Branch prediction in hardware

Motivation -- branch instructions can easily lead to stalls in the pipeline.
During the ID stage, we can calculate the target address of the branch, and we can also use an XOR to compare registers, but this means it's not until the end of the ID stage that we know the branch is taken or we fall through to the next instruction.

Strategy #1 -- Always assume the branch is fall through.
After all, the first time we encounter a branch instruction, we have no idea where it would branch to anyway, so this is a safe default.

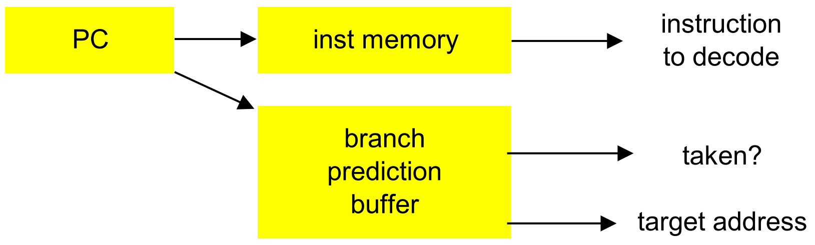
If we actually branch, we need to flush the pipeline:

```
7
8   7                           instruction 8 is the branch
9   8   7                       We realize too late that we shouldn't fetch 9.
3           7
    3           7
        3
            3
                3
```

However, in practice most branches are taken, because of loops. So this is a poor strategy because it's wrong most of the time. We can do better.

Strategy #2 -- Use a branch prediction buffer in the IF stage. A major improvement on the default.

While fetching, see if we've encountered this instruction before.
If it's a branch that we took the last time, then assume it will branch again.



A closer look at the branch prediction buffer, which typically has 32 entries (rows).

offset inst addr (e.g. last 8 bits)	taken?	target address	
00	no	0x…	
04	no	0x…	
08	no	0x…	
…	…	…	
1c	yes	0x00400008	the entry for instruction 8 points to inst 3
…		…	
fc	no	0x…	

We use the address of the instruction we're fetching as an index into the table. So the branch prediction buffer just needs 32 bits (the yes/no's) plus 32 words for the targets.

Strategy #3 -- Use 2 bits instead of 1. An improvement on the model:
If we are wrong once, don't switch the prediction until we're wrong twice in a row!
This will allow for hiccups. See example program loop.s