Solutions to Test 4

(1a) The cache has a total of $256 \times 1024/16 = 16 \times 1024 = 2^{14}$ blocks. There are four blocks per set, so we have a total of $2^{12}$ sets. The block size is 16 bytes, so we need 4 bits to specify the position in the block. Bits 4 to 15 specify the set number: these correspond to the hex positions 1 to 4, i.e., 0x010. In binary, this translates to 0000 0001 0000.

(1b) It is easy to see that there will be no conflict or capacity misses. Only cold-start misses will occur. a[0] is stored in the first byte of its block, so we will need to access a total of $[100/4] = 25$ distinct blocks. This is the number of misses that will occur.

(2)

Lh:  
    addi $sp, $sp, -8  # Make room on the stack for two words
    sw  $a0, 4($sp)    # Save $a0 on the stack
    sw  $ra, 8($sp)    # Save return address on the stack
    slti $t0, $a0, 1   # Set $t0 if n<1
    beq  $t0, $zero, L1 # Go to L1 if n>=1
    addi $v0, $zero, 1 # Put the result of 1 into $v0
    j    Lexit         # Go to the cleanup (restoration) phase

L1:  
    slti $t0, $a0, 3   # If n<3, set $t0
    beq  $t0, $zero, L2 # If n>=3 go to L2
    addi $v0, $zero, 2 # Prepare to return 2
    j    Lexit         # go to the cleanup (restoration) phase

L2:  
    addi $t1, $a0, -21 # calculate n-21
    slt  $t0, $t1, $zero # $t0=1 if n<21; 0 otherwise
    bne  $t0, $zero, L3  # go to L3 if n<21
    addi $v0, $zero, 20 # prepare to return 20
    j    Lexit         # go to the cleanup (restoration) phase

L3:  
    addi $a0, $a0, -1  # calculate n-1
    jal  Lh            # call h(n-1)
    addi $t2, $v0, 0   # place h(n-1) in $t2
    addi $a0, $a0, 4   # place n-1+4=n+3 in argument register
    jal  Lh            # call h(n+3)
    addi $t3, $a0, -3  # recover n
    sub  $t3, $t3, $t2  # n-h(n-1)
    add  $v0, $v0, $t3  # n-h(n-1)+h(n+3)

Lexit:  
    lw  $a0, 4($sp)    # restore argument register
    lw  $ra  8($sp)    # restore return address register
    addi $sp, $sp, 8   # restore stack pointer
    jr   $ra           # return to calling program

(3)
(a) | Cycles | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I0</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1</td>
<td>IF</td>
<td>ID</td>
<td>ID</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2</td>
<td>IF</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I3</td>
<td>IF</td>
<td>ID</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) | Cycles | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I0</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1</td>
<td>IF</td>
<td>ID</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2</td>
<td>IF</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I3</td>
<td>IF</td>
<td>ID</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) | Cycles | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I0</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1</td>
<td>IF</td>
<td>ID</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(4a) The smallest value of c for which \(6 + c + 1 \leq 2^c\) is 4.

(4b) Many mappings will work: here is one. The bottom line is that each data bit must be covered by at least two code bits; further, the set of code bits that cover each data bit must be unique. We can fill out the table as we showed during the discussion on May 4.

\[
\begin{align*}
c_0 &= d_0 \oplus d_2 \oplus d_3 \\
c_1 &= d_0 \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_5 \\
c_2 &= d_1 \oplus d_2 \oplus d_4 \oplus d_5 \\
c_3 &= d_3 \oplus d_4 \oplus d_5
\end{align*}
\]

(5a) It is easy to see that all the modules will be busy, so we’ll have 8 accesses served per 100 ns (over the long term). So, the throughput is 0.08 words per ns.

(5b) If the memory cycle time is 2 ns, the memory modules are no longer a bottleneck and we can serve requests as fast as they arrive. In such a case, the throughput will be 1 word per ns.

(6) The role of parity disk is spread out over the various disks in RAID 5. RAID 4 sets aside one disk to be the parity disk. Parity sectors often need to be updated with one or more of the data sectors are modified. Hence, in RAID 4, the parity disk will become a
performance bottleneck.
(7) Availability is the fraction of time over which that a computer is up. One application for which it is frequently used as a dependability measure is the telephone network.

(8)

(9a) (iii) 10 Watts to 150 Watts
(9b) (a) True.