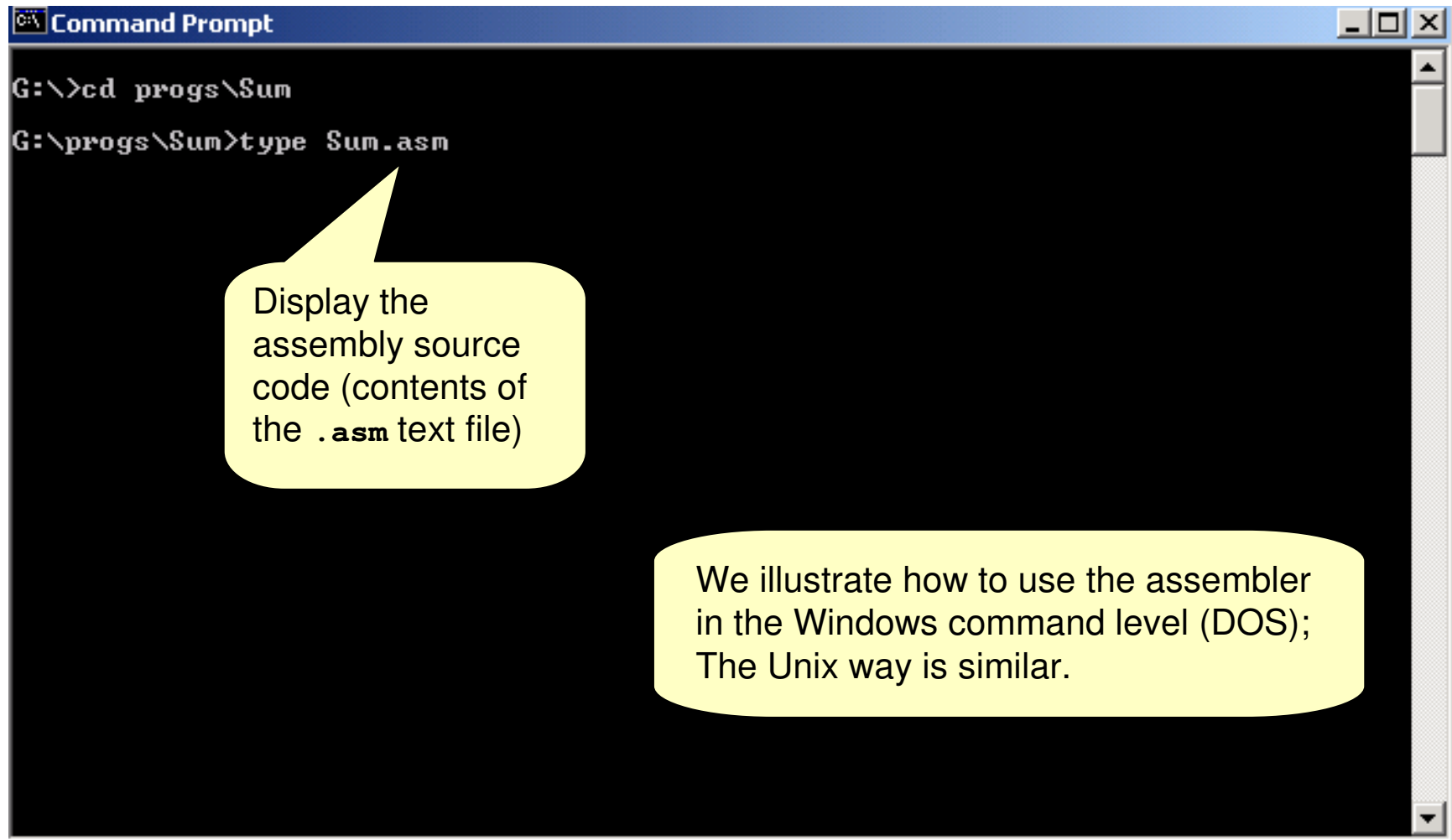
---





# The command-level assembler

---

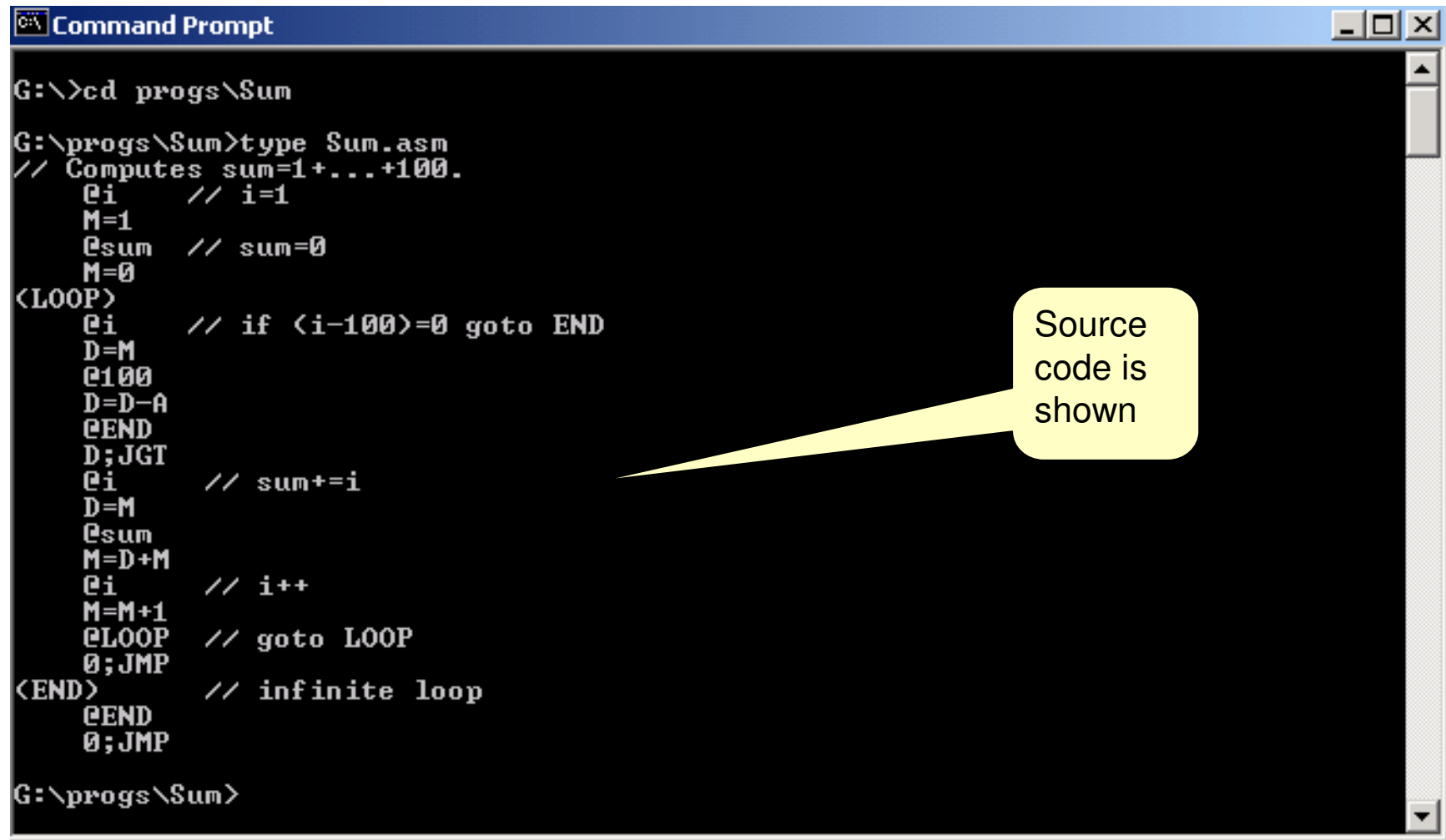


```
G:\>cd progs\Sum
G:\progs\Sum>type Sum.asm
```

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

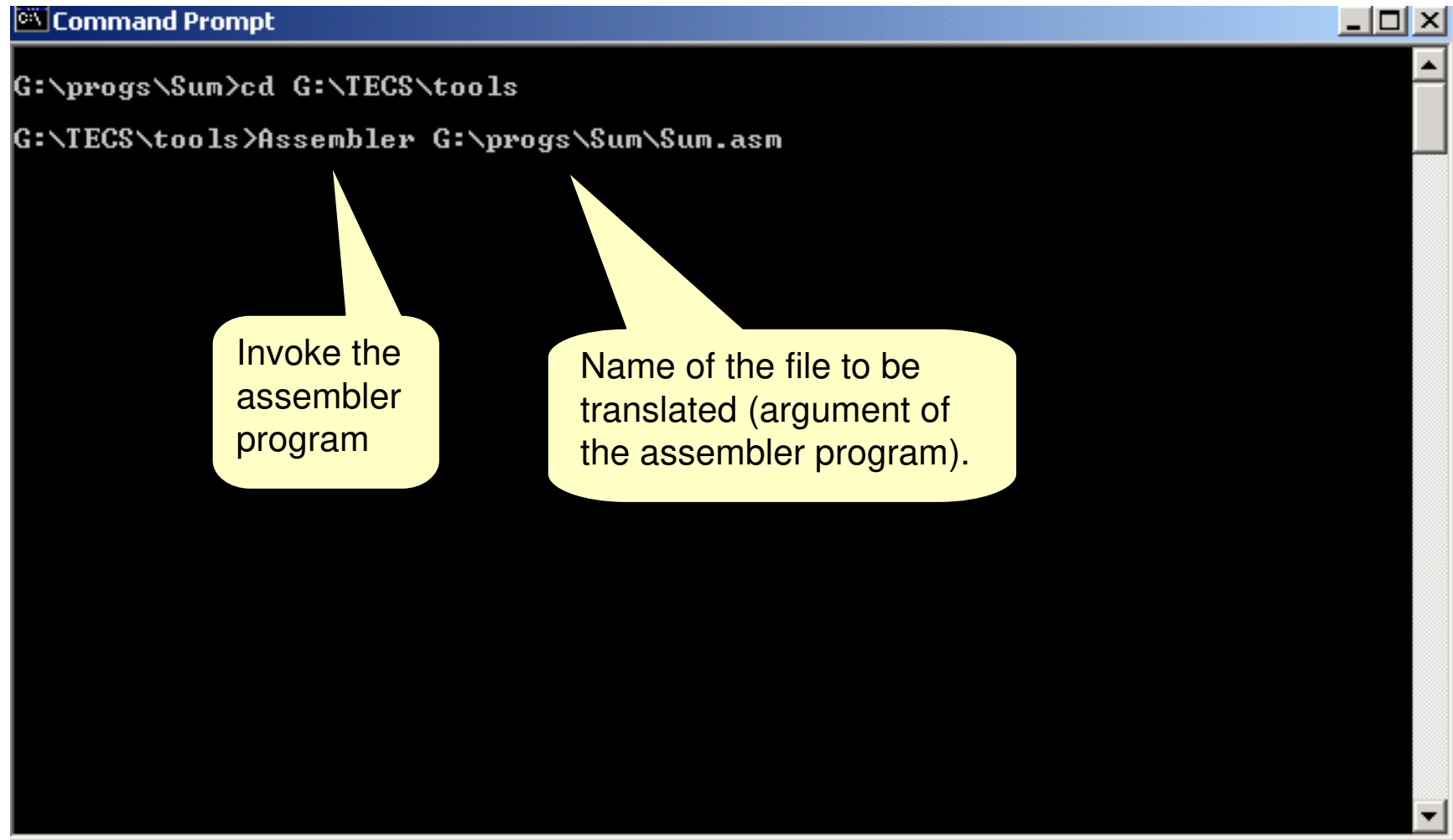
G:\progs\Sum>type Sum.asm
// Computes sum=1+...+100.
    @i      // i=1
    M=1
    @sum    // sum=0
    M=0
<LOOP>
    @i      // if <i-100>=0 goto END
    D=M
    @100
    D=D-A
    @END
    D;JGT
    @i      // sum+=i
    D=M
    @sum
    M=D+M
    @i      // i++
    M=M+1
    @LOOP   // goto LOOP
    @;JMP
<END>      // infinite loop
    @END
    @;JMP

G:\progs\Sum>
```

Source code is shown

# Invoking the Assembler

---

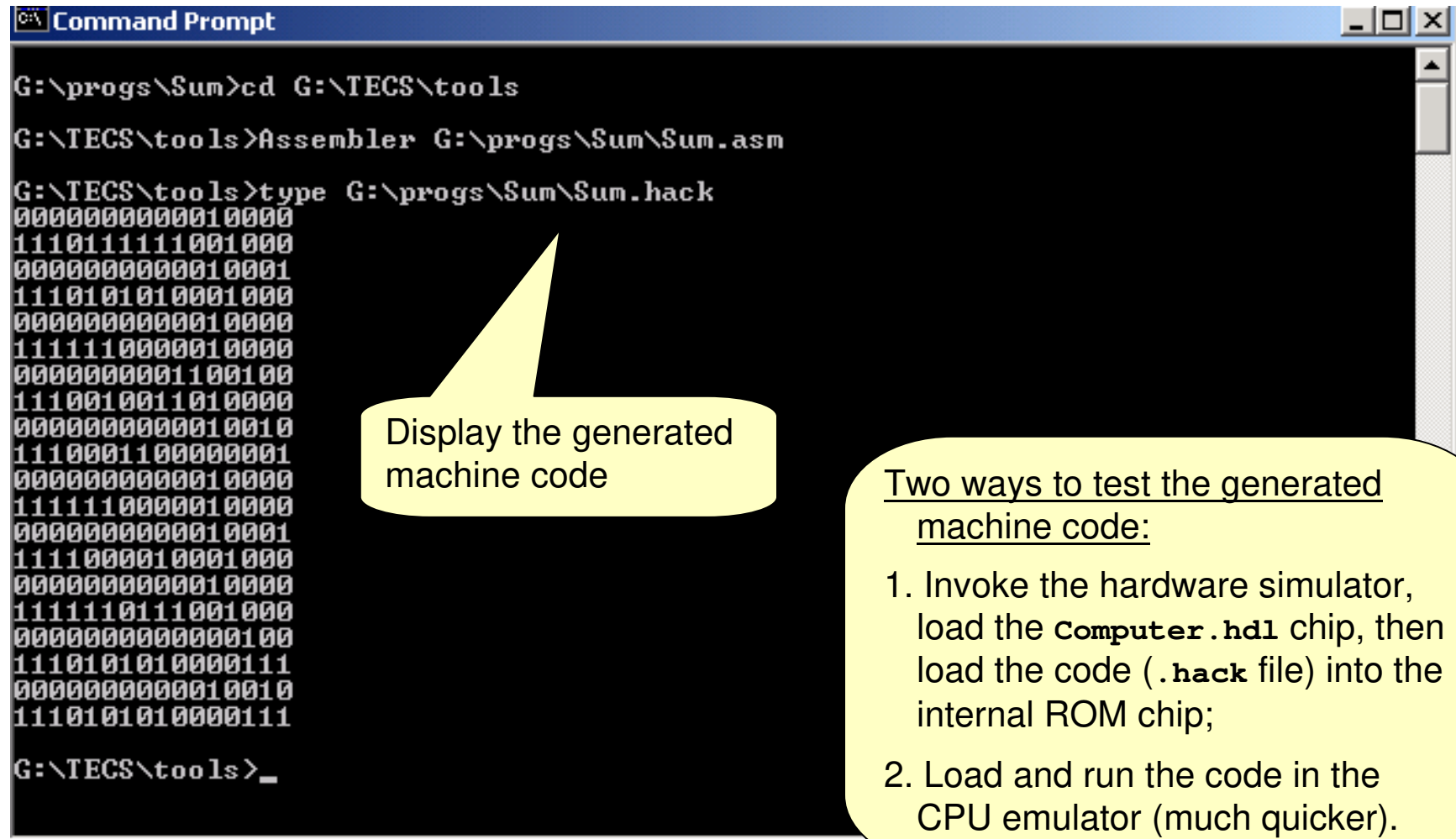


```
Command Prompt
G:\progs\Sum>cd G:\TECS\tools
G:\TECS\tools>Assembler G:\progs\Sum\Sum.asm
```

Invoke the assembler program

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

# Invoking the Assembler



```
Command Prompt

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
00000000000010000
1110111111001000
00000000000010001
1110101010001000
00000000000010000
11111100000010000
00000000001100100
1110010011010000
00000000000010010
11100011000000001
00000000000010000
11111100000010000
00000000000010001
1111000010001000
00000000000010000
1111110111001000
00000000000000100
1110101010000111
00000000000010010
1110101010000111

G:\TECS\tools>_
```

Display the generated machine code

Two ways to test the generated machine code:

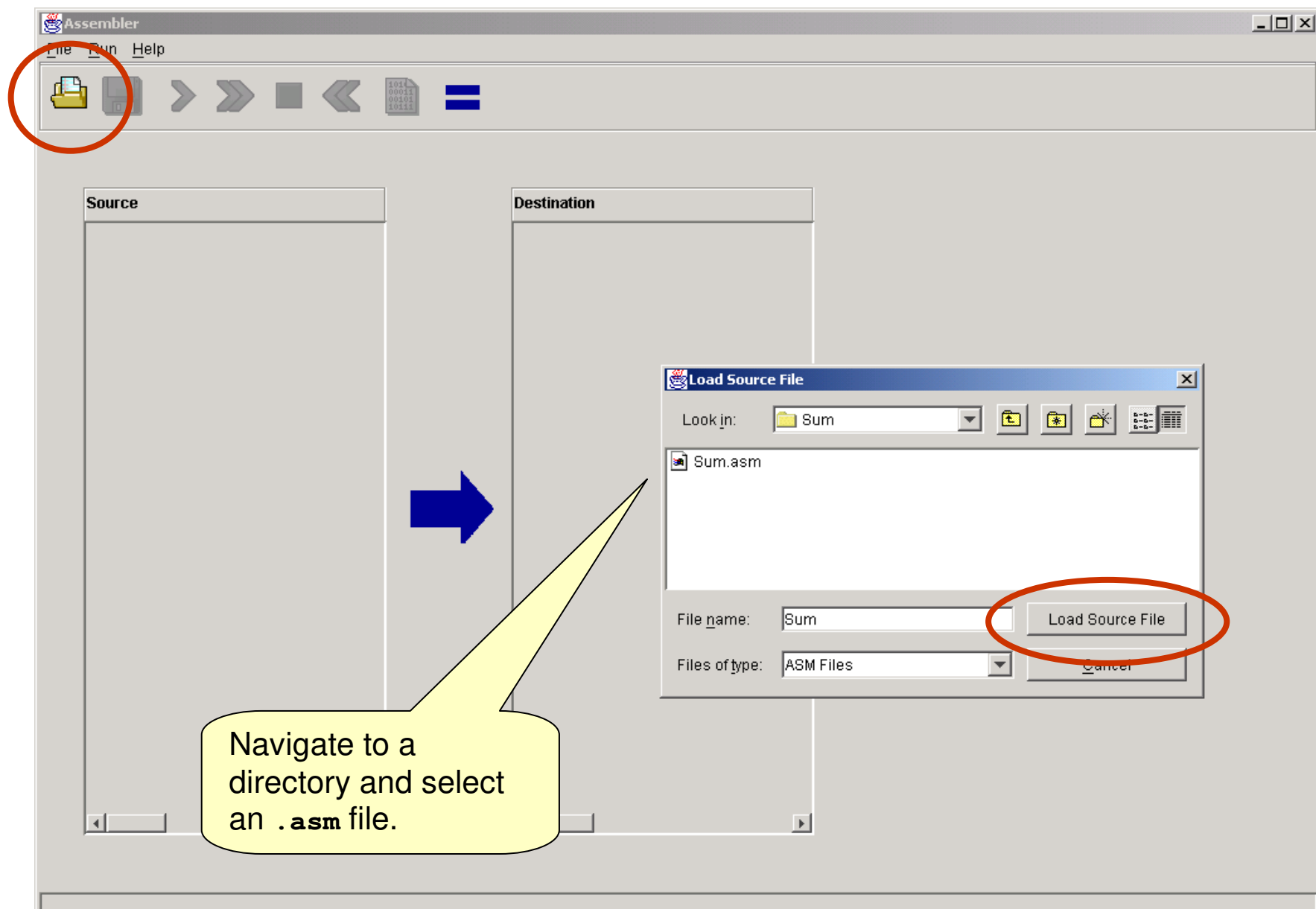
1. Invoke the hardware simulator, load the `Computer.hdl` chip, then load the code (`.hack` file) into the internal ROM chip;
2. Load and run the code in the CPU emulator (much quicker).

# Hardware Simulation Tutorial

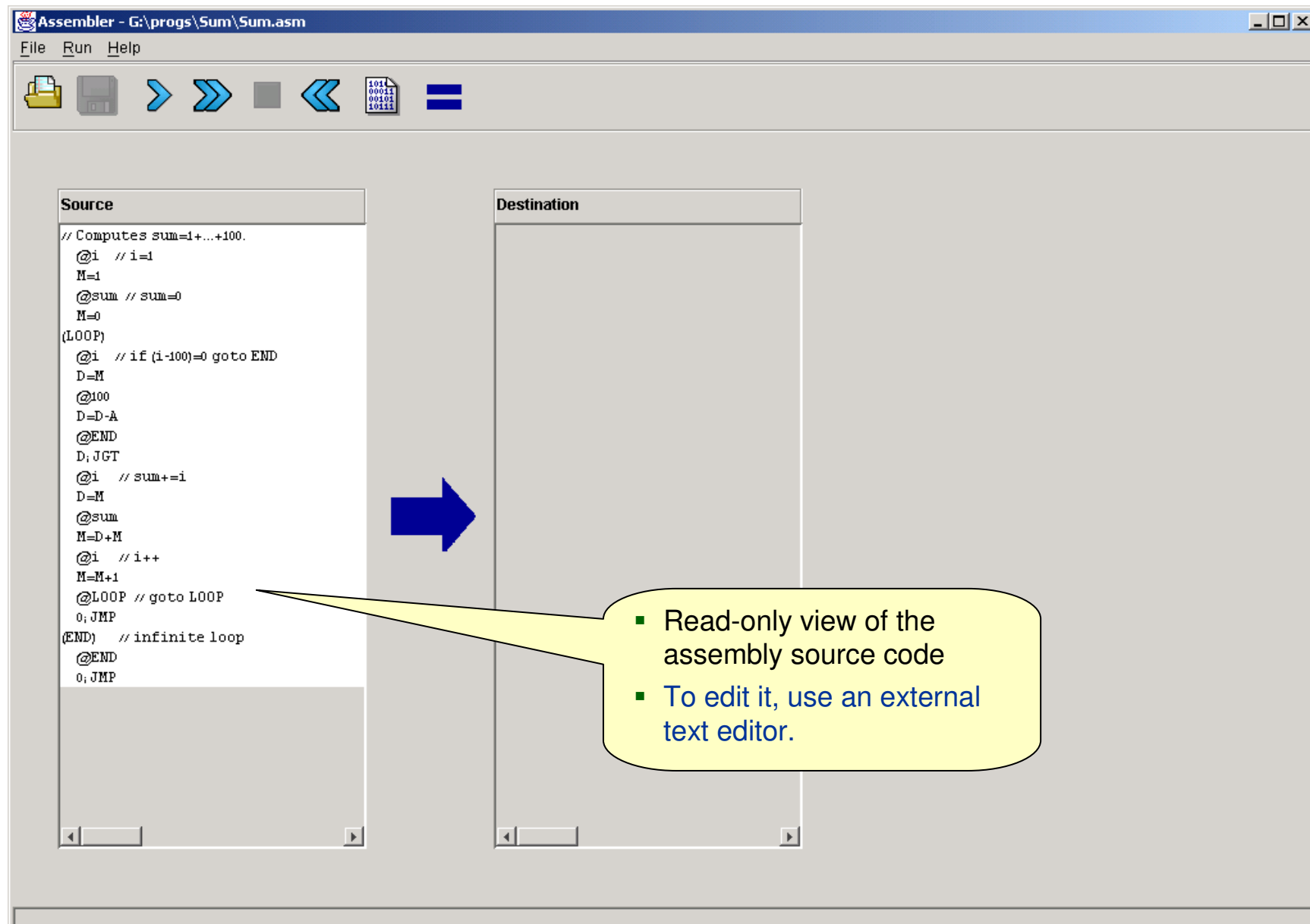
---



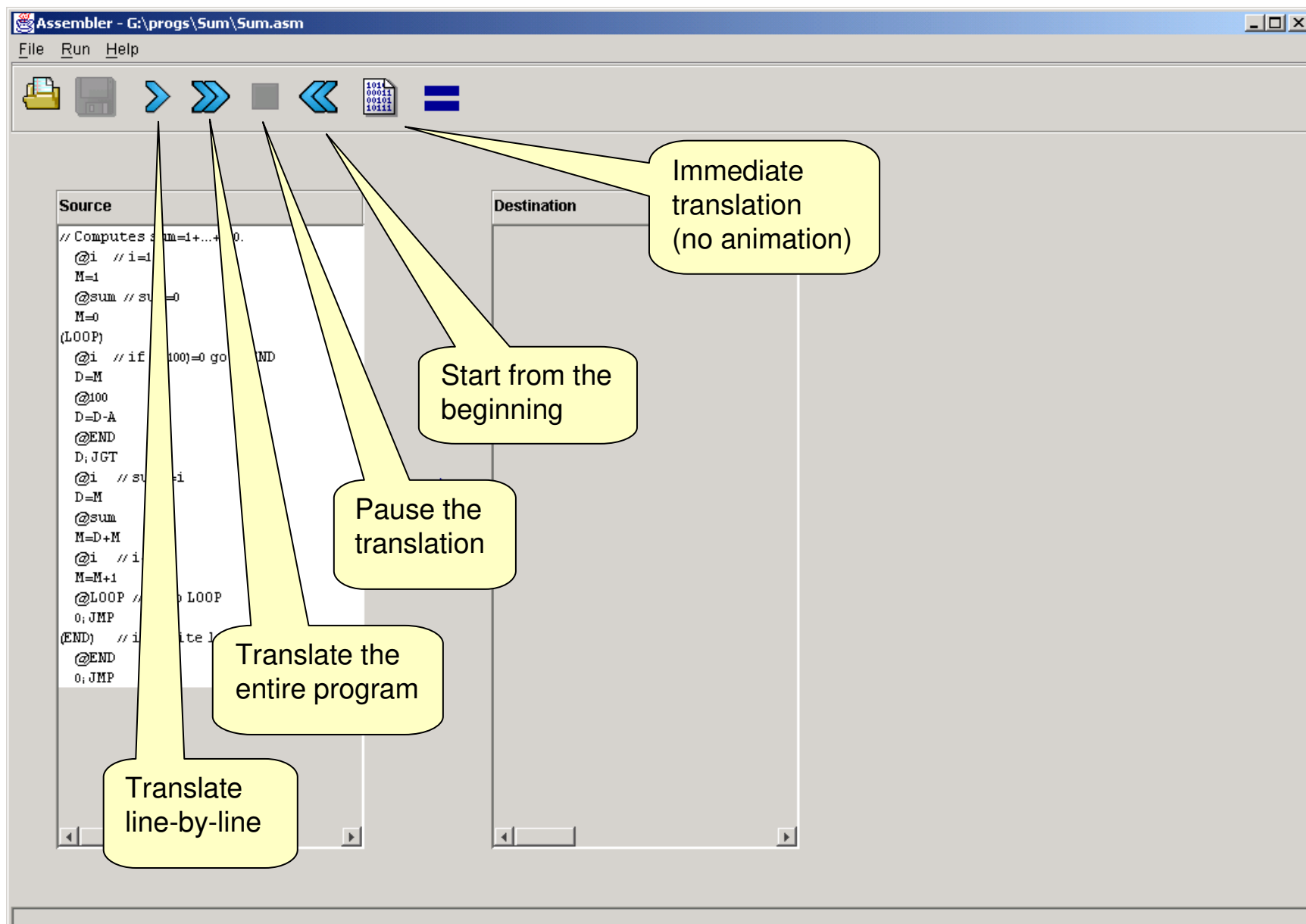
# Loading an assembly program



# Loading an assembly program



# Translating a program





# Inspecting the translation

**Assembler - G:\progs\Sum\Sum.asm**

File Run Help

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

```
0000000000001000  
1110111111001000  
0000000000001001  
1110101010001000  
0000000000001000  
1111110000001000  
00000000001100100  
1110010011010000  
00000000000010010  
1110001100000001  
00000000000010000  
11111100000010000  
00000000000010001  
1111000010001000  
00000000000010000  
1111110111001000  
00000000000000100  
1110101010000111  
00000000000010010  
1110101010000111
```

1. Click an assembly command

2. The corresponding translated code is highlighted

File compilation succeeded

# Saving the translated code

The screenshot shows the Assembler window with the file `G:\progs\Sum\Sum.asm`. The **Source** pane contains assembly code, and the **Destination** pane shows the translated binary code. A yellow callout bubble points to the save icon in the toolbar, stating: "Saves the translated code in a .hack file". A blue arrow points from the source code to the destination code. The status bar at the bottom indicates "File compilation succeeded".

**Source**

```
// Computes  
@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
```

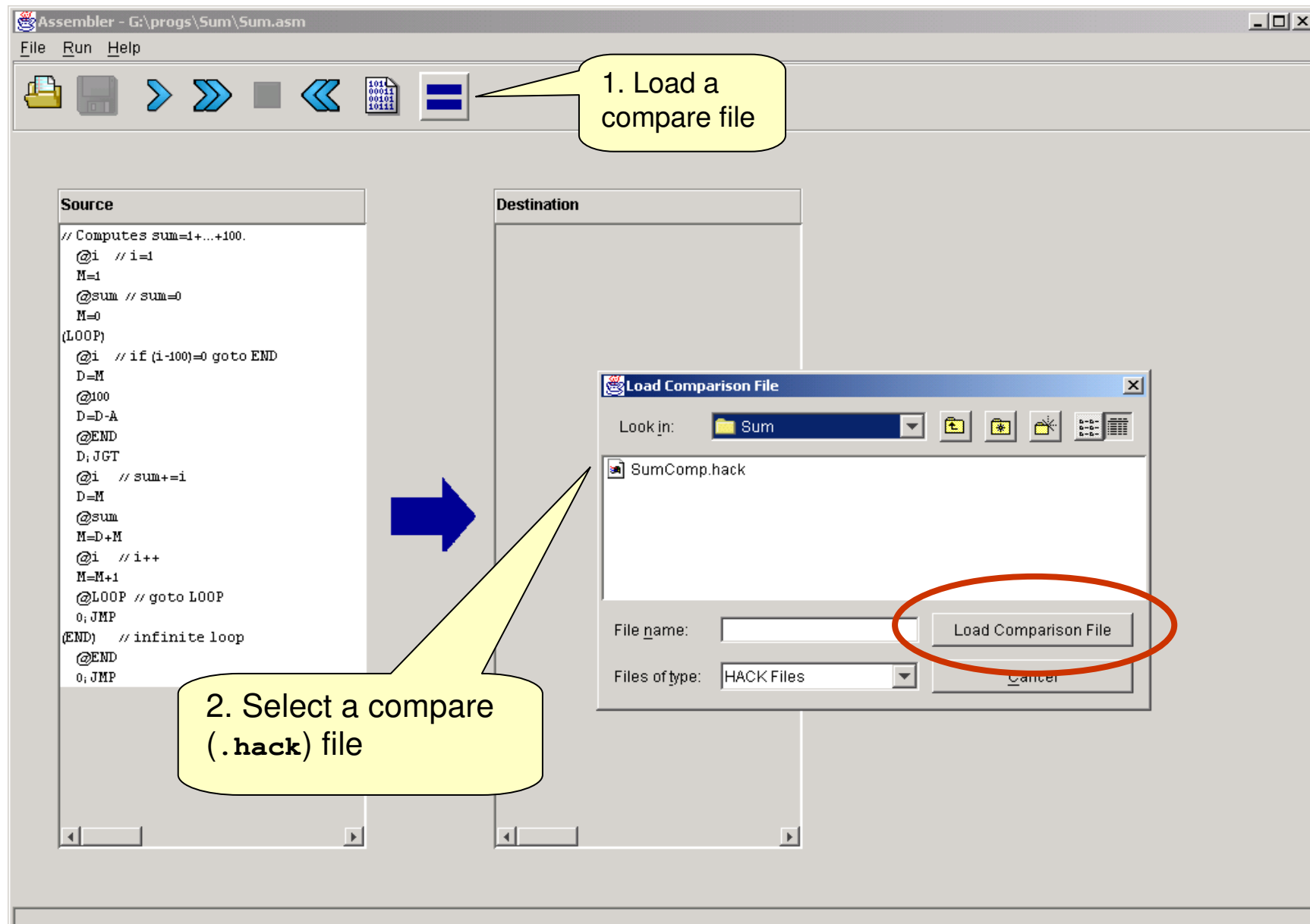
**Destination**

```
0000000000001000  
1110111111001000  
0000000000001001  
1110101010001000  
0000000000001000  
1111110000001000  
00000000001100100  
1110010011010000  
00000000000010010  
1110001100000001  
00000000000010000  
11111100000010000  
00000000000010001  
1111000010001000  
00000000000010000  
1111101110010000  
00000000000000100  
1110101010000111  
00000000000010010  
1110101010000111
```

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

File compilation succeeded

# Using Compare Files



# Using Compare Files

Assembler - D:\hack\instructor\Examples\sum\bad sum.asm

File Run Help

101  
00011  
00101  
10111

**Source**

```
// 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)
```

**Destination**

**Comparison**

```
0000000000010000  
1110111111001000  
0000000000010001  
1110101010001000  
0000000000010000  
1111110000010000  
0000000001100100  
1110010011010000  
0000000000010010  
1110001100000001  
0000000000010000  
1111110000010000  
0000000000010001  
1111000010001000  
0000000000010000  
1111101110010000  
0000000000001000  
1110101010000111
```

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

1. Compare file is shown

# Using Compare Files

The screenshot shows the 'Assembler - G:\progs\Sum\Sum.asm' window. It has three main panes: Source, Destination, and Comparison. The Source pane contains assembly code for a summing program. The Destination pane shows the binary translation of the source code. The Comparison pane shows the binary translation of a compare file. A blue arrow points from the highlighted line in the Source pane to the corresponding line in the Destination pane. A yellow speech bubble points to the comparison failure at the bottom.

**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**

```
00000000000010000  
1110111111001000  
00000000000010001  
1110101010001000  
00000000000010000  
11111100000010000  
00000000001100100  
1110010011010000  
00000000000010010  
11100011000000001  
00000000000010000  
11111100000010000  
00000000000010001  
1111000010001000  
00000000000010000  
1111110111001000  
00000000000000100  
1110101010000111  
00000000000010010  
1110101010000111
```

**Comparison**

```
00000000000010000  
1110111111001000  
00000000000010001  
1110101010001000  
00000000000010000  
11111100000010000  
00000000001100100  
1110010011010000  
00000000000010010  
11100011000000001  
00000000000010000  
11111100000010000  
00000000001010001  
1111000010001000  
00000000000010000  
1111110111001000  
00000000000000100  
1110101010000111  
00000000000010010  
1110101010000111
```

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

**Comparison failure**

## End-note: 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.