Full Name:______________________________________
Email:______________________________________

CS 15-740, Fall 1997

Final Exam

Dec. 9, 1997

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name on your exam copy.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 184(!) points.
- If you feel certain details are missing from the problem statement, you may make reasonable assumption, but state them clearly.
- This exam is OPEN BOOK. You may use any books or notes you like. You're even allowed to leave the room and do the exam elsewhere, although any form of collaboration is prohibited.

Do not write below this line

Superscalar	Consistency	Sorting Networks	Cilk	IP Routing	RAID
Total: 184: _____					

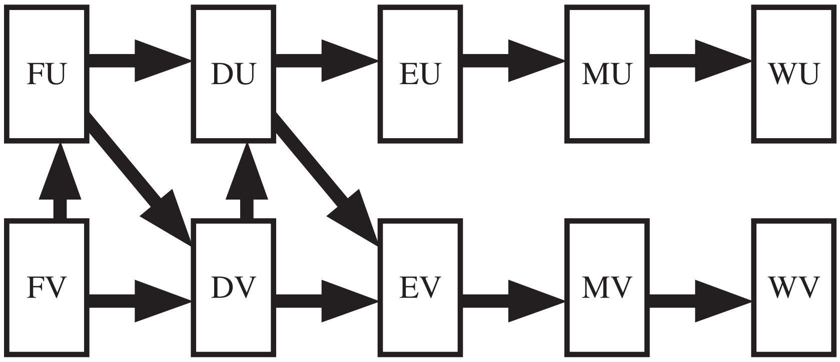

Figure 1: Superscalar MIPS Pipeline

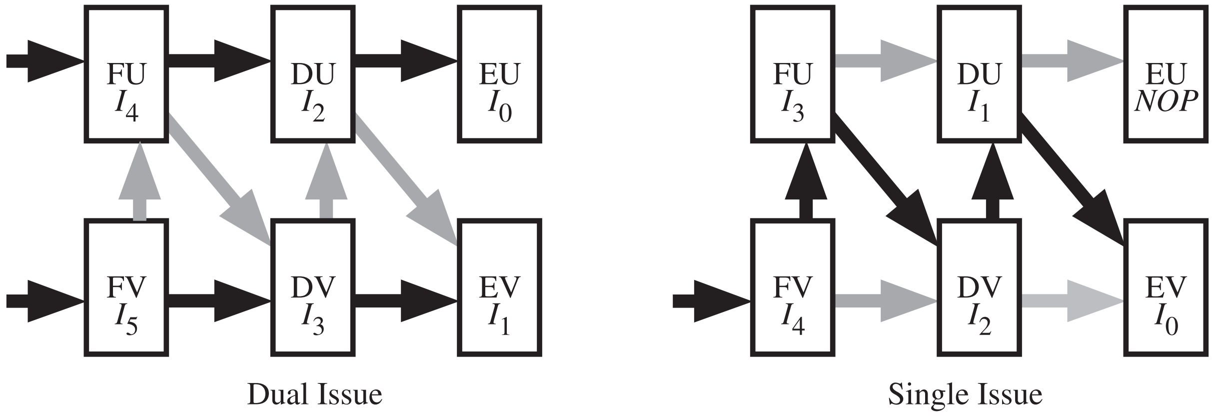

Figure 2: Data Transfers During Instruction Issue. Disabled connections shown in gray.

1. Superscalar Processor Implementation

The following set of problems concern a superscalar implementation of a MIPS processor. Like the MIPS-I architecture, we will assume a single load delay slot, i.e., the instruction immediately following a load cannot reference the loaded result. We will focus only on integer arithmetic, loads, and stores.

The overall structure, shown in Figure 1, consists of a pair of 5-stage pipelines, named “U” and “V” [1] For any pair of parallel pipe stages (e.g,. EU and EV), the instruction in the U pipe precedes the one in the V pipe in program order.

Each pipeline has the usual set of stages performing the same operations as we've studied in class and implemented with the MIPS simulator. The following restrictions are imposed on the two pipelines.

- The U pipe is only capable of executing integer instructions. Loads and stores must be handled by the V pipe.
- Once an instruction enters either of the execute stages (EU or EV), it is not allowed to stall. Extensive bypassing is provided to minimize the effects of hazards.

To handle these conditions, the processor contains "issue logic" which governs the movement of instructions from the decode to the execute stages. Assume initially that instructions I_0 and I_1 are in stages DU and DV, respectively, and that instructions I_2 and I_3 are in stages FU and FV. The two forms of issue are illustrated in Figure 2:

Double Instructions I_0 and I_1 are allowed to proceed down their respective pipelines. The next two instructions (I_2 and I_3) are moved into the decode stages, and a new pair of instructions (I_4 and I_6) is fetched.

Single Only instruction I_0 is allowed to proceed down the pipe, and it will be shifted to the V pipe (i.e., to EV). A bubble will be injected into the U pipe at EU (indicated as *NOP*). The decode stages will be filled by the current instruction in DV (I_1), and the current instruction in FU (I_2). The fetch stages will be filled by the current instruction in FV (I_3) and a new instruction fetched into FV (I_4).

Assume that the register file is fully ported (2 write ports and 4 read ports), that it performs writes before reads, and that bypassing is present wherever it is required.

Problem 1. (4 points):

How should the register file resolve the condition where both write ports are given the same register ID? Explain.

[1] Not the same as the Intel Pentium's U and V pipes.

Problem 2. (24 points):

Illustrate how the following instruction sequence would be executed by this pipeline by filling in the following pipeline diagram. Two copies of the diagram are included, in case you mess up the first one. Include the following information:

- The stage, if any, that the instruction is in.
- Bypassing, indicated by an arrow from stage generating the value to the stage consuming it. Label the arrow with the register number being bypassed.
- A heavy vertical line spanning two rows and following two decode stages, indicating the time points at which the processor must perform a single issue.

Instr	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
li \$4,16	FU	DU										
addu \$6,\$4,\$4	FV	DV										
li \$5,19		FU										
addu \$4,\$4,\$5		FV										
lw \$7,0(\$4)												
nop												
addu \$5,\$4,\$7												
sw \$5,0(\$4)												

Instr	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
li \$4,16	FU	DU										
addu \$6,\$4,\$4	FV	DV										
li \$5,19		FU										
addu \$4,\$4,\$5		FV										
lw \$7,0(\$4)												
nop												
addu \$5,\$4,\$7												
sw \$5,0(\$4)												

Problem 3. (8 points):

For each of the single issue cases you indicated in your pipeline diagram, explain why they occurred.

Problem 4. (8 points):

Extend the following instruction sequence to one for which all of the following bypass conditions arise: EU–EU, EU–EV, EV–EV, and EV–EU. Fill in the pipeline diagram as before.

Instr	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
li $4,16	FU	DU										
li $5,19	FV	DV										
		FU										
		FV										

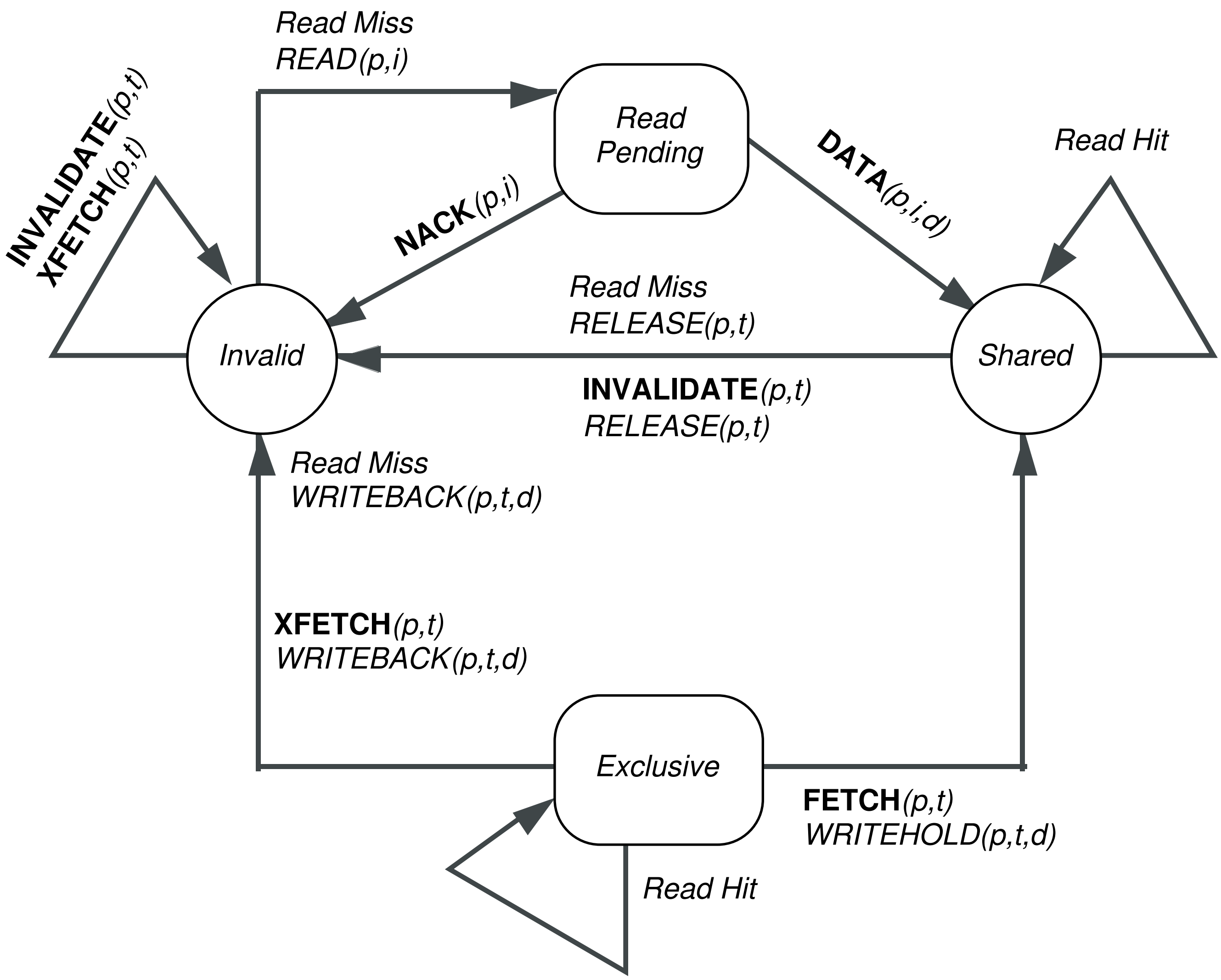

Figure 3: Cache Protocol for Read and Network Monitoring Operations

2. Network-Based Cache Consistency Protocol

The following problem concerns the network-based cache consistency protocol presented in class on November 24 and December 3. In the following discussion, we assume each cache has a single attached processor p, and therefore we use the terms "processor" and "cache" interchangeably. We assume each block a has a "home" i.e, a directory responsible for its management.

Figures 3 and 4 provide a condensed version of the protocol for handling read operations for the cache and the home directory, respectively. Also shown is the handling of unsolicited messages received by the cache and unsolicited evictions sent by the cache to the home directory. This version corresponds to the one presented in lecture on December 3 before any additional optimizations.

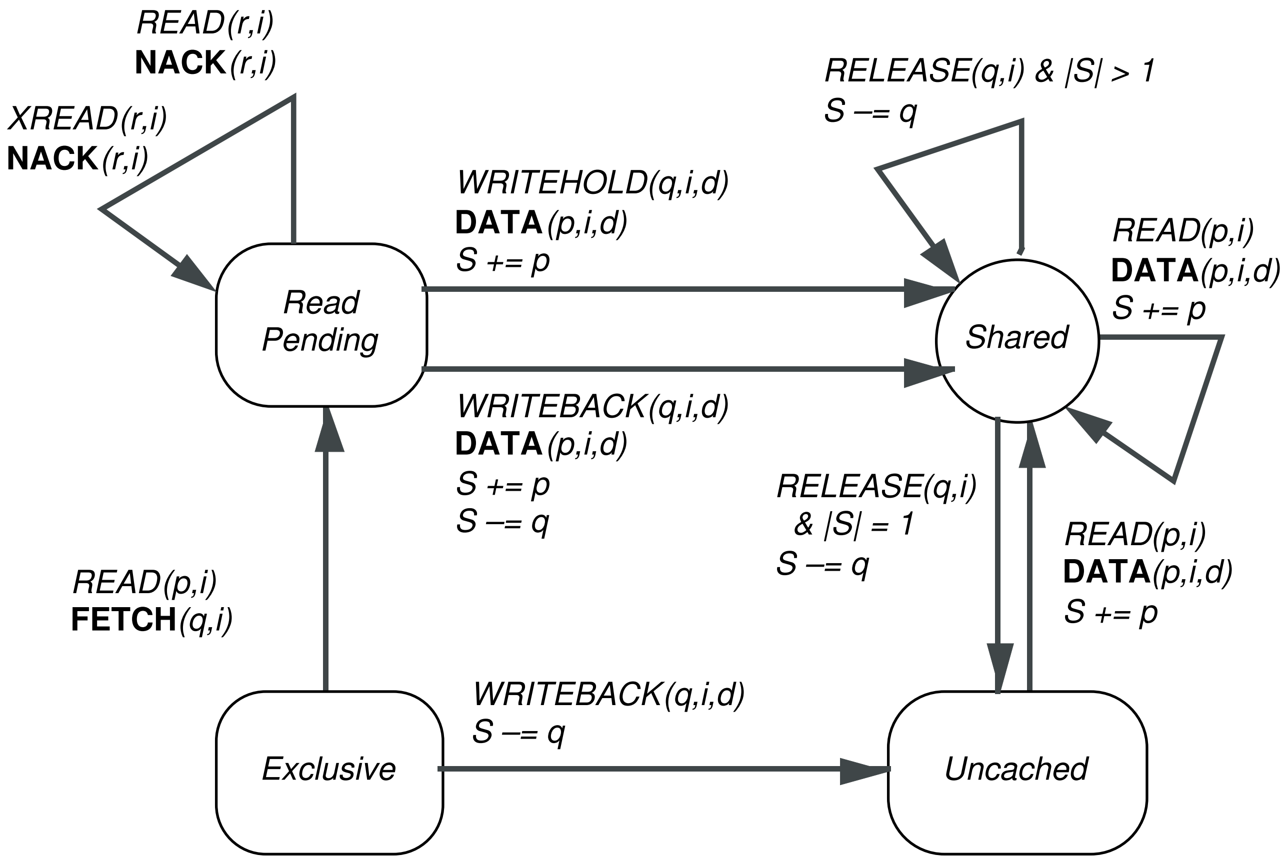

Figure 4: Directory Protocol for Read and Unsolicited Eviction Operations

These figures use the following notation:

- p The processor making a read or write request.
- q The current copy holder(s) of the block.
- r A different processor making a read or write request.
- i The block being read or written
- t The block currently held by the processor cache
- d The block data.
- S The set of copy holders

The following are the messages that may be sent from a cache to a home directory controller:

$READ(p, a)$	Processor p wants to read block a
$XREAD(p, a)$	Processor p wants to write block a
$WRITEBACK(p, a, d)$	Processor p evicts exclusive block a containing data d.
$WRITEHOLD(p, a, d)$	Processor p shares previously exclusive block a containing data d.
$RELEASE(p, a)$	Processor p evicts clean block a.

The following are the messages that may be sent from a home directory controller to a cache:

INVALIDATE(p, a)	Request processor p to give up its copy of block a.
FETCH(p, a)	Request processor p to provide a copy of block a.
XFETCH(p, a)	Request processor p to return its copy of block a.
DATA(p, a, d)	Provide data d for block a to processor p.
NACK(p, a)	Unable to fulfill request by processor p for block a.

Problem 5. (24 points):

Suppose we want to enhance the protocol to have the remote copy holder of an exclusive block forward data directly to a requesting cache, rather than passing it through the home directory. This feature is present in most real protocols, as discussed in H&P at the bottom of p. 685. You are to develop an extension of the protocol to implement this feature for supporting processor read operations. Assume that the protocol for write operations will not change, but it must continue to work correctly.

Document the following about your solution:

A. Any new message types you add, including its purpose, source, destination, and operands.

B. Any message types you can eliminate from the original protocol.

C. Any changes to the protocol state diagrams required. Show these directly on the diagrams.

D. An explanation of how your modified protocol handles the case where a cache evicts a block, but before the home receives the *WRITEBACK* message, it sends a request for the cache to forward its copy of the block to a second cache.

3. Sorting Networks and Nonblocking Networks

A sorting network is a multi-stage network that accepts one key value on each of its input ports, and permutes the keys so that they appear on its output ports in sorted order. The traditional way of drawing a sorting network is to connect horizontal lines that carry data (sometimes confusingly called *registers*) with vertical lines that represent comparison-exchange operations. A sorting network with two horizontal lines is shown below.

```
input         output
 key           key

  45 ----^---- 13
         |
  13 ----^---- 45
```

The network has one input port and one output port for each of its horizontal lines. Data flows from left to right along the horizontal lines. The comparator

```
         ^
         |
         ^
```

examines the two key values that arrive at its endpoints from the left, and exchanges them, if necessary, so that the smaller key is on top, and the larger key on bottom.

The network below sorts four keys.

```
input         output

     ----^----^-----------
         |    |
     ----^----|-^----^----
              | |    |
     ----^----^-|----^----
         |      |
     ----^------^---------
depth    1     2     3
```

Notice that a comparator can connect two horizontal lines that are not adjacent. Notice also that the two comparators labeled with depth 1 can be performed in parallel, since their endpoints are disjoint. The same is true for the two comparators labeled with depth 2. In general, the depth of a sorting network corresponds to the parallel time for the network to sort its keys. The size (or work) is defined as the total number of comparators, and represents the amount of hardware required.

Problem 6. (2 points):

Show the result of each comparison in the following sorting network. To help you get started, a few of the results are already drawn in. (This is Batcher's bitonic sorting network.)

```
1 ----^--1---^--1-^----------^----^------^----
      |      |    |          |    |      |
7 ----^--7-^-|--2-^--------^-|----|-^----^----
         | |             | |    | |
5 ----^--2-^-|--7-^------^-|-|----^-|----^----
      |      |    |      | | |      |    |
2 ----^--5---^--5-^----^-|-|-|------^----^----
                       | | | |
6 ----^------^---------^-|-|-|----^-----------
      |      |           | | |    |
3 ----^----^-|----^------^-|-|----|-^----^----
         | |    |        | |    | |    |
2 ----^----^-|----^--------^-|----^-|----^----
      |      |               |      |
5 ----^------^---------------^------^---------
```

Problem 7. (4 points):

Show that following network, which is based on the butterfly network, is not a sorting network.

```
----------^----^------^----
          |    |      |
--------^-|----|-^----^----
        | |    | |
------^-|-|----^-|----^----
      | | |      |    |
----^-|-|-|------^----^----
    | | | |
----|-|-|-^----^------^----
    | | |      |      |
----|-|-^------|-^----^----
    | |        | |
----|-^--------^-|----^----
    |            |    |
----^------------^----^----
```

In Assignment 5 we showed that a nonblocking network can be constructed out of two back-to-back majority networks. In this problem we will show that a nonblocking network can also be constructed out of two back-to-back sorting networks.

As before, our goal will be to route disjoint paths between the input ports and the output ports of the network. Our notion of disjoint will be cosmetically different, however. For this problem a set of paths is disjoint if at most one path uses each horizontal segment of each horizontal line, where a horizontal segment connects one comparator to the next on the same line. As an example, the set of paths taken by the keys through a sorting network is disjoint.

Our notion of nonblocking will also be slightly different, in a subtle way. In Assignment 5 we defined a network to be nonblocking if no matter what paths have already been established, it is possible to establish a new path, disjoint from all previously established paths, between any unused input port and any unused output port. For this problem we will weaken the definition a little. We will define a network to be nonblocking if there exists an algorithm for finding paths in the network so that no matter what paths have already been established *by that algorithm*, the algorithm can find a new path, disjoint from all previously established paths, between any unused input port and any unused output port.

Problem 8. (24 points):

Prove that two back-to-back sorting networks form a nonblocking network, and show how to find the paths in the network. To answer this question, you will have to specify three things:

1. What key values to use in each of the two sorting networks in order to establish a new path between a previously unused input port and previously unused output port.

2. What key values to use for the previously established paths.

3. What key values to use for the remaining unused input and output ports.

Note that once a path is established from an input port to an output port, it cannot be changed.
Here is a hint: All of the unused input and output ports can use the same key value.
Here is another hint: Paths between unused input and output ports can be changed at any time.
One last hint: There is a short proof, and it should fit in the space below.

4. Inverting Lower Triangular Matrices in Cilk

In this problem you will write a Cilk program to invert a lower triangular matrix. A lower triangular matrix is one in which all of the matrix elements above the diagonal are 0. The 4×4 matrix below, for example, is lower triangular.

```
2.5  0.0  0.0  0.0

1.5  1.0  0.0  0.0

0.4  0.0  0.5  0.0

2.3  4.4  4.5  1.7
```

Lower triangluar matrices seem to be easier to invert than matrices in general. There's a simple recursive algorithm based on the following identity. Suppose that A is an $n \times n$ matrix,

$$A = \begin{bmatrix} A_1 & 0 \\ A_3 & A_2 \end{bmatrix},$$

where A_1, 0, A_3, and A_2 are $(n/2) \times (n/2)$ submatrices. Then A^{-1} is given by the formula

$$A^{-1} = \begin{bmatrix} A_1^{-1} & 0 \\ -A_2^{-1}A_3A_1^{-1} & A_2^{-1} \end{bmatrix}.$$

Notice that A_1 and A_2 are both lower triangular, so A_1^{-1} and A_2^{-1} can be computed recursively. (It's also interesting to note that A^{-1} is lower triangular.)

Your task is to implement the following Cilk function.

```
cilk void inv_low_tri(int n, REAL *A, int an, REAL *C, int cn);
```

This function takes as input an $n \times n$ matrix A (where n is a power of 2), computes the inverse of `A\verb`, and puts the result in an $n \times n$ matrix `C`.

For this problem we won't worry about optimizing the representation of a lower triangular matrix by omitting the zero values above the diagonal. Instead, we'll store these zeros explicitly. As in Assignment 6, you are to assume that the matrices are stored in row-major order. The input parameters `an` and `cn` indicate the distance between the first element in one row and the first element in the next row of `A` and `C`, respectively. To access matrix element A_{ij}, you can use the macro `ELEM(A, an, i, j)`, which is predefined to be `A[i * an + j]`, for convenience.

You may assume that the following functions have already been written, and you may use them if necessary.

```
/* multiply matrices A and B, putting the result in C */

cilk void multiply(int n, REAL *A, int an, REAL *B, int bn,
                   REAL *C, int cn);

/* negate a matrix A (in place) */

cilk void negate_matrix(int n, REAL *A, int an);

/* Allocate a matrix of side-length n (therefore n^2 elements) */

REAL *alloc_matrix(int n);

/* free a matrix */

void free_matrix(REAL *A);
```

The first function computes the product of two $n \times n$ matrices A and B, and puts the result in matrix C. In Assignment 6 you implemented several versions of this function. The second negates each of the entries of an $n \times n$ matrix A

Problem 9. (20 points):

Write your Cilk code in the space provided below. Your code should exploit the parallelism inherent in the forumla for A^{-1}. Don't forget the base case!

```
cilk void inv_low_tri(int n, REAL *A, int an, REAL *C, int cn){








}
```

Problem 10. (6 points):

Write a recurrence for the depth of your algorithm. Use the function $D_{mult}(n)$ to denote the depth of the matrix multiplication algorithm. Be sure to include the base case!

$$D_{inv\_low\_tri}(n) =$$

Problem 11. (4 points):

Suppose that the matrix multiplication algorithm has depth $D_{mult}(n) = O(\log n)$ (as in Assignment 6, Part II, Problem 4.1(c)). Solve your recurrence for this case.

$$D_{inv\_low\_tri}(n) =$$

5. Routing in IP Networks

An internet router is a machine that is connected to two or more physical networks. For each network, the router has a separate network interface. The job of the router is to pass packets from one network to another. To do this, the router maintains a routing table that maps IP addresses to interfaces (and also to hardware addresses on the corresponding physical network).

The example below shows a small internet.

```
         _________          _________          _________
        |         |        |         |        |         |
        |net  1   |        |net  2   |        |net  3   |
    E---|         |---F----|         |---G----|         |---H
        |         |        |         |        |         |
        |_________|        |_________|        |_________|
```

In this figure, F and G are routers, and E and H are singly-homed hosts. Suppose that both E and F know how to route a packet to H. When E has a packet to send to H, it sends it to F across network 1, which then forwards it to G across network 2, which finally sends it to H across network 3. Now suppose that H has a packet to send to E, but that H does not know the route to E. In this case, H sends the packet to its default gateway, which in this example is G. G then forwards the packet its default gateway, F, which finally sends it to E,

Ben Bitdiddle, an aspiring 15-740 student with a summer job at Cisco, suggests a way to make the routers smarter. Whenever a router receives a packet with IP source address X on interface i, it will update its routing table so that future packets with destination X that arrive at the router will be forwarded out on interface i.

Ben's scheme is a bit risky, however. Consider the following scenario. E and F know how to route a packet to H. E's routing table indicates that a packet for H should be sent to F, and F's routing table indicates that a packet for H should be sent to G. Supposes that E sends a packet to H. H, however, has been down for some time, and the entry for H in G's routing table has expired. So when the packet arrives at G, G forwards it back to its default gateway F.

Problem 12. (4 points):

What happens to the packet next? Recall that each time an IP packet passes from one host to another, its TTL (time to live) counter is decremented by 1.

Problem 13. (4 points):

Next, suppose that host `H` wakes up, and sends a packet to `E`. What happens to this packet?

Problem 14. (4 points):

Suppose that `E` now decides to resend its packet to `H`. What happens to this packet?

Problem 15. (4 points):

Finally, suppose that after a short time `H` sends another packet to `E`. What happens to this packet?

Problem 16. (6 points):

After seeing these examples, Ben proposes a fix. He suggests that a router should never forward a packet back out through the same interface on which it arrived. Construct a small example in which this scheme fails as well.

Problem 17. (6 points):

All of these examples have something in common: a cycle in the routing tables for the IP address of `E`. Under what conditions will a cycle in the routing tables prevent packets from reaching a particular address `E`?

6. RAID: Redundant Array of Inexpensive Disks

One way to improve the throughput of a mass storage system is to construct it out of multiple disks, each of which can be accessed in parallel. Increasing the number of disks, however, also increases the likelihood that some disk will fail. To compensate for the increased risk of failure, some redundancy is built into the system, hence the name RAID (Redundant Array of Inexpensive Disks).

The figure below shows a data layout for RAID. The data is broken into stripes, numbered 0 through 3, each of which consists of multiple blocks, one for each of the four disks. The fourth block in each stripe holds the parity, which is the sum modulo two of the data in the other blocks.

0	0	0	parity 0
1	1	1	parity 1
2	2	2	parity 2
3	3	3	parity 3
disk 0	disk 1	disk 2	disk 3

Problem 18. (2 points):

Suppose that one block of one stripe is corrupted. Which data must be read in order to recover the block?

Problem 19. (2 points):

Suppose that an entire disk is corrupted. Which data must be read in order to recover the entire disk?

Problem 20. (2 points):

The RAID scheme pictured above has a potential bottleneck. Every write operation must access the same disk. For each stripe in the figure below, shade in one parity block. Your layout should not have the write bottleneck of the previous layout.

disk 0	disk 1	disk 2	disk 3
0	0	0	0
1	1	1	1
2	2	2	2
3	3	3	3

Problem 21. (4 points):

In either of the previous two schemes, recovering an entire disk requires reading a lot of data from the remaining disks. One way to reduce the cost of recovering a disk is to increase the amount of redundancy. In the figure below, there are four disks, but only three blocks per stripe, one of which is used for parity. Show how the stripes are laid out, and shade in one parity block for each stripe. Your layout should distribute the parity evenly across all disks.

disk 0	disk 1	disk 2	disk 3

Problem 22. (4 points):

How does your layout reduce the cost of recovering an entire disk?