Practice Exam 2

You will have 1 hour for this exam. Please allow some time to check your work. If you need extra space, write on the back. This exam is open book and note.

1. Convert the following C++ program to Pep/8 assembly language:

```
#include <iostream>
using namespace std;

void check(int & x, int & y)
{
    int temp;
    if (x > y) {
        temp = x;
        x = y;
        y = temp;
    }
}

int main()
{
    int a, b;
    cin >> a >> b;
    check(a, b);
    cout << a << " is no more than " << b << endl;
    return 0;
}
```

2. With a "two-address" architecture, most machine instructions take two addresses as operands. An instruction such as

```
add     X, Y
```

says to add the value stored at address `Y` to the value at address `X`, leaving the result in `X`. That is, it is roughly equivalent to the C++ statement `X += Y;`.

(a) How many memory reads are required to fetch and execute this instruction on a two-address architecture (where `X` and `Y` are direct-mode operands)?

(b) How many memory *writes* are required?

(c) Give an equivalent sequence of instructions for the Pep/8 architecture, and tell how many memory reads and writes are required for it.

3. Give a C++ function which is equivalent to the following Pep/8 code (that is, a C++-to-Pep/8 compiler might produce the following as output):

```
a:      .EQUATE 4
k:      .EQUATE 2
print:  LDX     k,s
        BRLE    L1
        SUBX    1,i
        ASLX
        DECO    a,sxf
        CHARO   '\n',i
        LDA     a,s
        STA     -2,s
        LDA     k,s
        SUBA    1,i
        STA     -4,s
        SUBSP   4,i
        CALL    print
        ADDSP   4,i
L1:     RET0
```

4. Write Pep/8 code for a call to the `print` function in the question above:

```
        BR      main
nums:   .BLOCK  20      ; an array of 10 ints
main:   ...
        ; some code to fill up nums with interesting data (don't write this)
        ...
        ; now call print with arguments a=nums and k=10
```

5. Consider the following recursive function in C++:

```
int M(int a[], int k)
{
    if (k == 1) return a[0];
    int j = k - 1;
    a[j - 1] = a[j - 1] + a[j];
    return M(a, j);
}
```

(a) Sketch the layout of items in a stack frame for this function:

(b) If each `int` occupies 2 bytes, and memory addresses also take 2 bytes, then how many bytes is each stack frame?

(c) Suppose there is a global `int` array called `nums`, which initially contains three items, 1, 2, and 3:

```
int nums[] = {1, 2, 3};
```

Sketch the relevant contents of the stack and global data storage at the point where the `return a[0];` statement is first executed, after `main` calls `M(nums, 3)`: