Problem 1: Estimate performance of the 8-way superscalar statically scheduled MIPS implementations described here. All are five stages, as used in class, and always hit the cache, as has been the case in class so far. Some of the implementations have no fetch group alignment restrictions, which means any eight contiguous instructions can be fetched. Some impose a fetch group alignment restriction, meaning if a CTI target is address a `IF` will fetch eight instructions starting at address $a\prime = 8 \times 4 \times \lfloor \frac{a}{8\times 4} \rfloor$ (for those preferring C: `aa = a & ~0x1f`). Instructions in $[a\prime, a)$ (or from `aa` to before `a`) will be squashed before reaching `ID`.

The implementations include a branch predictor that predicts when a branch is in `IF`, resolves (checks the prediction) when a branch is in `ID`, and if necessary recovers (squashes wrong-path instructions) when a branch is in `EX`. A branch is predicted when it is in `IF` and the prediction is used in the next cycle. Example 1, below, illustrates a correct taken prediction. The correctness of the prediction is checked, resolved, when the branch is in `ID`; if incorrect the wrong-path instructions are squashed and the correct path instructions are fetched in the next cycle (when the branch is in EX). This is illustrated in Example 2 for an incorrect taken prediction.

Note that due to alignment restrictions (if imposed) and branch placement the number of useful instructions fetched in a cycle can vary, and that is something to take into account in the subproblems below. The examples below illustrate *when* instructions will be fetched and squashed but they do not show *how many* will be fetched in every situation.

```
 # Example 1: Branch correctly predicted taken.  Target fetched in next cycle.
 #
 # Cycle            0  1  2  3  4  5  6
 beq  r1,r2, TARG   IF ID EX ME WB
 nop                IF ID EX ME WB
 ...

TARG:
 add r3, r4, r5        IF ID EX ME WB


 # Example 2: Branch wrongly predicted taken.
 # Target squashed, correct path (fall through) fetched in cycle after ID.

 # Cycle            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 beq  r1,r2, TARG   IF ID EX ME WB
 nop                IF ID EX ME WB
 sub r6, r7, r8           IF ID EX ME WB

TARG:
 add r3, r4, r5        IFx
 # Cycle            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
```

All implementations run the same program, which has not been specially compiled for the 8-way machine. In the program, which has no floating-point instructions, any two data-dependent instructions have at least seven instructions between them. This avoids some stalls, assume that there are no other stalls in the superscalar implementation **due to data dependencies**.

Let n_i denote the number of dynamic instructions in the program and let n_b denote the number of dynamic instructions that are branches. For the questions below show answers in terms of these symbols and also show values for $n_\mathrm{i} = 10^{10}$ and $n_\mathrm{b} = 2 \times 10^9$. Assume that half of the times a branch is executed it is taken.

(a) Suppose the 8-way implementation has perfect branch prediction and has no fetch alignment restrictions. Approximately how long (in cycles) will it take to run the program. State any assumptions. *Hint: Because*

of the branches it's $> \frac{n_i}{8}$.

(*b*) Repeat the question above for a predictor that always predicts not taken (which essentially means no predictor).

(*c*) Repeat the question above for a predictor with a 95% prediction accuracy. (Yes, that means 95% of the predictions are correct.)

(*d*) Once again, suppose the 8-way implementation has perfect branch prediction but now fetch is restricted to aligned groups. Approximately how long (in cycles) will it take to run the program. State any assumptions.

Problem 2: Consider a bimodal branch predictor with a 2^{10}-entry branch history table (BHT).

(*a*) What is the prediction accuracy on the branch below with the indicated behavior assuming no interference. Assume that the pattern continues to repeat. Provide the accuracy after warmup.

```
0x1000 beq r1, r2  TARG   t  t  t  n  t  t  n  n  n  t  t  t  n  t  t  n  n  n  ...
```

Problem 3: Suppose that for some crazy reason it's important that the branch at address `0x1000` be predicted accurately, even if that means suffering additional mispredictions elsewhere. The result of this crazyness is the code below, in which the branch in `HELPER` is intended to help the branch at `0x1000`.

(*a*) Choose an address for `HELPER` so that `0x1000` is helped.

(*b*) Given a correct choice for the address of `HELPER`, find the prediction accuracy of the branch at `0x1000`.

```
  jal HELPER
  nop
0x1000:
  beq r1, r2  TARG   t  t  t  n  t  t  n  n  n  t  t  t  n  t  t  n  n  n  ...
  ...
  ....

HELPER:
  beq r1, r2 SKIP
  nop
SKIP:
  jr r31
  nop
```