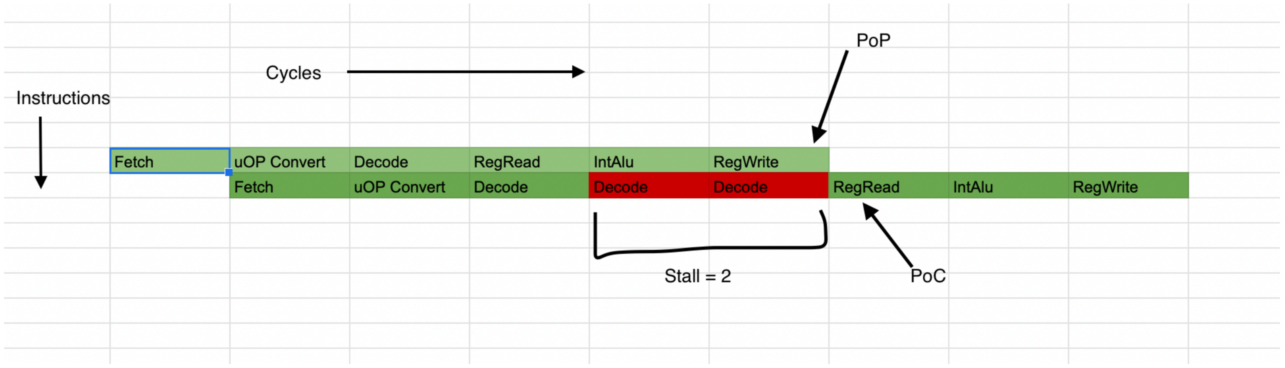
Computer Architecture Assignment 3 - u1417989

1. Data dependencies (30 pts)

PoP - Point of Production, **PoC -** Point of Consumption
Assuming each stage takes the entire cycle to do their work.
(1 excel box = 1 cycle/stage)

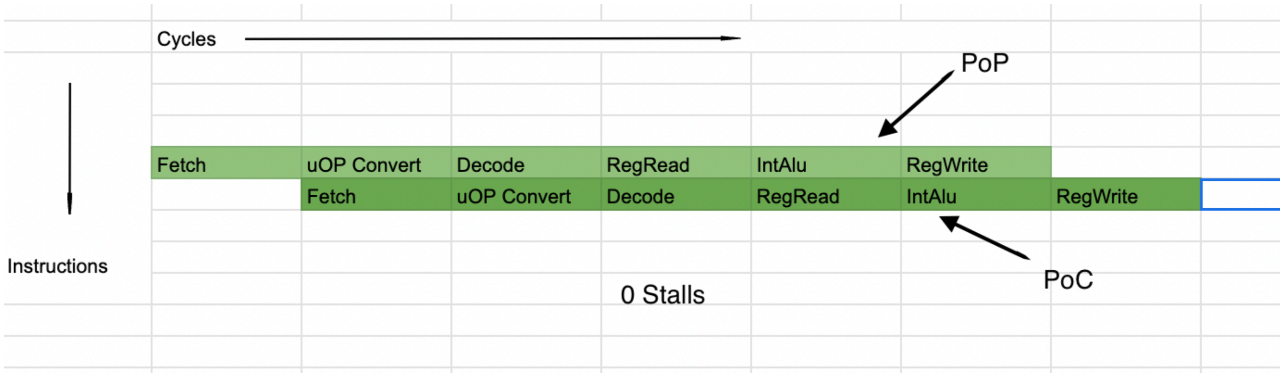
1. Int-add, followed by a dependent Int-add
   Example:
   IntAdd R3 <- R1, R2 IntAdd R5 <- R3, R4

   a. No Register Bypassing
   Stall = 2 cycles



Here, The first instruction would write the result in the register in the 6th cycle. Since there is no register bypassing allowed and the write & read takes an entire cycle, The RegRead for 2nd instruction takes place in the 7th cycle, resulting in a 2 cycle stall (as seen in the diagram)

   b. Full Bypassing



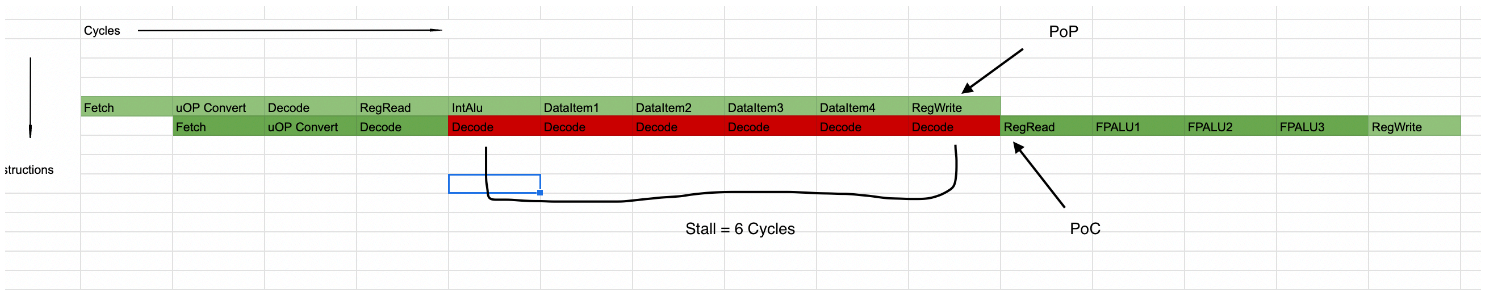
Since Register Bypassing is allowed, The IntALU from the 1st instruction produces the value that is to be consumed from the latch as one of the inputs for IntAdd in the 2nd instruction directly without waiting for register write, resulting in 0 Stalls

   Stall = 0 cycles

2. Load, followed by a dependent FP-add
   Example:

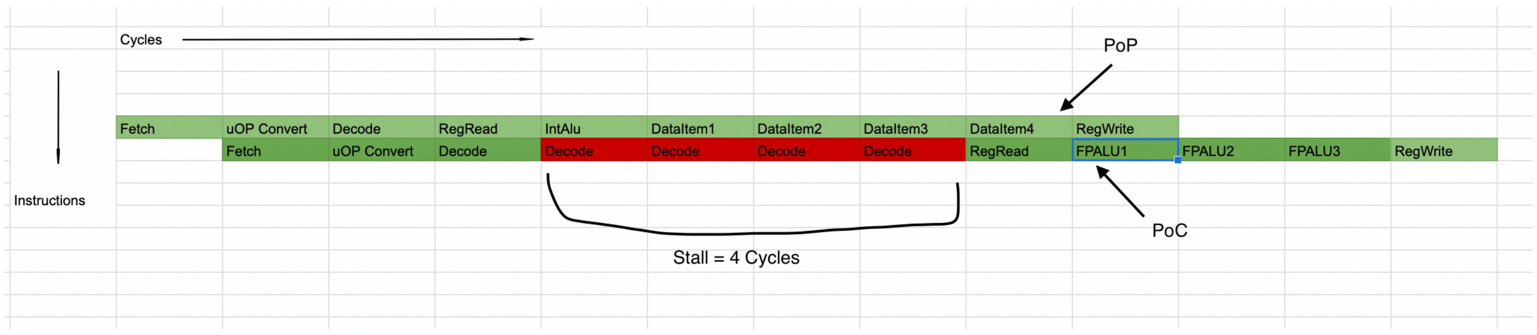
LD R2 <- [8] R1   FPAdd R4 <- R2, R3

a. No Register Bypassing
Stall = 6 cycles



Since no register bypassing, Data loaded from 1st instruction is consumed (11th cycle) only after register write(10th cycle). Hence, 6 stall cycles.

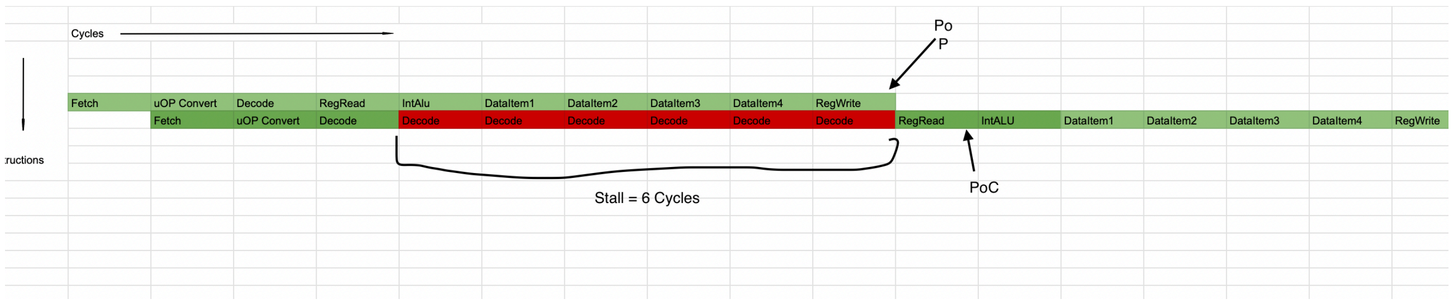
b. Full Bypassing
Stall = 4 cycles



Here, One of the operands for FPAdd is ready to be consumed in the FPALU1 stage when produced from the DataItem4 stage of the 1st Load Instruction (Stall = 4 cycles)

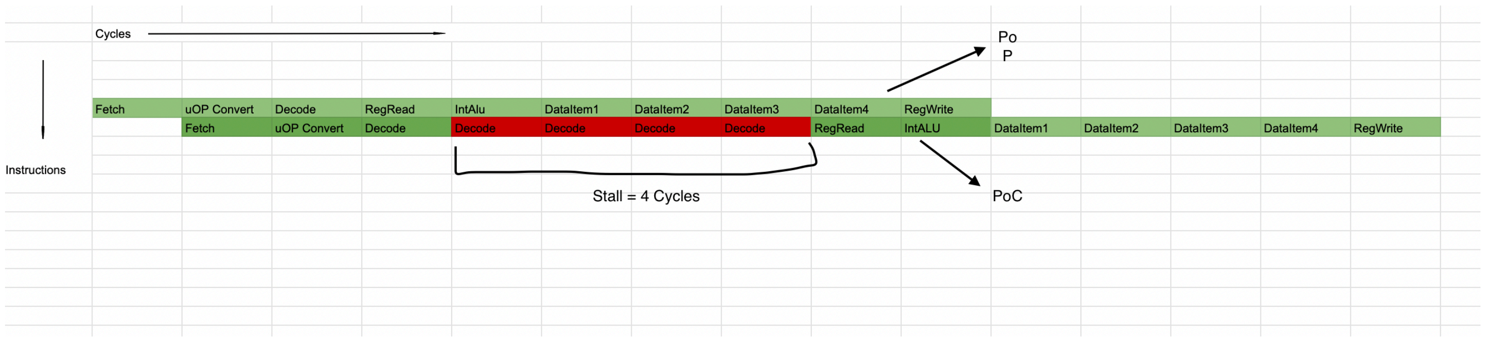
3. Load, providing the address operand for a store
Example:
LD R2 <- [8] R1   ST R3 -> [8] R2

a. No Register Bypassing
Stall = 6 cycles



Similar to other examples, we wait for register write in order to consume the value for no register bypassing, resulting in 6 stall cycles

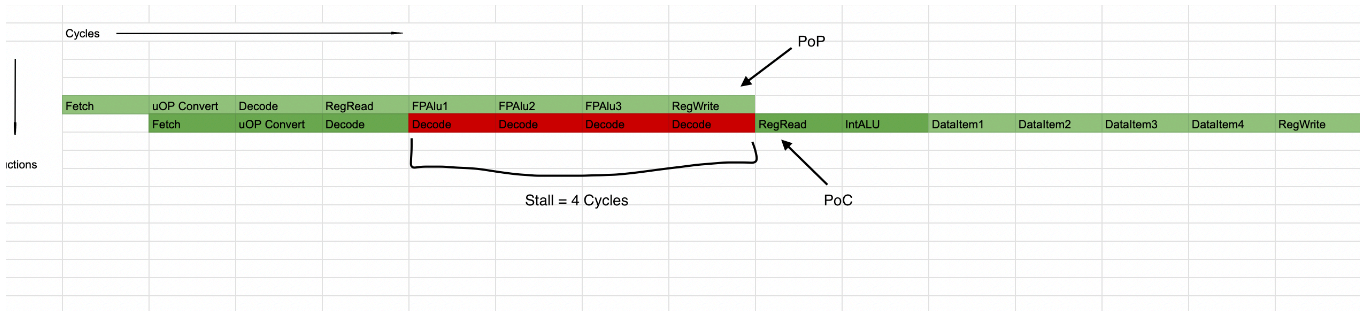
b. Full Bypassing
Stall = 4 cycles


Cycles
Po P
Fetch | uOP Convert | Decode | RegRead | IntAlu | DataItem1 | DataItem2 | DataItem3 | DataItem4 | RegWrite
Fetch | uOP Convert | Decode | Decode | Decode | Decode | Decode | RegRead | IntALU | DataItem1 | DataItem2 | DataItem3 | DataItem4 | RegWrite
Instructions
Stall = 4 Cycles
PoC

After the Load instruction produces the address for the dependent store, it can directly be consumed in the IntALu stage of store instruction for address computation without waiting for register write (Stall = 4 cycles)

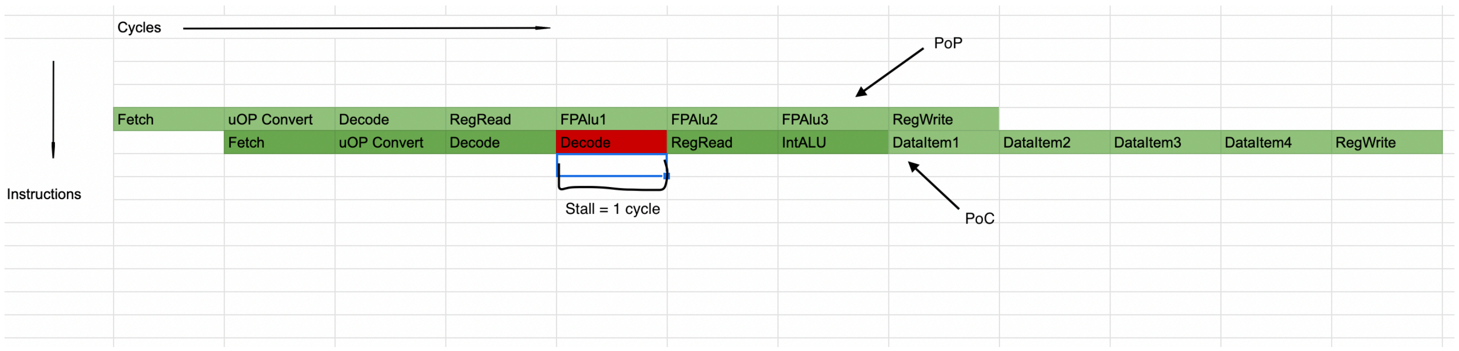
4. FP-add, providing the data operand for a store
   Example:
   FPADD R3 <- R1, R2 ST R3 -> [8] R4

a. No Register Bypassing
Stall = 4 cycles


Cycles
PoP
Fetch | uOP Convert | Decode | RegRead | FPAlu1 | FPAlu2 | FPAlu3 | RegWrite
Fetch | uOP Convert | Decode | Decode | Decode | Decode | Decode | RegRead | IntALU | DataItem1 | DataItem2 | DataItem3 | DataItem4 | RegWrite
ctions
Stall = 4 Cycles
PoC

RegRead needs to wait for Regwrite of prev instruction to complete. (Stall = 4 cycles)

b. Full Bypassing
Stall = 1 cycle


Cycles
PoP
Fetch | uOP Convert | Decode | RegRead | FPAlu1 | FPAlu2 | FPAlu3 | RegWrite
Fetch | uOP Convert | Decode | Decode | RegRead | IntALU | DataItem1 | DataItem2 | DataItem3 | DataItem4 | RegWrite
Instructions
Stall = 1 cycle
PoC

As soon as the data is ready after FPALU3 stage, it can be picked up directly for data write to memory without waiting for register write (1 stall)

2. **Branch delay slots and stalls (30 pts)**

Given:

a. A 10-stage in-order processor, where the instruction is fetched in the first stage, and the branch outcome is known after three stages. (IF = 1st stage, next 2 stages for determining branch outcome).

b. All stalls in the processor are branch-related and branches account for 15% of all executed instructions.

c. The branch is taken 90% of the time and not-taken 10% of the time.

For Each example, Assume that total number of instructions to be executed = 1000
(In ideal case (no stalls) there should be 1000 instructions in 1000 cycles = 1 CPI)

1. On every branch, fetch is stalled until the branch outcome is known.
Soln:

Since total instructions assumed is 1000
Branch Instructions = 15% * 1000 = 150
Since we stall until the branch outcome is known, there are 2 stall cycles for each branch instruction.
Total stalls = 150 * 2 = 300

Total cycles taken to complete 1000 instructions = 1000 + 300 = 1300

Average CPI = No. Of Cycles/ No. Of Instructions = (1300 / 1000)
= **1.3 CPI**

2. Every branch is predicted not-taken and the mis-fetched instructions are squashed if the branch is taken.
Soln:

Since total instructions assumed is 1000
Branch Instructions = 15% * 1000 = 150
Since we are predicting every time that the branch is not taken, we are right 10% of the time i.e. 10%*150 = 15 instructions.
For the remaining 135 branch instructions we predicted wrong and hence wasted 2 cycles (branch outcome time) for each instruction.
Total stall cycles = 135 * 2 = 270

Total cycles taken to complete 1000 instructions = 1000 + 270 = 1270

Average CPI = No. Of Cycles/ No. Of Instructions = (1270 / 1000)
= **1.27 CPI**

3. The processor has two delay slots and the two instructions following the branch are always fetched and executed, and

   1. You are unable to find any instructions to fill the delay slots.

Soln:

Since total instructions assumed is 1000
Branch Instructions = 15% * 1000 = 150

For each branch instruction we have 2 delay slots, and since there are No operations to fill these slots, we have total of 2 stall cycles per branching instruction.
Total stalls = 150 * 2 = 300

Total cycles taken to complete 1000 instructions = 1000 + 300 = 1300

Average CPI = No. Of Cycles/ No. Of Instructions = (1300 / 1000)
= **1.3 CPI**

2. You are able to move two instructions before the branch into the delay slots.

Soln:

Since total instructions assumed is 1000
Branch Instructions = 15% * 1000 = 150

Since we are able to move two instructions from before the branch delay slots into the slots, Irrespective of the branch outcome our two cycles would be utilized completely, resulting in no stalls.

Total Stalls = 0
Total cycles taken to complete 1000 instructions = 1000 + 0 = 1000

Average CPI = No. Of Cycles/ No. Of Instructions = (1000 / 1000)
= **1 CPI (Ideal case)**

3. You are able to move two instructions from the taken block into the delay slots.

Soln:

Since total instructions assumed is 1000
Branch Instructions = 15% * 1000 = 150

Since we are able to move two instructions from the taken block into the branch delay slots, We are right 90% of the time and would have stalls for the remaining 10% of the branching instructions.
In our case, We are wasting cycles for 10%*150 = 15 instructions.
Since no. of wasted stages per instruction is 2 stalls

Total Stalls = 15 * 2 = 30
Total cycles taken to complete 1000 instructions = 1000 + 30 = 1030

Average CPI = No. Of Cycles/ No. Of Instructions = (1030 / 1000)
= **1.03 CPI**

4. You are able to move two instructions from the not-taken block into the delay slots.

Soln:

Since total instructions assumed is 1000
Branch Instructions = 15% * 1000 = 150

Since we are able to move two instructions from the not-taken block into the branch delay slots, We are right only 10% of the time and would have stalls for the remaining 90% of the branching instructions.
In our case, We are wasting cycles for 90%*150 = 135 instructions.
Since no. of wasted stages per instruction is 2 stalls

Total Stalls = 135 * 2 = 270
Total cycles taken to complete 1000 instructions = 1000 + 270 = 1270

Average CPI = No. Of Cycles/ No. Of Instructions = (1270 / 1000)
= **1.27 CPI**

3. Deep Pipelines (40 pts)

Given:

Total time for unpipelined = 8ns
Latch overhead = 0.2 ns
PoP and PoC are separated by 4ns
No. Of independent instructions = 1/3rd
No. Of dependent instructions = 2/3rd

I. An Unpipelined Processor

Soln:

Total time taken for 1 instruction = T + Tovh
= 8 + 0.2
= 8.2 ns

Throughput = 1 / Total time taken per instruction
= 1 / (8.2 * 10^-9) s
= **0.122 BIPS**

II. A 10-stage pipeline

Soln:

- Independent Instructions

For Independent instructions the gap between two instructions is 1 stage, each instruction takes 1 stage
1 stage = 1 cycle
Gap between instructions (Total Time per instruction) = (T/n) + Tovh
= 8/10 + 0.2
= 1ns

- Dependent Instructions

For dependent instructions, The gap between PoP and PoC is given as 4ns
Since each instruction if they were independent takes 0.8ns (8/10)
Total gap in cycles = 4/0.8 = 5 cycles
Total time per instruction = 5 * 1 (Time per stage independent)
= 5ns

Total execution time per instruction = (1/3)* 1 + (2/3) * 5
= 0.33 + 3.33
= 3.67 ns

Throughput = 1 / Total time taken per instruction
= 1 / (3.67 * 10^-9) s
= **0.273 BIPS**

III. A 20-stage pipeline

Soln:

- Independent Instructions

For Independent instructions the gap between two instructions is 1 stage, each instruction takes 1 stage
1 stage = 1 cycle
Gap between instructions (Total Time per instruction) = (T/n) + Tovh
= 8/20 + 0.2
= 0.6ns

- Dependent Instructions

For dependent instructions, The gap between PoP and PoC is given as 4ns
Since each instruction if they were independent takes 0.4ns (8/20)
Total gap in cycles = 4/0.4 = 10 cycles

Total time per instruction = 10 * 0.6 (Time per stage independent)
= 6 ns

Total execution time per instruction = (1/3)* 0.6 + (2/3) * 6
= 0.2 + 4
= 4.2 ns

Throughput = 1 / Total time taken per instruction
= 1 / (4.2 * 10^-9) s
= **0.238 BIPS**