Computer Science 252 Quiz #2 **Name:__________________**

1 [10 points] Your task is to design the an I/O system with the best cost performance given a limited budget. Performance is measured as MB/s using average disk access sizes. Ignore the cost of the CPU, and assume that the main memory and processor has enough bandwidth to keep up with whatever I/O system you design. Also the operating system in the file server has no file cache because all the main memory is needed for the the application. Here are the ground rules:

- The storage capacity must be at least 10 GB.
- Output to the disk cannot be buffered (the data must be written on the disk before the write is considered complete)
- To maintain a reasonable response time, no resource can be used at more than 80% of its rated maximum bandwidth
- The maximum number of disks connected over a string to a disk controller is 10.
- The ratio of disk reads to disk writes is 2:1.
- The average size of a disk access is 4 KB
- The seek distribution is like the UNIX timesharing workload in Figure 9.40 on page 559.
- Assume that every disk I/O requires an average rotational delay.

Here are the costs and performance of the components:

Item	*Cost*	*Maximum performance*
Main Memory	\$1000/MB	250-ns for 8 bytes
I/O bus+rack	\$1000	Up to 15 disk controllers
Disk controller+string	\$3000	Extra overhead is 1 ms per I/O event and string bandwidth is 10 MB/sec
3.5 inch disk	\$2000	1 GB storage, rotates at 3600 RPM, seeks of 16 tracks or greater take 10 ms, seeks of 1 to 15 tracks take 5 ms, and there is no seek time for the same track. The maximum transfer rate is 2 MB/s.
2.5 inch disk	\$1100	0.5 GB storage, rotates at 5400 RPM, seeks of 16 tracks or greater take 8 ms, seeks of 1 to 15 tracks take 4 ms, and there is no seek time for the same track. The maximum transfer rate is 1.75 MB/s.
1.8 inch disk	\$600	0.25 GB storage, rotates at 7200 RPM, seeks of 16 tracks or greater take 6 ms, seeks of 1 to 15 tracks take 3 ms, and there is no seek time for the same track. The maximum transfer rate is 1.50 MB/s.

Your maximum budget is \$40,000.

A few comments on possible answers:

- **Memory was useful only if you wanted to add a disk cache**
- **The read to write ratio was only important if you were using a cache**
- **The 80% bandwidth limit primarily applied to the string limit, but it was hard to get 10 disks and a file cache to trasnsfer as much as 8 MB/s. Many people limited the disks to 80% of their maximum I/Os per second, which is a reasonable idea**

(a) [1 point] List the cost of the components of your I/O system

Item	*Number*	*Cost*
TOTAL		

- **The basic disk design options were:**
  10 3.5 inch disks+1 controllers+1 bus = $24,000
  20 2.5 inch disks+2 controllers+1 bus = $29,000
  40 1.8 inch disks+4 controllers+1 bus = $37,000
  The remaining money can be used as a disk cache to help reads

(b) [4 points] Draw a diagram of your I/O design

(c) [3 points] Calculate the performance of your I/O system. List all assumptions.

Common mistakes were
Not including 0 length seeks in average seek time
Not including rotational latency in disk access time
Not including block transfer time in disk access time

(d) [1 point] What is the MB/s per $1000 of your system?

(e) [1 point] What is the strengths of your I/O system compared to other options?

2. [28 points] The cost/performance of three DLX processors is to be examined, taking two approaches to performance Except where stated otherwise assume the same hardware resources as in the book for each approach:

Model A: The superscalar DLX issues up to one integer and one floating point instruction each clock cycle.
Model T: Similar to the model A, this superscalar DLX also issues up to one integer and one floating point instruction each clock cycle, but includes hardware for the Tomasulo Algorithm as described in Section 6.7.
Model V: The vector version of DLX, called DLXV in Chapter 7. It includes support for chaining.

All three models have an instruction cache containing 4096 bytes using 16 byte blocks that uses direct mapping. The models A and T also have a 4096-byte data cache with 128 byte blocks that is also direct mapped. This is a write through cache.

All models have the same hardware building blocks:
A fully pipelined adder whose latency is 3 clock cycles
A fully pipelined multiplier whose latency is 5 clock cycles
The memory is contains 16 banks, with each bank being 64-bits wide.
The latency for a memory access is 10 clock cycles. The memory system delivers one 64-bit quantity per bank after the initial memory latency

(a) [5 points]
Here is the cost of each resource needed to construct the models (a "microacre" is an imaginary unit of silicon area):

1 bit of static RAM:	0.001 microacres
1 bit of register:	0.002 microacres
64-bit pipelined floating point adder:	1 microacres
64-bit pipelined floating point multiplier:	1 microacres
Vector control unit (not including registers):	2 microacres
Tomasulo control unit:	2 microacres
Single Tomasulo reservation station:	0.1 microacres
Integer control unit and ALU (not including registers):	1 microacre
Cache controller (including address comparator and muxes but no RAM):	1 microacre

What is the cost of each model? Fill in the rest of the categories and sum the resources of each model. (Round to the nearest microacre.)

Common Mistakes

Forgetting to include tag memory space (-2)
Forgetting that 4KB cache is bytes, not bits (-1)
Miscalculating size of vector registers: 8*64*64 (-1)
Forgetting vector registers (-1)

Model A		Model T		Model V	
Integer control unit and ALU	1	Integer control unit and ALU	1	Integer control unit and ALU	1
Integer regs (32*32*2)	2	Integer regs (32*32*2)	2	Integer regs (32*32*2)	2
Scalar FP regs (16*64*2)	2	Scalar FP regs (16*64*2)	2	Scalar FP regs (16*64*2)	2
FP adder	1	FP adder	1	FP adder	1
FP multiplier	1	FP multiplier	1	FP multiplier	1
Instruction Cache Control	1	Instruction Cache Control	1	Instruction Cache Control	1
Data Cache Control	1	Data Cache Control	1	---	---
---	---	Tomasulo control unit	2	Vector control unit	2
(total so far)	(9)	(total so far)	(11)	(total so far)	(10)
Instruction Cache Tag RAM (21*256*1)	**5.4**	**Instruction Cache Tag RAM (32-lg(16)-lg(4k/16)+ 1 valid) *(4k/16)*1**	**5.4**	**Instruction Cache Tag RAM (20*256*1)**	**5.4**
Instruction Cache Data RAM (4k*8)	**32.8**	**Instruction Cache Data RAM(4k*8*1)**	**32.8**	**Instruction Cache Data RAM (4k*8)**	**32.8**
Data Cache Tag RAM (22*32*1)	**0.7**	**Data Cache Tag RAM (32-lg(128)-lg(4k/128)+ 1 valid) *(4k/128)*1**	**0.7**	**---**	---
Data Cache Data RAM(4k*8*1)	**32.8**	**Data Cache Data RAM(4k*8*1)**	**32.8**	**---**	---
		3 Tomasulo adder reservation stations	**0.3**	**Vector Registers (8*64*64*2)**	**65.5**
		2 Tomasulo multiplier reservation stations	**0.2**		
		Load Buffer (6*64*2)	**0.8**		
		Store Buffer (3*(32+ 64)*2)	**0.6**		
TOTAL	**81**	**TOTAL**	**85**	**TOTAL**	**114**

Here is the kernel that will be used for the comparison. Q,X, and Z are double precision floating point numbers, with the dimensions of X and Z being 1024. Q is in Fq, K is in Rk, the base address of array Z is in Rz, and the base address of array X is in Rx.

```
C*** KERNEL 3  INNER PRODUCT
C
          Q = 0.0
          DO 3 K = 1,1024
     3    Q = Q + Z(K)*X(K)
```

Each program should take no more than 20 lines. **Given the time available, do not spend much time on optimization of the code.** The loop at the bottom can be changed if necessary.

(b) [3 points] Write an efficient DLX version of this program for Model A. (The end of loop code is written at the bottom just to give you as much space as you need):

Label: Instruction (integer)		*Instruction (FP)*		*Comment*
1. ADDI	Rk,R0,#1	MOVI2FP	Fq,R0	*K = 1 and Q = 0*
2.		**MOVD**	**F10,Fq**	***F10 = 0 (to start loop)***
3.		**MOVD**	**F12,Fq**	***F12 = 0 (to start loop)***
4.Loop: **LD**	**F0,(Rz)**			***Load Z[k]***
5. **LD**	**F2,(Rx)**	**ADDD**	**Fq,Fq,F10**	***Load X[k]& Q=Q+Z*X[k+1]***
6. **LD**	**F6,(Rz)8**			***Load Z[k+1]***
7. **LD**	**F8,(Rx)8**	**MULTD**	**F4,F0,F2**	***Load X[k+1] & Z[k]*X[k]***
8. **ADDI**	**Rz,Rz,#2*8**			
9. ADDI	Rk,Rk,**#2**	**MULTD**	**F10,F6,F8**	***K=K+2 & Z[k+1]*X[k+1]***
10. SLE	R1,Rk,#1024			*<= 1024?*
11. BNE	R1, Loop			*Go to loop if <=*
12. **ADDI**	**Rx,Rx,#2*8**	**ADDD**	**F12,F12,F4**	***Q'=Q'+Z*X[k];delay slot***
13.		**ADDD**	**Fq,Fq,F12**	***Q=Q+Q'; outside loop***

(c) [2 points] What changes (if any) would you make to run this program on Model T. Why?

I wouldn't unroll loop as above since Tomasulo algorithm hardware gets the same effect by using buffers, renaming registers, and allowing out-of-order execution.

(d) [3 points] Write an efficient DLXV version of this program for Model V:

Label:	*Instruction*	*Comment*
1. LV	**V0,(Rx)zero**	***Set V0 = 0***
2. ADDI	Rk,R0,#1	*K = 1* ***(Overlap with LV)***
3. MOVI2FP	Fq,R0	*Q = 0* ***(Overlap with LV)***
4.Loop: **LV**	**V1,(Rz)**	***V1 = Z[k..k+63]***
5. **ADDI**	**Rz,Rz,#64*8**	***Increment Z base during vector load***
6. ADDI	Rk,Rk,#**64**	*Increment K by* ***64 (overlap with LV)***
7. SLE	R1,Rk,#1024	*<= 1024?* ***(overlap with LV)***
8. **LV**	**V2,(Rx)**	***V2 = X[k..k+63] (adjust for early increment)***
9. **MULTV**	**V3,V1,V2**	***V3 = V1* V2 (chained)***
10. **ADDV**	**V0,V0,V3**	***V0 = V0 + V3 (chained); partial sum***
11. BNE	R1, Loop	*Go to loop if <=*
12. **ADDI**	**Rx,Rx,#64*8**	***Increment X base (delay slot)***

13. **SV** **V0,(Rx)** ***Store V0 in memory so you can add it up***

14. **ADDI** **R2,Rx,#64*8** **Set ending address (overlap with SV)**

15.**Loop2: LD** **F2,(Rx)** ***Load element of V0***

16. **SLT** **R1,Rx,R2** ***Rx < 64*8 (load delay slot) ; early so < vs <=***

17. **ADDD** **Fq,Fq,F2** ***Q = Q + V0[i]***

18. **BNE** **R1, Loop2** ***Go to loop if <***

19. **ADDI** **Rx,Rx,#8** **Increment address (delay slot)**

(e) [1 point] What is the **total number** of instruction cache misses for each version of the program? (Note: this question is **not** the miss rate). Then give the number of clock cycles for instruction cache misses. Explain your calculation.

Simply the number of unique instructions divided by 4 (16 byte block / 4 bytes/instr) since there all the code fits in the cache. (You must round up if the instructions partially fill one block)
A (unrolled) = 18 instructions => 5 misses
T (rolled) = 12 instructions => 3 misses
V = 19 instructions => 5 misses

Instruction cache miss penalty is memory latency plus the block size in bytes divided by the number of bytes transfered per clock cycle.
= Memory latency + (block size / 8 – 1) = 10 + 16/8 – 1 = 11
(or could say that need an extra clock for the first 8 bytes)
Model A : 55 clock cycles
Model T: 33 clock cycles
Model V: 55 clock cycles

(f) [2 points] What is the **total number** of data misses for each version of the program? (Note: this question is **not** the miss rate). Then give the number of clock cycles for data cache misses. Explain your calculation.

Assuming arrays are not alligned in memory so that they line up exactly on top of each other in a cache (e.g., Z[0] does not use the same cache block as X[0]), you only have compulsory misses
=> 2*1024 references / (128 bytes in block/ 8 bytes per element)
= 2048 / 16 = 128 misses
Data cache miss penalty is memory latency plus the block size in bytes divided by the number of bytes transfered per clock cycle.
= Memory latency + (block size / 8 – 1) = 10 + 128/8 – 1 = 25
(or could say that need an extra clock for the first 8 bytes)
Model A : 3200 clock cycles
Model T: 3200 clock cycles
Model V: 0 clock cycles on data cache misses, since no data cache!

(g) [10 points] Calculate the number of clock cycles it takes to execute the program for each model of DLX. Show your work.

Model A:

The unrolled loop is scheduled so that there are no stalls due to structural hazards
Time in loop ignoring cache misses:
3*1 + 1024/2 *9 +1*3 = 4614
Total time including caches = 4614 + 55 + 3200 = 7869

Model T:
The rolled loop is avoids stalls automatically, with scalar and floating point overlaped. Let's ignore the extra cycles until it reaches steady state, and assume all integer instructions are overlaped with floating point. The limit to the number of operations completed per loop is then the number of buffers and reservation stations.
6 load buffers means at most 6 operands can be waiting
3 add reservation stations means at most 3 adds can be pending
2 mult reservation stations means at most 2 adds can be pending

Time in loop ignoring cache misses:
1 + 1024 * (floating point latency / number of pending operations)
= 1 + 1024 * (8/2) = 4097
Assuming that the Model T stalls during a cache miss, total time is
= 4097 +33 + 3200 = 7330

An alternate assumption is that the Tomasulo hardware does not stall during a cache miss, but continues. Then the question is what is the delay during a cache miss when operating at full efficiency. Assume that as soon as a load buffer is freed the access is made to the element of Z that causes the miss. During the 25 clock cycles of the miss the pipeline will drain (since it takes only about 10 clocks to complete the remaning operations). The Tomasulo hardware will then establish all the dependencies for the remaining operations. Since the pipeline is drained, it will take just as long to restablish the pipeline once the missing operands are fetched. Hence, there is at most a small performance advantage in allowing the cache to continue duing a miss. (In fact, a miss on Z will be followed immediately by a miss on X and you need an element from each to continue, so the time gap is really that of two cache misses and whatever potential overlap is possible during the two misses while waiting for both misses less the time to refill the pipeline after the misses.) To make an improvement you would need
to assume an "early restart" cache that did not wait until all the of the data of a block arrived. (And since a miss on Z is followed by a miss on X, you would still have to wait the full miss penalty on Z plus the latency on X.)

Model V:

	Instruction	*Start*	*Complete*
Prologue	LV	1	1+10+64 = 75
Loop1	LV	1	1+10+64 = 75
	LV	75+1 =76	75+1+10+64 = 150
	MULTV	76 + 1 + 10 = 87	150 + 1+ 5 = 156
	ADDV	87+1+5 = 93	156 + 1 + 3 = 160
Epilogue	SV	1	1+10+64 = 75
Loop2	LD	1	11 clock cycles (no cache!)
		12	15 (4 instructions, one clock each)

Vector time = Prologue + Loop1* 1024/64 + Epilogue + Loop2*64
= 75 + 160*16 + 75 + 15*64 = 3670 clock cycles

Give a short explanation why the winner is fastest:

Model V is fastest because
(1) The calculation is quicker because the floating point units are being run at the maximum pipeline rate a larger fraction of the time
(2) Model V pays two memory latencies every 64 elements (vector load start ups) vs. 2 memory latencies every 16 elements (determined by block size before another cache miss)
(3) Model V begins execution as soon as the first word from the second vector is fetched (11 clocks) as opposed to waiting for the full cache to be loaded (25 clocks)
(4) There is no reuse of the data, so the only benefit of the data cache is spatial locality, and the same effect if achieved by vector load
(5) While the code is not completely vectorizable, by doing the vector add to achieve the partial sum in the main loop most of the adds are performed at vector rates.
(In fact the vector operations is 64*16 = 1024 multiples + 64*16 = 1024 adds vs 64 scalar adds, giving a vectorization rate of 2048/2112 or 97%.)

(h) [2 points] Using the cost information from your answer to (a) and your performance calculation from your answer to (g), which model has the best cost-performance?

Assuming all the cost is in the processor

Model	Area	Area^3 (see p. 60) (≈ Cost)	Clocks	Fl. Pt. Ops per clock (≈ Performance)	Perf /Cost (*10^7)	Normalized to Model A
A	81	531,441	7869	.26	4.9	100%
T	85	614,125	7330	.28	4.5	93%
V	114	1,481,544	3670	.56	3.8	77%

Model A has the best cost performance

Also give a short explanation why you think the winner has the best cost performance.

Since cost goes with the 3rd power of area, the smaller area of A offsets the slightly higher speed of T. And while Model V is more than twice as fast as Model A, its larger area results in a cost that is almost three times that of A, hence model A still has better cost performance.