

# 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](http://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

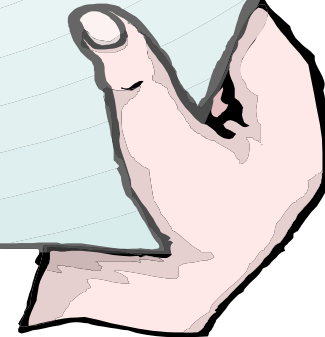
---

"The Elements of Computing Systems" evolves around the construction of a complete computer system, done in the framework of a 1- or 2-semester 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`, `VMEulator`):

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

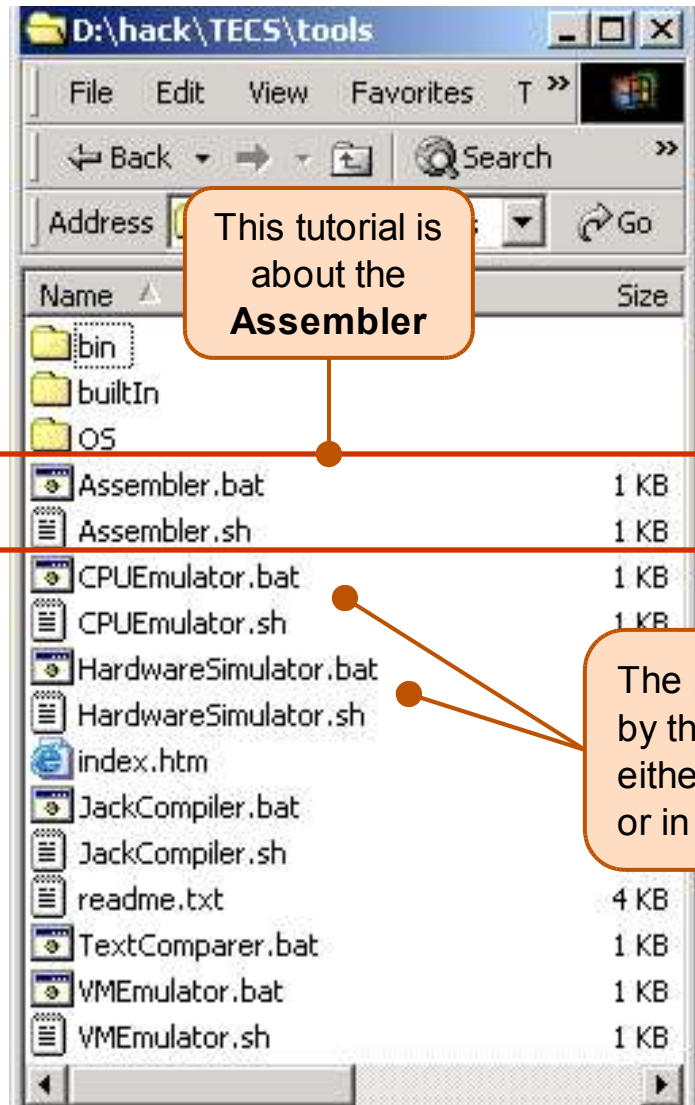
## Translators (`Assembler`, `JackCompiler`):

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

The machine code generated by the Assembler can be tested either in the Hardware Simulator or in the CPU Emulator.

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.



- I. [Assembly program example](#)
- II. [Command-level Assembler](#)
- III. [Interactive Assembler](#)

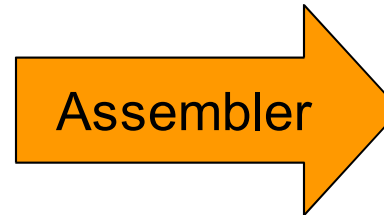
Relevant reading: Chapter 4: *Machine and Assembly Language*



# 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
  @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

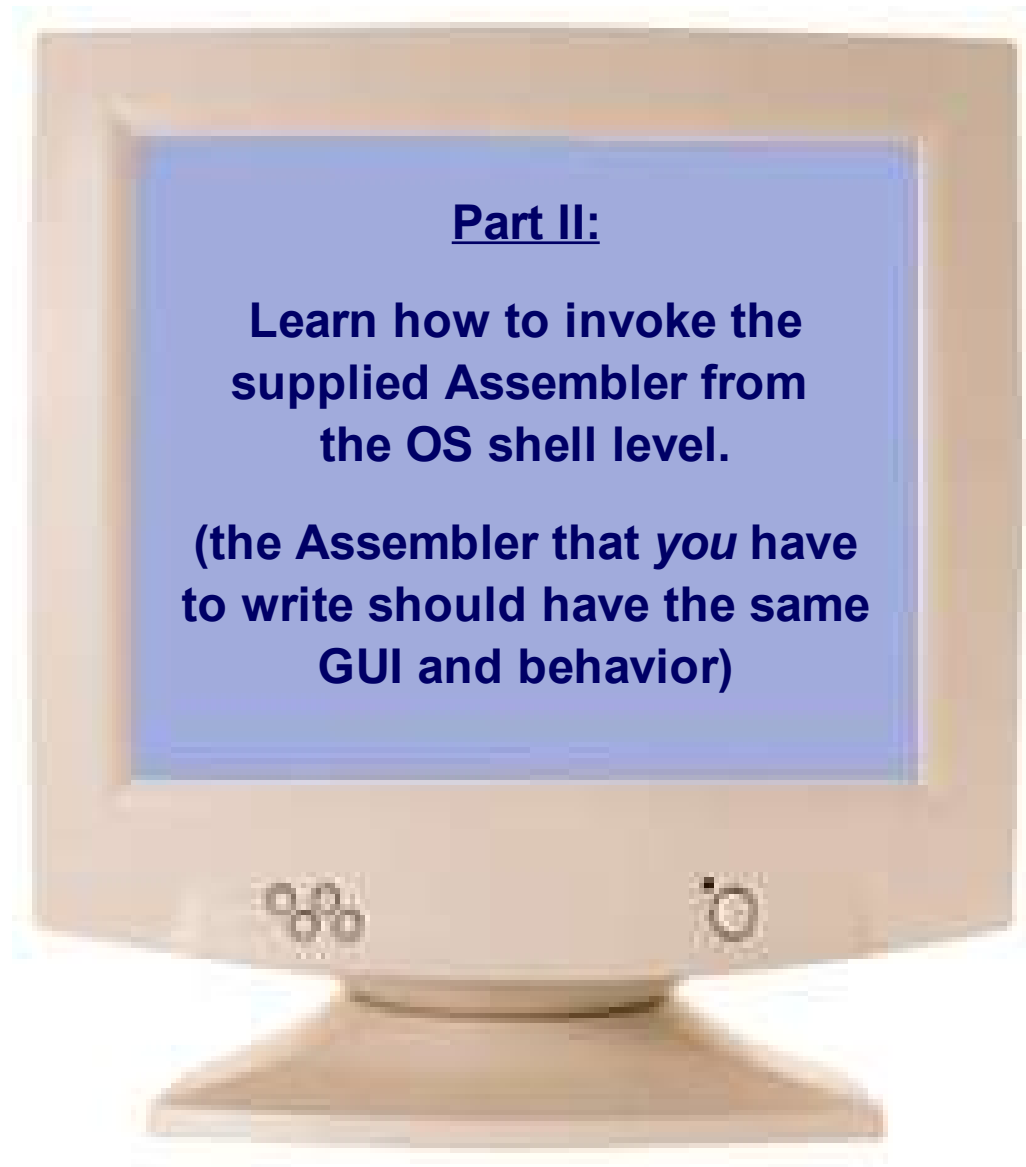
```
// Computes sum=1+...+100.
    @i      // i=1
    M=1
    @sum    // sum=0
    M=0
(LLOOP)
    @i      // if (i-100)=0 goto END
    D=M
    @100
    D=D-A
    @END
    D;JGT
    @i      // sum+=i
    D=M
    @sum
    M=D+M
    @i      // i++
    M=M+1
    @LOOP   // goto LOOP
    0;JMP
(END)    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

### The assembly program:

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

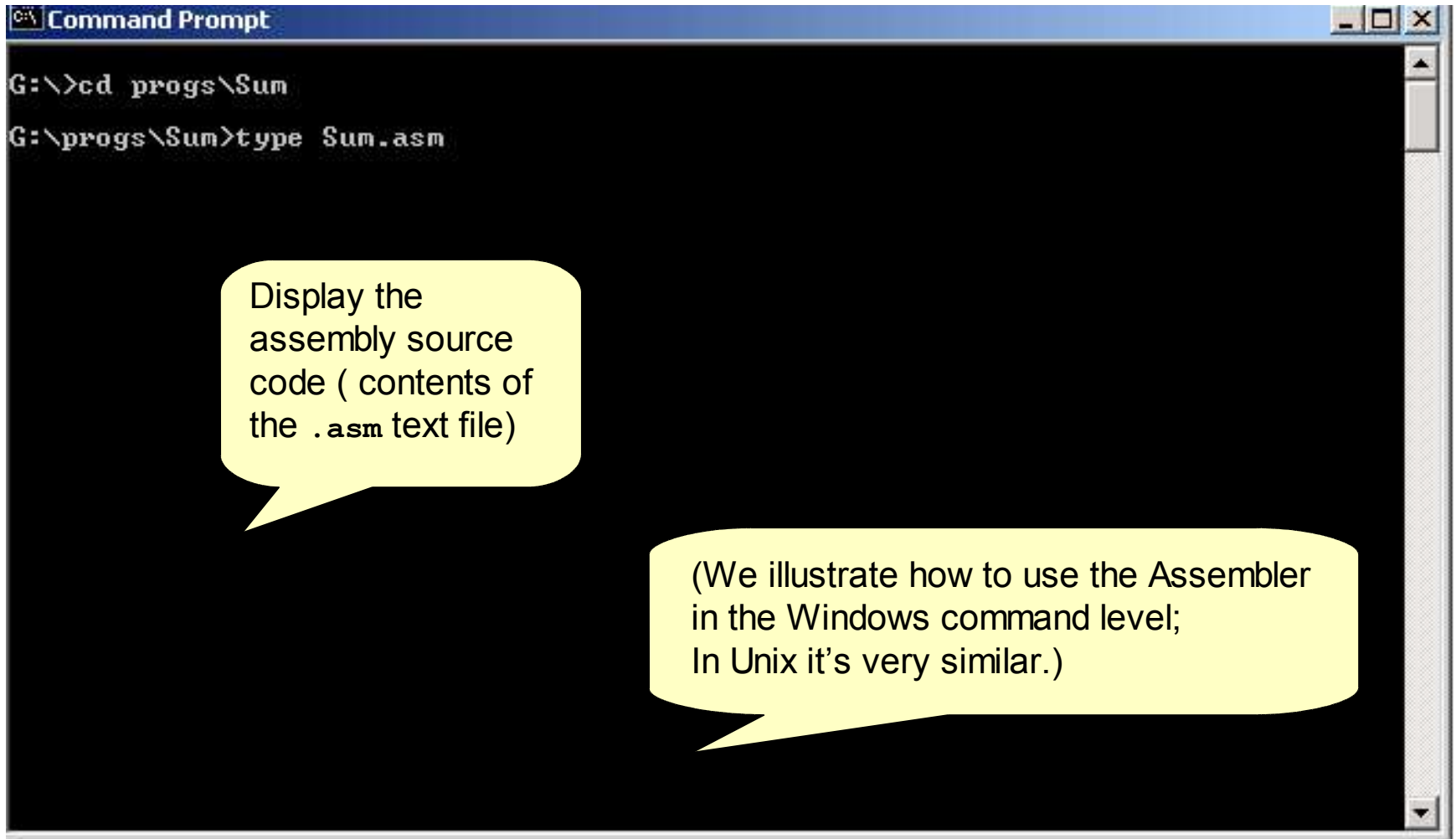
### The assembly process:

- 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.



# The command-level assembler

---



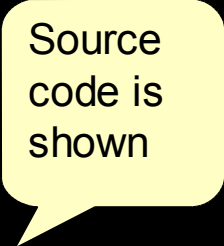
```
Command Prompt
G:\>cd progs\Sum
G:\progs\Sum>type Sum.asm
```

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

(We illustrate how to use the Assembler in the Windows command level; In Unix it's very similar.)

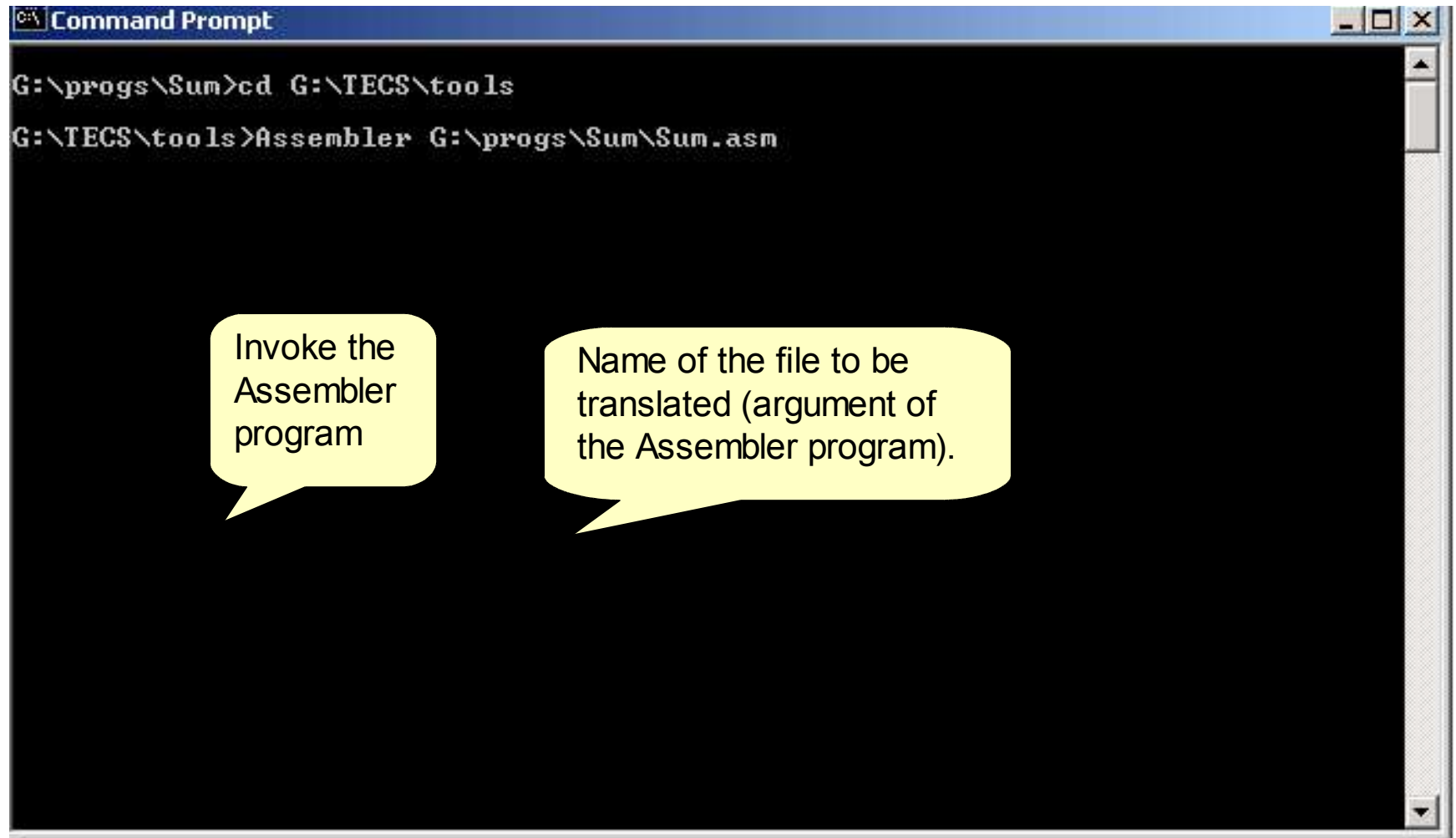
# Inspecting the source file

```
Command Prompt
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>
```



# 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
000000000000010000
1110111111001000
00000000000010001
1110101010001000
00000000000010000
11111100000010000
00000000001100100
1110010011010000
00000000000010010
1110001100000001
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.hd1` chip, then load the code (`.hack` file) into the internal ROM chip;
2. Load and run the code in the CPU Emulator (much quicker).

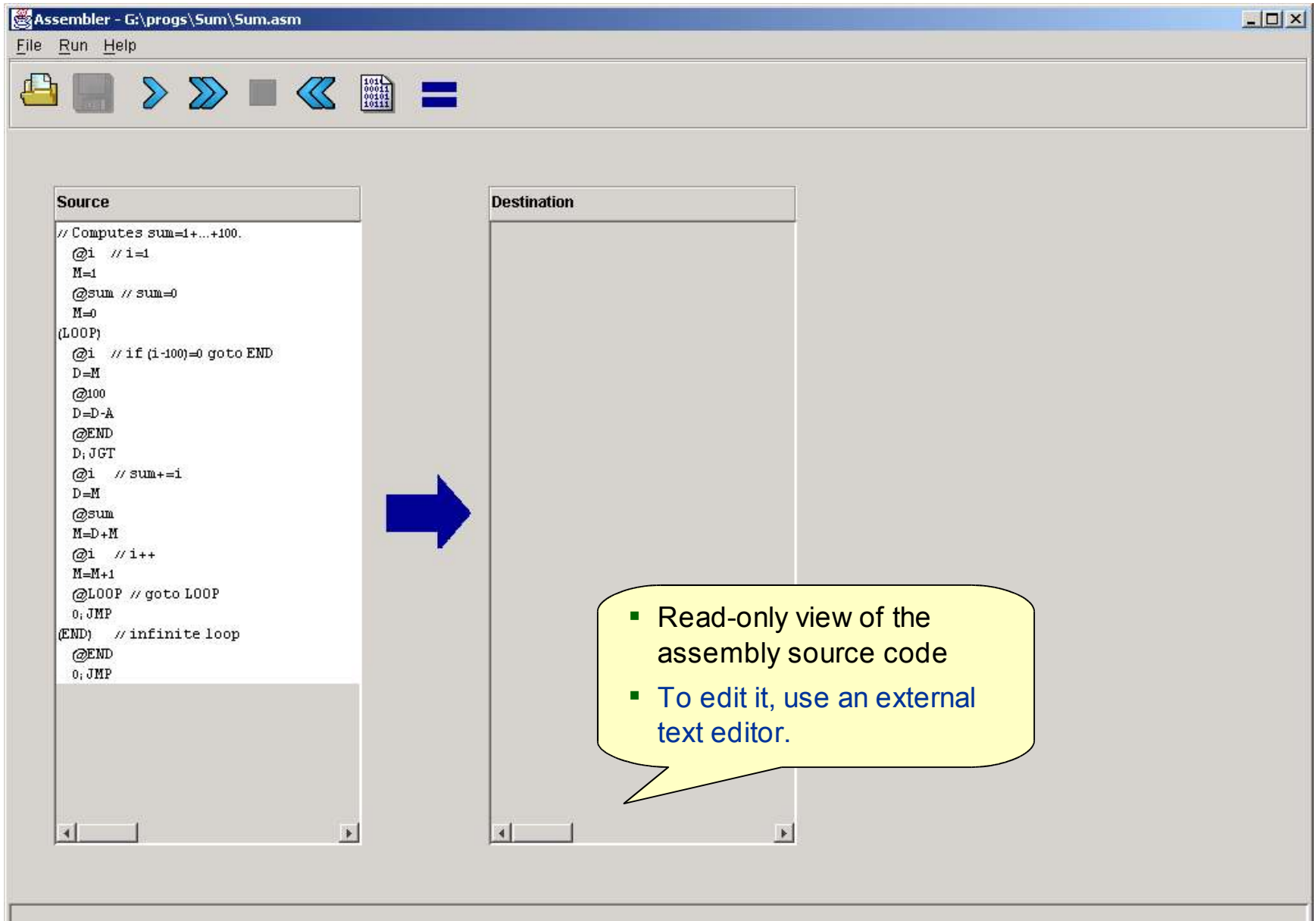


# Loading an assembly program

1. To load an assembly program, click here.

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

# Loading an assembly program



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

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

# Translating a program

The screenshot shows a window titled "Assembler - G:\progs\Sum\Sum.asm" with a menu bar (File, Run, Help) and a toolbar. The main area is split into two panes: "Source" and "Destination".

**Source Pane:**

```
// 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 Pane:**

The destination pane is currently empty. A callout bubble points to it with the text "Immediate translation (no animation)".

**Callout Bubbles:**

- "Immediate translation (no animation)" points to the Destination pane.
- "Start from the beginning" points to the top of the Source pane.
- "Pause the translation" points to the middle of the Source pane.
- "Translate the entire program" points to the bottom of the Source pane.
- "Translate line-by-line" points to the bottom of the Source pane.

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

**Destination**

```
000000000010000  
1110111111001000  
000000000010001  
1110101010001000  
000000000010000  
111111000010000  
000000001100100  
1110010011010000  
000000000010010  
111000110000001  
000000000010000  
111111000010000  
000000000010001  
1111000010001000  
000000000010000  
1111110111001000  
000000000000100  
1110101010000111  
000000000010010  
1110101010000111
```

**1. Click an assembly command**

**2. The corresponding translated code is highlighted**

File compilation succeeded

# Saving the translated code

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

File Run Help

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

```
000000000010000
1110111111001000
000000000010001
1110101010001000
000000000010000
111111000010000
000000001100100
1110010011010000
000000000010010
1110001100000001
000000000010000
111111000010000
000000000010001
1111000010001000
000000000010000
1111110111001000
000000000000100
1110101010000111
000000000010010
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.

# Comparing the translated code to a compare-file

The screenshot shows the 'Assembler' window with the file 'G:\progs\Sum\Sum.asm'. The 'Source' pane contains assembly code for a summing program. A blue arrow points from the source code to the 'Destination' pane. A yellow callout bubble above the toolbar says '1. Load a compare file'. A 'Load Comparison File' dialog box is open, showing the 'Sum' directory with the file 'SumComp.hack' listed. The 'Files of type' is set to 'HACK Files'. The 'Load Comparison File' button is circled in red. A yellow callout bubble below the source code says '2. Select a compare (.hack) file'.

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

File Run Help

1. Load a compare file

**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**

**Load Comparison File**

Look in: Sum

SumComp.hack

File name:  Load Comparison File

Files of type: HACK Files  Cancel

2. Select a compare (.hack) file

# Comparing the translated code to a compare-file

The screenshot shows the 'Assembler' window with the following content:

**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
```

**Toolbar:** A red circle highlights the compare icon (a document with binary code).

**Annotations:**

- Yellow speech bubble: "2. Translate the program (any translation mode can be used)"
- Yellow speech bubble: "1. Compare file is shown"

# Comparing the translated code to a compare-file

The screenshot shows the 'Assembler' window for 'G:\progs\Sum\Sum.asm'. It is divided into three main sections: Source, Destination, and Comparison.

**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+i  
@LOOP // goto LOOP  
0;JMP  
(END) // infinite loop  
@END  
0;JMP
```

**Destination:**

```
000000000010000  
1110111111001000  
000000000010001  
1110101010001000  
000000000010000  
1111110000010000  
000000001100100  
1110010011010000  
000000000010010  
1110001100000001  
000000000010000  
1111110000010000  
000000000010001
```

**Comparison:**

```
000000000010000  
1110111111001000  
000000000010001  
1110101010001000  
000000000010000  
1111110000010000  
000000001100100  
1110010011010000  
000000000010010  
1110001100000001  
000000000010000  
1111110000010000  
000000001010001  
1111000010001000  
000000000010000  
1111110111001000  
0000000000000100  
1110101010000111  
0000000000010010  
1110101010000111
```

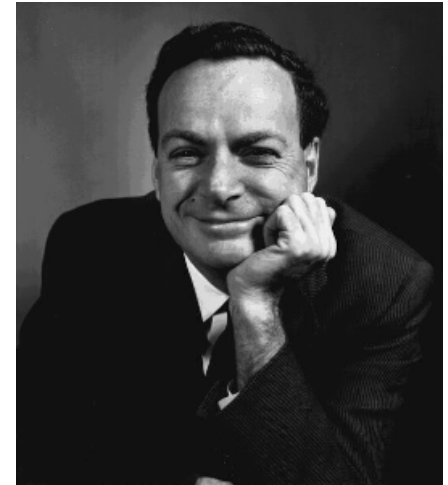
A blue arrow points from the highlighted line in the Source section to the Destination section. A blue equals sign is placed between the Destination and Comparison sections. A yellow speech bubble contains the text: "The translation of the highlighted line does not match the corresponding line in the compare file."

Comparison failure

## 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.