

# CS5000: Lecture 16

Vladimir Kulyukin

Department of Computer Science

Utah State University

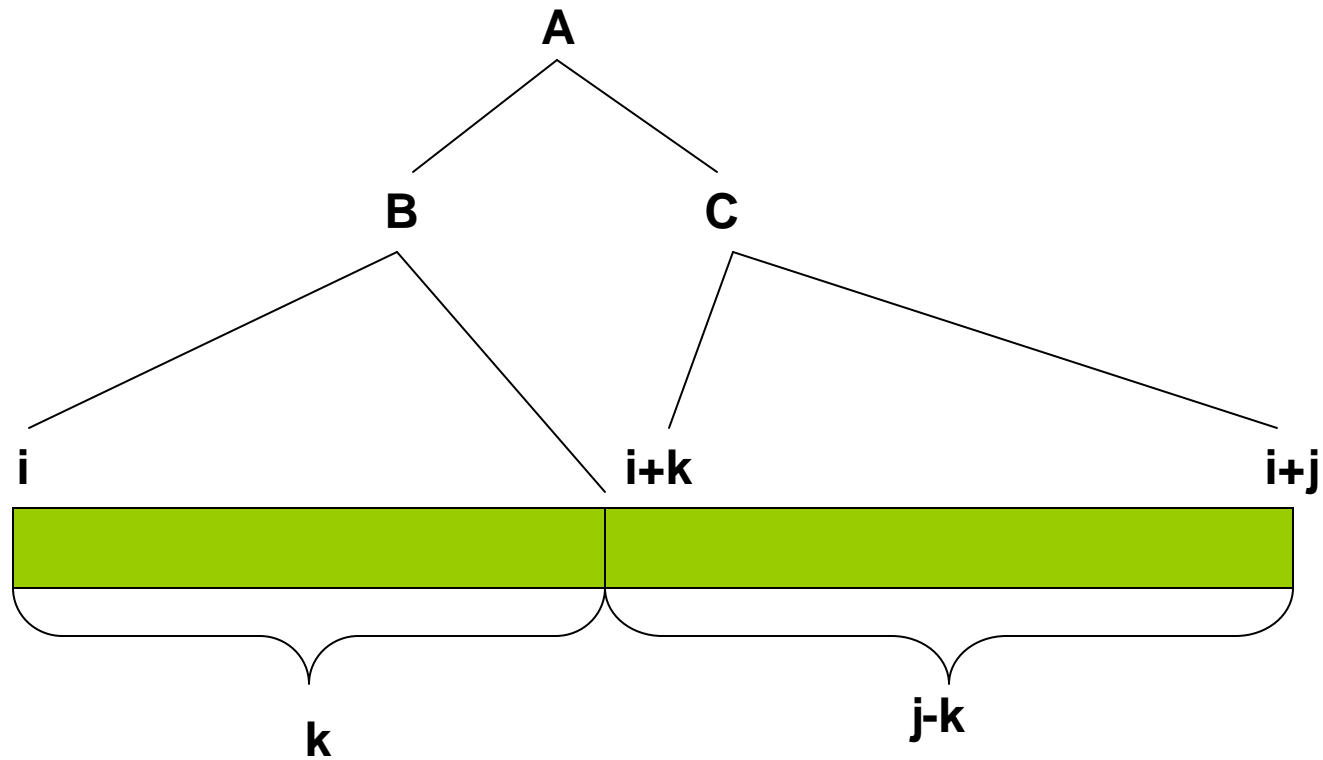
# Outline

- CYK Algorithm: Example
- Predicates and Quantifiers
- Programming language *L*

# CYK Algorithm: Review

- $V \rightarrow^* x_{i_1}$  if and only if  $V \rightarrow x_{i_1}$  is a production, since  $x_{i_1}$  is a string of length 1.
- For  $j > 1$ ,  $V \rightarrow^* x_{ij}$  if and only if there is some production  $V \rightarrow BC$  and some  $k$ ,  $1 \leq k < j$ , such that  $B \rightarrow^* x_{ik}$  and  $C \rightarrow^* x_{(i+k)(j-k)}$ .

# CYK Algorithm: Review



# CYK Algorithm: Review

- Since  $k$  and  $j-k$  are both less than  $j$ , we already know whether each of the last two derivations ( $B \rightarrow^* x_{ik}$  and  $C \rightarrow^* x_{(i+k)(j-k)}$ ) exists.
- When we reach  $j=n$ , we can determine if  $S \rightarrow^* x_{1n}$ .

# CYK Algorithm: Review

$V_{ij}$  is the set of variables  $A$  such that  $A \rightarrow^* x_{ij}$

for  $i=1$  to  $n$  {

$V_{i1} = \{V \mid V \rightarrow a \text{ is a production and the } i^{\text{th}} \text{ symbol of } x \text{ is } a\}$   
}

for  $j=2$  to  $n$  {

    for  $i=1$  to  $n-j+1$  {

$V_{ij} = \{\}$ ;

        for  $k=1$  to  $j-1$  {

$V_{ij} = V_{ij} + \{V \mid V \rightarrow BC \text{ is a production, } B \text{ is in } V_{ik} \text{ and } C \text{ is in } V_{(i+k)(j-k)}\}$ ;

        }

    }

}

# Example

$S \rightarrow AB|BC$

$A \rightarrow BA|a$

$B \rightarrow CC|b$

$C \rightarrow AB|a$

# Input: baaba

		i				
		1	2	3	4	5
j	1	B	A,C	A,C	B	A,C
	2	S, A	B	S,C	S,A	
	3	{	B	B		
	4	{	S,A,C			
	5	S,A,C				

# Computing $V_{24}$

- $V_{24}$ ,  $i=2$ ,  $j=4$ .
- $k$  goes from 1 to  $j-1=3$ .
- For  $k=1$ , we look for  $\{A \mid A \rightarrow BC, B \text{ in } V_{21}, C \text{ in } V_{33}\}$ .
- Entry  $V_{21}$  gives us  $A, C$ .
- Entry  $V_{33}$  gives us  $B$ .
- The right-hand side possibilities are the Cartesian product of  $V_{21}$  and  $V_{33}$ :  $AB, CB$ .
- $S \rightarrow AB$  and  $C \rightarrow AB$  are productions, so  $\{S, C\}$  are added to  $V_{24}$ .

# Computing $V_{24}$

- $V_{24}$ ,  $i=2$ ,  $j=4$ .
- For  $k=2$ , we compute  $\{A \mid A \rightarrow BC, B \text{ in } V_{22}, C \text{ in } V_{42}\}$ .
- Entry  $V_{22}$  gives us  $B$ .
- Entry  $V_{42}$  gives us  $S, A$ .
- The right-hand side possibilities are:  $BS, BA$ .
- $A \rightarrow BA$  is a production, so  $A$  is added to  $V_{24}$ . So now  $V_{24} = \{S, A, C\}$ .

# Computing $V_{24}$

- $V_{24}$ ,  $i=2$ ,  $j=4$ .
- For  $k=3$ , we compute  $\{A \mid A \rightarrow BC, B \text{ in } V_{23}, C \text{ in } V_{51}\}$ .
- Entry  $V_{23}$  gives us B.
- Entry  $V_{51}$  gives us A, C.
- The right-hand side possibilities are: BA, BC.
- $A \rightarrow BA$ ,  $S \rightarrow BC$  are productions, so A and S should be added to  $V_{24}$ , but they are already there. Thus,  $V_{24} = \{S, A, C\}$ .

# Predicates

A predicate  $P$  is a total Boolean - valued function on  $S$  such that for each  $a \in S$  either  $P(a) = T$  or  $P(a) = F$

# Characteristic Functions

Let  $R$  be a set.

$$P(x) = \begin{cases} 1 & \text{if } x \in R \\ 0 & \text{if } x \notin R \end{cases}$$

Then

$$R = \{x \mid P(x) = 1\}$$

$P(x)$  is the characteristic function of  $R$

# Bounded Existential Quantifier

Suppose that  $P(y, x_1, \dots, x_n)$  is a predicate.

$$Q(y, x_1, \dots, x_n) \Leftrightarrow P(0, x_1, \dots, x_n) \vee \\ P(1, x_1, \dots, x_n) \vee \dots \vee P(y, x_1, \dots, x_n)$$

$$Q \Leftrightarrow (\exists t)_{\leq y} P(t, x_1, \dots, x_n)$$

# (Unbounded) Existential Quantifier

$$Q(x_1, \dots, x_n) \Leftrightarrow (\exists t)P(t, x_1, \dots, x_n), t \in N.$$

# Bounded Universal Quantifier

Suppose that  $P(t, x_1, \dots, x_n)$  is a predicate.

$$\begin{aligned} (\forall t)_{\leq y} P(t, x_1, \dots, x_n) &\Leftrightarrow \\ P(0, x_1, \dots, x_n) &\& P(1, x_1, \dots, x_n) \& \\ P(2, x_1, \dots, x_n) &\& \dots \& P(y, x_1, \dots, x_n). \end{aligned}$$

# Universal Quantifier

$(\forall t)P(t, x_1, \dots, x_n)$  is true iff

$P(t, x_1, \dots, x_n)$  is true for all  $t \in N$ .

# De Morgan Identities

$$1) \neg(\exists t)_{\leq y} P(t, x_1, \dots, x_n) \Leftrightarrow (\forall t)_{\leq y} \neg P(t, x_1, \dots, x_n).$$

$$2) \neg(\forall t) P(t, x_1, \dots, x_n) \Leftrightarrow (\exists t) \neg P(t, x_1, \dots, x_n).$$

# Quantification Examples

$$(\exists y)(x + y = 4) \Leftrightarrow x \leq 4$$

$$(\exists y)(x + y = 4) \Leftrightarrow (\exists y)_{\leq 4}(x + y = 4)$$

$$(\forall y)(xy = 0) \Leftrightarrow x = 0$$

$$(\exists y)_{\leq z}(x + y = 4) \Leftrightarrow (x + z \geq 4 \ \& \ x \leq 4)$$

Programming Language *L*

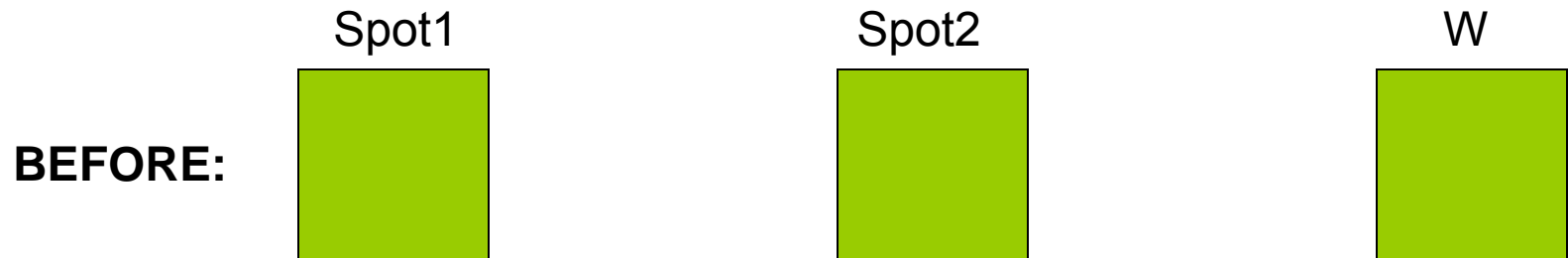
# 16f84a Assembler

<b>Spot1</b>	<b>equ</b>	<b>H'30'</b>
<b>Spot2</b>	<b>equ</b>	<b>H'31'</b>
	<b>movlw</b>	<b>D'5'</b>
	<b>movwf</b>	<b>Spot1</b>
	<b>movlw</b>	<b>D'2'</b>
	<b>addwf</b>	<b>Spot1,W</b>
	<b>movwf</b>	<b>Spot2</b>
	<b>movlw</b>	<b>D'3'</b>
	<b>subwf</b>	<b>Spot2,W</b>
	<b>movwf</b>	<b>Spot1</b>
	<b>clrw</b>	
	<b>clrf</b>	<b>Spot1</b>
	<b>clrf</b>	<b>Spot2</b>
	<b>nop</b>	
	<b>end</b>	

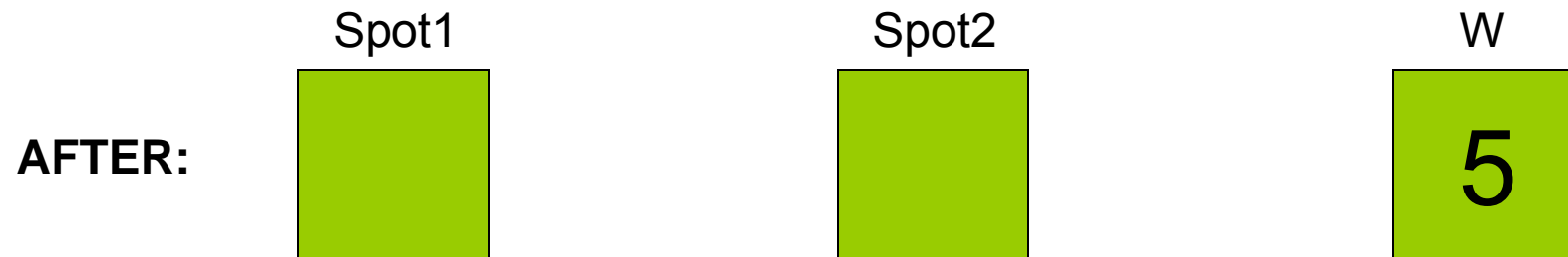
# Example

- In the next few slides, we will take a look at how instructions of a typical assembly language are executed.
- A typical assembly language has a fairly small number of instructions, e.g., 10, 20, 30.
- A program in a higher level language that wants to run on the same chip is compiled, i.e. translated, into these primitive instructions.

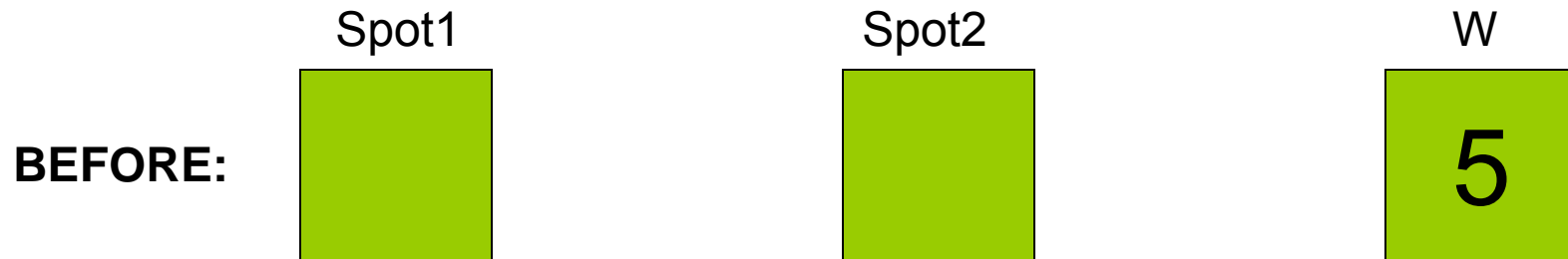
# Example: 16f84a Assembler



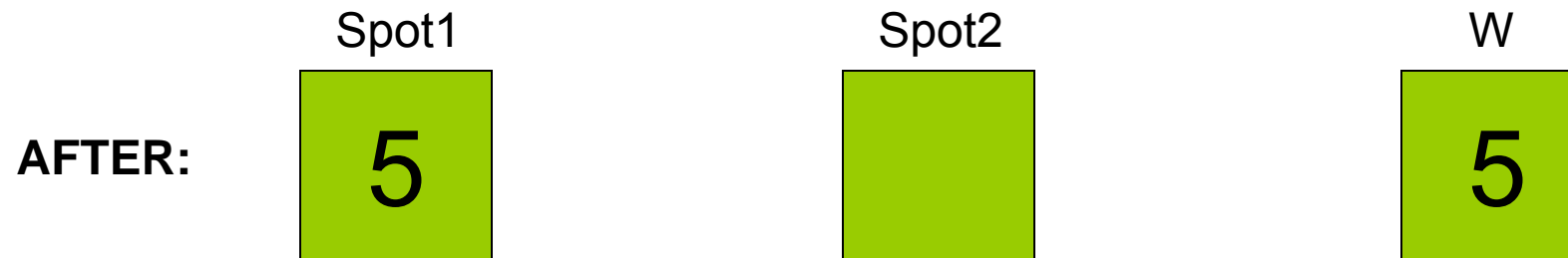
**movlw D'5'**



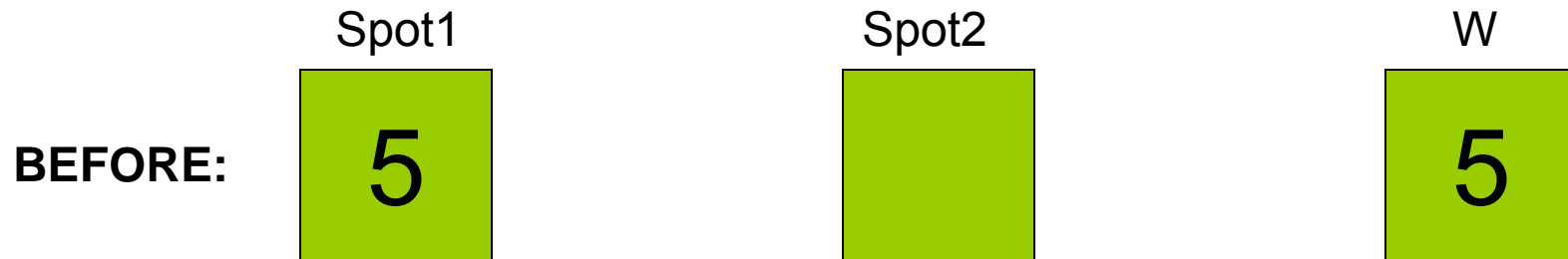
# Example: 16f84a Assembler



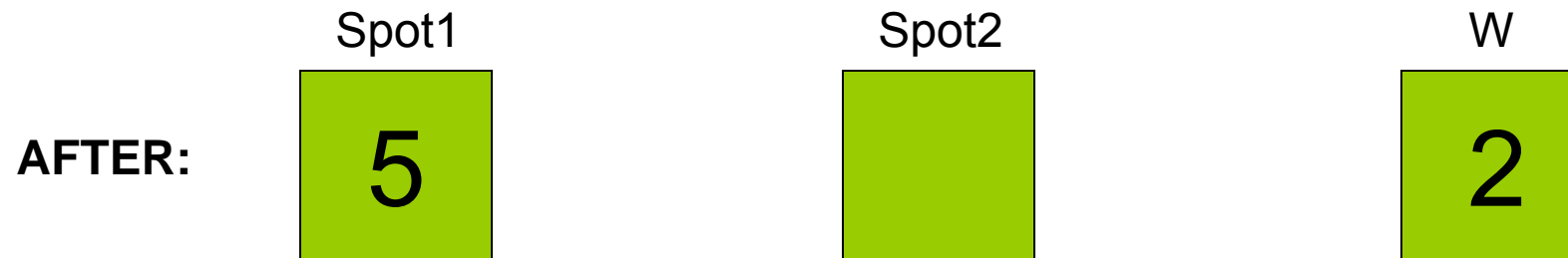
**movwf Spot1**



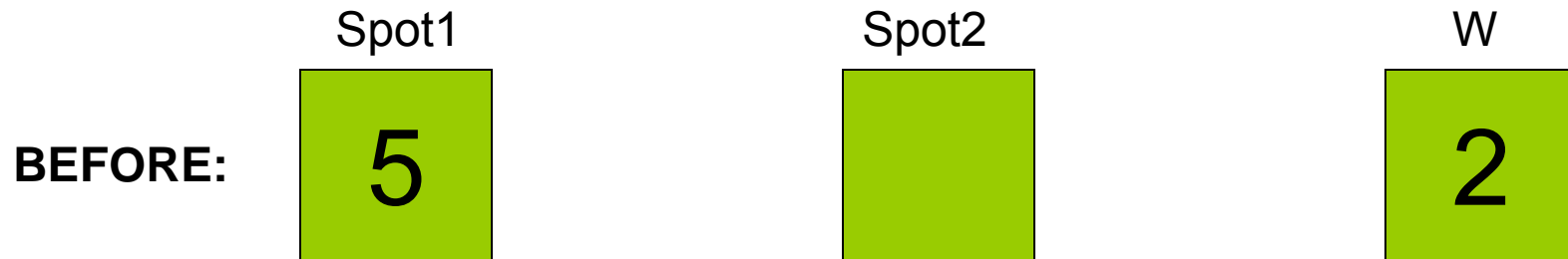
# Example: 16f84a Assembler



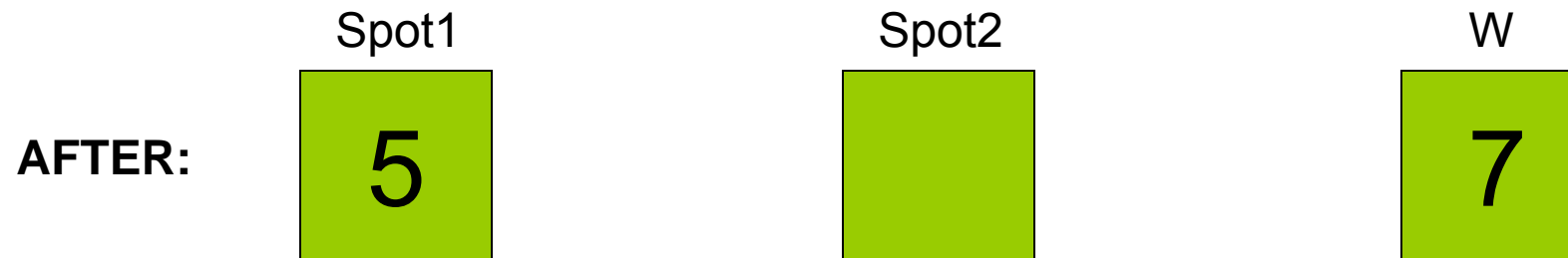
**movlw D'2'**



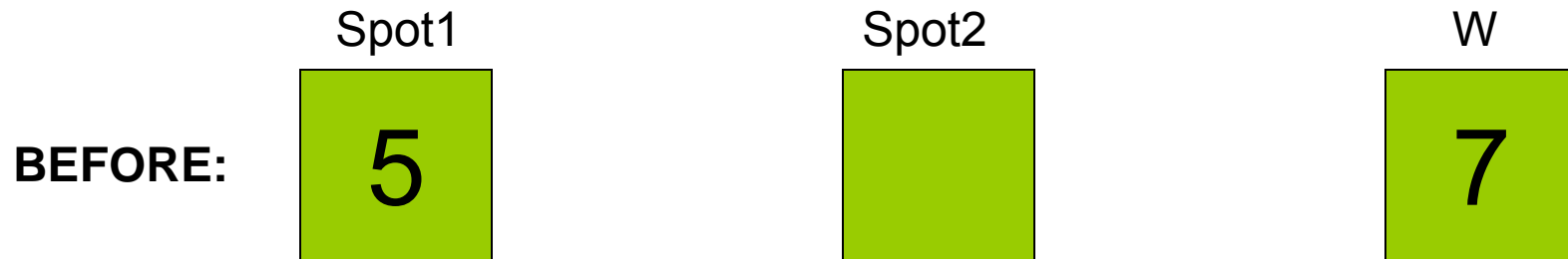
# Example: 16f84a Assembler



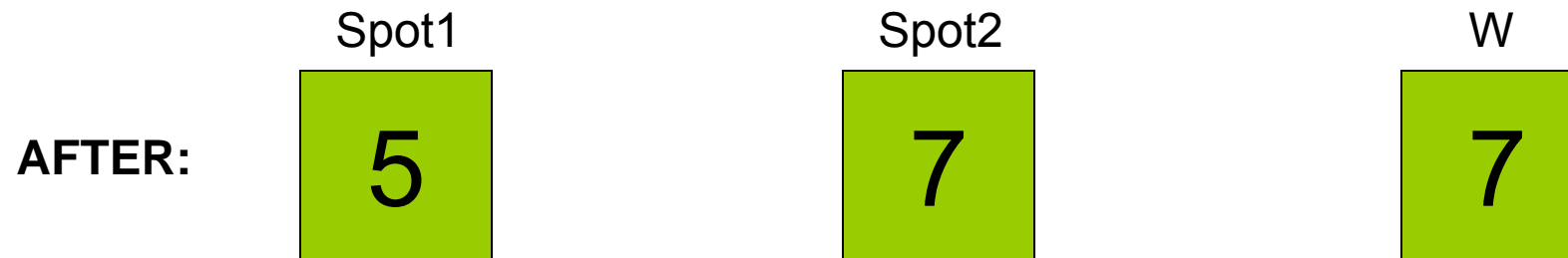
**addwf Spot1,W**



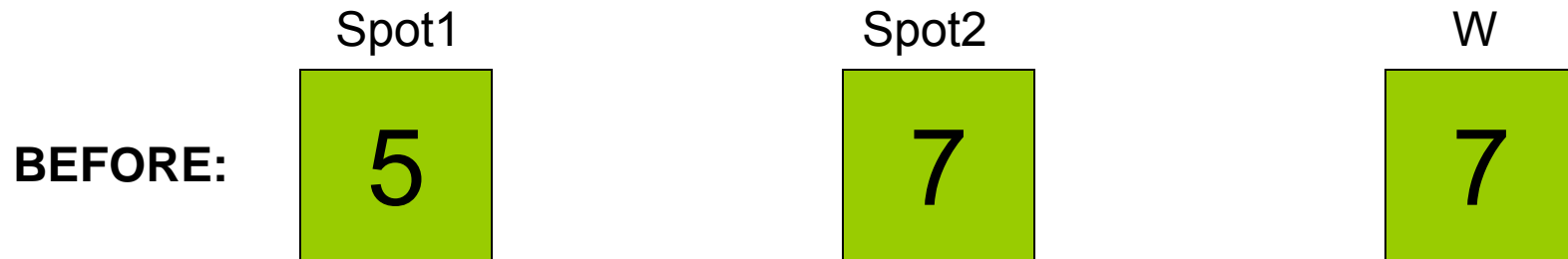
# Example: 16f84a Assembler



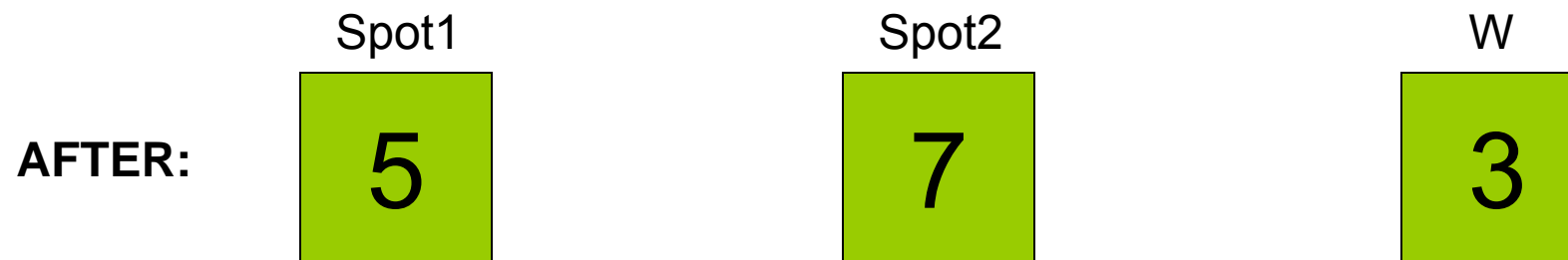
**movwf Spot2**



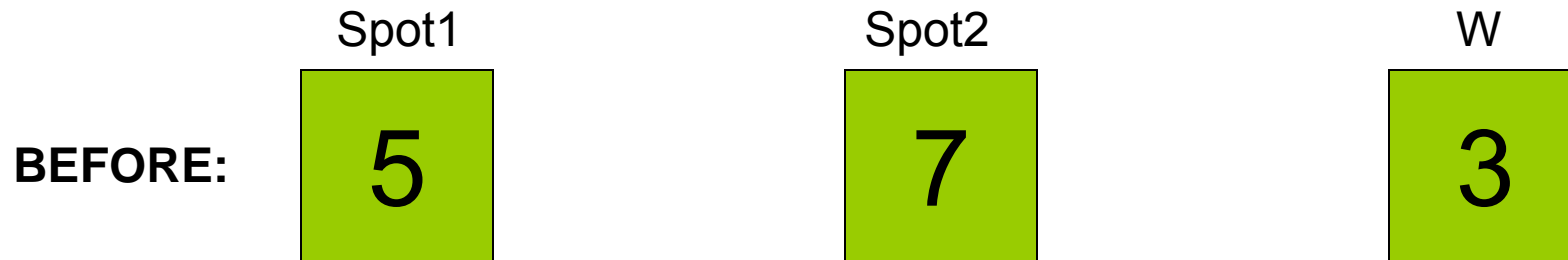
# Example: 16f84a Assembler



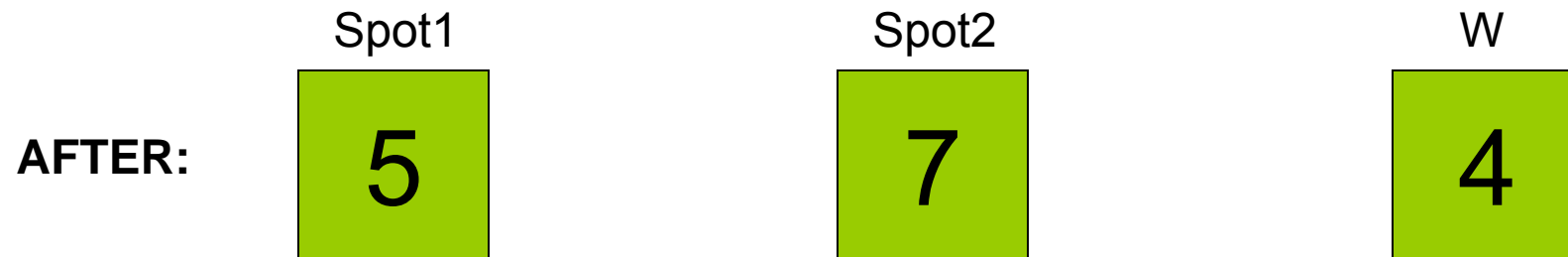
**movlw D'3'**



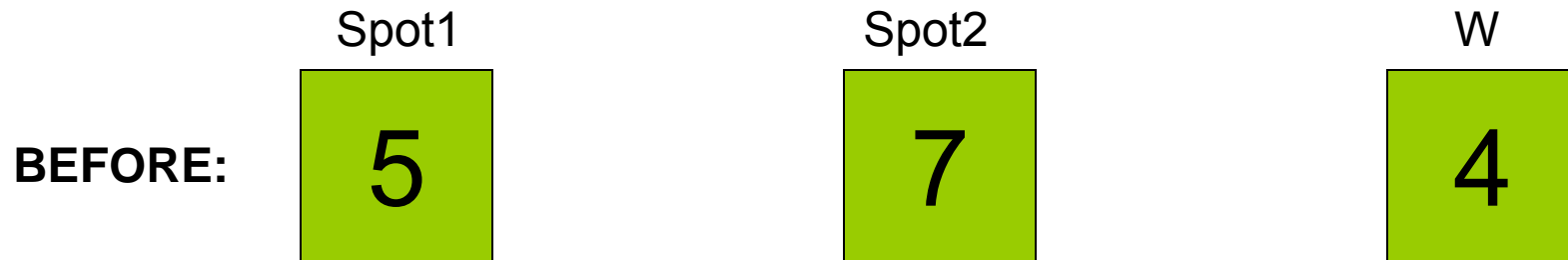
# Example: 16f84a Assembler



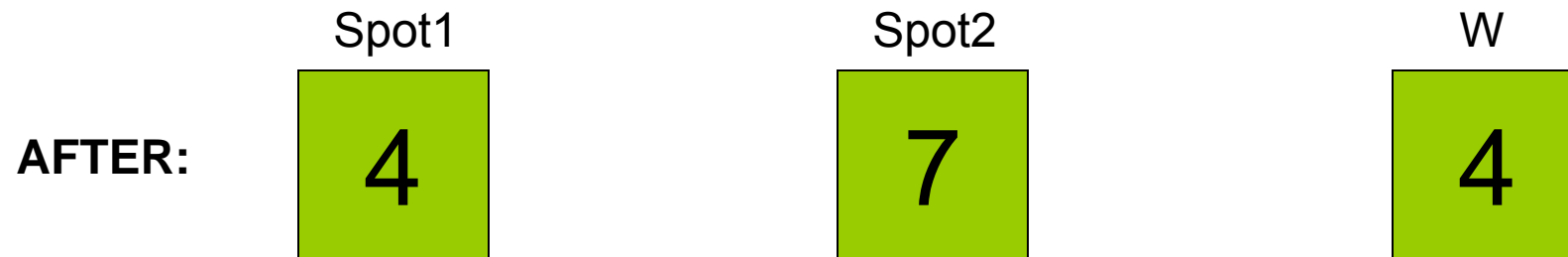
**subwf Spot2,W**



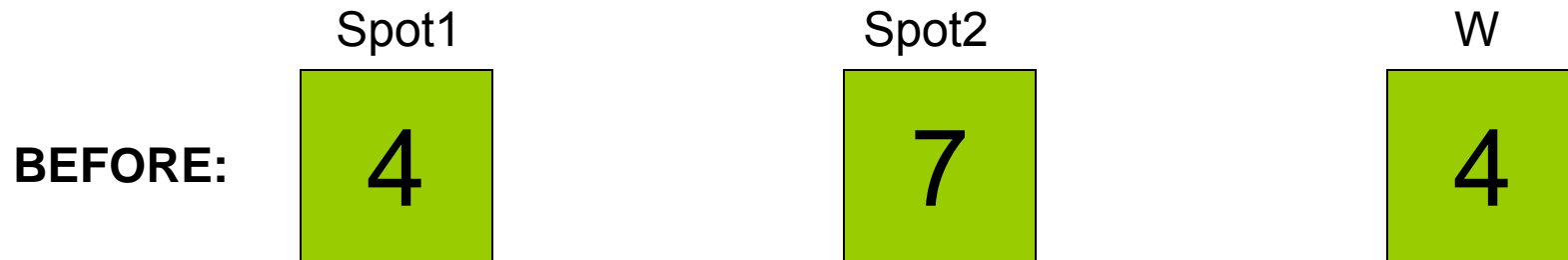
# Example: 16f84a Assembler



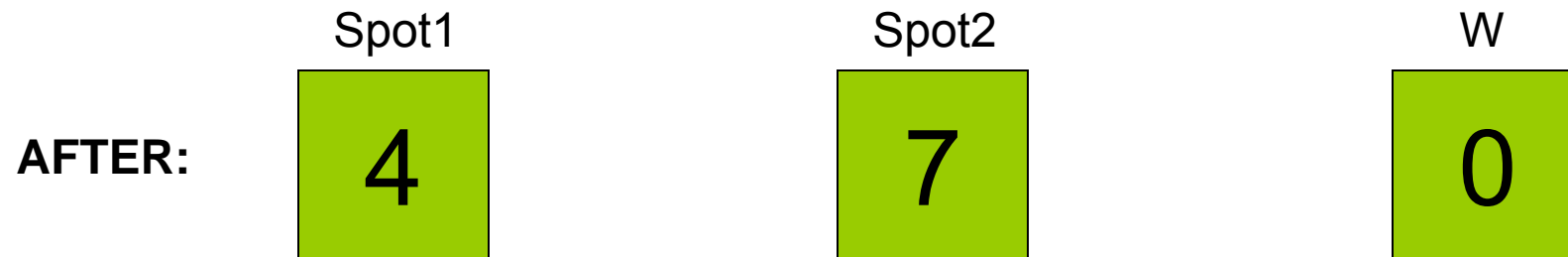
**movwf Spot1**



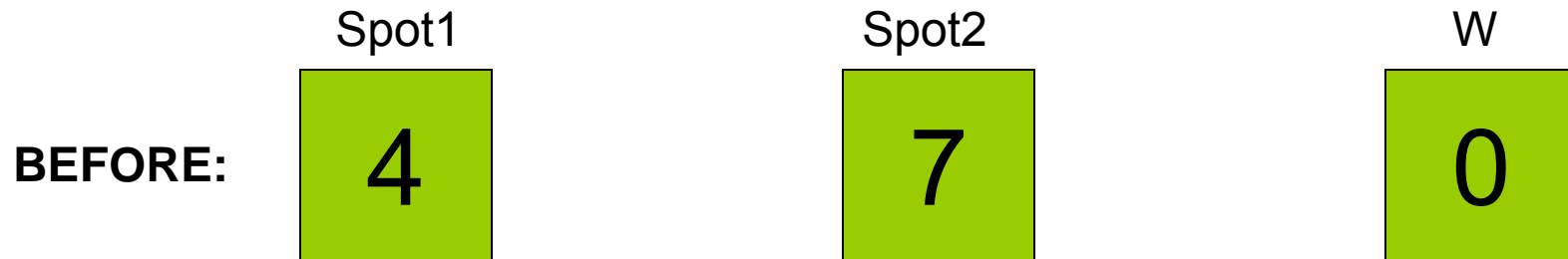
# Example: 16f84a Assembler



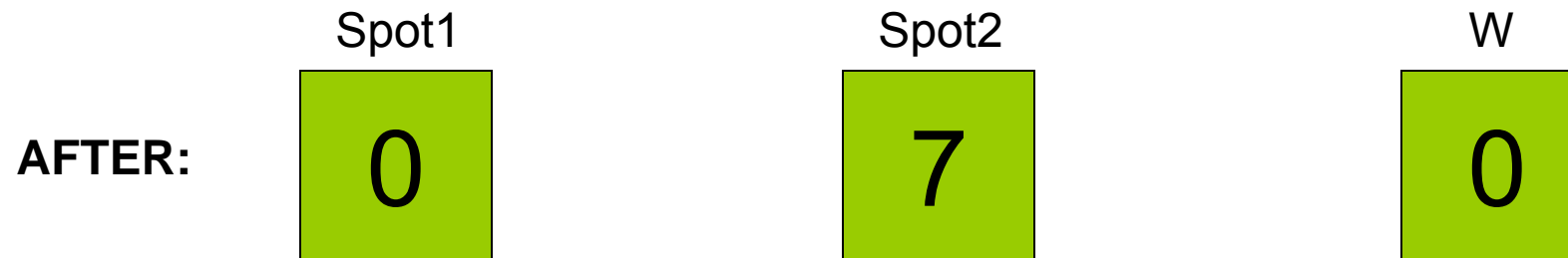
**clrw**



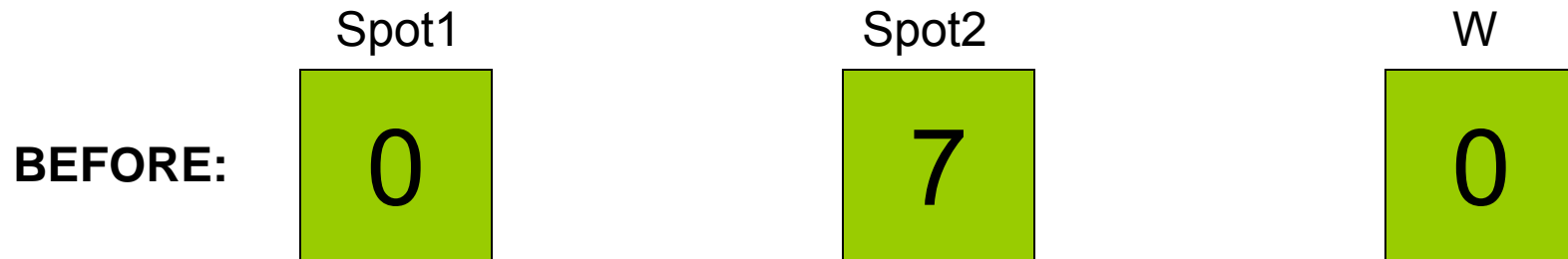
# Example: 16f84a Assembler



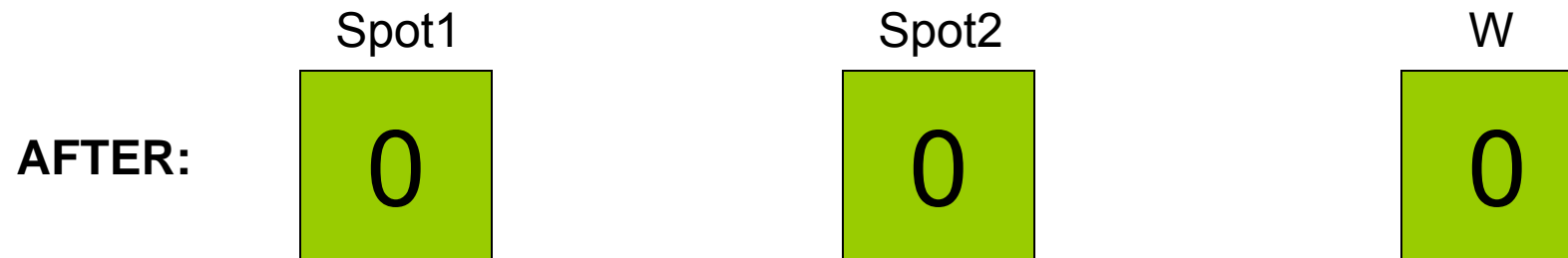
**clrf Spot1**



# Example: 16f84a Assembler



**clrf Spot2**



# Programming Language *L*

- In Chapter 2, Davis, Sigal, and Weyuker develop a theoretical programming language *L*.
- *L* is a simple theoretical prototype of an assembly language.

# Programming Language $L$ (Ch. 2)

Input variables :  $X_1, X_2, X_3, \dots$

Local variables :  $Z_1, Z_2, Z_3, \dots$

Output variable :  $Y$

Labels :  $A_1, B_1, C_1, D_1, E_1, A_2, \dots$

$X = X_1$

$Z = Z_1$

$A = A_1$

# Programming Language $L$

- The values of the variables are numbers (natural numbers).
- There is no upper limit on the values of these variables.
- The output variable ( $Y$ ) and the local variables ( $Z$ 's) have the initial values of 0.
- $L$  is a purely theoretical construct.

# Recommended Reading

- Section 2.1.