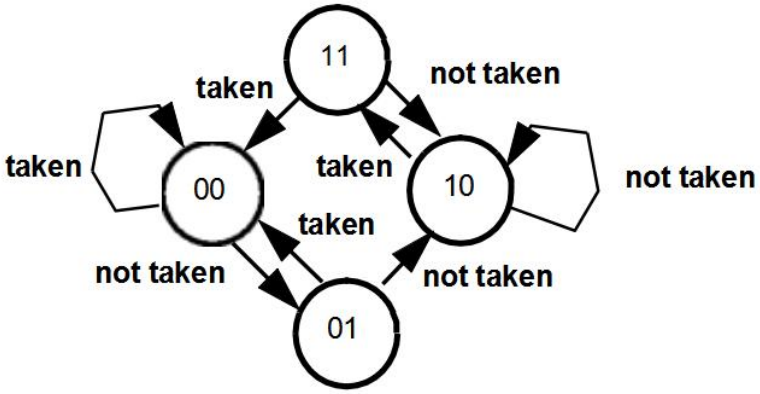
Problem Description

Consider the following program.

```
loop:
     LW   R4, 0(R3)
     ADDI R3, R3, 4
     SUBI R1, R1, 1
b1:
     BEQZ R4, b2
     ADDI R2, R2, 1
b2:
     BNEZ R1, loop
```

Assume the initial value of R1 is n (n>0).
Assume the initial value of R2 is 0 (R2 holds the result of the program).
Assume the initial value of R3 is p (a pointer to the beginning of an array of 32-bit integers).

We will be using a 2-bit predictor state machine, as shown below.



In state 1X we will guess not taken. In state 0X we will guess taken.

Assume that b1 and b2 do not conflict in the BHT.

Questions:

1. What does the program compute? That is, what does R2 contain when we exit the loop?

2. Now we will investigate how well our standard 2-bit branch predictor performs. Assume the inputs to the program are n=8 and p[0] = 1, p[1] = 0, p[2] = 1, p[3] = 0,… etc. That is the array elements exhibit an alternating pattern of 1's and 0's. Fill out Table 1 (note that the first few lines are filled out for you). What is the number of mispredicts?

Table 1 contains an entry for every branch (either b1 or b2) that is executed. The Branch Predictor (BP) bits in the table are the bits from the BHT. For each branch, check the corresponding BP bits (indicated by the bold entries in the examples) to make a prediction, then update the BP bits in the following entry (indicated by the italic entries in the examples).

System State		Branch Predictor		Branch Behavior	
PC	R3/R4	b1 bits	b2 bits	Predicted	Actual
b1	4/1	**10**	10	**N**	**N**
b2	4/1	*10*	**10**	**N**	**T**
b1	8/0	**10**	*11*	**N**	**T**
b2	8/0	*11*	**11**	**N**	**T**
b1	12/1	**11**	*00*		
b2	12/1				
b1					
b2					
b1					
b2					
b1					
b2					
b1					
b2					
b1					
b2					
b1					
b2					
b1					
b2					
b1					
b2					
b1					
b2					
b1					
b2					

Table 1