Questions

Question 1. Pipelining (18 MARKS)

We have the following MIPS program that swaps the contents of two arrays A and B, with base addresses in $s0 and $s1 respectively. Both arrays are of the same length, with the length given in register $s2.

```
        sll $t0, $s2, 2
        add $t0, $s0, $t0
        addi $t1, $s0, $0
        addi $t2, $s1, $0
loop:   slt $t3, $t1, $t0
        beq $t3, $0, exit
        lw $t3, 0($t1)
        lw $t4, 0($t2)
        sw $t3, 0($t2)
        sw $t4, 0($t1)
        addi $t1, $t1, 4
        addi $t2, $t2, 4
        j loop
```

a. Given that $s2 contains the value 8, how many instructions are executed in this program? (2 marks)

Before the loop: 4 instructions
Within the loop: 9 instructions
# of iterations = 8
On 9th iteration, slt and beq are executed.
Total = 4 + 8 x 9 + 2 = 78 instructions

Refer to the following tables for subsequent parts of this question.

Stage	Timing in nanoseconds (ns)
IF	5
ID	3
EX	3
MEM	5
WB	3

b. Assuming a NON-PIPELINED, single-cycle datapath implementation. How much time does it take to execute our program if $s2 contains the value 8? (2 marks)

Cycle time = 5 + 3 + 3 + 5 + 3 = 19ns
# of instructions = 78
Total time = 1482 ns.

c. Assuming a NON-PIPELINED, multi-cycle datapath implementation where all non-memory instructions skip the MEM stage and branch and jump instructions skip the WB (and of course MEM) stage. The sw instruction also skips the WB stage. How much time does it take to execute our program if $s2 contains the value 8? (3 marks)

Execution time for non-memory, non branch instructions = IF + ID + EX + WB
= 5 + 5 + 5 + 5 = 20 ns
Execution time for branch = IF + ID + EX
= 5 + 5 + 5 = 15 ns
Execution time for lw = 5 + 5 + 5 + 5 + 5 = 25 ns
Execution time for sw = 5 + 5 + 5 + 5 = 20 ns

	sll $t0, $s2, 2	**20**
	add $t0, $s0, $t0	**20**
	addi $t1, $s0, $0	**20**
	addi $t2, $s1, $0	**20**
loop:	**slt $t3, $t1, $t0**	**20**
	beq $t3, $0, exit	**15**
	lw $t3, 0($t1)	**25**
	lw $t4, 0($t2)	**25**
	sw $t3, 0($t2)	**20**
	sw $t4, 0($t1)	**20**
	addi $t1, $t1, 4	**20**
	addi $t2, $t2, 4	**20**
	j loop	**15**
exit:		

Time taken = 20 x 4 + 8 x (20 +15 +25 +25 +20 +20 +20 +20 + 15) + 20 + 15
= 80 + 8 x 180 + 35
= 1,555

d. What is the speedup between your results in part c versus your results in part b? Which implementation is faster? Why? Find your speedup to three decimal places. **Note: Speedup = (Time Taken for Single Cycle Datapath) / (Time Taken for Multicycle Datapath)** (2 mark)

Speedup = 1482/1555
= 0.953

Single cycle is faster due to stages taking longer in multicycle.

Now let's assume a pipelined implementation where each pipeline register adds a 1ns delay.

e. **Using the ideal pipeline equation,** How much time does the program take to execute on our MIPS pipeline if $s2 contains 8? Assume that the pipeline is initially empty. (2 marks)

Each stage must now take 5ns + delay time of 1ns. So cycle time is now 6ns. In an ideal pipeline, # of cycles = I + (N-1).

= 78 + 4 = 82 cycles.
Time = 82 x 6 = 492 ns

f. **Using the ideal pipeline equation,** How much time does the program take to execute on our MIPS pipeline If $s2 contains 800,000,000? Assume that the pipeline is initially empty. (2 marks)

# of cycles = 9 x 800,000,000 + 6 + 4 = 7,200,000,010
Time = 43,200,000,060ns

g. Compute the speedup to 3 decimal places for parts e. and f. against the single-cycle implementation in part b (note: You have to compare part f. against our program executing with $s2=800,000,000 on the single-cycle datapath implementation) **Note: Speedup = (Time Taken for Single Cycle Datapath) / (Time Taken for Pipelined CPU)** (2 marks)

For 8 instructions, speedup = 1482 / 492 = 3.012x

For 800,000,000 instructions, time taken on single cycle = (9 x 800,000,000 + 6) x 19 = 136,800,000,114ns

Speedup = 136,800,000,114/ 43,200,000,060
= 3.167x

h. In a 5-stage pipeline we hope to have an ideal speedup of 5x. Why is your best speedup much less than 5x? (2 marks)

Imbalanced pipeline, some stages taking longer than others, and 20% extra penalty for the pipeline registers.

Question 2 Caching (10 MARKS)

We consider a 2-way set associative cache of total size 16 words, with 2 words per block, on our MIPS machine.

a. How many offset, index and tag bits are there from our MIPS address? (2 marks)

2 words / block = 8 bytes / block.
Offset = log2(8) = 3 bits

Total size = 16 words, 2 words per block = 8 blocks. Two-way set associative = 4 sets.

# of index bits = log2(4) = 2 bits.

# of tag bits = 32 – 2 – 3 = 27 bits

b. Assuming that, in our program from Question 1, Array A starts at address 0x1000 and Array B at address 0x1034, draw the final state of our cache. What is the hit rate of our program, assuming that memory writes do not affect your answer in any way? Our cache uses a least-recently-used (LRU) replacement policy. (6 marks)

Start: 0001 0000 000 00 000_2

Array	Address in Hex	Address in Binary	Set
A[0], A[1]	0x1000 0x1004	0001 0000 000 00 000 0001 0000 000 00 100	0
A[2], A[3]	0x1008 0x100C	0001 0000 000 01 000 0001 0000 000 01 100	1
A[4], A[5]	0x1010 0x1014	0001 0000 000 10 000 0001 0000 000 10 100	2
A[6], A[7]	0x1018 0x101C	0001 0000 000 11 000 0001 0000 000 11 100	3

Array B:

Start: 0001 0000 0011 0100

Array	Address in Hex	Address in Binary	Set
??, B[0]	0x1030 0x1034	0001 0000 001 10 000 0001 0000 001 10 100	2
B[1], B[2]	0x1038 0x103C	0001 0000 001 11 000 0001 0000 001 11 100	3
B[3], B[4]	0x1040 0x1044	0001 0000 010 00 000 0001 0000 010 00 100	0
B[5], B[6]	0x1048 0x104C	0001 0000 010 01 000 0001 0000 010 01 100	1
B[7], ??	0x1050 0x1054	0001 0000 010 10 000 0001 0000 010 10 100	2

Cache Pattern

Set	Word 0	Word 1		Word 0	Word 1
0	A[0]	A[1]		B[3]	B[4]
1	A[2]	A[3]		B[5]	B[6]
2	~~??~~, B[7]	~~B[0]~~, ??		A[4]	A[5]
3	B[1]	B[2]		A[6]	A[7]

Access	Set	Hit/Miss
A[0]	0	M
B[0]	2	M
A[1]	0	H
B[1]	3	M
A[2]	1	M
B[2]	3	H
A[3]	1	H
B[3]	0	M
A[4]	2	M
B[4]	0	H
A[5]	2	H
B[5]	1	M
A[6]	3	M
B[6]	1	H
A[7]	3	H
B[7]	2	M

# of accesses = 16
# of hits = 7
Hit rate = 7/16 = 43.75%

c. Assuming that the cache has a hit rate of 0.46, the cache has an access time of 1 ns and the main memory has an access time of 50ns, what is the total time taken to read the arrays above? (Hint: Find the average memory access time, then multiply by the number of array reads) (2 marks)

AMAT = 0.46 x 1 + (1 – 0.46) x 51
= 0.46 + 27.54
= 28 ns

# of accesses = 16
Total time to read the two arrays = 448 ns

Question 3 Interprocess Communications (12 MARKS)

We have the following queue function in C:

```
#define MAX_QUEUE_LEN   16

typedef struct {
      int queue[MAX_QUEUE_LEN];
      int front, rear;
      TSema sema; // Semaphore for part b.
} TQueue;

void add2q(TQueue *q, int val) {
      if((q->rear + 1) % MAX_QUEUE_LEN == q->front) {
            printf("Error: Queue is full.\n");
      }
      else {
            q->queue[q->rear] = val;
            q->rear = (q->rear + 1) % MAX_QUEUE_LEN;
      }
}
int readfq(TQueue *q) {
      int ret = -1;
      if((q->front + 1) % MAX_QUEUE_LEN == q->rear) {
            printf("Error: Queue is empty.\n");
      }
      else {
            ret = q->queue[q->front];
            q->front = (q->front + 1) % MAX_QUEUE_LEN;
      }
      return ret;
}

TQueue myq = {{0}, 0, 0, 1};
```

a. Suppose we have the following two threads accessing the queue:

Thread 1:

```
... some operations ...
add2q(&myq, 3);
... some operations ...
```

Thread 2:

```
... some operations ...
add2q(&myq, 2);
... some operations ...
```

Write down four examples of possible values the members of myq (excluding sema) can take after executing both threads to completion, and indicate if the values are "correct", i.e. values that myq would take on if the add2q operations properly enforced critical sections.

Assume each example is independent of the others, and that each time the queue is completely empty at the beginning. The first box shows an example of how to fill in the answer: (8 marks)

Note: There are other possible values of rear not shown here.

Example 0: queue = {1, 2, 3, 4} front = 0 rear = 4 Correct? Yes	Example 1: queue = **{3, 2}** front = **0** rear = **2** Correct? **Yes**
Example 2: queue = **{2, 3}** front = **0** rear = **2** Correct? **Yes**	Example 3: queue = **{2}** front = **0** rear = **1** Correct? **No**
Example 4: queue = **{3}** front = **0** rear = **1** Correct?	

b. We are given two operations:

void pend_sema(TSema *sema); // Pend on the semaphore
void post_sema(TSema *sema); // Post on the semaphore

Modify the program above, making use of the semaphore in TQueue, to properly protect critical sections of add2q and readfq.

You may annotate directly on the source code in the Answer Book. (4 marks)

```
void add2q(TQueue *q, int val) {
      pend_sema(&q->sema);
      if((q->rear + 1) % MAX_QUEUE_LEN == q->front) {
            printf("Error: Queue is full.\n");
      }
      else {
            q->queue[q->rear] = val;
            q->rear = (q->rear + 1) % MAX_QUEUE_LEN;
      }
      post_sema(&q->sema);
}
int readfq(TQueue *q) {
      pend_sema(&q->sema);
      int ret=-1;
      if((q->front + 1) % MAX_QUEUE_LEN == q->rear) {
            printf("Error: Queue is empty.\n");
      }
      else {
            ret = q->queue[q->front];
            q->front = (q->front + 1) % MAX_QUEUE_LEN;
      }
      return ret;
      post_sema(&q->sema);
}
```

Question 4. File Management (16 MARKS)

We are given the following entries in a FAT16 based file system:

Directory entry:

Filename	**Starting Block**
test1.txt	3

FAT:

0	FREE
1	FREE
2	5
3	4
4	6
5	8
6	7
7	9
8	EOF
9	EOF

Assuming that each block is 1024 bytes long:

a. What is the maximum size of this FAT16 partition, assuming that the last three possible "block numbers" are reserved for the FREE, BAD and EOF flags? (2 marks)

Total number of possible blocks = 2^16 – 3 = 65533
Each block is 1024 bytes.
Maximum partition size = 65533 x 1024 = 63.99 MB

b. In most operating systems, the "seek" operation moves a "file pointer" to the requested byte position in the file, and reads and writes take place from that position onwards. So if we do "seek(1340)", this will cause reads and writes to take place from byte 1340 onwards in the file. Given the following operations:

seek(1340)
read 2048 bytes
seek(511)
read 30 bytes
seek(3111)
read 1120 bytes

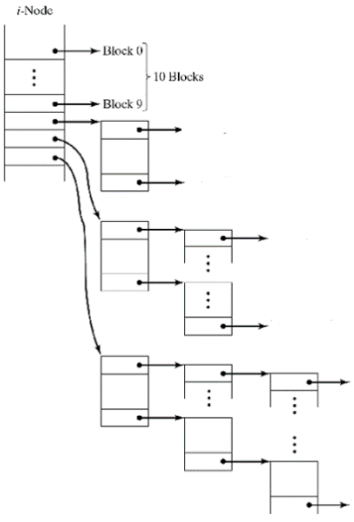
List down the block numbers accessed by these operations. Note that only the read operations result in block accesses. (6 marks)

We can map blocks to byte starting points.

Block	Starting Byte
3	0
4	1024
6	2048
7	3072
9	4096

Operation	Block(s) accessed
Seek(1340)	-
Read 2048 bytes	1340 + 2048 – 1 = 3387 4 (1340 to 2047) 6 (2048 to 3071) 7 (3072 to 3387)
Seek(511)	-
Read 30 bytes	3 (511 to 540)
Seek(3111)	-
read 1120 bytes	3111 + 1120 – 1 = 4230 7 (3111 to 4095) 9 (4096 to 4230)

Now assume that we have an inode based file system with 2048-byte data blocks, and block indexes are 32-bits long. Each inode has 10 direct access blocks, one single indirection block, one double indirection block, and one triple indirection block, as shown below:



c. If a data block is used to hold block pointers for the indirection levels, what is the maximum number of pointers each data block can hold, assuming that the blocks are fully utilized for pointers, and do not hold anything else. (2 marks)

Each pointer is 32 bits = 4 bytes. Each block is 2048 bytes. # of pointers = 2048 / 4 = 512 pointers)

d. What is the maximum size of a single file in gigabytes (2^{30} bytes)? Write your answer to 3 decimal places. (3 marks)

10 direct pointers = 2048 x 10 = 20,480 bytes
First level indirection: 512 pointers, = 512 x 2048 = 1,048,576 bytes
Second level: 512 x 512 pointers = 512 x 512 x 2048 = 536,870,912 bytes
Third level = 512 x 512 x 512 pointers = 512 x 512 x 512 x 2048 bytes
= 274,877,906,944 bytes
Total = 20,480 + 536,870,912 + 274,877,906,944 = 275,414,798,336 bytes
= 256.500 GB

e. What is the maximum size of the partition in gigabytes (2^{30} bytes)? Write your answer to 3 decimal places. Assume that every block in the partition can be used for data. (3 marks)

Maximum number of blocks possible = 2^{32}
Each block has 2048 bytes
# of bytes in total = 8,796,093,022,208
= 8192 GB

Question 5 Virtual Memory

a. We have a computer system with 4GB of main memory, 8GB of virtual memory, and 1MB page sizes. Complete the following table: (5 marks)

# of physical address bits	**32**
# of virtual address bits	**33**
# of bits in offset	**20**
# of bits in frame number	**12**
# of bits in virtual page number	**13**

Now consider a virtual memory system with 4 physical frames and 10 virtual pages and the following virtual page accesses:

2 3 0 2 1 3 5 2 4 1 5

b. How many page faults would we have if we used a FIFO replacement policy? (4 marks)

Access	Page Fault?
2	**Y**
3	**Y**
0	**Y**
2	**N**
1	**Y**
3	**N**
5	**Y**
2	**Y**
4	**Y**
1	**N**
5	**N**

Frame	1	2	3	4	5	6
1	**2**	**5**				
2	**3**	**2**				
3	**0**	**4**				
4	**1**					

NUMBER OF PAGE FAULTS: 7

c. How many page faults would we have if we used an LRU replacement policy? (5 marks)

Time Step	Access	Page Fault?
1	2	**Y**
2	3	**Y**
3	0	**Y**
4	2	**N**
5	1	**Y**
6	3	**N**
7	5	**Y**
8	2	**N**
9	4	**Y**
10	1	**Y**
11	5	**N**

Frame	1	2	3	4	5	6
1	**2(~~1~~, 4, 8)**					
~~2~~	**~~3(2, 6)~~**	**1(10)**				
3	**~~0(3)~~**	**5(~~7~~, 11)**				
4	**~~1(5)~~**	**4(9)**				

Timing is shown in brackts. E.g. 4(9) means address 4 at time 9.

NUMBER OF PAGE FAULTS: 7