3 Multiplier Microarchitecture

	Microarchitecture	Num Trans (#)	Cycle Time (τ)	Transaction Latency (cyc)	Transaction Latency (τ)	Transaction Throughput (trans/cyc)	Transaction Throughput (cyc/trans)	Total Execution Time (τ)
(a)	1-Cycle	60	62	1	62	1	1	3720
(b)	Iterative	60	20	4	60	0.25	4	3600
(c)	2-Cycle Unpiplined	60	32	2	64	0.5	2	3840
(c)	2-Cycle Pipelined	60	32	2	64	0.98	1.02	1952
(d)	4-Cycle Unpiplined	60	17	4	68	0.25	4	4080
(d)	4-Cycle Piplined	60	17	4	68	0.95	1.05	1071
(e)	Var-Lat Pipelined	60	20	1-5	20-100	0.59	1.7	2040

Figure 4: Evaluation of Various Multiplier Microarchitectures

3.a Iterative Microarchitecture

	Transaction	Cycle 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	`mul 0xdeadbeef, 0xf`	Z	Z	Z	Z												
2	`mul 0xf5fe4fbc, 0x7`					Z	Z	Z	Z								
3	`mul 0x0a01b044, 0x3`									Z	Z	Z	Z				
4	`mul 0xdeadbeef, 0xf`													Z	Z	Z	Z

Figure 5: Iterative Multiplier Transaction vs Time Diagram

3.b Two-Cycle Microarchitecture

	Transaction	Cycle 1	2	3	4	5	6	7	8
1	`mul 0xdeadbeef, 0xf`	X0	X1						
2	`mul 0xf5fe4fbc, 0x7`			X0	X1				
3	`mul 0x0a01b044, 0x3`					X0	X1		
4	`mul 0xdeadbeef, 0xf`							X0	X1

Figure 6: 2 Cycle Unpipelined Transaction vs Time Diagram

	Transaction	Cycle 1	2	3	4	5
1	`mul 0xdeadbeef, 0xf`	X0	X1			
2	`mul 0xf5fe4fbc, 0x7`		X0	X1		
3	`mul 0x0a01b044, 0x3`			X0	X1	
4	`mul 0xdeadbeef, 0xf`				X0	X1

Figure 7: 2 Cycle Pipelined Transaction vs Time Diagram

3.c Two-Cycle Microarchitecture

	Transaction	Cycle 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	`mul 0xdeadbeef, 0xf`	X0	X1	X2	X3												
2	`mul 0xf5fe4fbc, 0x7`					X0	X1	X2	X3								
3	`mul 0x0a01b044, 0x3`									X0	X1	X2	X3				
4	`mul 0xdeadbeef, 0xf`													X0	X1	X2	X3

Figure 8: 4 Cycle Unpipelined Transaction vs Time Diagram

	Transaction	Cycle 1	2	3	4	5	6	7
1	`mul 0xdeadbeef, 0xf`	X0	X1	X2	X3			
2	`mul 0xf5fe4fbc, 0x7`		X0	X1	X2	X3		
3	`mul 0x0a01b044, 0x3`			X0	X1	X2	X3	
4	`mul 0xdeadbeef, 0xf`				X0	X1	X2	X3

Figure 9: 4 Cycle Pipelined Transaction vs Time Diagram

3.d Variable-Latency Microarchitecture

	Transaction	Cycle 1	2	3	4	5	6	7	8	9	10
1	`mul 0xdeadbeef, 0xf`	Y	X0	X1	X2	X3					
2	`mul 0xf5fe4fbc, 0x7`		Y	Y	X1	X2	X3				
3	`mul 0x0a01b044, 0x3`				Y	Y	X2	X3			
4	`mul 0xdeadbeef, 0xf`						Y	X0	X1	X2	X3

Figure 10: Variable-Latency Microarchitecture Transaction vs Time Diagram

3.e Comparison of Microarchitectures

The 4 cycle pipelined architecture has the highest performance. It provided the best transaction throughput and therefore lowest total execution time. The 4 cycle pipelined multiplier allowed us to execute 4 multiply instructions at the same time provided that there are no data dependencies between them. For this specific type of mlutiplier (32 bit operand by 4 bit operand), anything more than a 4 cycle pipeline will not provide any benefits. The best performance is achieved when the number of cycles in the pipeline is equal to the number of bits in the second operand (ie. a 32 bit by 16 bit multiplier will have a 16-cycle pipeline). The variable latency pipelined multiplier can provide the same transaction throughput as the 4-cycle pipeline and better transaction latency, but only for certain data sets and instruction ordering (ie. a continuous stream of data that only needs the lowest 2 bits). Due to the necessity to stall in the Y stage, the variable latency multiplier resulted in a worse throughput and latency when compared to the 4-cycle pipeline in our tests. With a larger second operand (of n bits) and a more random data set, the variable latency multiplier will have roughly the same transaction throughput as the n-cycle pipeline, but a slightly better transaction latency.

In terms of area, the fixed latency multiplier will have the smallest area while the variable latency pipelined multiplier will have the largest area. The 4-cycle pipelined multiplier, with the second largest area, provided the best performance in this test due to the uniqueness of this data set. I believe that the area difference between these different architectures is not significant because they only involve the addition of registers and muxes, while the amount of combinational logic for arithmetic purposes remained roughly the same (same number of 32-bit adders and shifters).

While the 4 cycle pipelined multiplier provided the best throughput, it also had one of the worst latency. Therefore, if there are data dependencies between a succession of transactions, a fixed latency or variable latency pipelined multiplier will have provided better performance due to their lower transaction latency. In the case of the variable latency multiplier, the performance will vary depending on the data set as stated before.

Overall, for a 32-bit by 4-bit multiplier, the 4 cycle pipelined multiplier will provide the best performance in general. With good software scheduling, we can minimize the penalty for data dependencies and therefore maintain a

max throughput. However, if we wish to implement a larger multiplier such as a 32-bit by 16-bit multiplier, we should choose the variable-latency pipelined multiplier instead. With the larger operand, there are more upper bits that we can potentially skip, therefore the performance benefits of skipping them are also greater, especially when there are data dependencies. In that case, it is likely to outperform a 16-cycle pipelined multiplier.