Q1

Pros:

Cons:

Q2

Hazard types:

* CONTROL HAZARD

in EXEC1 there is a branch command that was taken (so PC is written in r[7]) but DEC1 or EXEC0 reads from r[7] stale data.

* solution is to forward the correct PC address to the ALU’s input.
* DATA HAZARD

in EXEC1 there is a LD command to a register which the command in DEC1 or EXEC0 reads from.

* solution is to forward the loaded value to the ALU’s input.

in DEC1 there is a ST command and DEC0 has a LD command

* solution is to stall the pipeline for 1 cycle.

in EXEC0 there is a LD command to a register which the command in DEC1 reads from

* solution is to stall the pipeline for 1 cycle.
* REG HAZARD

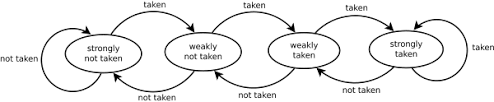
in EXEC1 there is a command that uses the ALU and writes the result to a register that the command in DEC1 reads from

* solution is to forward the result to the ALU’s input.

We didn’t see how a structural hazard is possible since the instruction and data memories are separated so there is no scenario where the same memory has read and write operations at the same time.

Q3

We included a 2-bit branch predictor for branch resolution which operates as follows:



the leftmost state corresponds to 0 and the rightmost to 3. The initial state is 0.

The predictor is a saturation counter. If the branch is taken, increment. otherwise, decrement.

for values of 0,1 assume branch not taken. for values of 2,3 assume branch taken.

Q6

1

The speedup is:

2

The CPI is still not 1 as we hoped. That’s because in the beginning of the execution the pipeline has to fill up, and at the end the pipeline has to empty.

3

Future improvements to increase IPC:

* use a more sophisticated branch predictor (we didn’t have the time to successfully implement one). A better predictor will have better predictions and fewer flushes which hurt the IPC.