



# 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

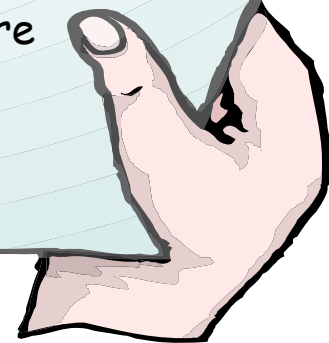
---

*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/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)

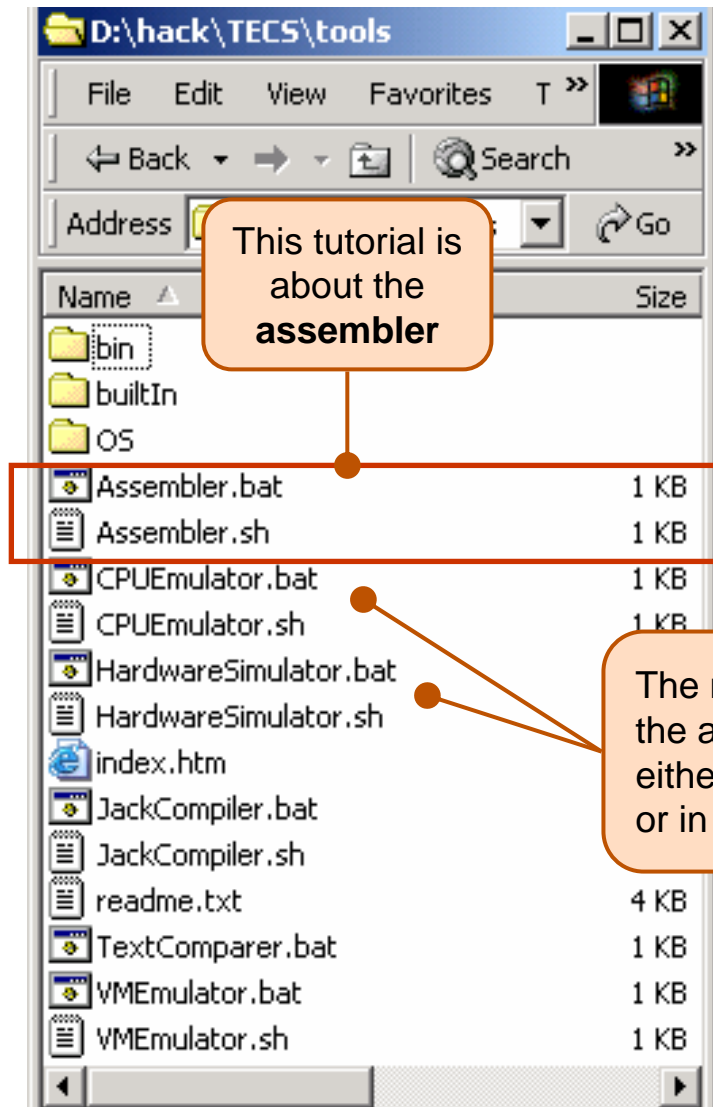
## Simulators

(`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](#)

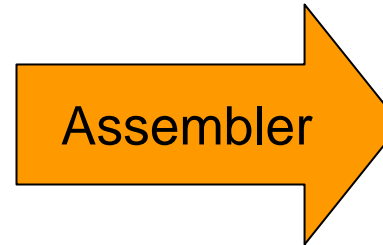
Relevant reading: Chapter 4: *Machine and Assembly Language*



# 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

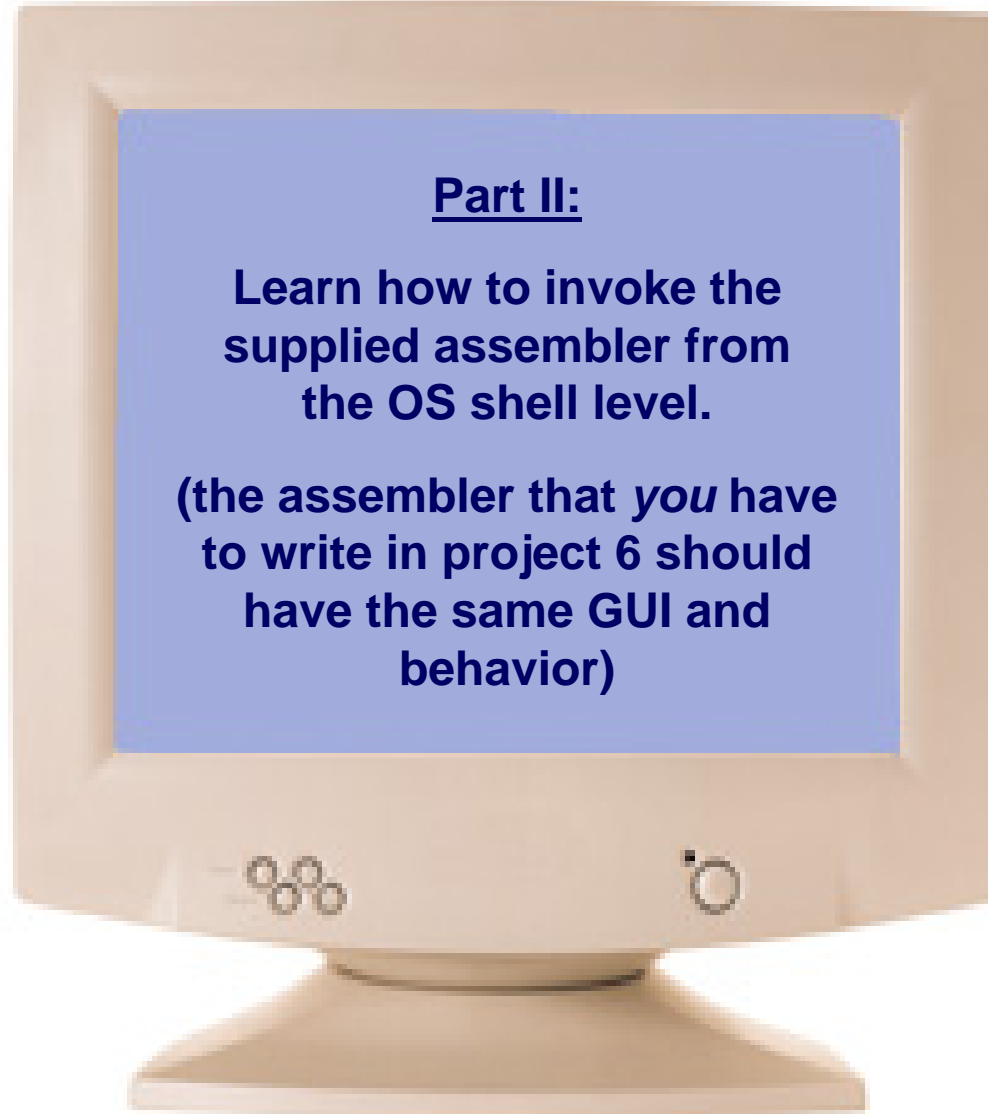
```
// Computes sum=1+...+100.
  @i      // i=1
  M=1
  @sum    // sum=0
  M=0
(LLOOP)
  @i      // if (i-100)=0 goto END
  D=M
  @100
  D=D-A
  @END
  D;JGT
  @i      // sum+=i
  D=M
  @sum
  M=D+M
  @i      // i++
  M=M+1
  @LLOOP  // goto LOOP
  0;JMP
(END)    // infinite loop
  @END
  0;JMP
```

### The assembly program:

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

### The assembly process:

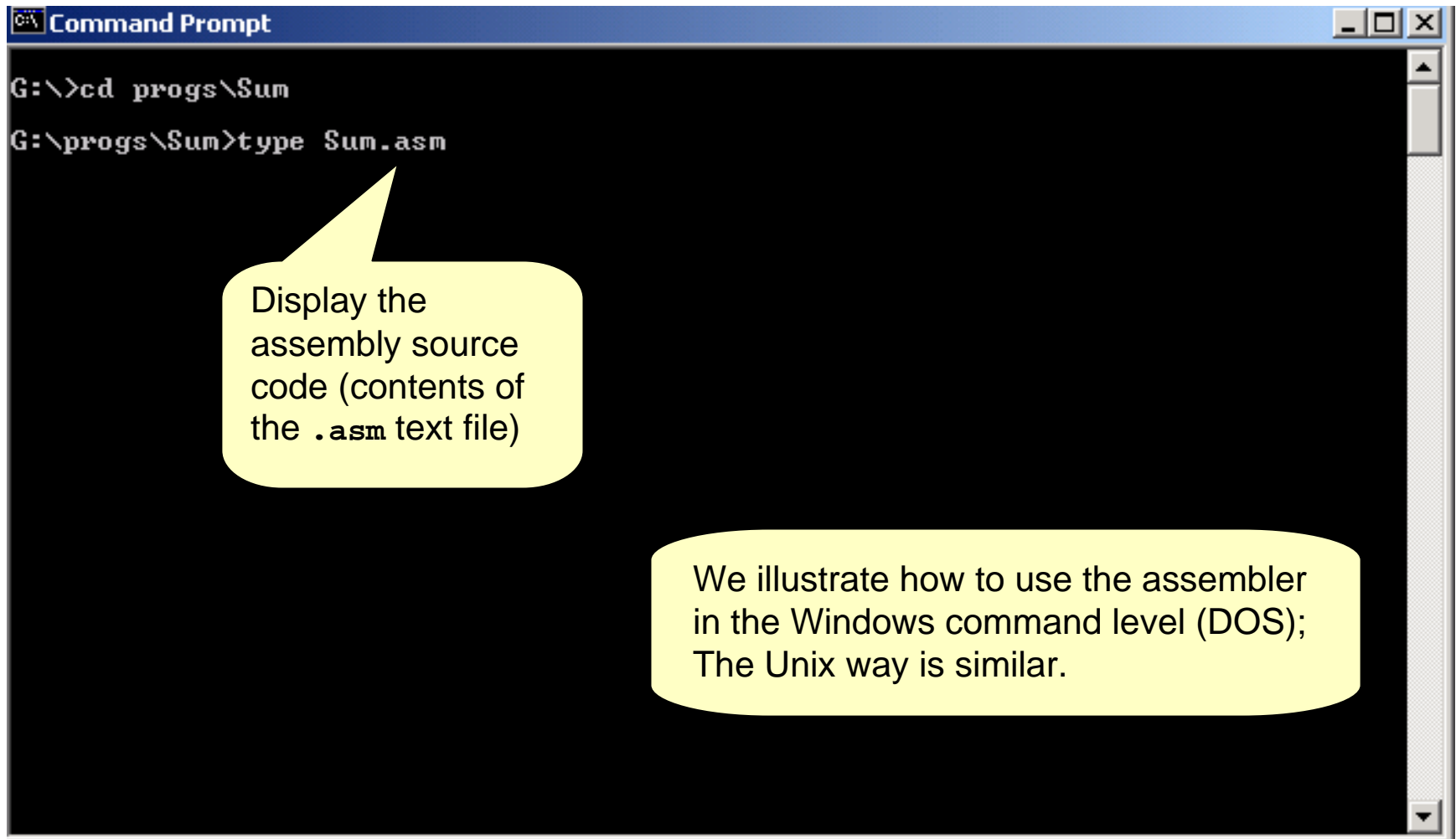
- Translates `Prog.asm` into `Prog.hack`
- Eliminates comments and white space
- Allocates variables (e.g. `i` and `sum`) to memory
- Translates each assembly command into a single 16-bit instruction written in the Hack machine language
- Treats label declarations like `(LOOP)` and `(END)` as pseudo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





# The command-level assembler

---

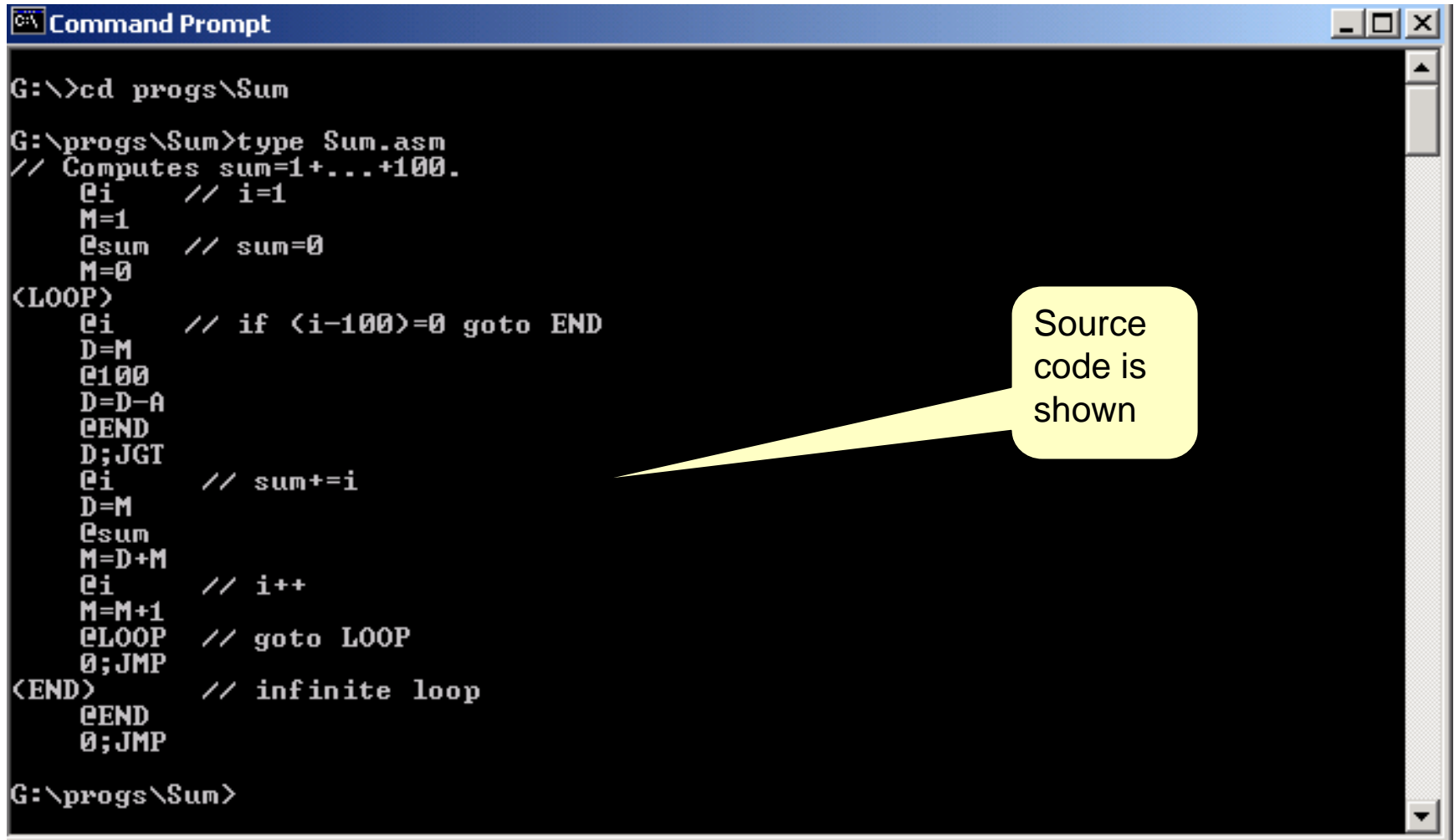


```
C:\ Command Prompt
G:\>cd progs\Sum
G:\progs\Sum>type Sum.asm
```

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

We illustrate how to use the assembler in the Windows command level (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
111011111001000
0000000000010001
1110101010001000
0000000000010000
111110000010000
0000000001100100
1110010011010000
0000000000010010
1110001100000001
0000000000010000
111110000010000
0000000000010001
1111000010001000
0000000000010000
111110111001000
0000000000000100
1110101010000111
0000000000010010
1110101010000111
G:\TECS\tools>_
```

Display the generated machine code

Two ways to test the generated machine code:

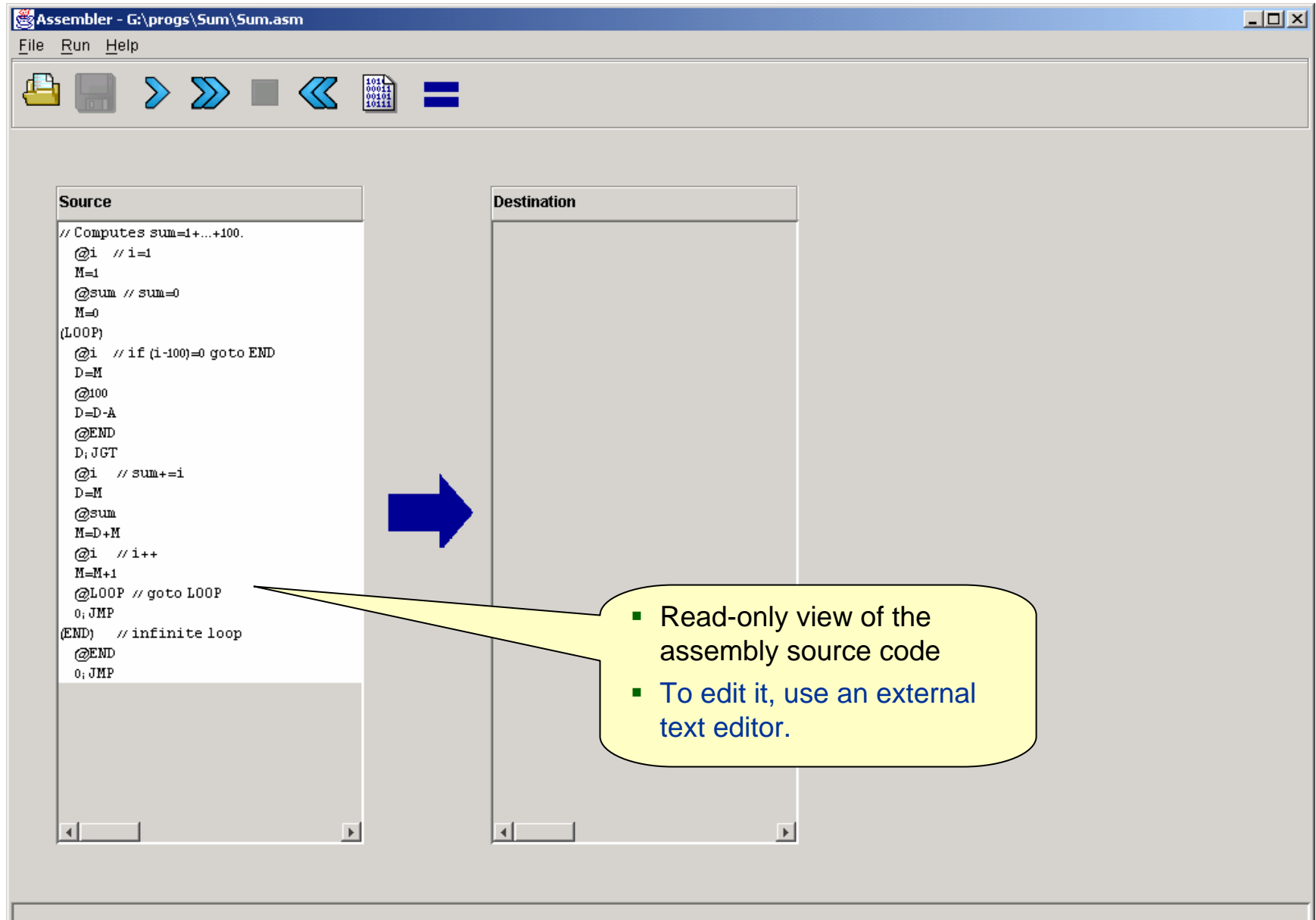
1. Invoke the hardware simulator, load the `computer.hd1` chip, then load the code (`.hack` file) into the internal ROM chip;
2. Load and run the code in the CPU emulator (much quicker).



# Loading an assembly program

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 the 'Assembler' window for a file named 'Sum.asm'. The interface is divided into two main panes: 'Source' on the left and 'Destination' on the right. The 'Source' pane contains assembly code for a program that computes the sum of integers from 1 to 100. The 'Destination' pane is currently empty. A toolbar at the top contains several icons for file operations and translation controls. Five yellow callout boxes with pointers identify these controls: 'Translate line-by-line' points to the first blue arrow icon; 'Translate the entire program' points to the second, larger blue arrow icon; 'Pause the translation' points to the grey square icon; 'Start from the beginning' points to the left-pointing blue arrow icon; and 'Immediate translation (no animation)' points to the icon showing binary code (10111, 00011, 00101, 10111) and an equals sign.

```
// Computes sum=1+...+100.
@i // i=1
M=1
@sum // sum=0
M=0
(LOOP)
@i // if (i-100)=0 go to 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 // go to LOOP
0; JMP
(END) // i=101
@END
0; JMP
```



# Inspecting the translation

The screenshot shows the 'Assembler - G:\progs\Sum\Sum.asm' window. The 'Source' pane contains assembly code, and the 'Destination' pane shows the corresponding binary code. A blue arrow points from the source to the destination. Two callout boxes provide instructions:

1. Click an assembly command

2. The corresponding translated code is highlighted

```
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
000000000010000
1110111111001000
000000000010001
1110101010001000
000000000010000
1111110000010000
0000000001100100
1110010011010000
000000000010010
1110001100000001
000000000010000
1111110000010000
000000000010001
1111000010001000
000000000010000
1111110111001000
0000000000000100
1110101010000111
0000000000010010
1110101010000111
```

# Saving the translated code

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

File Run Help

Source

```
// Computes
@i // i=1
M=1
@sum // sum
M=0
(L00P)
@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
@L00P // goto L00P
0; JMP
(END) // infinite loop
@END
0; JMP
```

Destination

```
000000000010000
1110111111001000
000000000010001
1110101010001000
000000000010000
1111110000010000
0000000001100100
1110010011010000
000000000010010
1110001100000001
000000000010000
1111110000010000
000000000010001
1111000010001000
000000000010000
1111110111001000
0000000000000100
1110101010000111
0000000000010010
1110101010000111
```

File compilation succeeded

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

# Using Compare Files

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
```

2. Select a compare (.hack) file

Load Comparison File

Look in: Sum

SumComp.hack

File name:

Files of type: HACK Files

Load Comparison File

Cancel

# Using Compare Files

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

File Run Help

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

```
000000000010000  
1110111111001000  
000000000010001  
1110101010001000  
000000000010000  
1111110000010000  
0000000001100100  
1110010011010000  
000000000010010  
1110001100000001  
000000000010000  
1111110000010000  
000000000010001  
1111000010001000  
000000000010000  
1111110111001000  
000000000000100  
1110101010000111
```

1. Compare file is shown

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

# Using Compare Files

The screenshot shows an Assembler window titled "Assembler - G:\progs\Sum\Sum.asm". The window is divided into three main sections: Source, Destination, and Comparison. The Source section contains assembly code for a program that computes the sum of integers from 1 to 100. The Destination section shows the binary translation of the source code. The Comparison section shows the binary translation of the compare file. A blue arrow points from the highlighted line in the Source section to the corresponding line in the Destination section. A yellow callout box points to the highlighted line in the Destination section, stating: "The translation of the highlighted line does not match the corresponding line in the compare file." The Comparison section shows a mismatch between the highlighted line in the Destination section and the corresponding line in the Comparison section.

```
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
000000000010000
1110111111001000
000000000010001
1110101010001000
000000000010000
1111110000010000
0000000001100100
1110010011010000
000000000010010
1110001100000001
000000000010000
1111110000010000
000000000010001
1111000010001000
000000000010000
1111110111001000
000000000000100
1110101010000111
0000000000010010
1110101010000111

Comparison
000000000010000
1110111111001000
000000000010001
1110101010001000
000000000010000
1111110000010000
0000000001100100
1110010011010000
000000000010010
1110001100000001
000000000010000
1111110000010000
0000000001010001
1111000010001000
000000000010000
1111110111001000
000000000000100
1110101010000111
0000000000010010
1110101010000111
```

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.



© Copyright California Institute of Technology. All rights reserved. Commercial use or modification of this material is prohibited.



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