**Lab 3**

Shai Weinstock 314964727 Eden Tsubery 316131796

Q1

Using the Harvard architecture has the following advantages:

* better performance since instructions and data are in different memories and can be read simultaneously (no bottleneck)
* less chance for data corruption as data and instruction use separate busses.

Disadvantages:

* memory is not flexible: free instruction memory cannot be used to extend data memory and vice versa.
* more complicated architecture, more hardware and more expensive

Even though more complicated than Von Neumann, using Harvard architecture is a good decision because the customer was unhappy with the performance, and Harvard architecture performs better than the Von Neumann architecture.

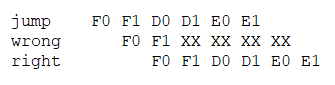
Q2

Hazard types:

* CONTROL HAZARD

in DEC0 there is a branch prediction is taken.

* solution is to flush old command and fetch new one from jump PC.



* DATA HAZARD

in EXEC1 there is a command that stores 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.

A picture containing text

Description automatically generated

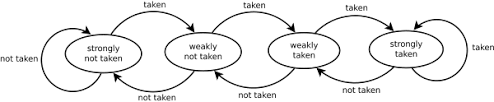
We don’t believe that structural hazards are a real possibility since the data and instruction memories are separated so no concurrent reads and write to the same memory.

DMA won’t interfere because if memory is busy (marked whenever LD, ST operation in pipeline) DMA will enter WAIT state until memory is freed.

All commands go through the same pipeline so no two commands competing for the ALU, so no structural hazard here neither.

Q3

We used 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.

In case a branch is taken then there is a control hazard described in Q2.

Q6

1

The speedup is calculated by

* ADD
* SQRTQ

2

The CPI is still not 1 as we hoped. That’s because there may be stalls during the execution and 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.
* using a cache can also help as memory addresses that are read frequently, can be fetched quicker (cache is faster than memory).

Q7