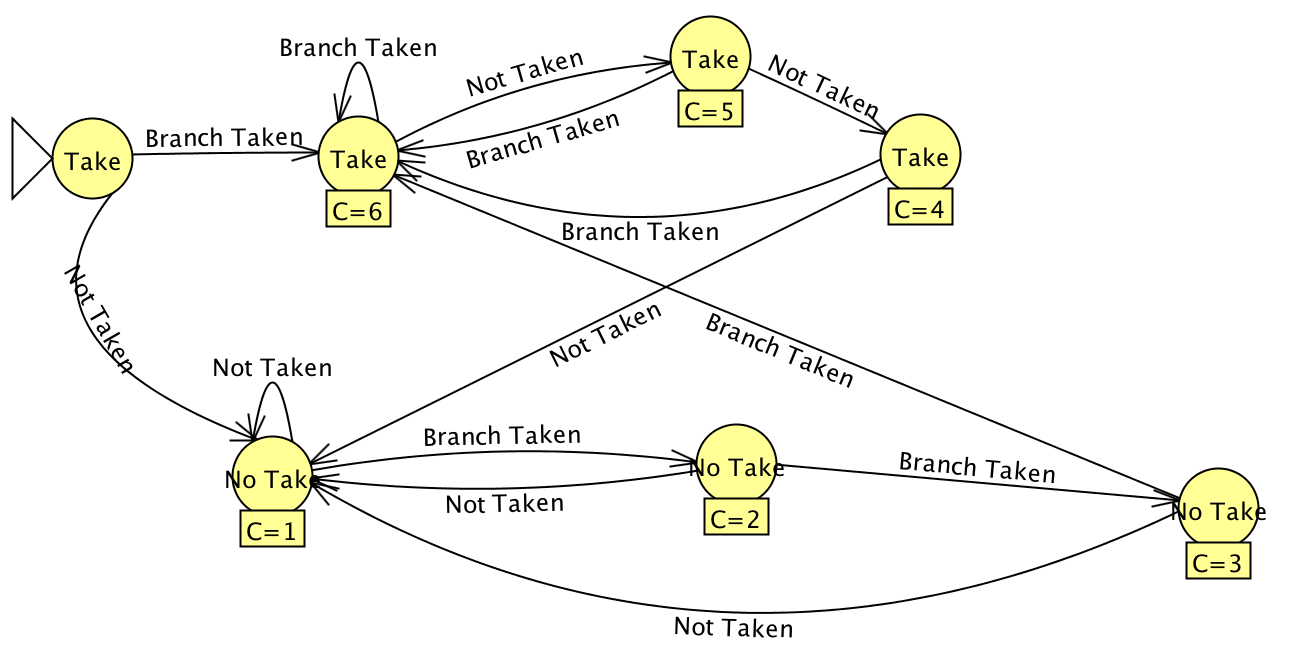
**CS320 – Fall 2017**

**Homework Assignment 4**

**Due: Wednesday, October 18th in class**

1. **[20%] Draw a finite state machine for branch prediction that is more tenacious than the two-bit predictor discussed in class. In your new design, prediction should only change after three consecutive mispredictions. Make sure to indicate the prediction for each of the states in your FSM.**



1. **[25%] Consider a gshare branch predictor with a 4-entry prediction table and a 2-bit global history register. Suppose that this predictor is used to predict the outcomes of the same branch instruction, and its actual behavior is listed in the sequence below. Assume that the last two bits of this branch’s PC are “10”. Assume that all predictor states start in the “strongly non-taken (00)” state. Demonstrate the operation of this 4-entry gshare predictor for this branch instruction and estimate the prediction accuracy. Actual Branch outcomes: TNTNTTNNNNN**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Time/Info | Global History | 00 | 01 | 10 | 11 | Prediction | Outcome |
| Initial State | 00 | 00 | 00 | 00 | 00 |  |  |
| 1 | 01 | 00 | 00 | **01** | 00 | **N** | T |
| 2 | 10 | 00 | 00 | 01 | **00** | **N** | N |
| 3 | 01 | **01** | 00 | 01 | 00 | **N** | T |
| 4 | 10 | 01 | 00 | 01 | **00** | **N** | N |
| 5 | 01 | **10** | 00 | 01 | 00 | **T** | T |
| 6 | 11 | 10 | 00 | 01 | **01** | **N** | T |
| 7 | 10 | 10 | **00** | 01 | 01 | **N** | N |
| 8 | 00 | **01** | 00 | 01 | 01 | **N** | N |
| 9 | 00 | 01 | 00 | **00** | 01 | **N** | N |
| 10 | 00 | 01 | 00 | **00** | 01 | **N** | N |
| 11 | 00 | 01 | 00 | **00** | 01 | **N** | N |

1. Answer the following questions:
2. **Design a branch predictor that would achieve perfect accuracy if the following pattern of branch outcomes were repeated forever: TNTTN [10%]**

We can have a 4-bit Global register (SR). Start it at 0000.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SRDEC | 0 | 1 | 2 | 5 | 11 | 6 | 13 | 10 | 5 | 11 | 6 | 13 | 10 | 5 | 11 | 6 |
| SRBIN | 0000 | 0001 | 0010 | 0101 | 1011 | 0110 | 1101 | 1010 | 0101 | 1011 | 0110 | 1101 | 1010 | 0101 | 1011 | 0110 |
| Result | X | T | N | T | T | N | T | N | T | T | N | T | N | T | T | N |

If SR = 0, 2, 5, 6, or 10: Predict Taken else Predict Not Taken

// If SR = 1, 11, or 13: Predict Not Taken

1. **What is the accuracy of your predictor if the pattern is the exact opposite (NTNNT)? [5%]**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SRDEC | 0 | 0 | 1 | 2 | 4 | 9 | 2 | 5 | 10 | 4 | 9 | 2 | 5 |
| SRBIN | 0000 | 0000 | 0001 | 0010 | 0100 | 1001 | 0010 | 0101 | 1010 | 0100 | 1001 | 0010 | 0101 |
| Result | X | N | T | N | N | T | N | T | N | N | T | N | T |
| A-Pred | -- | Wrong | Corr | Corr | Wrong | Wrong | Corr | Corr | Wrong | Wrong | Wrong | Corr | Corr |

If the pattern is the exact opposite, our predictor will be correct ~40% of the time.

2/3 + 2/5 \* K

**C) Now modify your predictor from part A) such that it eventually (after a warm-up period during which it can make wrong predictions) can perfectly predict both of the above patterns. [10%]**

We can modify this predictor such that we will start by going through a warm-up period of 12 instructions. We will have a secondary hash-register that will keep track of the values of the SR register. If after these 12 instructions happen there are no 11s or 13s recorded, we’re in the second pattern. If we record at least one 11 and one 13, we’re in the first pattern. If we are in the first pattern, we can simply apply the rule we came up with earlier (If SR = 0,2,5,6,10 => Taken else Not Taken). If we are in the second pattern, then If (SR = 2 or 4) predict Taken else Not Taken.

1. **Most of the time, the sharing of a single branch predictor counter between two different branch instructions hurts the prediction accuracy, but sometimes it helps. Give an example of a sequence of branch instructions such that the predictor sharing hurts the performance. Specifically indicate the order in which branch instructions are executed and estimate the prediction accuracy for both cases (shared predictor vs. separate predictors). You can assume any initial state you want. [10%].**

We start with a while loop that will recur on itself, say, 100 times. This loop is at PC 0x481000. We now have an if statement that is at PC 0x491000 and it should not be taken. The while loop primed the indexed table to take the loop (as seen by the 100 takes, 1 non-take) but this if statement needs to be skipped but uses the same location in the table for its information. Since the last few bits of each PC are the same, it will use the same branch predictor counter and thus will result in this misprediction.

1. **Repeat question 4) above, but this time show an example where predictor sharing actually helps the overall accuracy. [20%]**

We start with a while loop that will recur on itself, say, 100 times. This loop is at PC 0x481000. We now have another while loop that is at PC 0x491000 and it should recur on itself, say, 30 times. The first while loop primed the indexed table to take the loop, which helps the second while loop that uses the same index in the table so we have fewer mispredictions now. Since the last few bits of each PC are the same, it will use the same branch predictor counter and thus will result in this correct prediction and helpful outcome.