Midterm II
December 1, 2003
CS252 Graduate Computer Architecture

<table>
<thead>
<tr>
<th>Problem</th>
<th>Possible</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
[ This page left for $\pi$ ]

3.141592653589793238462643383279502884197169399375105820974944
Problem #1: Checking the bits

The error correction coding process can be viewed as a transformation from one space of bits (the unencoded data) to another (the coded data).

1a) For linear block codes, there are two matrices, the Generator (G) and Parity Check (H). What are they, how big are they (in terms parameters n and k), and how are they used?

1b) Define the minimum Hamming distance, \( d_{\text{min}} \), of an error correction code (make sure that this definition makes sense for codes like Reed-Solomon of RAID-5 that have more than one bit per symbol):

1c) Name a constraint on the columns of the parity check matrix (H) which would indicate that a code had minimum distance \( d_{\text{min}} \).

1d) For a code with \( d_{\text{min}} \) minimum distance, what is the formula for the maximum number of errors, \( E_{\text{detect}} \), that can be reliably detected? Why might you be able to detect more than the \( E_{\text{detect}} \) errors?
1e) For a code with $d_{\text{min}}$ minimum distance, what is the formula for the maximum number of random errors, $E_{\text{correct}}$, that can be corrected reliably?

1f) How does your answer to (1e) change if you have knowledge of which symbols are bad (i.e. which symbols have been erased). How might erased symbols be detected?

1g) What is a systematic error-correction code? Why is this desirable?

1h) Suppose we start with a non-systematic, linear code described by generator $G$ with distance $d_{\text{min}}$. How can we produce a systematic code with generator $G'$ from $G$ that still has distance $d_{\text{min}}$? (hint: what happens if you subtract two code words from the original code? Now think of this operation on $G$).
i) Suppose you are willing to use 5 parity bits. What is the maximum number of data bits \(k\) that you can protect with a Hamming code and still get a distance 4 code? What are the matrices \(G\) and \(H\) for this code? Prove that your code has distance 4.

j) Explain the function that a circuit that detected and corrected errors in the above code would have to perform. You should tell how to (1) get an “Error detected” signal, \(E_{\text{detect}}\) (2) get a double-error signal, \(E_{\text{double}}\), and (3) a bad bit identifier, \(E_{\text{bad}}\). Note that \(E_{\text{detect}}\) and \(E_{\text{double}}\) are Boolean (true/false) signals while \(0 \leq E_{\text{bad}} < n\) is a number that defines which bit is bad.
Problem #2: Queuing the DRAM

Suppose that we have a processor connected through a general network to a single DRAM “server”. Suppose that the computation on each of these processors have the following average behavior, and that each of the individual processors are completely uncorrelated (so that the combined behavior appears memoryless):

Instruction Mix: 20% loads, 10% stores, 20% floating-point, 20% integer, 30% branches
Base CPI without cache misses: 1.6.

Now, assume that each processor has a single-level of cache, with separate I and D caches. Assume a write-back cache, 8-word cache lines, a 99.5% hit rate on instructions, a 90% hit rate on loads and a 80% hit rate on stores. Cache hits happen at 1 per cycle with no stalls. Also, assume a write-allocate policy, and that 12.5% of the cache lines in the data cache are dirty and must be written back to memory on a miss. Assume no self-modifying code.

Assume that the network is one word (32-bits) wide and that it operates at 1 word/processor cycle bandwidth. When a cache line is loaded into the cache from the network or sent from the cache back into the network, the processor is tied up during the time that the data is actually transferred.

A one-way network trip to or from the DRAM takes 10 processor cycles. Assume that the DRAM can handle only one request at a time and that it has a service time of 6 processor cycles for the first word + 2 processor cycle for each additional word.

2a) What is the maximum rate at which the DRAM can service requests? (in accesses per processor cycle)? Do not account for the network yet, just the DRAM “server”:

2b) Assume no queuing in the network or DRAM (only one request outstanding at a time). The network and cache are pipelined for fills, however. Also, assume a single processor with a blocking cache. Assume that the time for sending data into the network or taking data from the network must be accounted for (i.e. when filling or replacing data in the cache). What is the penalty that a processor suffers when requesting data from memory?
2c) How long does it take to flush a dirty cache line from the processor cache into the network?

2d) Taking (2b) and (2c) into account, what is the CPI for the processor with a blocking cache? 
*Hint: Don’t forget instruction misses and dirty flushes from cache back to memory.*

2e) What is the RATE (in accesses per processor cycle) at which requests are made to the DRAM?

2f) What is the DRAM utilization? Can this ever be greater than one without queuing? Why or why not?
2g) Next, assume that we have queuing of requests at the DRAM. Assume that we have a non-blocking cache that can issue many requests (unbounded) without stalling. Assume that the processor can make forward progress with an outstanding instruction miss (weird assumption, but makes math easier). Finally, assume a fully pipelined network, but not fully pipelined DRAM. Cache fills and replacements still must happen at network bandwidth, but now we can overlap the network time.

Question: What is the rate of requests into the network now? What is the DRAM utilization? (Hint: now, the processor is only stalled for the time that data is transferred to/from the network, not during the network latency).

2h) Now, assume that the processor requests are exponentially distributed. The DRAM service time is deterministic (i.e. not exponentially distributed). What is the average time that a request spends at the DRAM, including any queuing time?

2i) What is the average number of cache misses for load instructions that the processor will have outstanding at any one time (hint: remember Little’s law and the average time to service a request as seen by the processor)?
Problem #3: Producing a Disk Subsystem

Suppose that we build a disk subsystem to handle a high rate of I/O by coupling many disks together. Properties of this system are as follows:

- Uses 10GB disks that rotate at 10,000 RPM, have a data transfer rate of 10 MBytes/s (for each disk), and have a 8 ms seek time, 4 KByte block size
- Has a SCSI interface with a 2ms controller command time.
- Has an average I/O size of 32 KByte
- Is limited only by the disks
- Has a total of 20 disks

Each disk can handle only one request at a time, but each disk in the system can be handling a different request. The data is not striped (all I/O for each request has to go to one disk).

3a) What is the average service time to retrieve a single disk block from a random location on a single disk, assuming no queuing time (i.e. the unloaded request time)?

3b) Assuming that the OS makes a series of single block requests (waiting for one to complete before requesting the next one), what is the bandwidth that can be achieved?

3c) What is the service time for a 32 KByte sequential request? What is the bandwidth that can be achieved on consecutive requests now?
3d) Given the average I/O size of 32KB again, what is the maximum number of I/Os per second (IOPS) for the whole disk system?

Suppose that we decide to improve the system by using new, better disks. For the same total price as the original disks, you can get 11 disks that have 19GB each, rotate at 12000 RPM, transfer at 12MB/s and have a 6ms seek time.

3e) What is the average unloaded service time for random 32KByte requests in the new system?

3f) What is the maximum number of IOPS in the new system?
Treat the entire system as a M/M/m queue (that is, a system with m servers rather than one), where each disk is a server. All requests are in a single queue. Assume that both systems receive an average of 800 I/O requests per second. Assume that any disk can service any request.

3g) What is the mean response time of the old system? The new one? You might find the following equation for an M/M/m queue useful:

\[
\text{Server Utilization (}\zeta\text{)} = \frac{\lambda}{\frac{1}{\text{Time}_{\text{server}} / m}} = \lambda \times \text{Time}_{\text{server}} \times \frac{\zeta}{m(1 - \zeta)}
\]

3h) Which system has a lower average response time? Why?
Question #4: Grab Bag

4a) What are the 3 C’s of cache-miss classification? What is the 4th C of multiprocessing?

4b) What is the difference between access time and cycle time in DRAM? Why should there be two such numbers?

4c) Explain why DRAM must be refreshed. What does the memory system have to do in order to refresh DRAM?

4d) How does utilizing multiple DRAM banks help the overall throughput of a memory system? Why should a prime number of banks be desirable?
4e) What is a victim cache, what does it help, and why is it a better engineering alternative than other options?

4f) How can parallelism (such as in a vector processor) be used to reduce to total energy consumed by a computation? Why doesn’t a superscalar processor get this advantage?

4g) What is the AMAT performance equation for a system with 2-levels of cache and DRAM?

4h) List three different ways in which the compiler can reduce cache misses in a program.
4i) Explain the difference between *coherence* and *consistency*:

4j) What does it mean for a multiprocessor to be sequentially consistent? Why is this a desirable requirement for a multiprocessor?

4k) Name the three conditions sufficient for sequential consistency that were presented in class and draw a diagram illustrating these conditions:

4l) What is *release consistency*, and why was it proposed as an alternative to sequential consistency?
[This is another blank page for you to use]