

Section 1: Book Problems

The Blue non-revised 4th edition was used

6.3.1)

a)

$$Average = seekrate + \frac{.5}{rps} + \frac{data}{rate}$$

$$Average = 11ms + \frac{.5}{7200} + \frac{1024}{34 * 2^{20}}$$

$$Average = 15.1954 ms$$
b)

$$13.1992 ms$$

$$Average = seekrate + \frac{.5}{rps} + \frac{data}{rate}$$

$$Average = 9ms + \frac{.5}{7200} + \frac{1024}{30 * 2^{20}}$$

$$Average = 13.1992 ms$$
6.3.2)
a) *minimum assumes the case where there is no seek time or rotational delay.

$$minimum = \frac{dada}{rate}$$

$$minimum = \frac{2048}{34 * 2^{20}}$$
b)

$$minimum = \frac{dada}{rate}$$

$$minimum = \frac{2048}{30 * 2^{20}}$$

6.15.1)

a)

b)

parity =	FEFE xor A387 xor F345 xor FF00
	parity = 513C

minimum = .03255

6.15.2)

a)

6.15.3)

Raid 4 is more efficient with small reads, because a single block can be accessed from just 1 disk instead of all the disks like raid 3 requires. Raid 4 also requires less reads to build a new parity block, making writing more efficient. Raid 3 has no real advantage over raid 4.

6.15.4)

Raid 5 constructs its parity blocks the same as raid 4, but it distributes them across the disks instead of storing them all on one disk. This prevents the parity disk from bottlenecking the performance during back to back writes. A single write will not see any improvement.

6.18.1)

a)

$$AFR = \frac{\frac{hours}{drive}}{hours}$$

$$AFR = \frac{8760}{1000000}$$

$$AFR = .876\%$$

$$failed \ disks = \frac{\left(drives * \left(\frac{hours}{drive}\right)\right)}{\frac{hours}{failure}}$$

$$failed \ disks = \frac{\left(1000 * (8760)\right)}{1000000}$$

$$failed \ disks = 8.76$$

b)

$$AFR = \frac{\frac{hours}{drive}}{hours}$$

$$AFR = \frac{10512}{1500000}$$

$$AFR = .7008\%$$

$$failed \ disks = \frac{\left(drives * \left(\frac{hours}{drive}\right)\right)}{\frac{hours}{failure}}$$

$$failed \ disks = \frac{\left(1000 * (10512)\right)}{1500000}$$

$$failed \ disks = 7.008$$

6.18.2)

a)

$$failed \ disks(7) = FR * (1.6667 + 3 + 2 + 4 + 8)$$

failed \ disks(7) = 159.14
failed \ disks(10) = FR * (1.6667 + 3 + 2 + 4 + 8 + 16 + 32 + 64)
failed \ disks(10) = 1140.26

$$failed \ disks(7) = FR * (1.6667 + 3 + 2 + 4 + 8)$$

failed \ disks(7) = 127.312
failed \ disks(10) = FR * (1.6667 + 3 + 2 + 4 + 8 + 16 + 32 + 64)
failed \ disks(10) = 912.208

Section 2: Cacti and SESC simulations

Block Size Tradeoffs

The first simulation involved testing different block sizes for the L1 instruction cache. The two charts below summarize the results from the experiment. The first chart shows the different miss rates depending on the block size, while the second chart shows the change in execution time. The miss rate makes a significant drop right away, with the decline becoming more gradual as time passes. This indicates that the most gain per area will be in the smaller block sizes, with the large ones offering similar miss rates. The execution time follows a different trend. As the block size increases, the execution time starts to go down but turns back up before long. This is caused by the larger latencies that cacti provided for the larger block sizes.





The second simulation involved testing different block sizes of the L1 data cache. The two charts below summarize the results from the experiment. The first chart shows the different miss rates depending on the block size, while the second chart shows the change in execution time. The results are significantly different from before. Increasing the block size actually increase the hit rate. This means the data cache does not have the same type of locality as the instruction cache. The fact that a larger block size decreases the total number of blocks available in the table (a fixed total cache size was used) could also be having a significant effect. The second chart provides some unexpected results. The execution time dips down in the beginning, just like in the previous example even though the miss rate is increasing. This is likely a byproduct of increasing the L2 cache at the same rate as the data cache. This brings the execution time gains from the previous simulation into question. Having the L2 cache independently set to a larger block size was causing SESC to lock up, meaning the configuration file was likely in error. Meaningful information can still be determined from the cache miss charts.





Cache Associativity Tradeoffs

This simulation involves simulating different cache associativities for the L1 data cache. The first chart shows the different miss rates depending on the associativity, while the second chart shows the change in execution time. Like in the first case, there is a significant initial drop in miss rate which then levels off as the associativity increases further. The case with an associativity of 32 has almost the same hit rate as the fully associativite case. The same trend can be seen in the execution time as was present in the previous simulations. As the associativity gets larger and larger, the cache latency increases. Considering that the hit rate decrease is negligible for the larger cases, it's only natural that the execution time should rise sharply.





Cache Size Tradeoffs

This simulation involves simulating different cache sizes for the L1 instruction cache. The first chart shows the different miss rates depending on the size, while the second chart shows the change in execution time. The first chart follows the same trend as two of the previous simulations. The hit rate starts out by dropping significantly, and then leveling off. The third test case is approximately as efficient as the largest. The execution time never actually drops however. The change in miss rate never outweighs the hefty increase in cache latency that is associated with increasing the cache size.





Overall Cache Performance Experiment

This simulation involves simulating three different test cases based of the previous results. Each test case represents an increase in cache latency, while the attributes are maximized to say within that latency. The data cache was first given an increase in associativitity, because it caused the most significant drop in the trials. The instruction cache was first made larger and given a larger block size, because those two attributes made the biggest changes in the previous simulations. The table below summarizes the 3 configurations used, while the chart below shows the relative execution times. The first case was the fastest by far. The smaller cache latencies must have been more important than the increased hit rates that went along with them. The fact that the miss rates were already so low (almost always below 5%) and the miss penalty is not too severe gave the configuration with the smaller latency a clear advantage.

Test	Data	Data Cache	Data Cache	Data Cache	Instruction	Instruction	Instruction	Instruction
Case	Cache Size	Associativity	Block Size	Latency	Cache Size	Associativity	Block Size	Latency
1	32768	8	32	4	65536	2	256	4
2	131072	16	32	5	65536	2	512	5
3	65536	32	32	6	262144	2	512	6



Section 3: Implementing a Simple Data Cache

This section deals with the addition of a basic two way associativite cache for the data memory of the pipelined MIPS processor designed in the previous lab. A write through policy and no-write allocate policy were used, so the cache only effects reads. The diagram below represents the cache system without the control logic. Whenever a write occurs, the data is written directly to ram (and updated in the cache if that address is present) so very little has changed during write cycles. Read cycles are controlled by a simple FSM. Whenever a cache miss, The FSM begins a 4 cycle process that reads each of the 4 words bytes associated with that address into one of the two possible locations that data can be present in the cache. The location is randomly chosen. During this process, the rest processor is stalled. When the data is done being read into the cache, the processor resumes. The total time needed to read the data into the cache is 4 cycles, which is 3 more than the single cycle needed to read from the cache.

The point of this cache is to reduce the effect of high memory latency. This could be very use full in a system where the processor runs much faster than its main working memory, or when the memory has a relatively long latency. The current implementation on the FPGA would likely see no benefit and only a performance decrease because the memory is capable of running at the same rate as the CPU with no latency. It is likely that other technologies exit where even a simple cache system like this would result in a significant increase in performance (though a write buffer would be necessary to speed up the write process).



Simulations

Several different simulations are below that show various cache utilizations.

Case 1: Cache write through.

Case 1: Cac	che wri	te through.			
clk	1				
PC	004000	00400010	00400014	00400018	0040001C
instruction	214A00	A14A0000	214A0001	1541FFFD	3C0A1000
ra2(2)	000000	0000000			
ra2(3)	000000	0000000			
ra2(4)	000000	0000000			
ra2(5)	000000	0000000			
ra2(10)	100000	10000002		1000003	
ra2(19)	000000	000000F			
mem_stall	0				
data	{000000	000000000000000000000000000000000000000		0000003-{000000	
ra2	{00000	{0000000}} {	0000000} {00000		00000000} {D
valid	00	00			
address	000000	0000000			
wren	0				
DM_in	000000	80808080	0000000	03030303	
byteena	0000	0000		1000	000
		The instruct through op the cache, The highlig \$10 into the signals as s	tion above performants eration. The dates to so no data has to hted instruction and dress point seen by the RAM	orms a simple ta location is to be changed is storeing the ed to by \$10. VI are highligh	: write not in I there. : value in The ted

Case 2: Cache Miss

🔷 dk	1										
	004000	100400020	00400024	00400028				0040002C	00400030		
	081000	8D420000	9143000F	9144007E				9145007F	A153007C		
-🔷 ra2(2)	030201	0000000								03020100	
-🔶 ra2(3)	000000	0000000									
-🔷 ra2(4)	000000	0000000									
-🔶 ra2(5)	000000	0000000									
-🔶 ra2(10)	100000	1000007F		10000080					10000000		
-🔶 ra2(19)	000000	000000F									
🔶 mem_stall	0										
-🔶 data(0)	0F0E0D	XXXXXXXXXXXXXX			XXXXXXXX	XXXXXXXX	XXXXXXXXX	0F0E0D0C0	BOA0908070	6050403020	100
-🔶 data	{7F7E71	{\xxxxxxxxxxxxxxx		xxxxxx	{XXXXXXX	{XXXXXXX	{XXXXXXX	<u>{xxxxxxxxxxx}</u>	XXXXXXXXX	<u>{}xxxxxxx</u>	{XXX
-🔶 ra2	{00000	{0000000} {	0000000	{00000000}	{00000000}	{00000000	} {00000000	} {00000	{000000	{00000000}	{000
📥 valid	81	00			01					81	
- ddress	000111	00011111		00000000	00000001	00000010	00000011		00011100	00011101	0001
📥 wren	0										
- DM in	000000	00800080	10000080	00000000							
	0000	0000		11111				0000	11111		
		This instruc	tion load	the word	etored at \$	10 and m	nte it into 9	2 The s	tall signal	can he see	
								1 1 1			
		going high	n cycle 2	for three of	ycles to g	ive the cad	the time to	load the	data. The	e data set	can
		be seen full	y loaded i	nto the ca	che in cyc	le 7. The	output int	o register	2 is also b	ighlighted	
		The total o	peration ta	akes 8 cvo	les instead	l of the re	mlar 5				
		The total o	perauona	inces o cyc	acs motout	I OF LIC IC	Baia 2.				

Case3: Cache Hit

	▶ dk	0											
-	PC	004000		0040002	2C	00400030				00400034			
->	instruction	081000		9145007	7F	A153007C				0810000D			
-	ra2(2)	030201	00	000000			03020100						
->	ra2(3)	000000	00	000000							0000000F		
->	> ra2(4)	000000	00	000000								0000007E	
->	> ra2(5)	000000	00	000000									0000007F
->	ra2(10)	100000	10	000080		10000000							
->	ra2(19)	000000	00	00000F									
	mem_stall	0											
->	data(7)	7F7E7D	XХ	XXXXXX	XXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	7F7E7D7C7	87A7978	7F7E7D0F76	37A7978777
->	🕨 data	{7F7E7		{XXXXXX	XXX	XXXXXXXX	{XXXXXX	{XXXXXXX	{XXXXXXX	{7F7E7D7C	/B7A797	{7F7E7D0F7	B7A797877
->	ra2	{00000	{0	0000000)	{000000	{00000000}	{00000000}	{00000000	} {00000	{000000	{000000	{00000000}
->	valid	81	01				81						
-	address	000111	00	000011		00011100	00011101	00011110	00011111				
4	wren	0											
-	DM_in	000000	00	000000							0F0F0F0F	00000000	
-	byteena	0000		0000		1111				0000	0001	0000	
			Γ										
				_									
				Here, a	ı ca	che hit ca	n be obse	rved, thou	gh there is	still a sta	l due to th	e fact that	the
			i	nstruct	ion	was fetch	ed before	the stall fi	om the pr	evious ins	truction th	at filled th	s
				cache ł	oloc	k was im	plemented	The inst	ruction loa	ads the va	ule stored	in \$10 +	127
				nto \$5	т	he datas l	ocation in	the cache	is highligh	ted as m	all as the o	autrout	
				nto 95.	. 1	ne datas r		uie eache	is ingingi	icu, as w	uic uic u	uipui.	

Case 4: Cache Write

	▶ dk	0										
->	PC	004000	0040003	0			00400034					
-	instruction	081000	A153007	7C			0810000D					
-	ra2(2)	030201	0000	03020100								
->	ra2(3)	000000	0000000	0				0000000F				
-	ra2(4)	000000	0000000	0					0000007E			
->	ra2(5)	000000	0000000	0						0000007F		
->	ra2(10)	100000	1000000	0								
->	ra2(19)	000000	0000000	F								
	mem_stall	0										
-	data(7)	7F7E7D	XXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	7F7E7C 7C	B7A7978	7F7E7067	37A7978777	6757473727	170
->	🕨 data	{7F7E7	{XXX	{XXXXXXX	{XXXXXXX	{XXXXXXX	{7F7E7D7C	7B7A797	{7F7E7D0F7	B7A797877	7675747372	7170}
-	ra2	{00000	{000	{00000000}	{00000000}	{00000000	} {00000	{000000	{000000	{00000000}	{00000000}	{000}
-	valid	81	01	81								
-3	address	000111	0001	00011101	00011110	00011111						
4	wren	0										
-3	DM_in	000000	0000000	0				0F0F0F0F	0000000			
-) byteena	0000	1111				0000	0001	0000			
			This	nstruction	stores the	contents	from regi	ster \$19 in	to an add	ress curre	ntly stored	
			in ca	he. The	number no	t only get	s written o	irectly to	RAM, bu	t also to th	e cache.	
			The o	hange in t	he cache	s highlight	ed, as we	as the w	rite signals	s that go to	the ram.	
			771		C1	11	.1	11	1	·	1	
			Ine	nstruction	was fetch	ed durring	g the stall	caused by	a cache n	uss, so the	eiong	
			delay	durring t	he instruct	ions fetch	phase is t	mrelated t	o the instr	action itse	f.	