NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

CZ3001

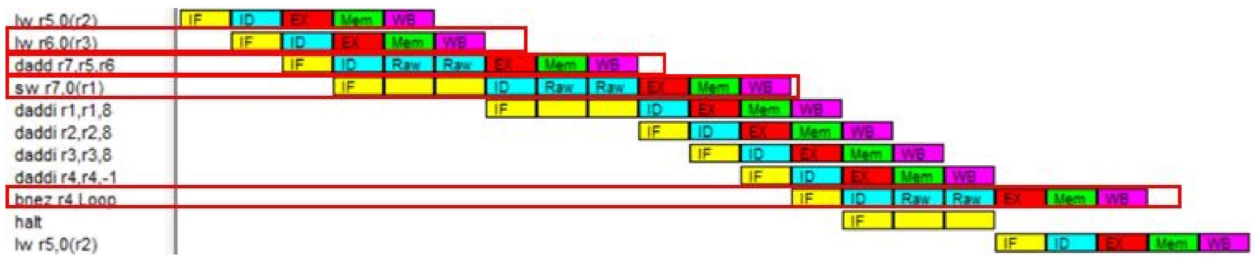
Advanced Computer Architecture

(Sem 1 2020-2021)

Team Members:
Singh Kirath (U1822329J)
Arya Shashwat (U1822436J)

1. Prediction: There would be 3 stall cycles for the RAW dependencies of loading R6, adding R7 and 3 stall cycles for branch prediction without data forwarding.

   Our prediction turned out to be wrong as there are 2 stall cycles being used for both the data RAW dependencies and branch prediction in the actual execution.This could be because the architecture allows read and write to happen simultaneously which requires only 2 stall cycles for RAW dependencies. Also, the branch prediction by BNEZ is being done in the instruction decode stage (including the updation of the program counter), making it available in the execute stage. Hence only 1 stall cycle is used.



   There are 2 types of data hazards in the code-

   a. There are 3 RAW data hazards:

      LW R6 , 0(R3)
      DADD R7 , R5 , R6

      DADD R7, R5, R6
      SW R7, 0(R1)

      DADDI R4, R4, -1
      BNEZ R4, Loop

   b. Control hazard:
      There is 1 stall cycle as the architecture assumes the branch is always not taken. So when the branch is taken, that instruction needs to be flushed.

      BNEZ R4, Loop
      END: halt

2.

$$CPI = \frac{(IC_{hot}+S_{hot})*L+N_{out}+S_{out}}{IC_{hot}*L+N_{out}} \xrightarrow{L \to +\infty} CPI_{asymptotic} = \frac{IC_{hot}+S_{hot}}{IC_{hot}}$$

$$= \frac{(9+7)*6+5+0}{9*6+5}$$

$$= 1.711$$

The CPI from the simulation results comes out to be 1.76 which is about 0.05 times higher than our calculated steady state CPI. The simulator CPI is higher because it uses extra stall cycles for the one-off start-up cost in clock cycles needed to initially fill the pipeline.

The steady state CPI is more effective in describing the performance of a loop in a pipelined architecture. It takes multiple factors into account including various types of stall cycles, nop instructions, etc.
Conventional CPI measures the performance in a single-cycled architecture without any pipeline.

3.

$$CPI = \frac{(IC_{hot}+S_{hot})*L+N_{out}+S_{out}}{IC_{hot}*L+N_{out}} \xrightarrow{L \to +\infty} CPI_{asymptotic} = \frac{IC_{hot}+S_{hot}}{IC_{hot}}$$

$$= \frac{(9+3)*6+5+0}{9*6+5}$$

$$= 1.305$$

Even with data forwarding, there are still stall cycles in the program. There are a total of 2 RAW hazards and 1 control hazard.
The stall cycles with data forwarding comes down to 1 and so does the branch prediction. Some of the stall cycles occur after the ID stage rather than the IF because through data forwarding, the data is forwarded to the instruction in the execution stage, rather than the decode stage. The previous instruction calculates the result in its execute stage and passes the result on to the execute stage of the next instruction in the next clock cycle.

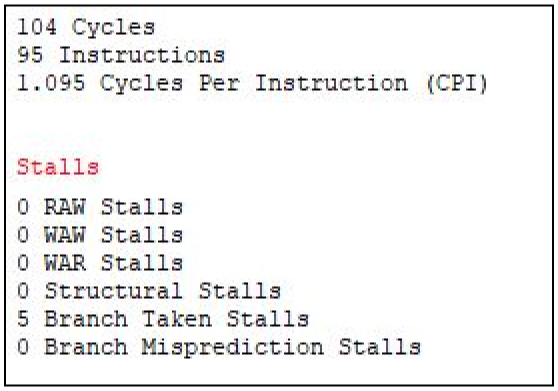
4.

$$CPI = \frac{(IC_{hot}+S_{hot})*L+N_{out}+S_{out}}{IC_{hot}*L+N_{out}} \xrightarrow{L \to +\infty} CPI_{asymptotic} = \frac{IC_{hot}+S_{hot}}{IC_{hot}}$$

$$= \frac{(15+0)*6+5+5}{15*6+5}$$

$$= 1.052$$

After adding NOP instructions to the code, the RAW hazards get eliminated to get better performance. This eliminated all the stall cycles from RAW hazards replacing them with instructions.


```
104 Cycles
95 Instructions
1.095 Cycles Per Instruction (CPI)


Stalls
0 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
5 Branch Taken Stalls
0 Branch Misprediction Stalls
```

However, there are still 5 control hazards present.

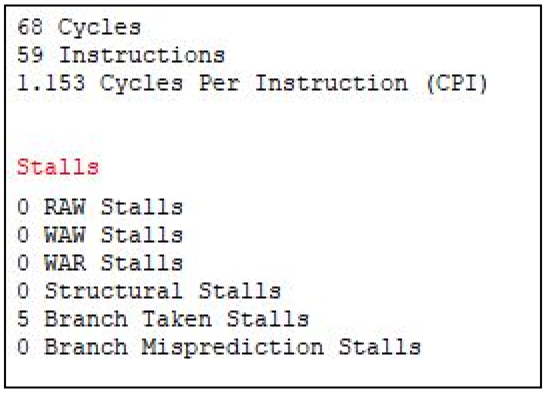
5.

$$CPI = \frac{(IC_{hot}+S_{hot})*L+N_{out}+S_{out}}{IC_{hot}*L+N_{out}} \xrightarrow{L \to +\infty} CPI_{asymptotic} = \frac{IC_{hot}+S_{hot}}{IC_{hot}}$$

$$= \frac{(9)*6+5+5}{9*6+5}$$

$$= 1.084$$

After reordering the instructions, all the RAW stall cycles were eliminated and the CPI improved from earlier. The only stall cycles remaining are the 5 branch taken stalls.



```
68 Cycles
59 Instructions
1.153 Cycles Per Instruction (CPI)

Stalls
0 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
5 Branch Taken Stalls
0 Branch Misprediction Stalls
```

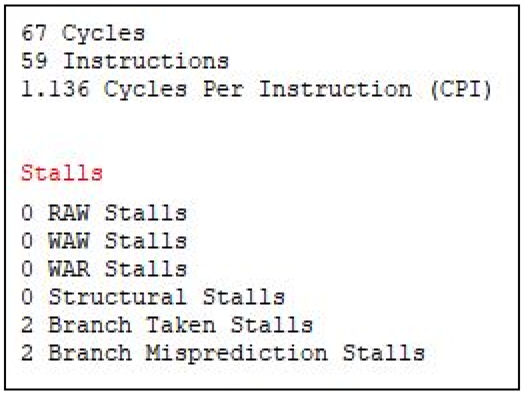
6. After enabling the “Branch target buffer” and “Data Forwarding”, the performance improved significantly from before.

$$CPI = \frac{(IC_{hot}+S_{hot})*L+N_{out}+S_{out}}{IC_{hot}*L+N_{out}} \xrightarrow{L \to +\infty} CPI_{asymptotic} = \frac{IC_{hot}+S_{hot}}{IC_{hot}}$$

$$= \frac{(9+0)*6+5+4}{9*6+5}$$

$$= 1.067$$

As compared to forwarding only in Q3, this program has no RAW hazards which would be eliminated by reordering all instructions. Also, Q3 has 5 control hazards while Q6 only has 4 by using a branch target buffer which tries to predict the branch target to reduce stall cycles.



```
67 Cycles
59 Instructions
1.136 Cycles Per Instruction (CPI)

Stalls
0 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
2 Branch Taken Stalls
2 Branch Misprediction Stalls
```

As compared to rescheduling only in Q5, the performance has minimal change. In Q5, there were 5 branch taken stall cycles. Whereas, in Q6 there are only 2. But 2 extra branch misprediction stalls were also added as a result of using a branch target buffer. Therefore, we only have a reduction of 1 net stall cycle by using both data-forwarding and branch target buffer.

7. **Assumptions:**
    - Data forwarding and branch target buffer are disabled
    - Adding extra inputs to arrays “b” and “c” excluding “a” as it is used for storing the results and not as an input array
    - Updating the contents of register R4 from 6 to 12 to reflect the 6 extra inputs

    After adding additional 6 inputs, the steady state CPI:

$$CPI = \frac{(IC_{hot}+S_{hot})*L+N_{out}+S_{out}}{IC_{hot}*L+N_{out}} \xrightarrow{L \to +\infty} CPI_{asymptotic} = \frac{IC_{hot}+S_{hot}}{IC_{hot}}$$

$$= \frac{(9+7)*12+5+0}{9*12+5}$$

$$= 1.743$$

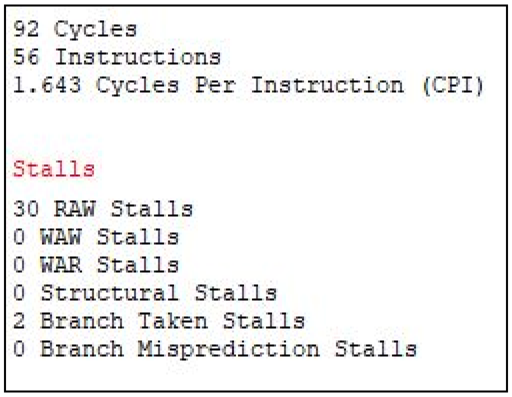
There is a loss in performance in the steady state CPI. This is because there are 7 stall cycles per loop (6 RAW hazards and 1 control hazard).
Instruction count = 9 x 12 + 5 = 113
New normal CPI = (# Cycles) / (# Instructions)
= 113 / 113
= 1
The new normal CPI would be 1 as it uses a single cycle per instruction. So, there would be no stalls and data hazards.

8.

$$CPI = \frac{(IC_{hot}+S_{hot})*L+N_{out}+S_{out}}{IC_{hot}*L+N_{out}} \xrightarrow{L \to +\infty} CPI_{asymptotic} = \frac{IC_{hot}+S_{hot}}{IC_{hot}}$$

$$= \frac{(17+11)*3+5+0}{17*3+5}$$

$$= 1.589$$



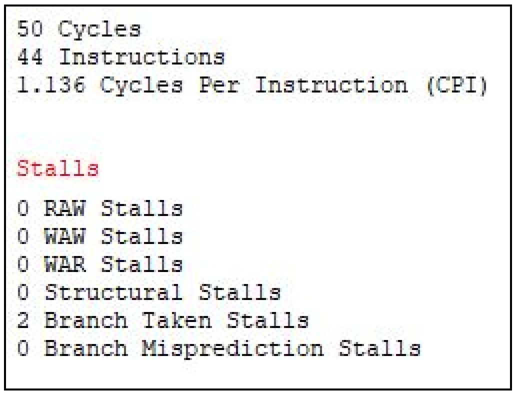
```
92 Cycles
56 Instructions
1.643 Cycles Per Instruction (CPI)


Stalls

30 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
2 Branch Taken Stalls
0 Branch Misprediction Stalls
```

The CPI shows in the simulator is 1.643 which is higher as a result of the extra cycles needed for the one-off startup cost.

By loop unrolling by a factor of 2, the CPI has improved as compared to Q2. There are a total of 11 stalls in each loop interaction (10 RAW hazards and 1 control hazard) amounting to a total of 33 stall cycles in total.
In Q2, there were a total of 42 stall cycles. Therefore, we have a net reduction of 9 stall cycles by using loop unrolling.

9. After instruction rescheduling, there is a significant improvement to the CPI.

Reordering the instructions eliminated all the stall cycles. Only 2 branch taken stall cycles are left. AS compared to the unordered code in Q8, there are 31 fewer stall cycles. Therefore the CPI improves significantly.



```
50 Cycles
44 Instructions
1.136 Cycles Per Instruction (CPI)


Stalls
0 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
2 Branch Taken Stalls
0 Branch Misprediction Stalls
```

$$CPI = \frac{(IC_{hot}+S_{hot})*L+N_{out}+S_{out}}{IC_{hot}*L+N_{out}} \xrightarrow{L\ \rightarrow\ +\infty} CPI_{asymptotic} = \frac{IC_{hot}+S_{hot}}{IC_{hot}}$$

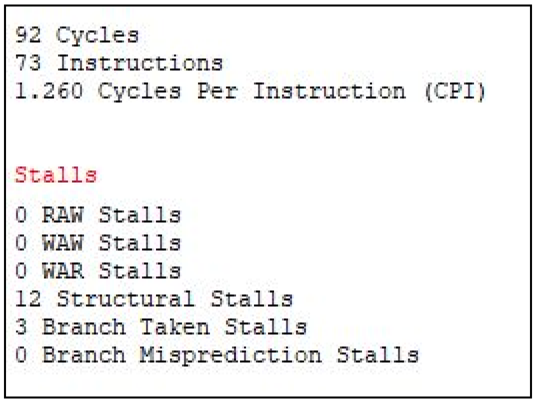
$$= \frac{(13+0)*3+5+2}{13*3+5}$$

$$= 1.045$$

Excluding the stalls for one-off startup cost, the calculated CPI is very close to the one in the simulator.

10.

A. To remove all the RAW hazards and stall cycles, the loop has to be unrolled by a factor of 3.



```
92 Cycles
73 Instructions
1.260 Cycles Per Instruction (CPI)


Stalls
0 RAW Stalls
0 WAW Stalls
0 WAR Stalls
12 Structural Stalls
3 Branch Taken Stalls
0 Branch Misprediction Stalls
```

The addition in floating point numbers is different from integer addition in the MIPS architecture. Floating point addition takes 4 execution cycles instead of 1. This gives rise to structural hazards as more than one instruction tries to access the memory at a time.

We weren't able to remove all the stall cycles as a decrease in the structural stalls resulted in an increase in the RAW stalls. So our code removed all RAW cycles, sacrificing structural stalls.

B. Latency (execution time) - Latency is used to denote the time required to finish a fixed task.[1] In this question, the fixed problem is the iteration in each loop. Hence the latency for this problem is 17 (total number of instructions) + 3 (Structural Stalls) + 1 (Branch taken Stalls) = 21 clock cycles.

C. Total LOC = 22 [including the initialization and halt statements]

References

[1] *Upenn.edu*. [Online]. Available: https://www.cis.upenn.edu/~milom/cis501-Fall08/lectures/02_performance.pdf. [Accessed: 26-Nov-2020].

Signatures

Kirath Singh (U1822329J) - Kirath Singh

Shashwat Arya (U1822436J) - Shashwat Arya