3 Impact of Cache Access Time and Replacement Policy

3.a Miss Rate Analysis

Transaction Address	tag	idx	m/h	L0	L1	L2	L3	L4	L5	L6	L7
0x024	0x0	0x2	m	-	-	-	-	-	-	-	-
0x030	0x0	0x3	m			0x0					
0x07c	0x0	0x7	m				0x0				
0x070	0x0	0x7	h								0x0
0x100	0x2	0x0	m								
0x110	0x2	0x1	m	0x2							
0x204	0x4	0x0	m		0x2						
0x214	0x4	0x1	m	0x4							
0x308	0x6	0x0	m		0x4						
0x110	0x2	0x1	m	0x6							
0x114	0x2	0x1	h		0x2						
0x118	0x2	0x1	h								
0x11c	0x2	0x1	h								
0x410	0x8	0x1	m								
0x110	0x2	0x1	m		0x8						
0x510	0xa	0x1	m		0x2						
0x110	0x2	0x1	m		0xa						
0x610	0xc	0x1	m		0x2						
0x110	0x2	0x1	m		0xc						
0x710	0xe	0x1	m		0x2						
Number of Misses = 16											
Miss Rate = 0.8											

Figure 5: Direct-Mapped Cache Contents Over Time

Transaction				Set 0		Set 1		Set 2		Set 3	
Address	**tag**	**idx**	**m/h**	**Way 0**	**Way 1**	**Way 0**	**Way 1**	**Way 0**	**Way 1**	**Way 0**	**Way 1**
0x024	0x0	0x2	m	-	-	-	-	-	-	-	-
0x030	0x0	0x3	m					0x0			
0x07c	0x1	0x3	m							0x0	
0x070	0x1	0x3	h								0x1
0x100	0x4	0x0	m								
0x110	0x4	0x1	m	0x4							
0x204	0x8	0x0	m			0x4					
0x214	0x8	0x1	m		0x8						
0x308	0xc	0x0	m				0x8				
0x110	0x4	0x1	h	0xc							
0x114	0x4	0x1	h								
0x118	0x4	0x1	h								
0x11c	0x4	0x1	h								
0x410	0x10	0x1	m								
0x110	0x4	0x1	h				0x10				
0x510	0x14	0x1	m								
0x110	0x4	0x1	h				0x14				
0x610	0x18	0x1	m								
0x110	0x4	0x1	h				0x18				
0x710	0x1c	0x1	m								
Number of Misses = 12											
Miss Rate = 0.6											

Figure 6: Two-Way Set-Associative Cache Contents Over Time with LRU Replacement

Transaction				Set 0		Set 1		Set 2		Set 3	
Address	**tag**	**idx**	**m/h**	**Way 0**	**Way 1**	**Way 0**	**Way 1**	**Way 0**	**Way 1**	**Way 0**	**Way 1**
0x024	0x0	0x2	m	-	-	-	-	-	-	-	-
0x030	0x0	0x3	m					0x0			
0x07c	0x1	0x3	m							0x0	
0x070	0x1	0x3	h								0x1
0x100	0x4	0x0	m								
0x110	0x4	0x1	m	0x4							
0x204	0x8	0x0	m			0x4					
0x214	0x8	0x1	m		0x8						
0x308	0xc	0x0	m				0x8				
0x110	0x4	0x1	h	0xc							
0x114	0x4	0x1	h								
0x118	0x4	0x1	h								
0x11c	0x4	0x1	h								
0x410	0x10	0x1	m								
0x110	0x4	0x1	h				0x10				
0x510	0x14	0x1	m								
0x110	0x4	0x1	m			0x14					
0x610	0x18	0x1	m								
0x110	0x4	0x1	m				0x18				
0x710	0x1c	0x1	m								
Number of Misses = 14											
Miss Rate = 0.7											

Figure 7: Two-Way Set-Associative Cache Contents Over Time with FIFO Replacement

3.b Sequential Tag Check then Memory Access

Component	Delay Equation	Delay(τ)
addr_reg_M0	1	1
tag_decoder	3 + 2×2	7
tag_mem	10 + [(4+27)/16]	12
tag_cmp	3 + 2[log2(26)]	13
tag_and	2 - 1	1
data_decoder	3 + 2×3	9
data_mem	10 + [(8+128)/16]	19
rdata_mux	3[log2(4)] + [32/8]	10
rdata_reg_M1	1	1
Total		**73**
addr_reg_M0	1	1
tag_decoder	3 + 2×2	7
tag_mem	10 + [(4+27)/16]	12
tag_cmp	3 + 2[log2(26)]	13
tag_and	2 - 1	1
data_decoder	3 + 2×3	9
data_mem	10 + [(8+128)/16]	19
Total		**62**

Figure 8: Critical Path and Cycle Time for 2-Way Set-Associative Cache with Serialized Tag Check before Data Access

The reason that the 2-way set-associative microarchitecture is slower than the direct-mapped microarchitecture is the need for the tag check result to go through the data_decoder. It happens that the data_decoder?s delay is relatively significant (9τ). This connection is needed so that the data can be outputted from the correct way.

3.c Parallel Read Hit Path

Component	Delay Equation	Delay(τ)
addr_reg_M0	1	1
addr_mux	3[log2(2)] + [5/8]	4
data_decoder	3 + 2×3	9
data_mem	10 + [(8+128)/16]	19
rdata_mux	3[log2(4)] + [32/8]	10
rdata_reg_M1	1	1
Total		**44**

Figure 9: Critical Path and Cycle Time for Direct Mapped Cache with Parallel Read Hit

Component	Delay Equation	Delay(τ)
addr_reg_M0	1	1
addr_mux	3[log2(2)] + [5/8]	4
data_decoder	3 + 2×2	7
data_mem	10 + [(8+128)/16]	19
rdata_mux	3[log2(4)] + [32/8]	10
way_mux	3[log2(2)] + [32/8]	7
rdata_reg_M1	1	1
Total		**49**

Figure 10: Critical Path and Cycle Time for 2-Way Set-Associative Cache with Parallel Read Hit

The reason that the 2-way set-associative microarchitecture is slower than the direct-mapped microarchitecture is the way_mux, which is needed to output the data from the correct way. This mux has a delay of 7τ, which is relatively significant.

3.d Pipelined Write Hit Path

Component	Delay Equation	Delay(τ)
addr_reg_M0	1	1
tag_decoder	3 + 2×3	9
tag_mem	10 + [(8+26)/16]	13
tag_cmp	3 + 2[log2(25)]	13
tag_and	2 - 1	1
wen_and	2 - 1	1
wen_reg_M1	1	1
Total		**39**

Figure 11: Critical Path and Cycle Time for Direct Mapped Cache with Pipelined Write Hit

Component	Delay Equation	Delay(τ)
addr_reg_M0	1	1
tag_decoder	3 + 2×2	7
tag_mem	10 + [(4+27)/16]	12
tag_cmp	3 + 2[log2(25)]	13
tag_and	2 - 1	1
wen_and	2 - 1	1
wen_reg_M1	1	1
Total		**36**

Figure 12: Critical Path and Cycle Time for 2-Way Set-Associative Cache with Pipelined Write Hit

The reason that the direct-mapped microarchitecture is slower than the 2-way set-associative microarchitecture is the larger tag_decoder, which is needed to fetch the correct tag from the tag memory. Because the direct mapped cache uses a 3 to 8 decoder instead of a smaller 2 to 4 decoder of the 2-way set associative cache, the critical path is longer. There is also 1τ extra delay for the tag_mem due to it being 8 rows instead of 4 rows in the 2-way set-associative cache.

3.e Average Memory Access Latency

Associativity	μarch	Replacement Policy	Hit Time (τ)	Miss Rate (ratio)	Miss Penalty (τ)	AMAL (τ)
Direct Mapped	Seq	n/a	68	0.8	300	308
2-way Set Assoc	Seq	LRU	73	0.6	300	253
2-way Set Assoc	Seq	FIFO	73	0.7	300	283
Direct Mapped	PP	n/a	44	0.8	300	284
2-way Set Assoc	PP	LRU	49	0.6	300	229
2-way Set Assoc	PP	FIFO	49	0.7	300	259

Figure 13: Average Memory Access Latency for Six Cache Configurations

The pipelined 2-way set-associative cache with a LRU replacement policy has the lowest average memory access time. I think that this conclusion is fairly general due to several factors. Set associative caches generally performed better

than direct mapped caches due to its better ability to handle sparser data. Due to the pipelined write and parallel read hit, the cycle time is also much lower than the sequential architecture. The LRU policy is very beneficial for real world programs that generally display temporal locality in data use. I think that in most cases, it is safe to say that we should choose this configuration. However, in some specific cases, this may not be the best choice. For example, for programs that have very structured spatial locality, it might be better to use a direct mapped cache due to its larger index. This can improve performance slightly with the lower cycle time. Also in technology where energy consumption and heat dissipation is a larger concern, the parallel read hit design can be bad due to wasted power in reading the data memory even though there was no tag hit.