The code below is the assembly for the toUpper example. Figure out the prediction accuracy for each of the three branch instructions below (individually) with a 1-bit predictor, as well as for the first branch (beq) with a 2-bit predictor. Thus, your answer will be 4 percentages. You can assume:

- The code has been executing for a long time, so the predictors are “warm”.
- The predictors do not conflict in the branch history table.
- The string is 40 characters long (including the final null).
- Each of the 39 non-null characters in the string are randomly distributed in the range 1-255.
- ASCII code (the numeric representation of characters) is given on page 122.

```assembly
// string is a pointer held at Memory[100].
// X0-index, 'A' = 65, 'a' = 97, 'z' = 122
LDUR X0, [X31, #100] // index = string

LOOP:
    LDURB X1, [X0, #0] // load byte *index
    CBZ X1, END // exit if *index == 0
    CMPI X1, #97 // is *index < 'a'? 
    B.LT NEXT // don't change if < 'a'
    CMPI X1, #122 // is *index > 'z'? 
    B.GT NEXT // don't change if > 'z'
    SUBI X1, X1, #32 // X1 = *index + ('A' - 'a')
    STURB X1, [X0, #0] // *index = new value;

NEXT:
    ADDI X0, X0, #1 // index++;
    B LOOP // continue the loop

END:
```

**1-bit:**
\[
\text{Accuracy} = \frac{\text{Predict } T + \text{Not } T \times \text{Predict } }{\text{Predict } T + \text{Not } T} = \frac{96 + 157 \times 159}{255} = 95.6\% \text{ Right}
\]

**2-bit:**
\[
\text{Accuracy} = \frac{\text{Predict } T + \text{Not } T \times \text{Predict } }{\text{Predict } T + \text{Not } T} = \frac{96^2 + 159^2}{255^2} = 97.5\% \text{ Right}
\]

---

**B.LT**
\[X1 \in \{1..255\} \]
\[x_1 < 97 \Rightarrow T \]
\[x_1 \geq 97 \Rightarrow \text{Not } T \]
\[
\text{Accuracy} = \frac{96}{255} \times \frac{96}{255} + \frac{157}{255} \times \frac{159}{255} = \frac{96^2 + 159^2}{255^2} = 53\% \text{ Right}
\]

---

**B.GT**
\[X1 \in \{97..255\} \]
\[x_1 > 122 \Rightarrow T \]
\[x_1 \leq 122 \Rightarrow \text{Not } T \]
\[
\text{Accuracy} = \frac{133}{159} \times \frac{133}{159} + \frac{26}{159} \times \frac{26}{159} = \frac{133^2 + 26^2}{159^2} = 72.6\% \text{ Right}
\]
For the code below draw the constraint graph. Label each edge with the cause of the constraint (i.e. “RAW X2”, etc.). Then adjust the code to run as fast as possible on a pipelined processor like that in lab #4. Note that if you carefully adjust the program you should be able to fill all the delay slots of the program (tricky!).

**Problem:** Nothing to fill 8’s delay slot

**However:** Adjust addresses & move STUR after SUBI

```
LOOP:
  1: LDUR X2, [X10, #0]
  2: SUB X4, X2, X3
  3: STUR X4, [X10, #0]
  4: LDUR X5, [X10, #8]
  5: SUB X6, X5, X3
  6: STUR X6, [X10, #8]
  7: SUBI X10, X10, #16
  8: CBNZ X10, LOOP

RAW X2  \       \ RAW X5
  \     \     \   \\
  2     5     6

RAW X4  \       \ RAW X6
  \     \     \   \\
  3     6

WAR X10  \       \ WAR X10
  \     \     \   \\
  7     7     8

RAW X10 control
```

```
Loop:
  1: LDUR X2, [X10, #0]
  2: SUB X4, X2, X3
  3: STUR X4, [X10, #16] // Before STURs -> fix addresses!
  4: LDUR X5, [X10, #8]
  5: SUB X6, X5, X3
  6: STUR X6, [X10, #24]
  7: SUBI X10, X10, #16
  8: CBNZ X10, LOOP
```