Problem 1

A)

ADDR	Instruction	Iteration 1	Iteration 2
	`loop:`		
`0x108`	`addi x1, x1, -1`	***compulsory***	***hit***
`0x10c`	`addi x2, x2, 1`	***hit***	***hit***
`0x110`	`jal x0, foo`	***compulsory***	***conflict***
	`...`		
	`foo:`		
`0x218`	`addi x6, x6, 1`	***compulsory***	***conflict***
`0x21c`	`bne x1, x0, loop`	***hit***	***hit***

B) hit latency = 1 cycle
miss penalty = 15 cycles
miss rate = (3+2*63 misses) / (5*64 accesses) = 129 misses / 320 accesses = **0.4031**
AMAL = hit latency + (miss rate * miss_penalty) = 1+(0.4031*15) = **7.0469** cycles

*As seen in the table above, **conflict misses** are occurring twice in almost every iteration of the loop, thus are dominating the AMAL.*

C) *In a two-way set associative cache architecture, we would reduce the number of conflict misses, because we would have both `0x110` and `0x210` be available in our two-way cache line, thus all of the conflict misses from iteration 2 to 64, would become hits. This would lead to only compulsory misses in our cache, because we would be able to store all of the loop instructions in our cache after the first iteration.*

hit latency = 1 cycle
miss penalty = 15 cycles
miss rate = (3 misses) / (5*64 accesses) = 3 misses / 320 accesses = **0.009375**
AMAL = hit latency + (miss rate * miss_penalty) = 1+(0.009375*15) = **1.1406** cycles

Problem 2

A.

Virtual Address		
V PN		
L1	L2	page offset
2b	2b	12b

		Page - Table Entry	
idx	Physical address of PTE	Valid	Physical address
11	0xffffc	1	0xfffe0
10	0xffff8	0	
01	0xffff4	0	
00	0xffff0	1	0xfffb0
11	0xfffec	1	0x05000
10	0xfffe8	1	0x07000
01	0xfffe4	0	
00	0xfffe0	0	
11	0xfffdc	0	
10	0xfffd8	0	
01	0xfffd4	0	
00	0xfffd0	0	
11	0xfffcc	0	
10	0xfffc8	0	
01	0xfffc4	0	
00	0xfffc0	0	
11	0xfffbc	1	0x01000
10	0xfffb8	1	0x04000
01	0xfffb4	1	0x00000
00	0xfffb0	0	

B.

Transaction Address	VPN	Page Offset	m/h	Total Num Mem Acc	TLB Way 0		TLB Way 1	
					VPN	PPN	VPN	PPN
0xeff4	0xe	0xff4	m	3	-	-	-	-
0x2ff0	0x2	0xff0	m	3	0xe	0x07	-	-
0xeff8	0xe	0xff8	h	1	0xe	0x07	0x2	0x04
0x2ff4	0x2	0xff4	h	1	0xe	0x07	0x2	0x04
0xeffc	0xe	0xffc	h	1	0xe	0x07	0x2	0x04
0x2ff8	0x2	0xff8	h	1	0xe	0x07	0x2	0x04
0xf000	0xf	0x000	m	3	0xe	0x07	0x2	0x04
0x2ffc	0x2	0xffc	h	1	0xf	0x05	0x2	0x04
0xf004	0xf	0x004	h	1	0xf	0x05	0x2	0x04
0x3000	0x3	0x000	m	3	0xf	0x05	0x2	0x04
0xf008	0xf	0x008	h	1	0xf	0x05	0x3	0x01
0x3004	0x3	0x004	h	3	0xf	0x05	0x3	0x01
0xf00c	0xf	0x00c	h	1	0xf	0x05	0x3	0x01
0x3008	0x3	0x008	h	1	0xf	0x05	0x3	0x01
							0x3	0x01
Number of Misses:				**4**				
Miss Rate:				**0.2857**				

Question 3

Part A Direct Mapped

Transaction Address	tag	idx	m/h	L0	L1	L2	L3	L4	L5	L6	L7
0x024	0x00	0x2	m	-	-	-	-	-	-	-	-
0x030	0x00	0x3	m			0x00					
0x07c	0x00	0x7	m				0x00				
0x070	0x00	0x7	h								0x00
0x100	0x02	0x0	m								*
0x110	0x02	0x1	m	0x02							
0x204	0x04	0x4	m		0x02						
0x214	0x04	0x1	m					0x04			
0x308	0x06	0x0	m		0x04						
0x110	0x02	0x1	m	0x06							
0x114	0x02	0x1	h		0x02						
0x118	0x02	0x1	h		*						
0x11c	0x02	0x1	h		*						
0x410	0x08	0x1	m		*						
0x110	0x02	0x1	m		0x08						
0x510	0x0a	0x1	m		0x02						
0x110	0x02	0x1	m		0x0a						
0x610	0x0c	0x1	m		0x02						
0x110	0x02	0x1	m		0x0c						
0x710	0x0e	0x1	m		0x02						
					0x0e						

NUMBER OF MISSES	16
MISS RATE	0.80

LRU Policy

Trans				SET 0			SET 1			SET 2			SET 3		
Addr	tag	idx	m/h	U	Way 0	Way1	U	Way 0	Way 1	U	Way 0	Way 1	U	Way 0	Way 1
0x024	0x00	0x2	m	–	–	–	–	–	–	–	–	–	–	–	–
0x030	0x00	0x3	m							0	0x00				
0x07c	0x01	0x3	m										0	0x00	
0x070	0x01	0x3	h										1		0x01
0x100	0x04	0x0	m										1		*
0x110	0x04	0x1	m	0	0x04										
0x204	0x08	0x0	m				0	0x04							
0x214	0x08	0x1	m	1		0x08									
0x308	0x0c	0x0	m				1		0x08						
0x110	0x04	0x1	h	0	0x0c										
0x114	0x04	0x1	h				0	*							
0x118	0x04	0x1	h				0	*							
0x11c	0x04	0x1	h				0	*							
0x410	0x10	0x1	m				0	*							
0x110	0x04	0x1	h				1		0x10						
0x510	0x14	0x1	m				0	*							
0x110	0x04	0x1	h				1		0x14						
0x610	0x18	0x1	m				0	*							
0x110	0x04	0x1	h				1		0x18						
0x710	0x1c	0x1	m				0	*							
							1		0x1c						

NUMBER OF Ms	12
MISS RA TE	0.6

FIFO Policy

Trans				SET 0		SET 1		SET 2		SET 3	
Addr	tag	idx	m/h	Way 0	Way1	Way 0	Way 1	Way 0	Way 1	Way 0	Way 1
0x024	0x00	0x2	m	-	-	-	-	-	-	-	-
0x030	0x00	0x3	m					0x00			
0x07c	0x01	0x3	m							0x00	
0x070	0x01	0x3	h								0x01
0x100	0x04	0x0	m								*
0x110	0x04	0x1	m	0x04							
0x204	0x08	0x0	m			0x04					
0x214	0x08	0x1	m		0x08						
0x308	0x0c	0x0	m				0x08				
0x110	0x04	0x1	h	0x0c							
0x114	0x04	0x1	h			*					
0x118	0x04	0x1	h			*					
0x11c	0x04	0x1	h			*					
0x410	0x10	0x1	m			*					
0x110	0x04	0x1	m			0x10					
0x510	0x14	0x1	m				0x04				
0x110	0x04	0x1	h			0x14					
0x610	0x18	0x1	m				*				
0x110	0x04	0x1	m				0x18				
0x710	0x1c	0x1	m			0x04					
							0x1c				
NUMBER OF Ms				14							
MISS RA TE				0.7							

PART B

Direct Mapped Critical Path Table

Component	Delay Equation	Delay (T)
add_reg_M0	1	1
tag_decoder	4+2n = 4+2*3	10
tag_mem	10+[(n+m)/16] = 10+[(8+25)/16]	13
tag_cmp	3+2[log2(n)] = 3+ 2* [log2(25)]	13
tag_and	n-1 = 2 - 1	1
wen_and	n-1 = 2 - 1	1
data_mem	10+[(n+m)/16] =10+[(8+128)/16]	19
rdata_mux	3[log2(n)]+[m/8] = 3*[log2(4)] + [32/8]	10
rdata_reg_M1	1	1
TOTAL		69

2-Way Set Associative Critical Path Table

Component	Delay Equation	Delay (T)
add_reg_M0	1	1
tag_decoder	4+2n = 4+2*2	8
tag_mem	10+[(n+m)/16] = 10 +[(4+26)/16]	12
tag_cmp	3+2[log(n)] = 3+2*[log2(26)]	13
tag_and	n-1 = 2-1	1
data_decoder	4+2n = 4+2*3	10
data_mem	10+[(n+m)/16] = 10+[(8+128)/16]	19
rdata_mux	3[log(n)]+[m/8] = 3*[log2(4)] + [32/8]	10
rdata_reg_M1	1	1
TOTAL		75

The primary reason for the slowness of 2-way set associative design is that its critical path includes data_decoder which depends on the tag_and output. This component was not included in direct mapped, because data_decode and tag_decode would be executed in parallel.

PART C

Associativity	Replacement Policy	Hit Time (T)	Miss Rate (ratio)	Miss Penalty (T)	AMAL (T)
Direct Mapped	N/A	69	0.8	300	309
2-way Set Assoc	LRU	75	0.6	300	255
2-way Set Assoc	FIFO	75	0.7	300	285

2-way set associative cache with LRU policy has the lowest AMAL. This configuration is definitely better than direct mapped config, because it takes advantage of spatial locality. The LRU policy is a situation dependent. In this example it took advantage of the temporal locality to reduce the miss rate. This is evident in Part A LRU table, where the second half of the instructions referenced the same tag intermittently, thus keeping the recently used tags in the cache helped reduce the miss ratio.

Question 3

Part A

Transaction Address	tag	idx	m/h	L0	L1	L2	L3	L4	L5	L6	L7
0x024	0x00	0x2	m	-	-	-	-	-	-	-	-
0x030	0x00	0x3	m			0x00					
0x07c	0x00	0x7	m				0x00				
0x070	0x00	0x7	h								0x00
0x100	0x02	0x0	m								*
0x110	0x02	0x1	m	0x02							
0x204	0x04	0x4	m		0x02						
0x214	0x04	0x1	m					0x04			
0x308	0x06	0x0	m		0x04						
0x110	0x02	0x1	m	0x06							
0x114	0x02	0x1	h		0x02						
0x118	0x02	0x1	h		*						
0x11c	0x02	0x1	h		*						
0x410	0x08	0x1	m		*						
0x110	0x02	0x1	m		0x08						
0x510	0x0a	0x1	m		0x02						
0x110	0x02	0x1	m		0x0a						
0x610	0x0c	0x1	m		0x02						
0x110	0x02	0x1	m		0x0c						
0x710	0x0e	0x1	m		0x02						
					0x0e						

NUMBER OF MISSES	16
MISS RATE	0.80

Trans				SET 0			SET 1			SET 2			SET 3		
Addr	tag	idx	m/h	U	Way 0	Way1	U	Way 0	Way 1	U	Way 0	Way 1	U	Way 0	Way 1
0x024	0x00	0x2	m	-	-	-	-	-	-	-	-	-	-	-	-
0x030	0x00	0x3	m							0	0x00				
0x07c	0x01	0x3	m										0	0x00	
0x070	0x01	0x3	h										1		0x01
0x100	0x04	0x0	m										1		*
0x110	0x04	0x1	m	0	0x04										
0x204	0x08	0x0	m				0	0x04							
0x214	0x08	0x1	m	1		0x08									
0x308	0x0c	0x0	m				1		0x08						
0x110	0x04	0x1	h	0	0x0c										
0x114	0x04	0x1	h				0	*							
0x118	0x04	0x1	h				0	*							
0x11c	0x04	0x1	h				0	*							
0x410	0x10	0x1	m				0	*							
0x110	0x04	0x1	h				1		0x10						
0x510	0x14	0x1	m				0	*							
0x110	0x04	0x1	h				1		0x14						
0x610	0x18	0x1	m				0	*							
0x110	0x04	0x1	h				1		0x18						
0x710	0x1c	0x1	m				0	*							
							1		0x1c						

NUMBER OF Ms	12
MISS RATE	0.6

Trans				SET 0		SET 1		SET 2		SET 3	
Addr	tag	idx	m/h	Way 0	Way1	Way 0	Way 1	Way 0	Way 1	Way 0	Way 1
0x024	0x00	0x2	m	-	-	-	-	-	-	-	-
0x030	0x00	0x3	m					0x00			
0x07c	0x01	0x3	m							0x00	
0x070	0x01	0x3	h								0x01
0x100	0x04	0x0	m								*
0x110	0x04	0x1	m	0x04							
0x204	0x08	0x0	m			0x04					
0x214	0x08	0x1	m		0x08						
0x308	0x0c	0x0	m				0x08				
0x110	0x04	0x1	h	0x0c							
0x114	0x04	0x1	h			*					
0x118	0x04	0x1	h			*					
0x11c	0x04	0x1	h			*					
0x410	0x10	0x1	m			*					
0x110	0x04	0x1	m			0x10					
0x510	0x14	0x1	m				0x04				
0x110	0x04	0x1	h			0x14					
0x610	0x18	0x1	m				*				
0x110	0x04	0x1	m				0x18				
0x710	0x1c	0x1	m			0x04					
							0x1c				
NUMBER OF Ms				14							
MISS RA TE				0.7							

PART B

Direct Mapped Critical Path Table

Component	Delay Equation	Delay (T)
add_reg_M0	1	1
tag_decoder	4+2n = 4+2*3	10
tag_mem	10+[(n+m)/16] = 10+[(8+25)/16]	13
tag_cmp	3+2[log2(n)] = 3+ 2* [log2(25)]	13
tag_and	n-1 = 2 - 1	1
wen_and	n-1 = 2 - 1	1
data_mem	10+[(n+m)/16] =10+[(8+128)/16]	19
rdata_mux	3[log2(n)]+[m/8] = 3*[log2(4)] + [32/8]	10
rdata_reg_M1	1	1
TOTAL		69

2-Way Set Associative Critical Path Table

Component	Delay Equation	Delay (T)
add_reg_M0	1	1
tag_decoder	4+2n = 4+2*2	8
tag_mem	10+[(n+m)/16] = 10 +[(4+26)/16]	12
tag_cmp	3+2[log(n)] = 3+2*[log2(26)]	13
tag_and	n-1 = 2-1	1
data_decoder	4+2n = 4+2*3	10
data_mem	10+[(n+m)/16] = 10+[(8+128)/16]	19
rdata_mux	3[log(n)]+[m/8] = 3*[log2(4)] + [32/8]	10
rdata_reg_M1	1	1
TOTAL		75

The primary reason for the slowness of 2-way set associative design is that its critical path includes data_decoder which depends on the tag_and output. This component was not included in direct mapped, because data_decode and tag_decode would be executed in parallel.

PART C

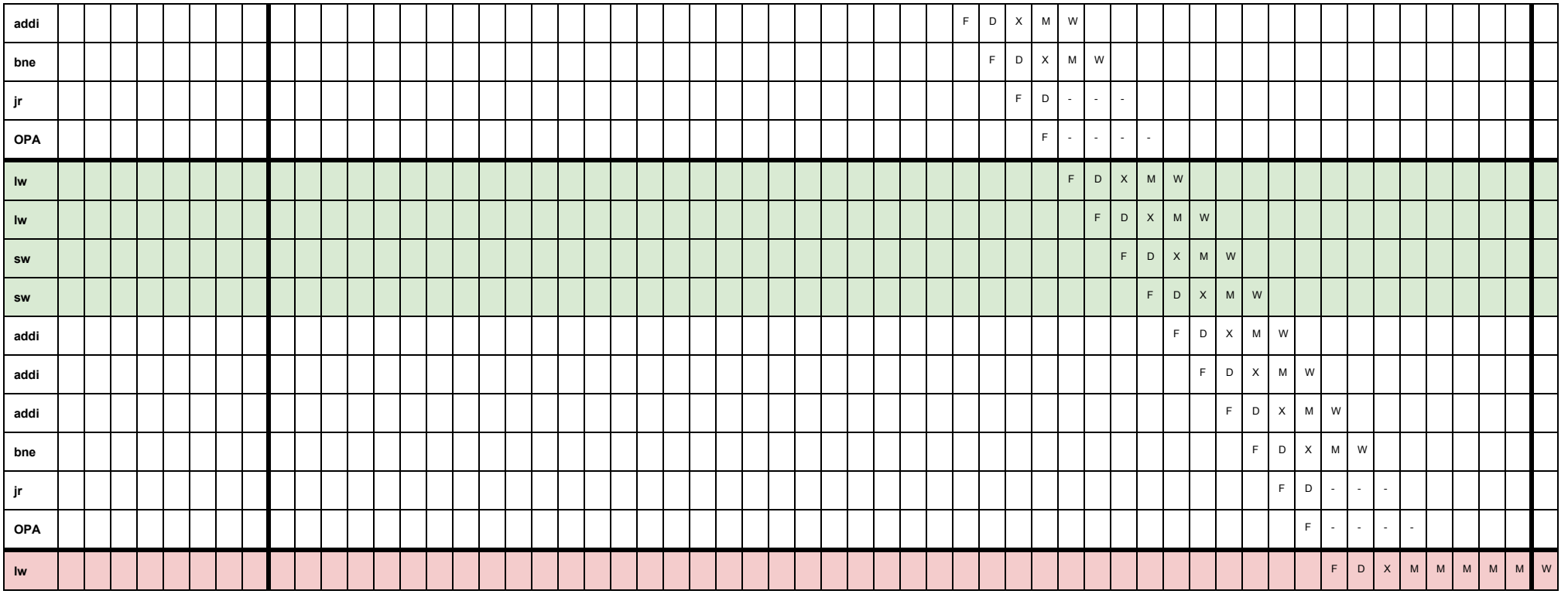
Associativity	Replacement Policy	Hit Time (T)	Miss Rate (ratio)	Miss Penalty (T)	AMAL (T)
Direct Mapped	N/A	69	0.8	300	309
2-way Set Assoc	LRU	75	0.6	300	255
2-way Set Assoc	FIFO	75	0.7	300	285

2-way set associative cache with LRU policy has the lowest AMAL. This configuration is definitely better than direct mapped config, because it takes advantage of spatial locality. The LRU policy is a situation dependent. In this example it took advantage of the temporal locality to reduce the miss rate. This is evident in Part A LRU table, where the second half of the instructions referenced the same tag intermittently, thus keeping the recently used tags in the cache helped reduce the miss ratio.

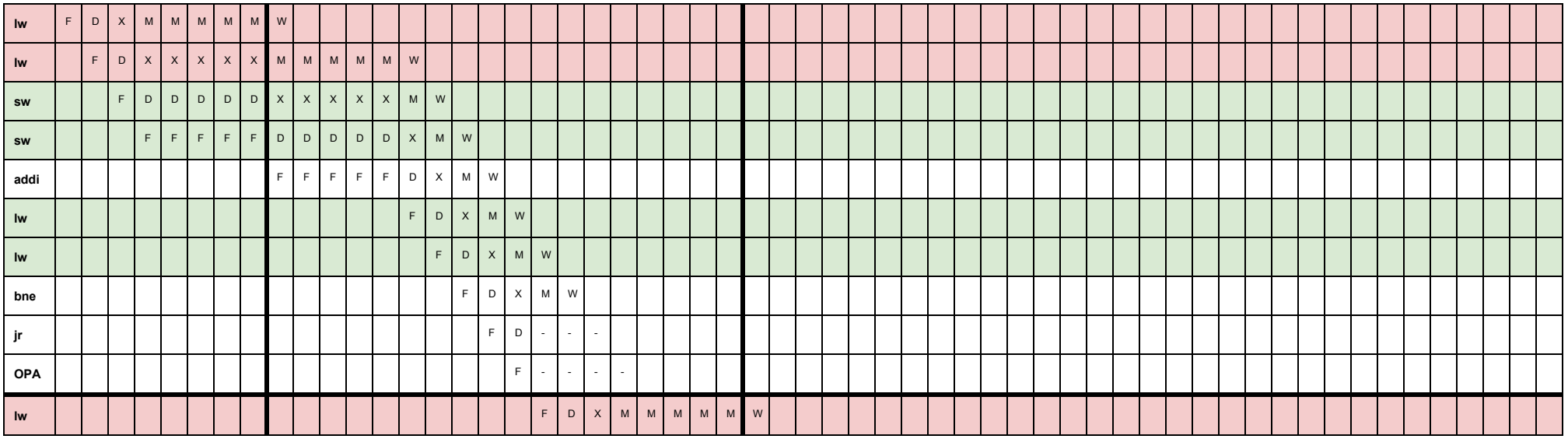
Question 4

PART A

lw	F	D	X	M	M	M	M	M	W																																																
lw		F	D	X	X	X	X	X	M	M	M	M	M	W																																									
sw			F	D	D	D	D	D	X	X	X	X	X	M	W																																								
sw				F	F	F	F	F	D	D	D	D	D	X	M	W																																							
addi									F	F	F	F	F	D	X	M	W																																						
addi														F	D	X	M	W																																					
addi															F	D	X	M	W																																				
bne																F	D	X	M	W																																			
jr																	F	D	-	-	-																																		
OPA																		F	-	-	-	-																																	
lw																			F	D	X	M	W																																
lw																				F	D	X	M	W																															
sw																					F	D	X	M	W																														
sw																						F	D	X	M	W																													
addi																							F	D	X	M	W																												
addi																								F	D	X	M	W																											
addi																									F	D	X	M	W																										
bne																										F	D	X	M	W																									
jr																											F	D	-	-	-																								
OPA																												F	-	-	-	-																							
lw																													F	D	X	M	W																						
lw																														F	D	X	M	W																					
sw																															F	D	X	M	W																				
sw																																F	D	X	M	W																			
addi																																	F	D	X	M	W																		
addi																																		F	D	X	M	W																	


addi F D X M W
bne F D X M W
jr F D - - -
OPA F - - - -
lw F D X M W
lw F D X M W
sw F D X M W
sw F D X M W
addi F D X M W
addi F D X M W
addi F D X M W
bne F D X M W
jr F D - - -
OPA F - - - -
lw F D X M M M M M W

PART B


lw F D X M M M M M W
lw F D X X X X X M M M M M W
sw F D D D D D X X X X X M W
sw F F F F F D D D D D X M W
addi F F F F F D X M W
lw F D X M W
lw F D X M W
bne F D X M W
jr F D - - -
OPA F - - - -
lw F D X M M M M M W

PART	Number of Instructions	CPI	Execution Time (cyc)	CPI Breakdown			
				Useful Work	RAW Stalls	Control Squashes	Memory Stalls
Part 4.A	256	1.5	384	32/32 = 0.8	0	8/32 = 0.25	8/32 = 0.25
Part 4.B	256	2.25	576	8/8 = 1.0	0	2/8 = 0.25	8/8 = 1.0

Part A
Num Instrs = (64/2) iters * (8 instrs/iter) = 256 instrs
Useful Work = 32 instr/stable iter
Cycles per stable Iter = 48 cycles
Control Squashes = 8 squashes/stable iter
Mem Stalls = 2misses*4 stalls/miss = 8 stalls/stable iter
Total instr per stable iter = 32 instr/stable iter
CPI = 48 cycles/32 instrs = 1.5

Part B
Num Instrs = (64/2) iters * (8 instrs/iter) = 256 instrs
Useful Work = 8 instr/stable iter
Cycles per stable Iter = 18 cycles
Control Squashes = 2 squashes/stable iter
Mem Stalls = 2misses*4 stalls/miss = 8 stalls/stable iter
Total instr per stable iter = 8 instr/stable iter
CPI = 18 cycles/8 instrs = 2.25

Part C

Array data structure performs better in this example, because each time we have a compulsory miss, we can load 4 array elements into the cache, so that the next three iterations of the loop will have cache hits. Whereas, linked list data structure is large thus will take up the entire cache line, thus we each individual node will result in a compulsory miss, and we must pay the price on each iteration.

Execution time would not change as a function of cache-line size for linked list structure, since we are assuming that there will always be one node per cache line. However, execution time should decrease for array structure in an inversely fashion because we would reduce the number of compulsory misses for the number of elements we are accessing.

Execution time would then decrease inversely since we would be able to load multiple nodes in each compulsory miss. Assuming spatial locality holds, meaning these nodes that are pointing to each other are also physically located adjacently to each other, we should be able to reduce the number of misses by a factor of 1/n where n is the number of nodes that can be loaded per cache line.

Execution of time for both data structures would increase linearly. Execution time would be however worse for a linked-list structure by a certain factor that is determined by the cache line size which in turn signifies number of cache misses we encounter per memory access. Both data structures will result in theoretical asymptotic behavior of O(n). It is clear that array structure can result O(n) behavior, and linked-list's theoretical behavior is O(cn) where c is the factor we talked about. O(cn) is approximated to O(n).

Execution time would then be independent of the memory stalls, because we would always hit on the memory accesses. With an assumption that we can access the memory within 1 cycle, the execution would then be dependent on the data and control hazards and number of instructions in the loop. With memory access delays disregarded, we can assume that our CPI would be very close to one. Asymptotic behavior for both data structures would be the same, O(n).

We can conclude that arrays are the data structures that make best use of spatial locality. Thus, if cache aligned, arrays will result in the lowest cache latency. This might not necessarily be true for other data structures, since they are large and have intricate structure that can only be exploited with algorithmic gains. When considering cache behavior, irregular structures might not necessarily offer the lowest latency.