1. Referring back to Question 3 of Section 2.3, if the machine used the pipeline technique discussed in the text, what will be in “the pipe” when the instruction at address 0xAA is executed? Under what conditions would pipelining not prove beneficial at this point in the program?

Suggested approach:

In question 3 of Section 2.3, we see the following:

Suppose the Vole memory cells at addresses 0xA4 to 0xB1 contain the bit patterns given in the following table:

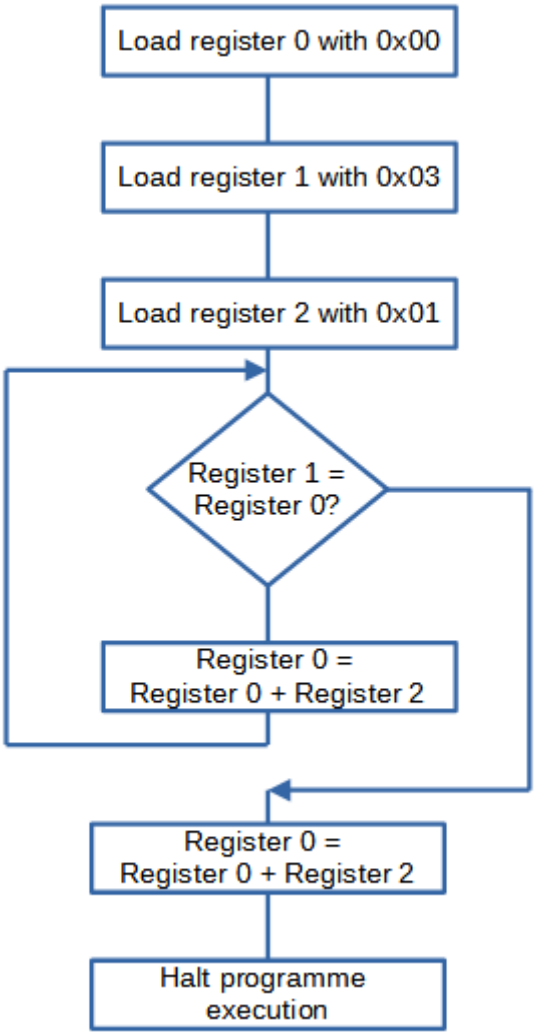
Address	Contents
0xA4	0x20
0xA5	0x00
0xA6	0x21
0xA7	0x03
0xA8	0x22
0xA9	0x01
0xAA	0xB1
0xAB	0xB0
0xAC	0x50
0xAD	0x02
0xAE	0xB0
0xAF	0xAA
0xB0	0xC0
0xB1	0x00

Based on the definition of the instructions (see Appendix C), analyse and number the instructions seen in the above table:

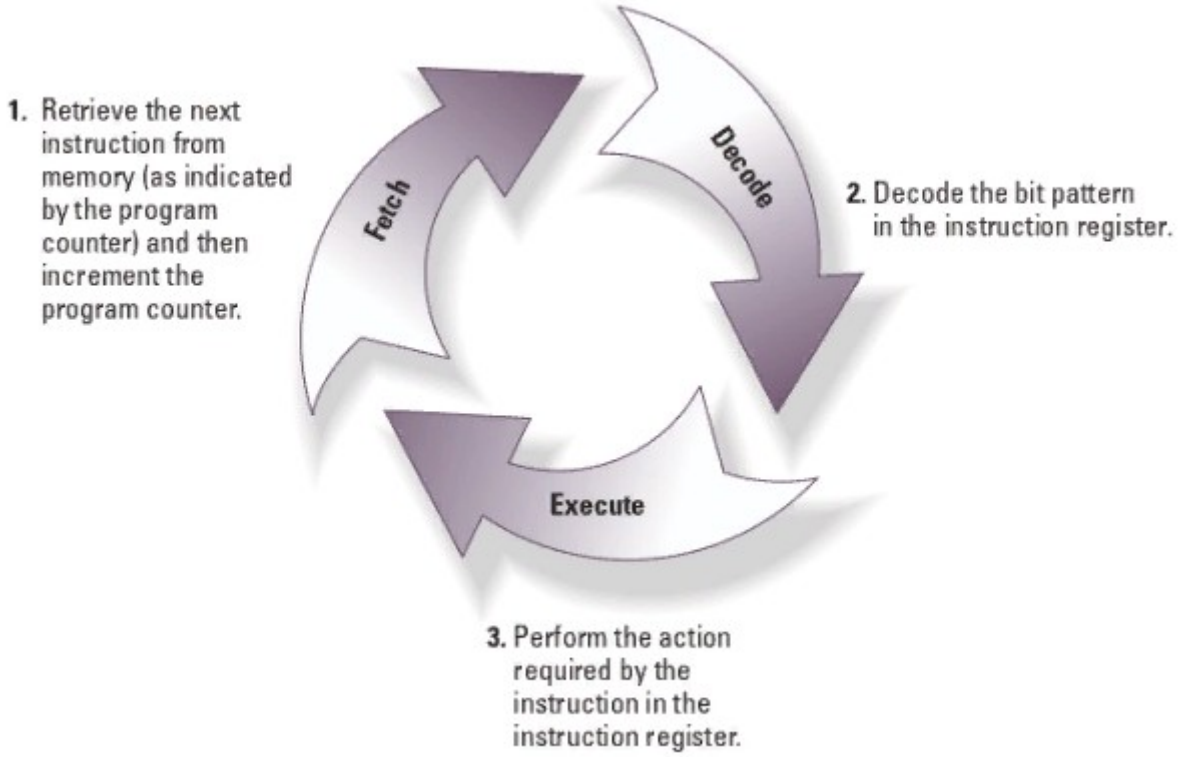
Address	Contents	Instruction number	Meaning
0xA4	0x20	I1	**LOAD** the register R with the bit pattern XY. 0x2000: load the value 0x00 to register 0.
0xA5	0x00		
0xA6	0x21	I2	**LOAD** the register R with the bit pattern XY. 0x2103: load the value 0x03 to register 1.
0xA7	0x03		
0xA8	0x22	I3	**LOAD** the register R with the bit pattern XY. 0x2201: load the value 0x01 to register 2.
0xA9	0x01		
0xAA	0xB1	I4	**JUMP** to the instruction located in the memory cell at address XY if the bit pattern in register R is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution. (The jump is implemented by copying XY into the program counter during the execute phase.) 0xB1B0: compares the contents of register 0x1 (in the first loop execution: 0x03) with the contents of register 0x0 (in the first loop execution: 0x00). If the two are not equal, normal execution is continued. If the two are equal, the PC is be loaded with 0xB0.
0xAB	0xB0		
0xAC	0x50	I5	**ADD** the bit patterns in registers S and T as though they were two's complement representations and leave the result in register R. 0x5002: add the content of register 0x2 to that of register 0x0 and place the sum in register 0x0
0xAD	0x02		
0xAE	0xB0	I6	**JUMP** 0xB0AA: compares the contents of register 0x0 with the contents of register 0x0. If they are equal, the PC would be loaded with 0xAA. This is the implementation of unconditional branch. Since the language does not include an unconditional branch instruction, the alternative option is to compare the content of register 0x0 to the content of itself in order to force the branch.
0xAF	0xAA		
0xB0	0xC0	I7	**HALT** instruction. 0xC000: programme execution is stopped.
0xB1	0x00		

Practically, 1 is added to the content of register 0 until it becomes equal to 3. When that occurs, programme execution is halted.

If we were to visualise the workflow of the programme, it would be:



In a previous section of the book, we see that the proposed pipeline consists of:



Setting up a table to show the pipeline:

Cycle Nr	Fetch	Decode	Execute	Decription
1	I1			
2	I2	I1		
3	I3	I2	I1	Register 0 = 0x00
4	I4	I3	I2	Register 1 = 0x03
5	I5	I4	I3	Register 2 = 0x01
6	I6	I5	I4	Jump to address 0xB0 if Register 1 = Register 0
7	I7	I6	I5	Register 0 = Register 0 + Register 2
8	No fetch	I7	I6	Jump to address 0xAA if Register 0 = Register 0
9	I4			
10	I5	I4		
11	I6	I5	I4	Jump to address 0xB0 if Register 1 = Register 0
12	I7	I6	I5	Register 0 = Register 0 + Register 2
13		I7	I6	Jump to address 0xAA if Register 0 = Register 0
14	I4			
The loop is repeated until Register 1 = Register 0:				
N_{end} - 3	I6	I5	I4	Jump to address 0xB0 if Register 1 = Register 0
N_{end} - 2	I7			
N_{end} - 1		I7		
N_{end}			I7	Programme execution is halted

Therefore, pipelining is not beneficial when the branching condition is true and the instruction at the target address must be fetched, decoded and executed.