

4.9 A program that contains a loop

In principle, programs are executed so that those instructions which are at the beginning of a program are executed first, and the program execution proceeds instruction by instruction towards the end of the program. By using jump instructions it is possible to violate this principle. A jump instruction can cause a jump from the end of a program to the beginning of the program. We say that jump instructions can create a loop in a program. Loops are sequences of instructions which are repeated many times during the execution of a program. In program **hello_loop.iml** in Figure 4-8c we already saw a loop which prints the characters of the text "Hello!". **abcde.iml** in Figure 4-10a is another program in which a loop is used to print characters.

Only the textual IML instructions are shown for the program in Figure 4-10a. For you to learn how the program operates, it might be useful to write the numerical instructions by hand besides the textual instructions. You could thus compile the program by hand. Figure 4-10b shows what happens on the screen and in the registers of the imaginary processor when the program is being executed. Figure 4-10c can help you to hand-compile the program. You can also exploit tables 4-1 and 4-2.

A key idea in program **abcde.iml** is that it prints letters A, B, C, D, and E because the ASCII codes of these letters are sequential numbers 41H, 42H, 43H, 44H, and 45H. First the program loads the ASCII code of letter A, 41H, to register B. When register B is incremented inside the loop of the program, it first increments to 42H, then to 43H, then to 44H, then to 45H, and finally to 46H. Value 46H is never printed. The other general purpose register of the processor, register A, is used to count how many letters are left to be printed to the screen. Register A is 5 at the beginning. As it is decremented inside the loop, it first decrements to 4, then to 3, then to 2, then to 1, and finally to 0. When register A reaches value 0, a jump to address name **end_of_program** takes place, and the program terminates.

```
// abcde.iml    (c) 1998-2000 Kari Laitinen

// This program prints the letters ABCDE to the screen.
// Register A is used to count how many characters have
// been printed. The character code being output to the
// screen is held in register B. The first code is 'A'
// which means 41H, the ASCII code of letter A.
// As the ASCII code in register B is incremented after
// each printing, a different letter will be printed
// each time.

beginning_of_program:
    load_register_a_with_value    5
    load_register_b_with_value    'A'

print_next_letter:
    output_byte_from_register_b
    increment_register_b
    decrement_register_a
    jump_if_register_a_zero        end_of_program
    jump_to_address                print_next_letter

end_of_program:
    stop_processing
```

Two jump instructions are usually needed to construct a loop. Here the first jump instruction terminates the loop at the right moment. The second jump instruction continues the loop.

Figure 4-10a. A program that prints letters ABCDE in a loop.

Here the first jump in the program takes place. Instruction 41H, "jump to address", is executed. The execution modifies PROGRAM POINTER (PP) so that the address of the next instruction is 04H.

Register A (RA) reaches value 0 at the end. At that moment register B has already value 46H, the ASCII code of letter F. That letter is, though, never printed because the program terminates when register A has value 0.

Status	PP	IC	IO	RA	RB	MP	SP	FE	FF	Screen contents
RESET	00	ff	ff	ff	ff	ff	ff	ff	ff	
FETCH	01	15	ff	ff	ff	ff	ff	ff	ff	
FETCH	02	15	05	ff	ff	ff	ff	ff	ff	
EXECUTE	02	15	05	05	ff	ff	ff	ff	ff	
FETCH	03	17	05	05	ff	ff	ff	ff	ff	
FETCH	04	17	41	05	ff	ff	ff	ff	ff	
EXECUTE	04	17	41	05	41	ff	ff	ff	ff	
FETCH	05	94	41	05	41	ff	ff	ff	ff	
EXECUTE	05	94	41	05	41	ff	ff	ff	ff	A
FETCH	06	18	41	05	41	ff	ff	ff	ff	A
EXECUTE	06	18	41	05	42	ff	ff	ff	ff	A
FETCH	07	1a	41	05	42	ff	ff	ff	ff	A
EXECUTE	07	1a	41	04	42	ff	ff	ff	ff	A
FETCH	08	45	41	04	42	ff	ff	ff	ff	A
FETCH	09	45	0b	04	42	ff	ff	ff	ff	A
EXECUTE	09	45	0b	04	42	ff	ff	ff	ff	A
FETCH	0a	41	0b	04	42	ff	ff	ff	ff	A
FETCH	0b	41	04	04	42	ff	ff	ff	ff	A
EXECUTE	04	41	04	04	42	ff	ff	ff	ff	A
FETCH	05	94	04	04	42	ff	ff	ff	ff	A
EXECUTE	05	94	04	04	42	ff	ff	ff	ff	AB
FETCH	06	18	04	04	42	ff	ff	ff	ff	AB
EXECUTE	06	18	04	04	43	ff	ff	ff	ff	AB
FETCH	07	1a	04	04	43	ff	ff	ff	ff	AB
EXECUTE	07	1a	04	03	43	ff	ff	ff	ff	AB
FETCH	08	45	04	03	43	ff	ff	ff	ff	AB
FETCH	09	45	0b	03	43	ff	ff	ff	ff	AB
EXECUTE	09	45	0b	03	43	ff	ff	ff	ff	AB
FETCH	0a	41	0b	03	43	ff	ff	ff	ff	AB
FETCH	0b	41	04	03	43	ff	ff	ff	ff	AB
EXECUTE	04	41	04	03	43	ff	ff	ff	ff	AB
FETCH	05	94	04	03	43	ff	ff	ff	ff	AB
EXECUTE	05	94	04	03	43	ff	ff	ff	ff	ABC
FETCH	06	18	04	03	43	ff	ff	ff	ff	ABC
EXECUTE	06	18	04	03	44	ff	ff	ff	ff	ABC
FETCH	07	1a	04	03	44	ff	ff	ff	ff	ABC

Because of space limitation, 16 lines have been left out here.

FETCH	0a	41	0b	01	45	ff	ff	ff	ff	ABCD
FETCH	0b	41	04	01	45	ff	ff	ff	ff	ABCD
EXECUTE	04	41	04	01	45	ff	ff	ff	ff	ABCD
FETCH	05	94	04	01	45	ff	ff	ff	ff	ABCD
EXECUTE	05	94	04	01	45	ff	ff	ff	ff	ABCDE
FETCH	06	18	04	01	45	ff	ff	ff	ff	ABCDE
EXECUTE	06	18	04	01	46	ff	ff	ff	ff	ABCDE
FETCH	07	1a	04	01	46	ff	ff	ff	ff	ABCDE
EXECUTE	07	1a	04	00	46	ff	ff	ff	ff	ABCDE
FETCH	08	45	04	00	46	ff	ff	ff	ff	ABCDE
FETCH	09	45	0b	00	46	ff	ff	ff	ff	ABCDE
EXECUTE	0b	45	0b	00	46	ff	ff	ff	ff	ABCDE
FETCH	0c	b2	0b	00	46	ff	ff	ff	ff	ABCDE
STOP	0c	b2	0b	00	46	ff	ff	ff	ff	ABCDE

Figure 4-10b. Step-by-step execution of the program in Figure 4-10a.