Part 1: The total score of this part is 50. The number in the paratheses indicates its score.

1. (15) For a von Neumann machine, with what kind of hardware mechanism and software, we can perform the following procedures?
   (a) (5) Jump subroutine (JSUB) and Return from subroutine (RET).
   (b) (5) Start an External Vector Interrupt (INT n) and Return from Interrupt (IRET)
   (c) (5) Initialize and End a Direct Memory Access (DMA).
2. (15) Explain the differences between the following terms.
   (a) (3) Processor and Coprocessor.
   (b) (3) Isolated I/O and Memory-Mapped I/O.
   (c) (3) Supervisor mode and User mode.
   (d) (3) Write through and Write once.
   (e) (3) Synchronous and Asynchronous bus timings.
3. (10) Assume we want to implement a 24-hour real-time clock in a computer system (or microprocessor system) by interrupt.
   (a) (5) Using a system block diagram to explain what kind of hardware components are needed in this computer system.
   (b) (5) Using a flow chart to show the program structure of this real-time clock.
4. (10) For a $256K \times 1$ bit dynamic RAM IC chip shown in Figure 1,
   (a) (5) Using a $2\frac{1}{2}D$ organization to show the internal organization of this chip.
   (b) (3) Using a timing diagram to explain the memory cycle time and the memory access time.
   (c) (2) Explain by what kind of testing procedure you can determine whether this chip is good or bad.

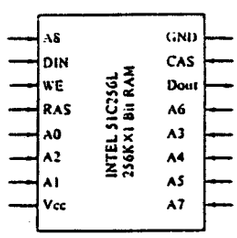

FIGURE 1

Part 2: The total score of this part is 50. The number in the paratheses indicates its score.

1. (10) Recursion is a programming technique in which a program routine (or function) may call itself directly or indirectly.

   (a) (4) Is it true that in general a recursive program is more efficient than its nonrecursive counterpart in terms of CPU time? How about memory usage? You must explain why.

   (b) (6) Write a recursive routine (or function in C) to calculate $n!$ for any given positive integer n. You may use either PASCAL or C.

2. (10) Determine the time complexity of the following programs using the big O notation (such as $O(n)$ for a linear time complexity). Describe how you obtain your results.

   (a) (5)

```
int bsearch(int x, int v[], int n)
{
  int low, high, mid;

  low = 0;
  high = n-1;
  while (low <= high) {
    mid = (low + high) / 2;
    if (x < v[mid])
      high = mid - 1;
    else if (x > v[mid])
      low = mid + 1;
    else
      return mid;
  }
  return -1;
}
```

   (b) (5)

```
tower(int i, int j, int k, int n)
{
  if (n==1) printf("move from %d to %d\n", i, j);
  else {
    tower(i,k,j,n-1);
    printf("move from %d to %d\n", i, j);
    tower(k,j,i,n-1);
  }
}
```

3. (17) Assume you want to insert n distinct numbers $a_1, a_2, \ldots, a_n$ to a binary search tree in the order that a_1 is inserted first, a_2 second, a_3 third, and so on. Initially the tree is empty. It is possible that due to the relations among $a_1, a_2, \ldots, a_n$, the resulting tree contains only one leaf (or terminal), i.e., each node in the tree, except the terminal node, has exactly one son. For example, if $n = 5$, and $a_1 = 5$, $a_2 = 4$, $a_3 = 3.5$, $a_4 = 3$, $a_5 = 2$, then the resulting tree has only one leaf which contains a_5.

   (a) (5) What is the time complexity for constructing a binary search tree with n nodes but only one leaf?

   (b) (5) Give an example of $a_1, a_2, \ldots, a_n$ such that neither $a_1 < a_2 < \ldots < a_n$ nor $a_1 > a_2 > \ldots > a_n$ holds, but the resulting tree still has only one leaf.

   (c) (7) Assume the relation between any two numbers a_i and a_j, $i < j$, is denoted as O_{ij}. For example if $a_3 > a_5$ then $O_{35} =$ ">". Derive the relations among all O_{ij}, $i < j$ such that after $a_1, a_2, \ldots, a_n$ are inserted, the resulting tree contains only one leaf.

4. (13) Dynamic allocation.

   (a) (5) Dynamic allocation is a powerful technique to manage memory in some programming languages such as C and PASCAL. Describe the main advantage of this technique.

   (b) (8) Assume you have 10,000 character strings in an ASCII file. Each string may have a length from 1 to 100,000. Obviously if you used a 2-dimensional array to store all the strings, the size of the array would be $10,000 \times 100,000$, which would be beyond the capacity of most computers. Describe a feasible procedure to read all strings into a program and store them in a dynamically allocated data structure such that each string can be randomly accessed. Since dynamic allocation is quite time consuming, you should try to minimize the number of calls to the dynamic allocation routine. You may make any reasonable assumption.