Instructions: This is a take home exam. There are four questions as outlined in the study guide. Each question is worth 25 points and some questions have multiple parts. All students must do the exam by themselves. If there are questions, you may send email to the instructor for clarification.

The following program is to be used in questions 1 and 2.

```c
/* Matrix transposition.
   * Inputs:
   *   n : positive integer.
   *   A : n x n matrix, stored in row-major order in contiguous space of size n^2.
   * Side Effect:
   *   The array A is modified so that the modified array is equal to A^t (transpose).
   * Recall that the (i,j) element of A^t is equal to the (j,i) element of A.
*/

void transpose(int *A, int n)
{
    int i, j;
    int t;
    for (i=0; i<n; i++)
        for (j=i+1; j<n; j++) {
            t = *(A + i*n + j);
            *(A + i*n + j) = *(A + j*n + i);
            *(A + j*n + i) = t;
        }
}
```

The following assembly code was obtained using Sun’s C compiler on king (4 processor Sun Enterprise 3000 with UltraSPARC I processors running SunOS 5.9).

```
$ cc -V
cc: Sun WorkShop 6 update 2 C 5.3 2001/05/15
cc -S -xO2 -xarch=v9 transpose.s

! FILE transpose.c

!    1    !void transpose(int *A, int n)
!    2    !{
```
SUBROUTINE transpose
!
OFFSET SOURCE LINE LABEL INSTRUCTION

.global transpose
transpose:
/* 000000 2 */ save %sp,-176,%sp
/* 0x0004 */ sra %i1,0,%l4

! 3 ! int i,j;
! 4 ! int t;
! 5 ! for (i=0;i<n;i++)

/* 0x0008 5 */ cmp %l4,0
/* 0x000c */ bge,a,pt %icc,.L77000010
/* 0x0010 */ nop
.L77000015:
/* 0x0014 5 */ or %g0,0,%l3
/* 0x0018 */ or %g0,0,%l2
! 6 ! for (j=i+1;j<n;j++) {

/* 0x001c 6 */ or %g0,1,%l1
.L900000105:
/* 0x0020 6 */ cmp %l1,%l4
/* 0x0024 */ bge,a,pt %icc,.L900000106
/* 0x0028 9 */ add %l2,%l4,%l2
! 7 ! t = *(A + i*n+j);

/* 0x002c 7 */ smul %l1,%i1,%o7
/* 0x0030 */ sra %l2,0,%o2
/* 0x0034 6 */ or %g0,%l1,%l0
! 8 ! *(A + j*n+i) = *(A + i*n+j);

/* 0x0038 9 */ sra %l3,0,%o1
/* 0x003c 8 */ sllx %o2,2,%o3
/* 0x0040 9 */ sllx %o1,2,%o5
/* 0x0044 8 */ add %i0,%o3,%o4
.L77000014:
/* 0x0048 7 */ sra %l0,0,%g1
/* 0x004c 9 */ add %l0,1,%l0
/* 0x0050 */ sllx %g1,2,%g4
/* 0x0054 9 */ cmp %l0,%l4
/* 0x0058 8 */ sra %o7,0,%g1
/ * 0x005c 7 */   ld  [%o4+%g4],%g5 ! volatile
/ * 0x0060 8 */   sllx %g1,2,%g2
/ * 0x0064  */   add %i0,%g2,%o0
/ * 0x0068  */   ld  [%o5+%o0],%g3 ! volatile
/ * 0x006c  */   st  %g3,[%o4+%g4] ! volatile
/ * 0x0070 9 */   st  %g5,[%o5+%o0] ! volatile
/ * 0x0074  */   bl,pt %icc,.L77000005
/ * 0x0078  */   add %o7,%14,%o7
.L77000007:
/ * 0x007c 9 */   add %12,%14,%12
.L9000000106:
/ * 0x0080 9 */   or  %g0,%11,%13
/ * 0x0084  */   cmp %11,%14
/ * 0x0088  */   bl,pt %icc,.L9000000105
/ * 0x008c 6 */   add %11,1,%11
.L77000010:
/ * 0x0090 9 */   ret  ! Result =
/ * 0x0094  */   restore %g0,%g0,%g0
<table>
<thead>
<tr>
<th>α</th>
<th>α₁</th>
<th>α₂</th>
<th>α₃</th>
<th>β₁</th>
<th>β₂</th>
<th>β₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>12</td>
<td>34</td>
<td>106</td>
<td>18</td>
<td>18</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 1: Instruction constants for the Pentium III using gcc version 2.91.66

1. **Instruction Count – 25 points**

Derive a formula for the number of machine instructions that are executed by the function transpose as a function of the input size \( n \). Use the assembly code provided to derive your result.

2. **Cache Design, Cache Misses, and Algorithm Optimization – 25 points**

Assume that the above transpose function is executed on a machine with a direct mapped cache of size \( C = 2^c \) and block size 1. Assume that each element in the cache holds a single int.

(a) Derive a formula for the number of cache misses under the assumption that only read accesses to the input array \( A \) are considered (you may ignore all other variables and instruction accesses).

(b) How does your formula change with increased block size?

(c) How can the computation be modified to better take advantage of increased block size?

3. **WHT Instruction Count Model – 25 points**

Let \( \mathcal{W}_{2^n} \) be a WHT algorithm, and let \( A(n) \) be the number of times the recursive WHT procedure is called, \( A_i(n) \) the number of times the straight-line code for \( \text{WHT}_{2^n} \) is called, \( L_1(n) \) the number of times the outer loop is executed, \( L_2(n) \) the number of times the middle loop is executed, \( L_3(n) \) the number of times the innermost loop is executed. Then the number of instructions required to execute \( \mathcal{W}_{2^n} \) is equal to

\[
\alpha A(n) + \sum_{i=1}^{8} \alpha_i A_i(n) + \sum_{i=1}^{3} \beta_i L_i(n),
\]

where \( \alpha \) is the number of instructions for the code in the compiled WHT procedure executed outside the loops, \( \alpha_i, i = 1, \ldots, 8 \) is the number of instructions in the compiled straight-line code implementations of small WHT’s of size one through eight, and \( \beta_i, i = 1, 2, 3 \) is the number of instructions executed in the outer-most, middle, and inner-most loops in the compiled WHT procedure.

\[
A(n) = \begin{cases} 
1 + \sum_{i=1}^{t} 2^{n-n_i} A(n_i) & \text{if } n = n_1 + \cdots + n_t \\
0 & \text{if } n \text{ is a leaf.} 
\end{cases} 
\]

\[
A_i(n) = \begin{cases} 
\sum_{i=1}^{t} 2^{n-n_i} A(n_i) & \text{if } n = n_1 + \cdots + n_t \\
1 & \text{if } n \text{ is a leaf.} 
\end{cases} 
\]

\[
= l_i 2^{n-l_i} \text{where } l_i \text{ is the number of leaf nodes with value } l \text{ in the tree corresponding to } \mathcal{W}_{2^n} 
\]

\[
L_1(n) = \begin{cases} 
t + \sum_{i=1}^{t} 2^{n-n_i} L_1(n_i) & \text{if } n = n_1 + \cdots + n_t \\
0 & \text{if } n \text{ is a leaf} 
\end{cases} 
\]

\[
L_2(n) = \begin{cases} 
\sum_{i=1}^{t} (L_2(n_i) + 2^{n_1 + \cdots + n_{i-1}}) & \text{if } n = n_1 + \cdots + n_t \\
0 & \text{if } n \text{ is a leaf} 
\end{cases} 
\]

\[
L_3(n) = \begin{cases} 
\sum_{i=1}^{t} (L_3(n_i) + 2^{n-n_i}) & \text{if } n = n_1 + \cdots + n_t \\
0 & \text{if } n \text{ is a leaf} 
\end{cases} 
\]
(a) Using the constants in Table 1 and the recurrence relations for \( A(n) \), \( L(n) \), and \( A_l(n) \) in Equations 1, 2, 3, 4, and 5 derive a formula for the number of instructions used by the right-recursive algorithm and the iterative algorithm of size \( N = 2^n \). You should be able to derive a closed-form solution to the general recurrence relations involving \( n \) and the constants in Table 1 when you specialize the general recurrences to the right-recursive and iterative algorithms. Recall that the right-recursive algorithm corresponds to the factorization

\[
\text{WHT}_{2^n} = (\text{WHT}_2 \otimes I_{2^{n-1}})(I_2 \otimes \text{WHT}_{2^{n-1}}), \tag{6}
\]

and the iterative algorithm corresponds to the factorization

\[
\text{WHT}_{2^n} = \prod_{i=1}^{n} (I_{2^i-1} \otimes \text{WHT}_2 \otimes I_{2^{n-i}}). \tag{7}
\]

(b) Using the formulas you obtained in the first part of this question, compute the ratio of the number of instructions for the recursive algorithm divided by the iterative algorithm for \( n = 1, 2, \ldots, 10 \). What is the ratio in the limit \( n \to \infty \)?

4. WHT Benchmarking – 25 points

Use queen to measure the computing time and instruction count for the iterative and recursive WHT algorithms of size \( 2^n \) for \( n = 1, 2, \ldots, 18 \). You will use the WHT package (see http://www.ece.cmu.edu/~spiral/wht.html) to do the timings. The WHT package has been installed and the commands you need to use “wht.measure” and “wht.inst” are available in /home/jjohnson/bin. Note that wht.inst has been created for this question and uses CPC to count instructions and cycles.

The WHT package uses a simple grammar to specify different WHT algorithms (see the wht paper from the lecture on the WHT). The iterative algorithm can be specified using

\[
\text{split[small[1],\ldots,small[1]]}
\]

where small[1] is listed \( n \) times for the iterative algorithm of size \( 2^n \). The recursive algorithm can be specified using

\[
\text{recursive} \quad ::= \quad \text{split[small[1],recursive(n-1)]} \\
\text{recursive} \quad ::= \quad \text{small[1]}
\]

For example, when \( n = 3 \), the iterative algorithm is \text{split[small[1],small[1],small[1]]} and the recursive algorithm is \text{split[small[1],split[small[1],small[1]]]}. The instruction and cycle count for these algorithms may be obtained using the commands:

\[
$ /home/jjohnson/bin/wht_inst -w "\text{split[small[1],small[1],small[1]]}" \\
\quad \text{401 instructions in} \quad \text{1203 cycles}
\]

\[
$ /home/jjohnson/bin/wht_inst -w "\text{split[small[1],\text{split[small[1],small[1]]}]"} \\
\quad \text{534 instructions in} \quad \text{1604 cycles}
\]

(a) Produce a table of cycle counts and instruction counts for the iterative and recursive algorithms of size \( n = 1, 2, \ldots, 18 \). You may want to write a simple program to generate the strings for the recursive and iterative algorithms and use a shell script to do the measurements.

(b) Produce a table of ratios of instruction counts and cycle counts for the recursive divided by the iterative algorithms. Compare the ratios to those obtained from the instruction count model in the previous question.

(c) Write a brief description of your observations from the data and the conclusions you can make.