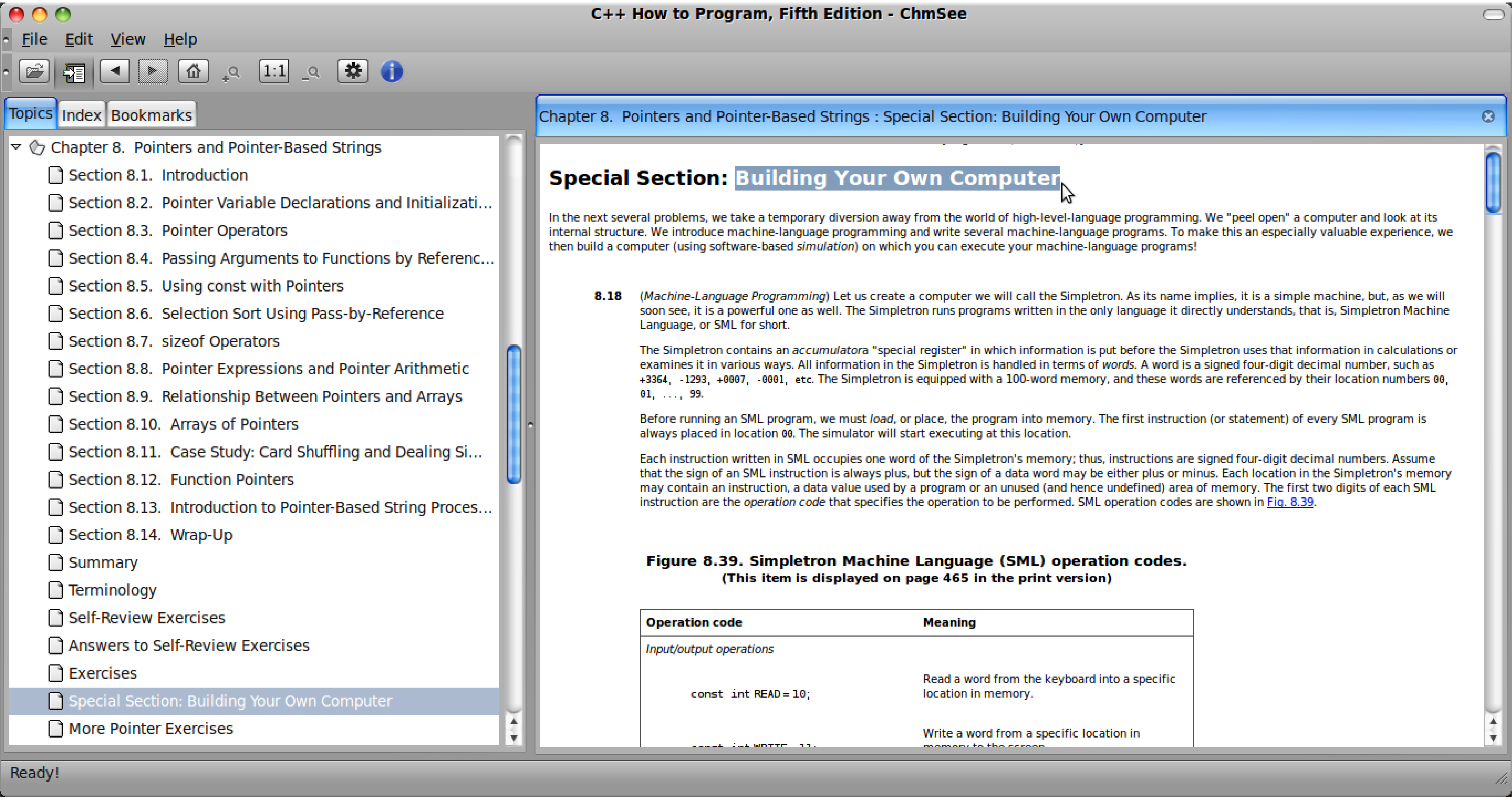
Building Your Own Computer

Objectives

- Introduce machine-language programming
- Build a computer (using software-based simulation) to execute machine-language programs!
- Have a fun ;)

Chapter 8 of "C++: How to Program" by Deitel...



Chapter 8. Pointers and Pointer-Based Strings : Special Section: Building Your Own Computer

Special Section: Building Your Own Computer

In the next several problems, we take a temporary diversion away from the world of high-level-language programming. We "peel open" a computer and look at its internal structure. We introduce machine-language programming and write several machine-language programs. To make this an especially valuable experience, we then build a computer (using software-based *simulation*) on which you can execute your machine-language programs!

8.18 (*Machine-Language Programming*) Let us create a computer we will call the Simpletron. As its name implies, it is a simple machine, but, as we will soon see, it is a powerful one as well. The Simpletron runs programs written in the only language it directly understands, that is, Simpletron Machine Language, or SML for short.

The Simpletron contains an *accumulator*a "special register" in which information is put before the Simpletron uses that information in calculations or examines it in various ways. All information in the Simpletron is handled in terms of *words*. A word is a signed four-digit decimal number, such as `+3364`, `-1293`, `+0007`, `-0001`, etc. The Simpletron is equipped with a 100-word memory, and these words are referenced by their location numbers `00`, `01`, `...`, `99`.

Before running an SML program, we must *load*, or place, the program into memory. The first instruction (or statement) of every SML program is always placed in location `00`. The simulator will start executing at this location.

Each instruction written in SML occupies one word of the Simpletron's memory; thus, instructions are signed four-digit decimal numbers. Assume that the sign of an SML instruction is always plus, but the sign of a data word may be either plus or minus. Each location in the Simpletron's memory may contain an instruction, a data value used by a program or an unused (and hence undefined) area of memory. The first two digits of each SML instruction are the *operation code* that specifies the operation to be performed. SML operation codes are shown in Fig. 8.39.

Figure 8.39. Simpletron Machine Language (SML) operation codes.
(This item is displayed on page 465 in the print version)

Operation code	Meaning
Input/output operations	
`const int READ = 10;`	Read a word from the keyboard into a specific location in memory.
`const int WRITE = 11;`	Write a word from a specific location in memory to the screen.

The Simpletron

- Runs programs written in Simpletron Machine Language
- Equipped with 100-word memory
  - A *word* is a signed 4-digit decimal number, such as +3364, -1293, +0007, -0001, etc.
  - A word contains an instruction of SML or a piece of data
- Has a bunch of registers like Accumulator, instructionRegister, counter, operationCode, operand

  A *register* is a memory location to store data inside a computer (fastest and most expensive memory)

SML operation set

```
• const int READ = 10; // reads data from keyboard to memory
• const int WRITE = 11; // writes data from memory to display
• const int LOAD = 20; // loads data from memory to accumulator
• const int STORE = 21; // stores data from accumulator to memory
• const int ADD = 30; // adds a value from memory to accumulator
• const int SUBTRACT = 31; // …
• const int DIVIDE = 32; // …
• const int MULTIPLY = 33; // …
• const int BRANCH = 40; // branches to a specific location in memory
• const int BRANCHNEG = 41; // … if accumulator is negative
• const int BRANCHZERO = 42; // … if accumulator is zero
• const int HALT = 43; // halts the program
```

Instruction Format

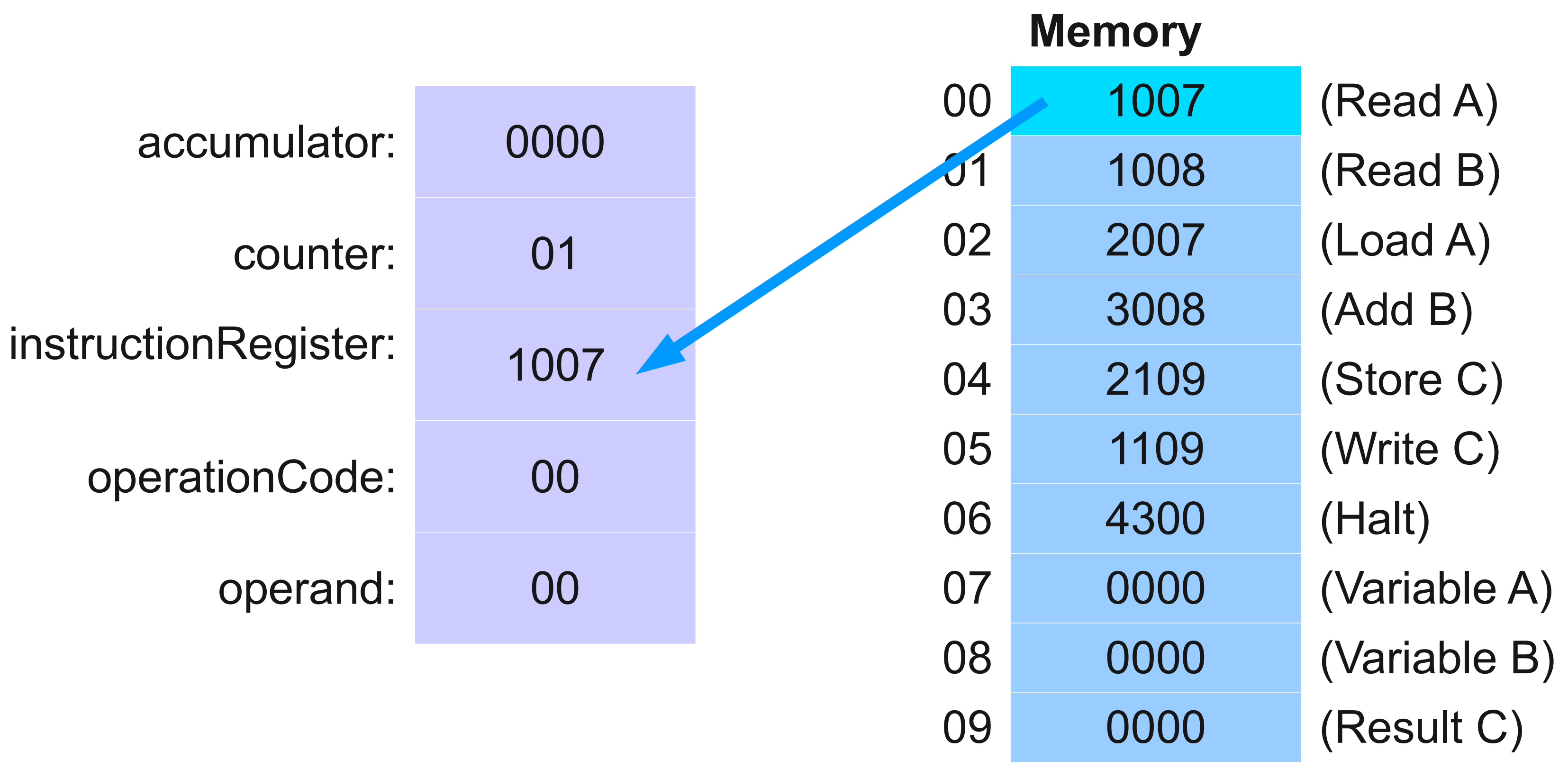
- Each instruction occupies 1 word = 4 digits
  - 2 digits for operation code
  - 2 digits for operands
    - In the Simpletron operands are memory addresses
- Example: `const int READ = 10;`
  - Instruction 1025 reads a “word” from keyboard to 25th location in memory
- Example: `const int ADD = 30;`
  - Instruction 3012 adds a “word” from 12th location in memory to accumulator

SML example 1
step 0: initialization ;)

accumulator:	0000
counter:	00
instructionRegister:	0000
operationCode:	00
operand:	00

	Memory	
00	1007	(Read A)
01	1008	(Read B)
02	2007	(Load A)
03	3008	(Add B)
04	2109	(Store C)
05	1109	(Write C)
06	4300	(Halt)
07	0000	(Variable A)
08	0000	(Variable B)
09	0000	(Result C)

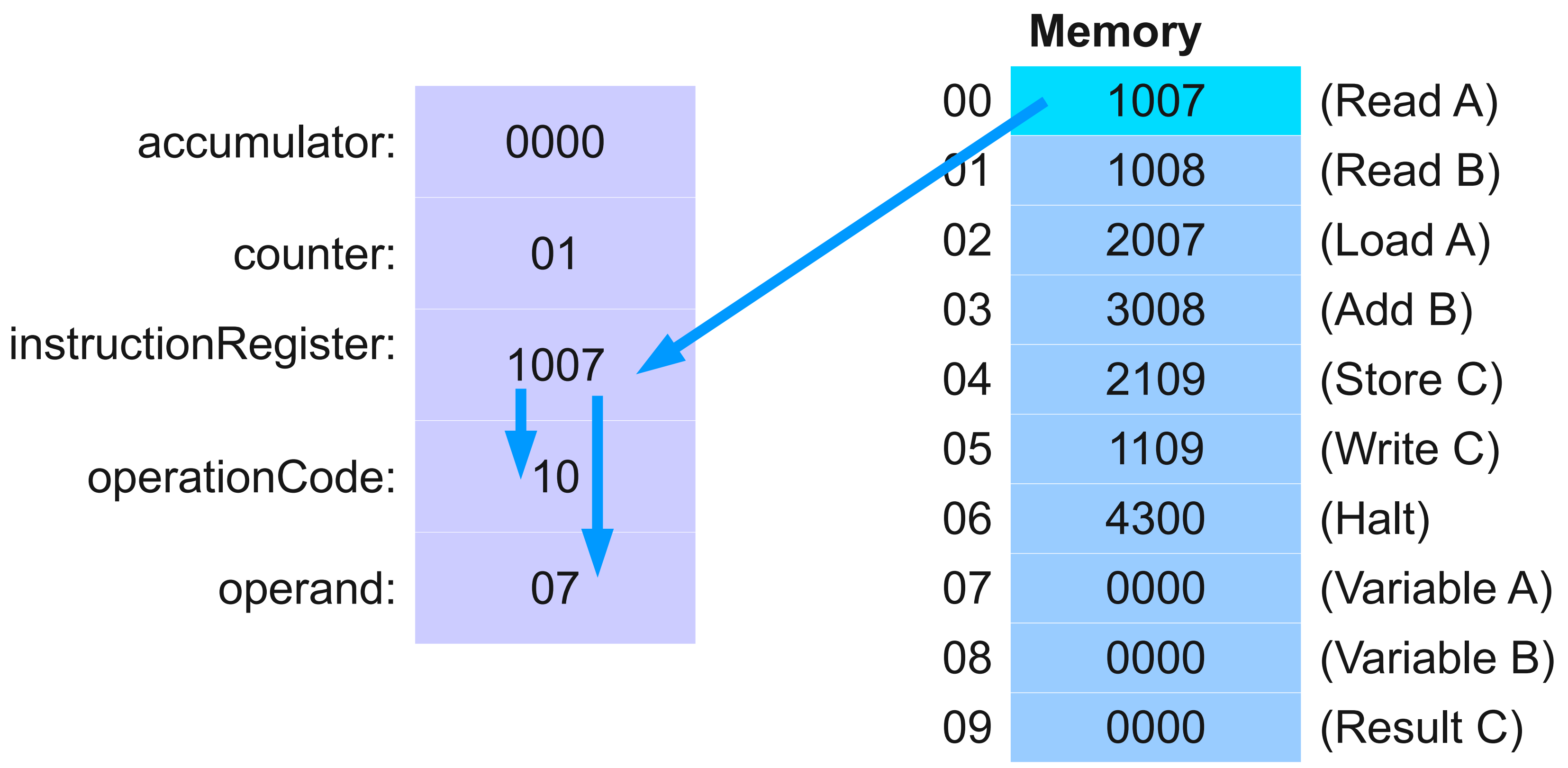
SML example 1
step 1: fetch instruction



accumulator:	0000
counter:	01
instructionRegister:	1007
operationCode:	00
operand:	00

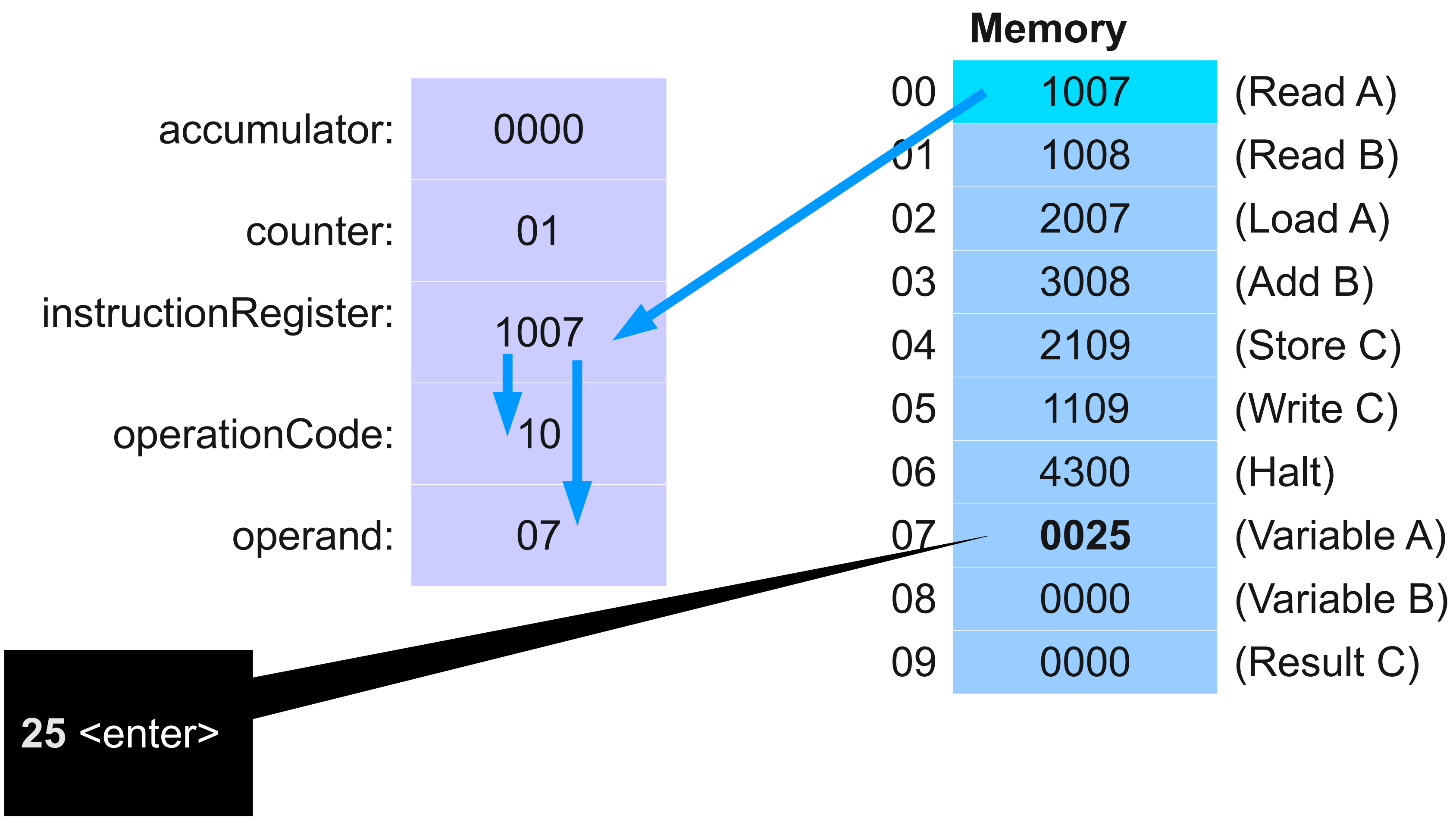
	Memory	
00	1007	(Read A)
01	1008	(Read B)
02	2007	(Load A)
03	3008	(Add B)
04	2109	(Store C)
05	1109	(Write C)
06	4300	(Halt)
07	0000	(Variable A)
08	0000	(Variable B)
09	0000	(Result C)

SML example 1
step 2: “digest” instruction



	Memory	
00	1007	(Read A)
01	1008	(Read B)
02	2007	(Load A)
03	3008	(Add B)
04	2109	(Store C)
05	1109	(Write C)
06	4300	(Halt)
07	0000	(Variable A)
08	0000	(Variable B)
09	0000	(Result C)

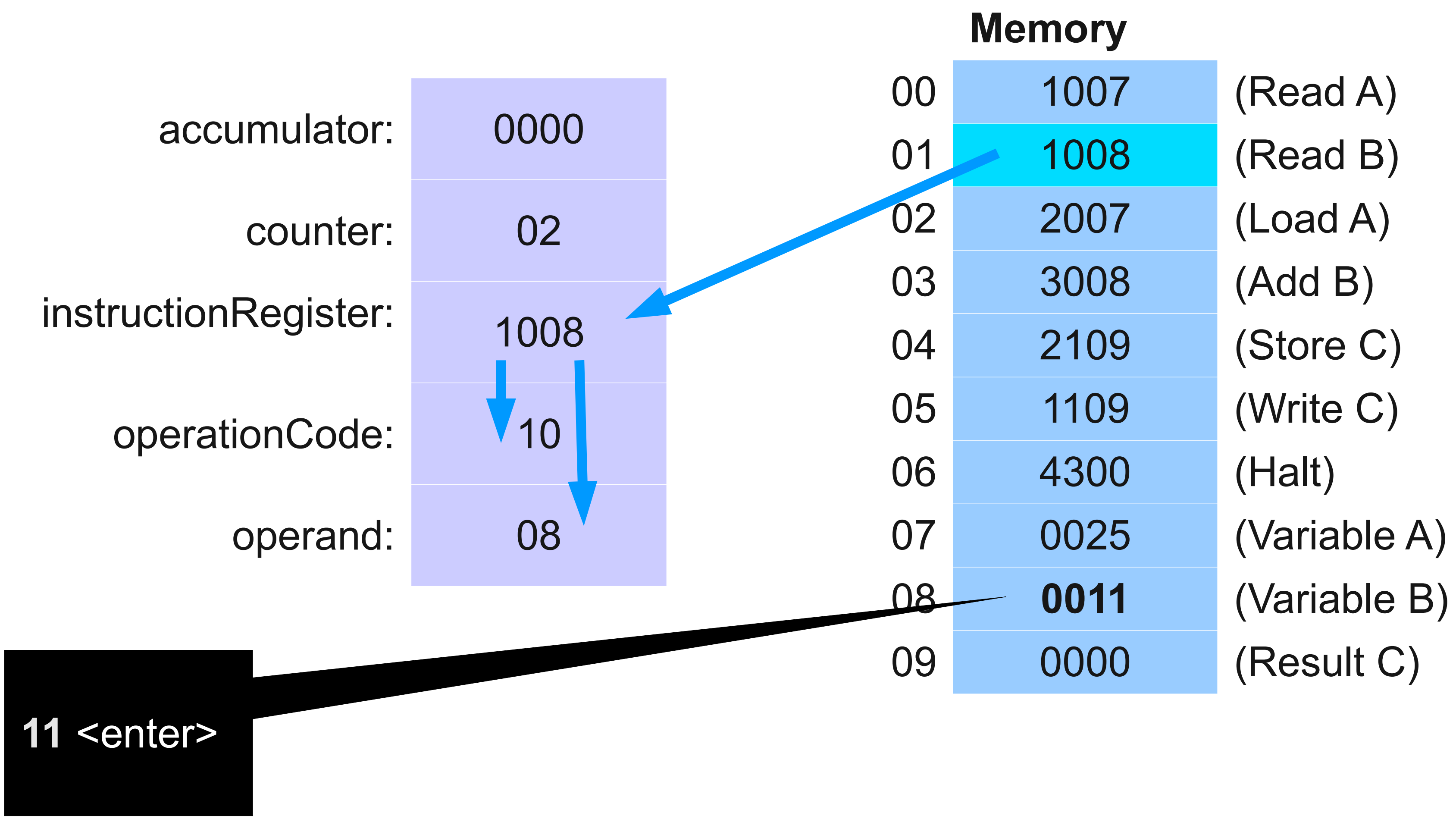
SML example 1
step 3: execute instruction



Memory

00	1007	(Read A)
01	1008	(Read B)
02	2007	(Load A)
03	3008	(Add B)
04	2109	(Store C)
05	1109	(Write C)
06	4300	(Halt)
07	**0025**	(Variable A)
08	0000	(Variable B)
09	0000	(Result C)

SML example 1
fetch+digest+execute

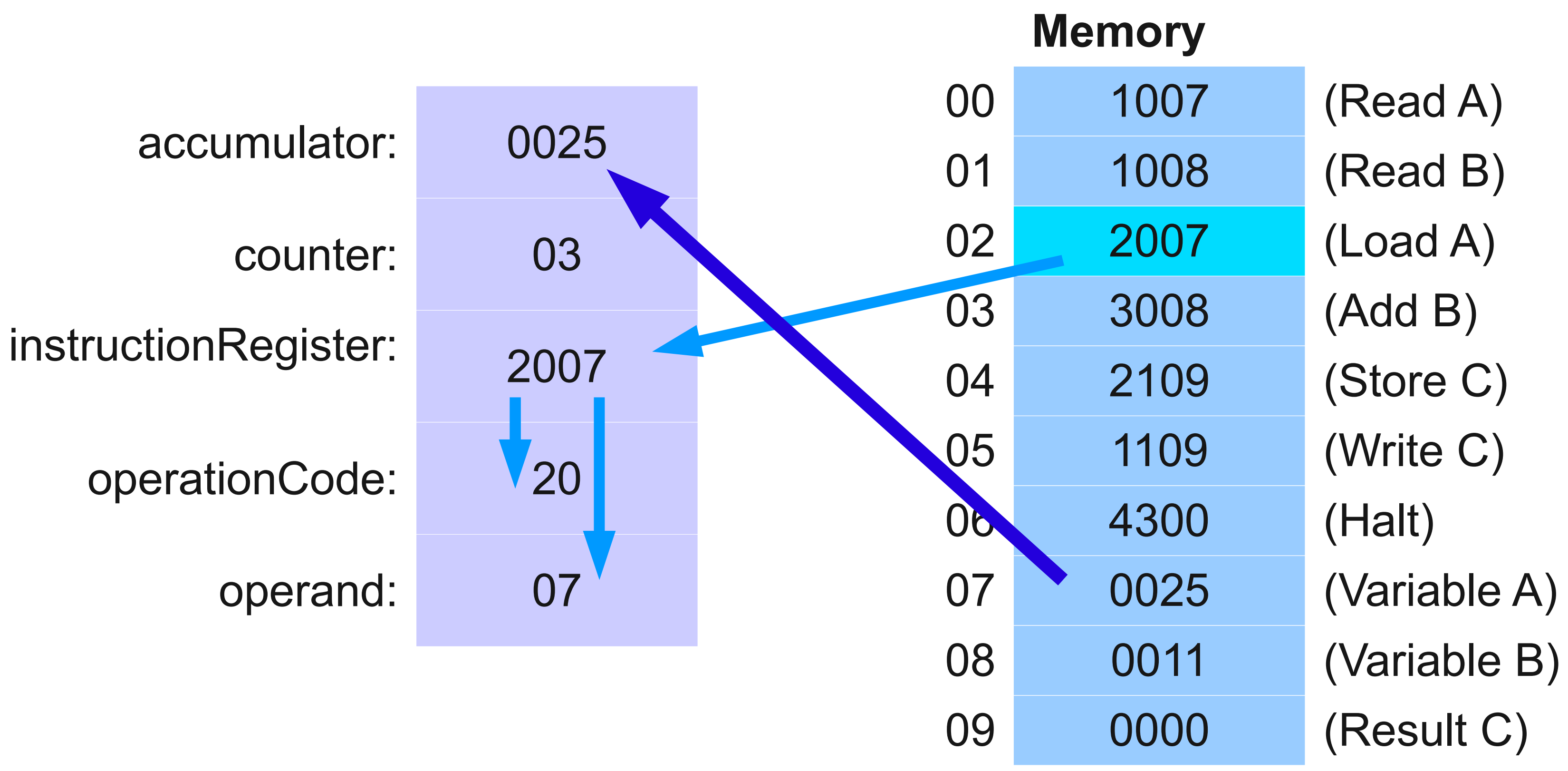


accumulator:	0000
counter:	02
instructionRegister:	1008
operationCode:	10
operand:	08

Memory

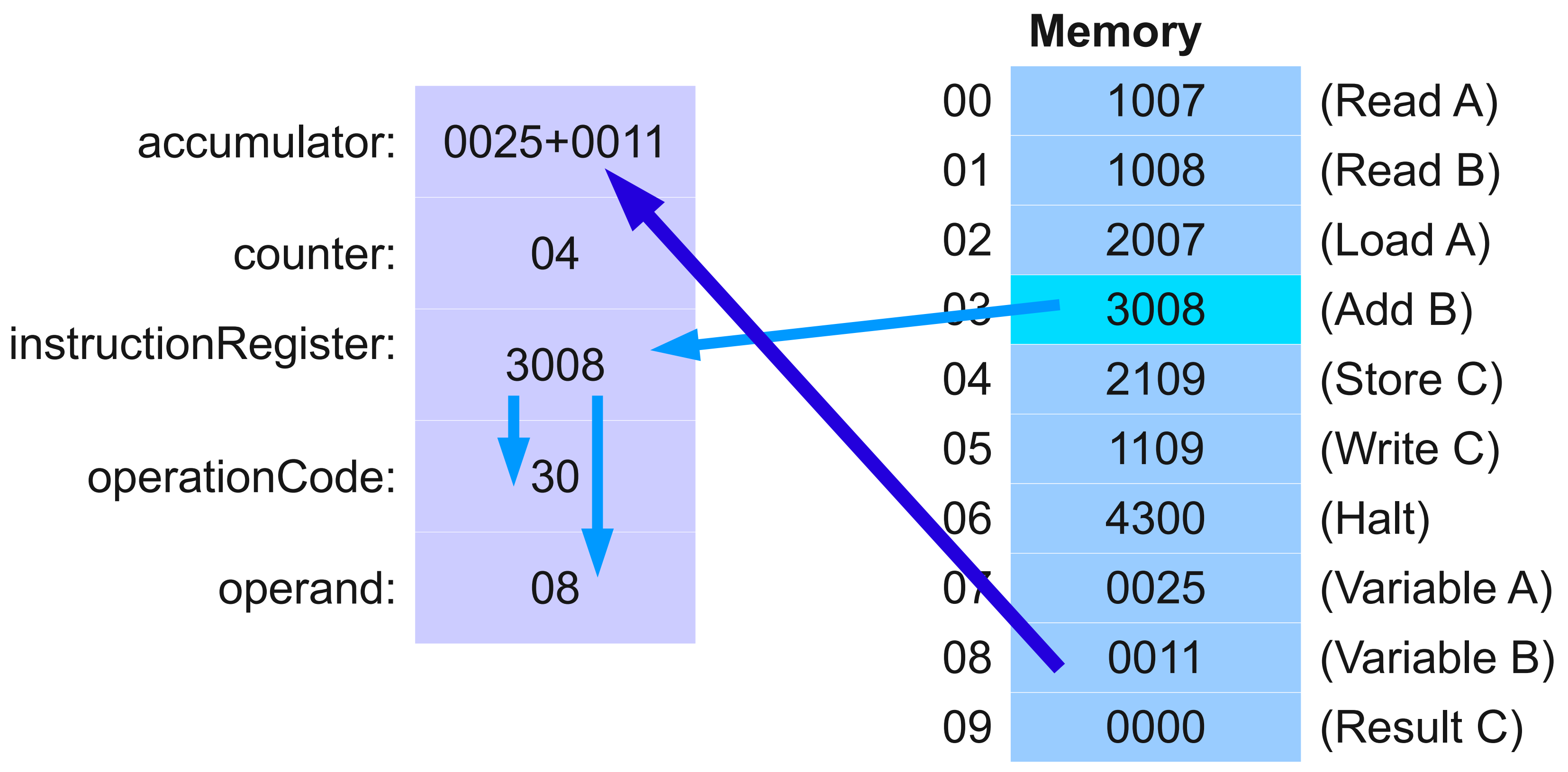
00	1007	(Read A)
01	1008	(Read B)
02	2007	(Load A)
03	3008	(Add B)
04	2109	(Store C)
05	1109	(Write C)
06	4300	(Halt)
07	0025	(Variable A)
08	**0011**	(Variable B)
09	0000	(Result C)

SML example 1
fetch+digest+execute



	Memory	
00	1007	(Read A)
01	1008	(Read B)
02	2007	(Load A)
03	3008	(Add B)
04	2109	(Store C)
05	1109	(Write C)
06	4300	(Halt)
07	0025	(Variable A)
08	0011	(Variable B)
09	0000	(Result C)

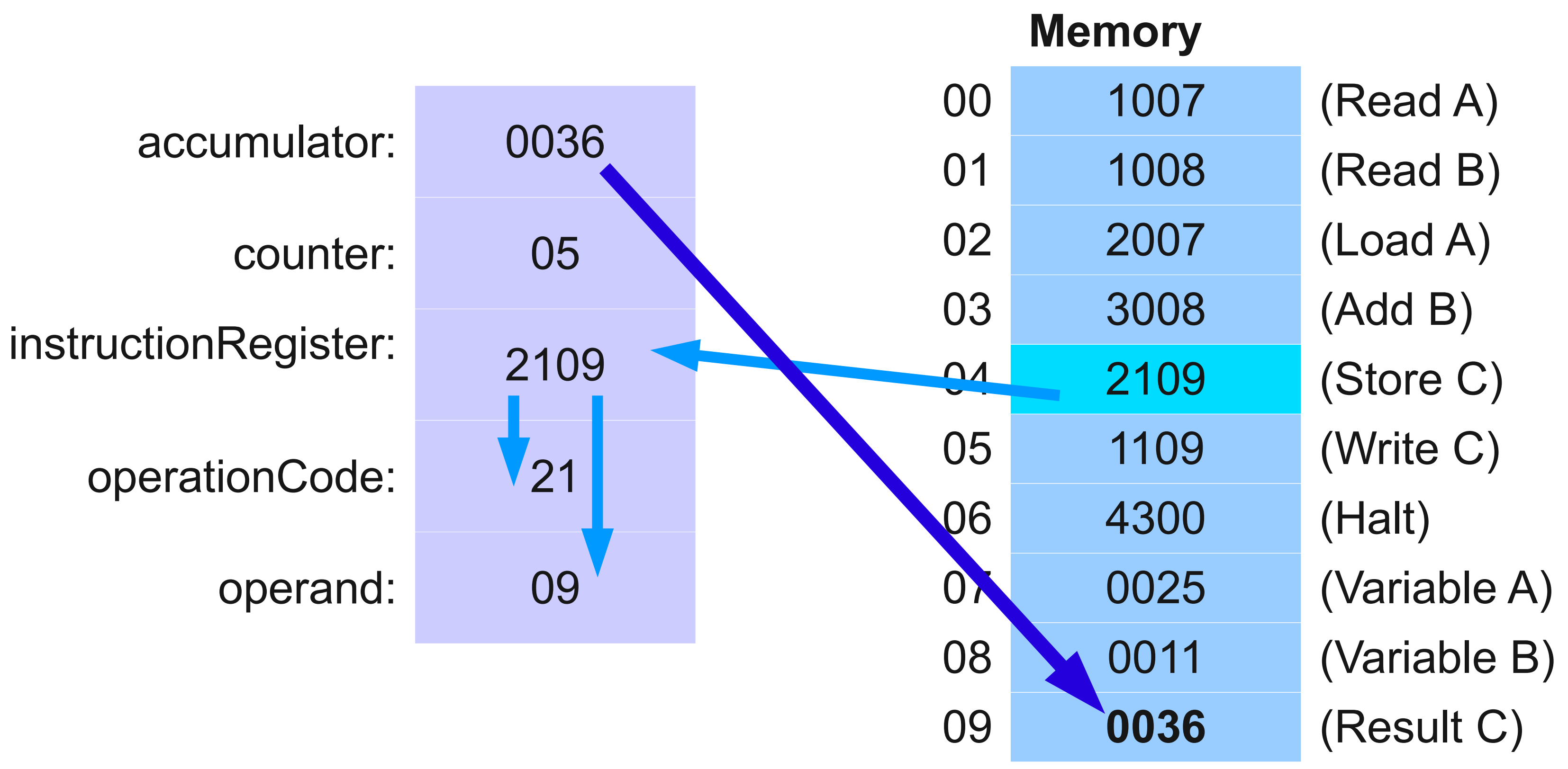
SML example 1
fetch+digest+execute



Memory

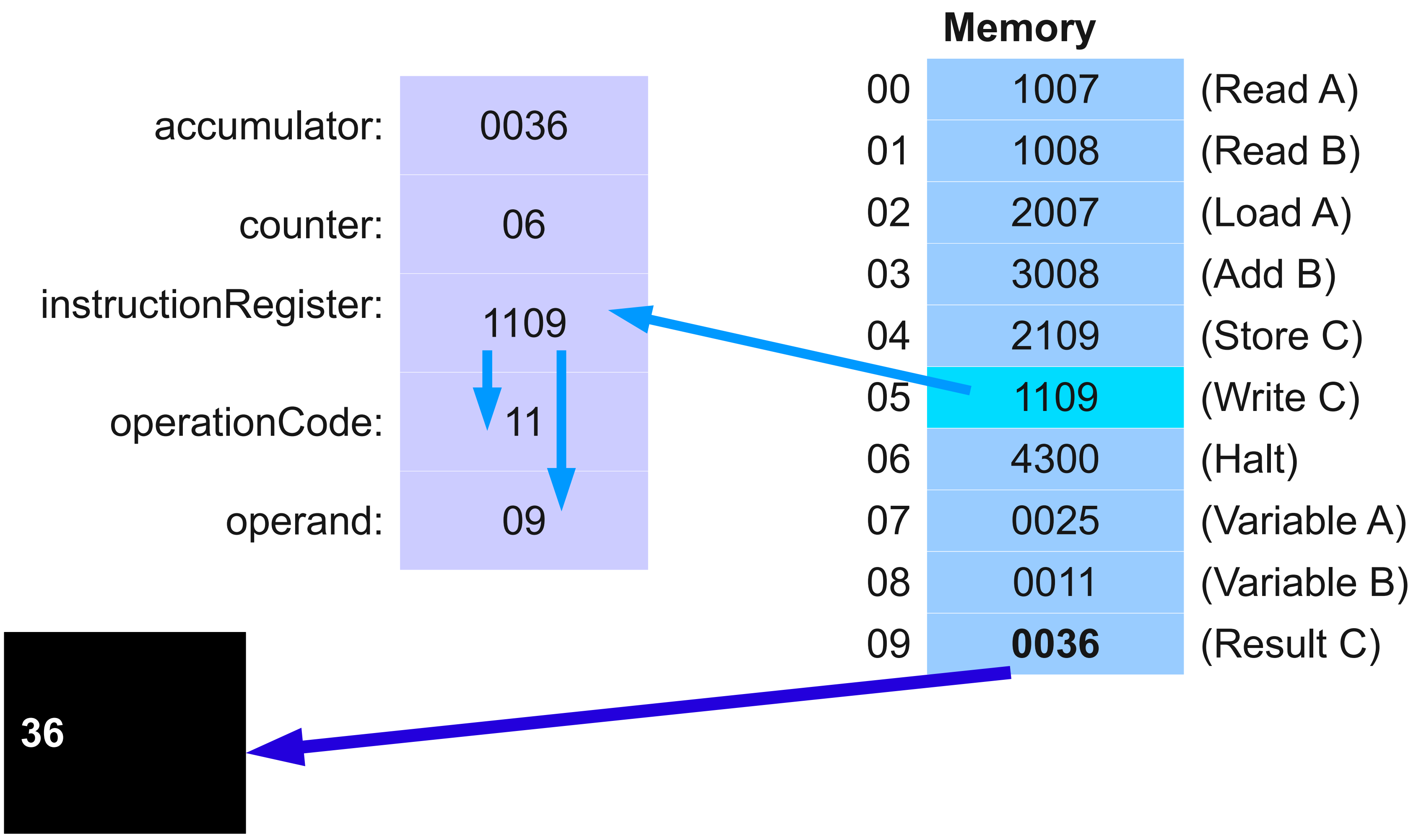
00	1007	(Read A)
01	1008	(Read B)
02	2007	(Load A)
03	3008	(Add B)
04	2109	(Store C)
05	1109	(Write C)
06	4300	(Halt)
07	0025	(Variable A)
08	0011	(Variable B)
09	0000	(Result C)

SML example 1
fetch+digest+execute



	Memory	
00	1007	(Read A)
01	1008	(Read B)
02	2007	(Load A)
03	3008	(Add B)
04	2109	(Store C)
05	1109	(Write C)
06	4300	(Halt)
07	0025	(Variable A)
08	0011	(Variable B)
09	**0036**	(Result C)

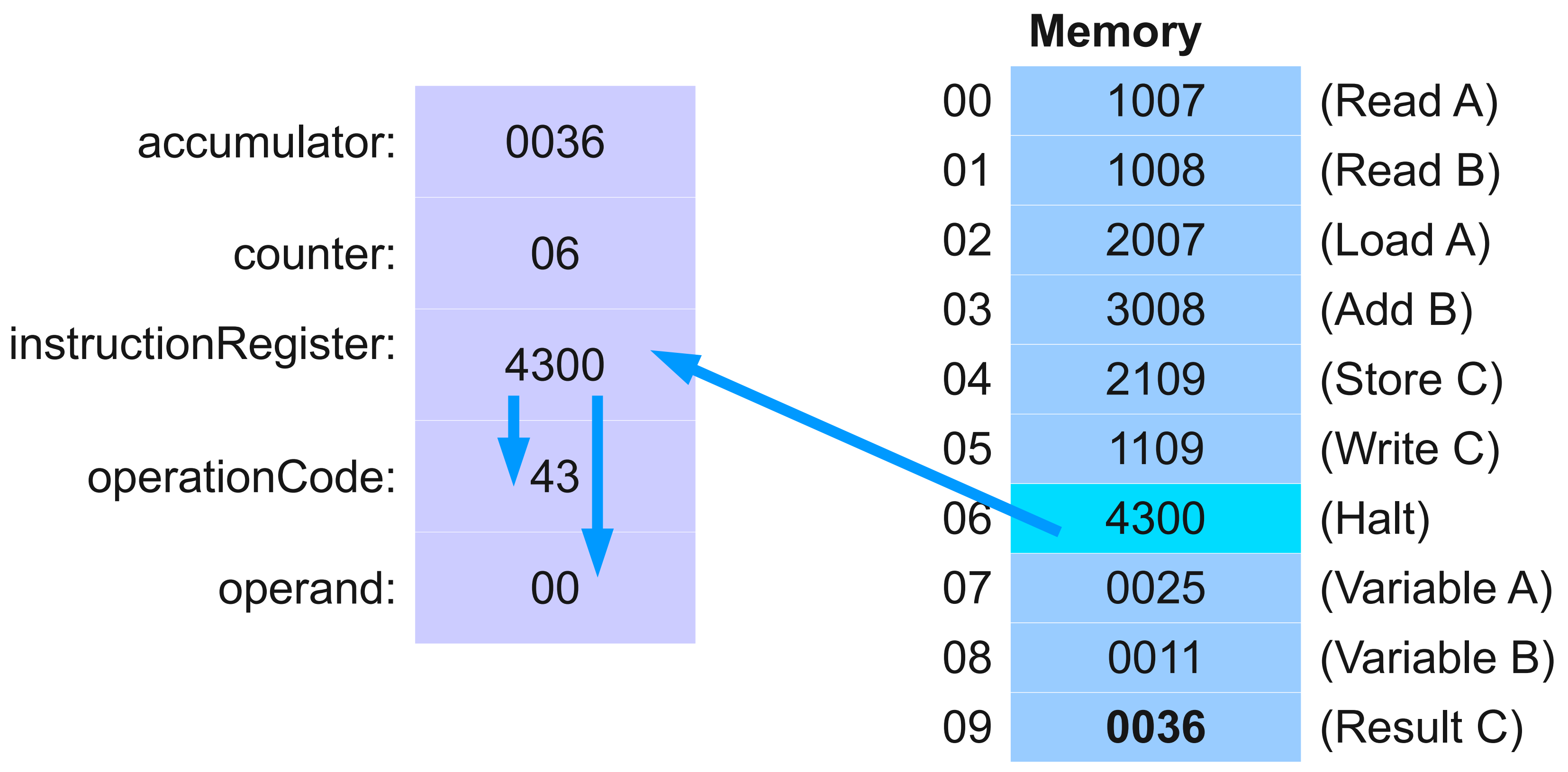
SML example 1
fetch+digest+execute



Memory

00	1007	(Read A)
01	1008	(Read B)
02	2007	(Load A)
03	3008	(Add B)
04	2109	(Store C)
05	1109	(Write C)
06	4300	(Halt)
07	0025	(Variable A)
08	0011	(Variable B)
09	**0036**	(Result C)

SML example 1
fetch+digest+execute



accumulator:	0036
counter:	06
instructionRegister:	4300
operationCode:	43
operand:	00

Memory

00	1007	(Read A)
01	1008	(Read B)
02	2007	(Load A)
03	3008	(Add B)
04	2109	(Store C)
05	1109	(Write C)
06	4300	(Halt)
07	0025	(Variable A)
08	0011	(Variable B)
09	**0036**	(Result C)

SML example 2 (what is it doing?)

accumulator:	0000
counter:	00
instructionRegister:	0000
operationCode:	00
operand:	00

	Memory	
00	1009	(Read A)
01	1010	(Read B)
02	2009	(Load A)
03	3110	(Subtract B)
04	4107	(Branch negative to 07)
05	1109	(Write A)
06	4300	(Halt)
07	1110	(Write B)
08	4300	(Halt)
09	0000	(Variable A)
10	0000	(Variable B)

Programs to write in SML

- Use a sentinel-controlled loop to read positive numbers and compute and print their sum. Terminate input when a negative number is entered.
- Use a counter-controlled loop to read seven numbers, some positive and some negative, and compute and print their average.
- Read a series of numbers, and determine and print the largest number. The first number read indicates how many numbers should be processed.

Programs to write in C++

- Write the simulator to run SML code as described in Chapter 8.
- Run the SML solutions of the previous problems in the simulator.
- Have a fun.