```
# loop.s -- example for branch prediction in hardware.

        .text
main:
        li $s1, 1               # 1
outer:
        li $s2, 1               # 2     ###### begin outer loop
inner:
        li $v0, 1               # 3     #       ###### begin inner loop
        move $a0, $s2           # 4     #       #
        syscall                 # 5     #       #

        addi $s2, $s2, 1        # 6     #       #
        slti $t0, $s2, 11       # 7     #       #
        bnez $t0, inner         # 8     #       ###### end of inner loop

        addi $s1, $s1, 1        # 9     #
        slti $t0, $s1, 11       # 10    #
        bnez $t0, outer         # 11    ###### end of outer loop

        li $v0, 10              # 12
        syscall                 # 13

#-------------------------------------------------------------------------
#
# Behavior of instruction #8, which is a conditional branch instruction:
#       taken 9 times   \
#       fall thru once   \
#       taken 9 times     \
#       fall thru once     } There are 10 outer iterations, so the branch
#       ...               /  is taken 90 times and falls thru 10 times.
#       taken 9 times    /
#       fall thru once  /
#
# If our branch prediction strategy is "always assume fall thru", then we are
# correct 10 times out of 100 (10%), which is pretty poor performance.
#
#-------------------------------------------------------------------------
# If our branch prediction strategy is to use a 1-bit predictor:
# We initially assume the branch is fall thru.
#
#  Reality: TTTTTTTTTF TTTTTTTTTF TTTTTTTTTF ... TTTTTTTTTF
#  Predict: FTTTTTTTTT FTTTTTTTTT FTTTTTTTTT ... FTTTTTTTTT
#
#  wrong:   *        * *        * *        *     *        *
#
# So we will be wrong 20 times == 80/100 correct = 80%.
#
#-------------------------------------------------------------------------
# If our strategy uses a 2-bit predictor, again assume initially fall thru.
#
#  Reality: TTTTTTTTTF TTTTTTTTTF TTTTTTTTTF ... TTTTTTTTTF
#  Predict: FFTTTTTTTT TTTTTTTTTT TTTTTTTTTT ... TTTTTTTTTT
#
#  wrong:   **       *          *          *              *
#
# So we will be wrong 12 times == 88/100 correct = 88%.
```