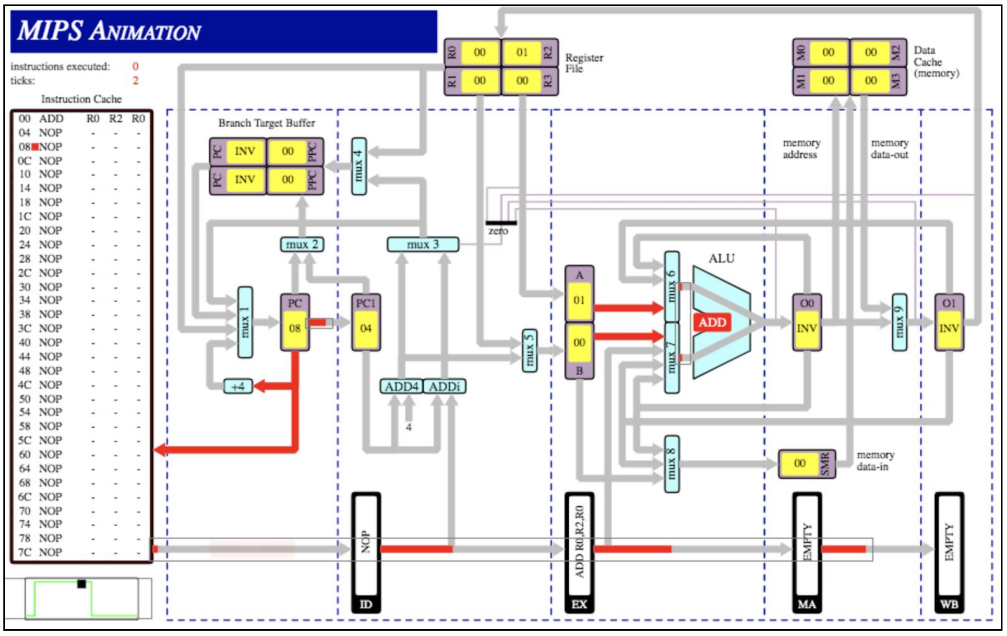
CS3021 Computer Architecture II - Tutorial 4

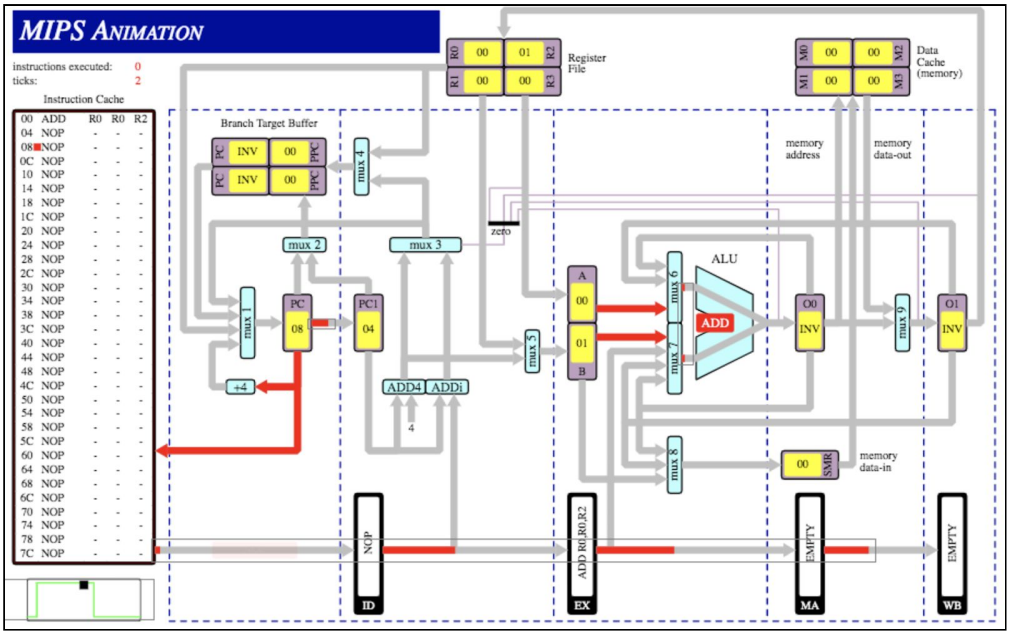
Student # 16327446 - Brandon Dooley

1) DLX/MIPS Processor - Instruction Screenshots

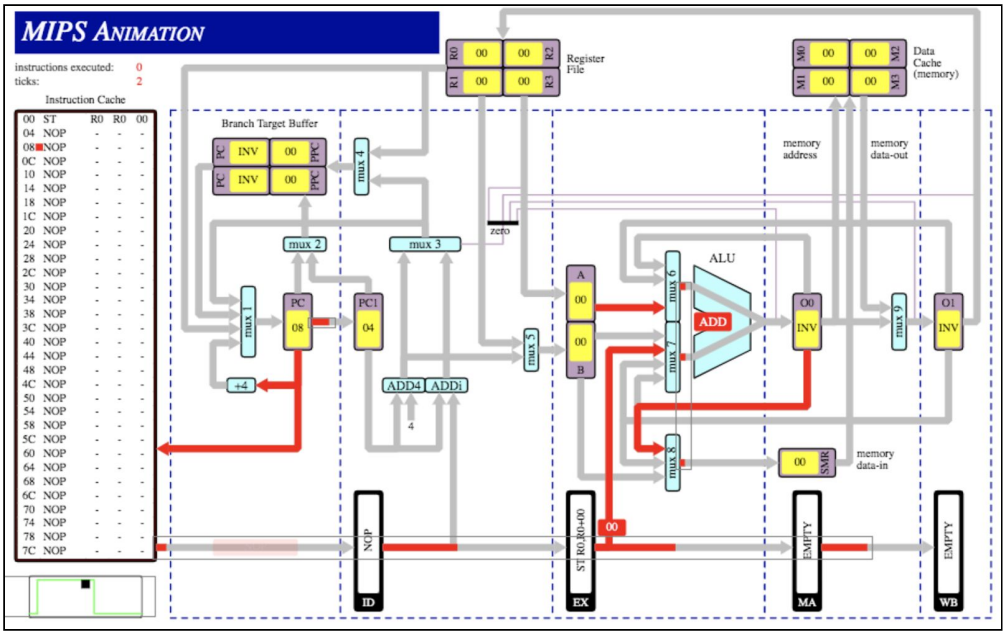
[1] - 01 to MUX6



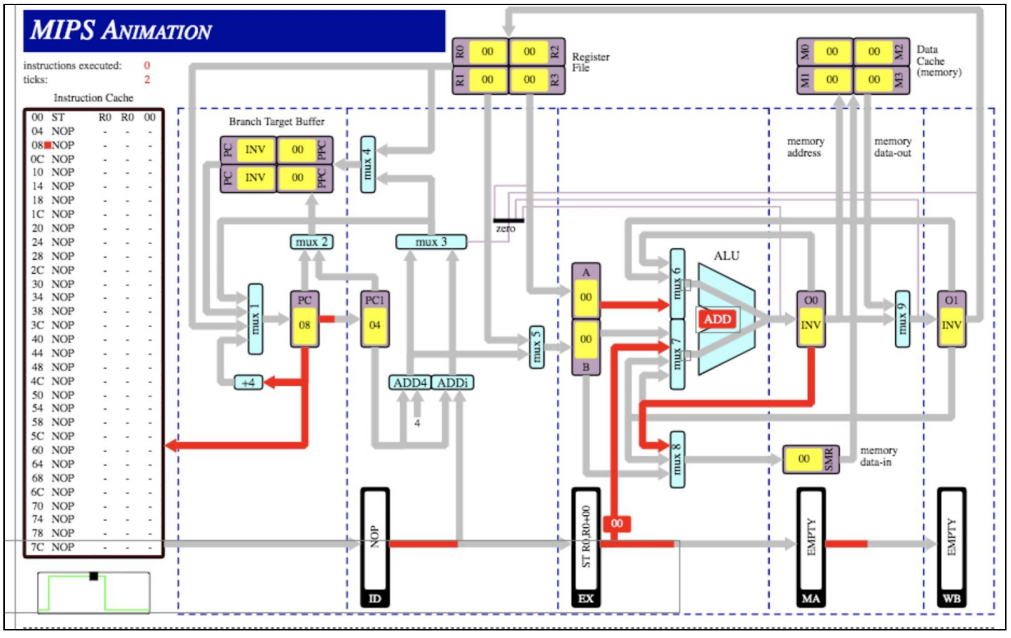
[2] - 00 to MUX7 and 01 to MUX6 (simultaneously)



[3] - 00 to MUX8



[4] - EX to MUX7



2) DLX/MIPS Processor - ALU Fowarding Tests

[1] - ALU Fowarding Enabled

- **Resulting Value of R1** = 15
- **Number of Clock Cycles** = 10
- **Instructions Executed** = 6

With ALU Forwarding enabled there is a total of 10 clock cycles (6 instructions + 4-stage pipeline delay). Having ALU Forwarding enabled allows the value of a previous ALU function to be passed directly back into the ALU as an input for the next instruction rather than having to store it back in memory in a register file and retrieve it again. This reduces the need for extra ST and LD instructions to be executed to retrieve the value of a register and allows the pipeline to flow smoothly and without stalls.

[2] - ALU Fowarding Disabled

- **Resulting Value of R1** = 15
- **Number of Clock Cycles** = 18
- **Instructions Executed** = 6

With ALU Forwarding disabled there is a total of 18 clock cycles. This can be broken down into 6 instructions resulting in 6 clock cycles, a 4-stage pipeline delay of 4 clock cycles and 2 x STALL instructions for each arithmetic instruction after the first instruction (2 x 4). Having ALU Forwarding disabled means that the result of each arithmetic operation must first be passed back into the register file before the next arithmetic operation can be performed. This requires STALL's to be implemented in the pipeline whilst the resultant value is being stored back and then loaded from the register file again.

The results between **[1]** and **[2]** are the same (15) as they perform the exact same arithmetic operations just with performance differences as a result of STALL's being needed when ALU Forwarding is disabled.

[3] - ALU Fowarding Enabled & CPU Data Dependency Interlock Disabled

- **Resulting Value of R1** = 6
- **Number of Clock Cycles** = 10
- **Instructions Executed** = 6

With ALU Forwarding enabled & CPU Data Dependency Interlock disabled there are a total of 10 clock cycles (6 instructions + 4-stage pipeline delay). However, by disabling CPU Data Dependency Interlock the system does not predict/realise that the next instruction in the pipeline depends on the result of the previous instruction and continues on with the arithmetic. It performs the arithmetic using the old/original values in the register file that have not yet been updated by the result of the previous arithmetic operation. As a result of this the output at the end of the execution is incorrect.

3) DLX/MIPS Processor - Multiplication Program

[1] - Branch Prediction (Original)

- **Instructions Executed =** 39
- **Clock Cycles =** 51

With Branch Prediction enabled the processor tries to guess/predict which way a branch will go before this is known definitively. In the given multiplication program this means that the jump at the end of the loop will only need to stall on the first occurance. This stall is used to calculate the offset of the necessary jump instruction and to store the result within the branch target buffer for later use. Another 4 of the additional clock cycles are produced as a result of the 4-stage pipeline delay. The multiplication loop is also executed 4 times which executes the LD instruction 4 times producing 4 STALL's.

The inner BEQZ's are true twice during the execution and as a result act as a jump operation and thus need to calculate the offset of the necessary jump. This produces a further 2 STALL's. Similarly, the outer BEQZ is true once performing the same STALL and giving us our final extra clock cycle. In summary the additional clock cycles can be described as follows:

12 Addtional Clock Cycles:

- 4-Stage Pipeline Delay (x4)
- LD Instructions (x4)
- J Instructions (x1)
- BEQZ Instructions (x3)

[2] - Branch Interlock

- **Instructions Executed =** 39
- **Clock Cycles =** 53

By enabling Branch Interlock this produces a further 2 clock cycles than the previous implementation using branch prediction. This is as a result of moving the branch target buffer which allows the processor to cache/save destinations/offsets of precalculated branch destinations. As a result the processor must now calculate the jump offset every iteration it is called within the loop (4 times) rather than using the branch target buffer to access the previously calculated offset. Similar to the previous explanation there are also 4 x LD instructions which produce 4 stalls and a 4-stage pipeline delay producing 4 more clock cycles.

14 Addtional Clock Cycles:

- 4-Stage Pipeline Delay (x4)
- LD Instructions (x4)
- J Instructions (x4)
- BEQZ Instructions (x2)

[3] - Branch Prediction Enabled (Swapped Shift Instructions)

- **Instructions Executed =** 39
- **Clock Cycles =** 47

By swapping the shift operations there is no longer a CPU Data Dependency between the two operations *LD R2, R0, 00* and *SRLi R2, R2, 01*. Previously, the processor had to implement a stall once every iteration of the loop (x4) when the *LD R2* operation was followed by *SRLi R2* as the processor had to wait for the value of R2 before it could perform its arithmetic. By swapping the shift operations *LD R2, R0, 00* is now followed by *SLLi R3, R3, 01* thus removing this dependency and eliminating the extra 4 stalls and reducing the clock cycles down to 47.

8 Total Addtional Clock Cycles:

- 4-Stage Pipeline Delay (x4)
- ~~LD Instructions (x4)~~
- J Instructions (x1)
- BEQZ Instructions (x3)