Short Answer Questions [9 points]

3 points for each question; 1 point for each sub-question.
Please answer each question with a short answer of no more than two sentences.

1. Amdahl's Law

a. What does Amdahl's Law state? (In words, not a formula.)
The speedup is limited by the portion of the application you are accelerating.

b. A program spends 75% of its time doing multiply instructions. If the multiplier is sped up by 3x, how much faster does the application run? (Report your answer as "the program is X times faster.")
The program is 2x faster. (75% of the time is 3x faster, so that 75% now takes up 1/3 of that time, or only 25% of the original execution time. We have effectively eliminated 50% of the total execution time so the program is 2x faster.)

c. What is the limit on the maximum speedup if the multiplier in question 1b was infinitely fast? (Report your answer as "the program would be X times faster.")
The program would be 4x faster. (If the 75% of the time goes infinitely fast it will take 0 time, so the total program execution time will be reduced to 25% of what it originally was, or 4x faster.)

2. Two's Complement

a. How do you do subtraction with two's complement? (In words, not a formula.)
Using a regular adder, you invert the number to be subtracted and add one. (Set the carry in to true.)

For a 4-bit two's complement number, show how to represent the following operations and their results. If you cannot, explain why.

b. 4-7
4-7 = 4+-7 = 0100+1001 = 1101 = -3

c. 11-3
Cannot do this because 11 can not be represented in a 4-bit two's complement number. The range is -8 to 7.

3. Pipeline Depth

a. Why shouldn't we use a million pipeline stages if an operation can be divided up into a million steps, even if we can keep the pipeline full?
The overhead of the pipeline latch would be too high. (We would spend too much time moving data relative to computing.)

Assume an operation can be divided into 1, 10, or 50 pipeline stages with no overhead and the pipeline can be kept full.

b. What degree of pipelining would provide the optimal throughput? What is the throughput relative to the un-pipelined version?
50 stages would provide 50x the throughput.

c. What degree of pipelining would provide the optimal latency? What is the latency relative to the un-pipelined version?
1 stage would provide the lowest latency, which is the same as the un-pipelined version.

False/True [6 points]

Circle either false or true or neither.
0 points for no answer, -1 point for an incorrect answer, +1 point for a correct answer.

4a. For forwarding you need only look at the data available in the WB stage.
False True
False. Data must also be forwarded from the MEM stage.

4b. A dynamic branch predictor is always better than a static one.
False True
False. A 1 bit dynamic predictor will get an alternating branch 100% wrong while a static one will get it 50% wrong.

4c. A perfect branch predictor combined with data forwarding would allow a processor to always keep its pipeline full.
False True
False. TLB, cache misses, and exceptions will all keep the pipeline from being full.

4d. DMA is worse for large transfers than interrupts due to the overhead of setting up the transfer.
False True
False. For large transfers the overhead of setting up a DMA transfer is far less than interrupts on every piece of data received.

4e. An LRU replacement policy will always be better than a random replacement policy for managing virtual memory pages.
False True
False. We saw a pathological case where data almost fit and an LRU policy caused severe thrashing in lecture.

4f. The average number of memory accesses per instruction is less than one if not all instructions are loads or stores.
False True
False. We need to access memory to load the instruction itself as well.

MIPS Assembly [12 points]

5a. Write MIPS assembly for the algorithm below. [4 points]

```
c = a + b + 4;
do {
      c -= a;
      b++;
} while (c > 3)
```

a is in $t0, b is in $t1, c is in $s0. Clear comments are required for each line of code.

Code	Comments
`add $s0, $t0, $t1`	`c = a + b`
`addi $s0, $s0, 4`	`c = c + 4`
`Loop:`	
`sub $s0, $s0, $t0`	`c -= a`
`addi $t1, $t1, 1`	`b++`
`slti $t2, $s0, 3`	`Set $t2 to 1 if c <= 3`
`bneq $zero, $t2, Loop`	`If $t2 is zero, then c>3 and`
	`we don't branch`

5b. Modify the code for the procedure below to follow MIPS calling conventions. [5 points]

Code	Comments
addi $sp, $sp, -16	*Increment the stack pointer*
sw $ra, 12($sp)	*Store the return address*
addi $t0, $a0, 3	Uses the first procedure argument
sw $s0, 8($sp)	*Store s0 on the stack*
sw $s6, 4($sp)	*Store s6 on the stack*
add $t1, $a1, $t0	Uses the second procedure argument
add $s0, $t1, $t1	
loop:	
addi $s6, $s0, -4	
sw $t1, 0($sp)	*Store $t1 on the stack*
sub $t1, $s6, $t1	
add $a0, $s6, $zero	*Copy $s6 to $a0 for the call*
jal sub_routine	Needs $s6 as its first argument
lw $t1, 0($sp)	*Restore $t1 from the stack*
bne $s6, $v0, loop	Uses return value from sub_routine
lw $ra, 12($sp)	
lw $s0, 8($sp)	*Restore $s0*
sub $v0, $s6, $t1	Store result as return value
lw $s6, 4($sp)	*Restore $s6*
addi $sp, $sp, 16	*Pop the stack pointer back*
jr $ra	

5c. How much additional stack space is required for the procedure in 5b? [1 point]
4 words, or 16 bytes. ($ra, $s0, $s6, $t1)

5d. If the line `sub $t1, $s6, $t1` was removed from the procedure in question 5b, how could you optimize the stack operations? [2 points]
You do not need to put $t1 on the stack on every loop iteration. You can move its push/pop outside the loop.

Caches [16 points]

6. Show the contents of the three caches below for the specified address stream. [9 points; 3 for each cache type]

Three, 4-entry, LRU caches are shown below, with the most recently used entry at the top of each set. They are fully-associative (FA), direct-mapped (DM), and 2-way set-associative (SA). Each cache has a block/line size of 1 byte/address. The DM and SA caches use the modulo (LSB-based) indexing function discussed in class. Show how the contents of the caches change with the access pattern 0, 1, 2, 3, 4, 0, 4, 0 ,2 by writing the memory address in each location of the cache at each time step and circling whether the access is a hit or miss.

Cycle 0: Address: 0

FA	DM	SA
0 - MISS	*0 - MISS*	*0 - MISS*
Hit Miss	Hit Miss	Hit Miss

Cycle 1: Address: 1

FA	DM	SA
1 - MISS	*0*	*0*
0	*1 - MISS*	
		1 - MISS
Hit Miss	Hit Miss	Hit Miss

Cycle 2: Address: 2

FA	DM	SA
2 - MISS	*0*	*2 - MISS*
1	*1*	*0*
0	*2 - MISS*	*1*
Hit Miss	Hit Miss	Hit Miss

Cycle 3: Address: 3

FA	DM	SA
3 - MISS	*0*	*2*
2	*1*	*0*
1	*2*	*3 - MISS*
0	*3 - MISS*	*1*
Hit Miss	Hit Miss	Hit Miss

Cycle 4: Address: 4

FA	DM	SA
4 - MISS	*4 - MISS*	*4 - MISS*
3	*1*	*2*
2	*2*	*3*
1	*3*	*1*
Hit Miss	Hit Miss	Hit Miss

Cycle 5: Address: 0

FA	DM	SA
0 – MISS	*0 - MISS*	*0 - MISS*
4	*1*	*4*
3	*2*	*3*
2	*3*	*1*
Hit Miss	Hit Miss	Hit Miss

Cycle 6: Address: 4

FA	DM	SA
4 – HIT	*4 - MISS*	*4 – HIT*
0	*1*	*0*
3	*2*	*3*
2	*3*	*1*
Hit Miss	Hit Miss	Hit Miss

Cycle 7: Address: 0

FA	DM	SA
0 – HIT	*0 - MISS*	*0 – HIT*
4	*1*	*4*
3	*2*	*3*
2	*3*	*1*
Hit Miss	Hit Miss	Hit Miss

Cycle 8: Address: 2

FA	DM	SA
2 – HIT	*0*	*2 - MISS*
0	*1*	*0*
4	*2 – HIT*	*3*
3	*3*	*1*
Hit Miss	Hit Miss	Hit Miss

(Extra)

FA	DM	SA
Hit Miss	Hit Miss	Hit Miss

7. Calculate the effective CPI (taking into account both instruction execution and memory access) for the following unified instruction and data cache. [7 points]

Instruction execution CPI = 1.0
Percent of instructions that are loads or stores = 33%
Cycles to access main memory = 100
Cycles to access cache = 2
Miss rate for cache 2.0%

7a. What is the total memory access rate? (memory accesses per cycle) [1 point]

CPI(data memory accesses+instruction memory accesses) =*
1.0(0.33+1.00) = 1.33*

7b. How many cycles per instruction are spent handling misses? [2 points]

*1.33 memory accesses per instruction * 2.0% miss rate * 100 cycles per miss =*
*1.33*0.02*100 = 2.66*

7c. How many cycles per instruction are spent handling hits? [2 points]

*1.33 memory accesses per instruction * (100%-2.0%) hit rate * 1 cycles per hit =*
*1.33*0.98*2 =2.6*

7d. What is the effective CPI for the system with the cache? [2 points]

cycles executing + cycles for hits + cycles for misses =
1.0 + 2.66 + 2.6 = 6.26m

Pipelining [12 points]

8a. Identify all data dependencies in the following code. [2 points]

Fill in the table below for each data dependency you find. For example, if instruction 2 depends on register 14 from instruction 0, you would write "R14 from 2". If there is no data dependency leave the table entry blank.

Instruction	Depends on Register from Instruction	Depends on Register from Instruction	Depends on Register from Instruction
I1: `add $r2, $r1, $r3`			
I2: `sub $r4, $r2, $r1`	*R2 from 1*		
I3: `and $r5, $r1, $r2`	*R2 from 1*		
I4: `sub $r6, $r2, $r4`	*R2 from 1*	*R4 from 2*	
I5: `add $r7, $r2, $r3`	*R2 from 1*		

Assume a standard 5-stage MIPS pipeline discussed in class (IF, ID, EX, MEM, WB). Fill out the schedule for executing the above instructions correctly. The cycles (time) are on the horizontal axis. Fill in IF, ID, EX, MEM, and WB in the appropriate time for each instruction. Label all stalls with * and draw arrows on the scheduling to specify what is forwarded and where. The shading and lines are just to make it easier to grade and have nothing to do with the problem.

8b. Fill out the schedule for a pipeline with no forwarding. [5 points]

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
I1	*IF*	*ID*	*EX*	*MEM*	*WB*										
I2		*IF*	*ID*	*	*	*EX*	*MEM*	*WB*							
I3			*IF*	*ID*	*	*	*EX*	*MEM*	*WB*						
I4				*IF*	*ID*	*	*	*	*EX*	*MEM*	*WB*				
I5					*IF*	*ID*	*	*	*	*EX*	*MEM*	*WB*			

8b. Fill out the schedule for a pipeline with forwarding from just the EX/MEM stage. [5 points]

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
I1	*IF*	*ID*	*EX*	***MEM***	*WB*										
I2		*IF*	*ID*	***EX***	*MEM*	*WB*									
I3			*IF*	*ID*	*	*EX*	*MEM*	*WB*							
I4				*IF*	*ID*	*	*EX*	*MEM*	*WB*						
I5					*IF*	*ID*	*	*EX*	*MEM*	*WB*					

$r2 is forwarded from I1's MEM in T3 to I2's EX in T3.