Due: Monday, July 25th. Written 4pm, Programs 11:59pm.
File names: sort.asm, sorting2.c, sort3.asm (extra credit), mult.mps

Written (30 points)

Written from Chapter 11 (24 points): 1; 2 b, d, f, g; 3 b, e, g, h; 4 b, d; 5, 6, 8, 11, 13.

MIPS Written: (6 points) What are the machine codes (in hexadecimal) for the instructions in the following MIPS code fragment?
loop:  addu $4, $8, $11
      lw  $5, 16($sp)
      beq  $8, 0, loop
      nop
      addi $7, $9, -2
      j  loop

Programs (60 points)

1) (25 points) Name: sort.asm. Write a 32-bit Intel assembly language integer sorting function for the nasm assembler that relies on insertion sort. I chose 32-bit because many students have 32-bit machines at home.

We will test your function by calling it from within the sorting1.c program available in ~ssdavis/50/p4. The function will receive two parameters: 1) The address of an array of int[25]; and 2) The number of items in the array. You should look at the assembly code in sorting1.s, using gcc -m32 -masm=intel -fno-asynchronous-unwind-tables -S sorting1.c to determine the placement and data length of the parameters on the stack. Your function must be register neutral, with extensive comments similar to those in selsort.asm. Looking at selsort.asm, you will notice that the code is quite inefficient, with many registers being unnecessarily re-copied. Your code must be efficient, and use registers instead of local variables wherever possible. With four general registers, as well as DI and SI, there is little need for memory storage. Besides the original parameters, your code should rely on only one local variable on the stack.

To create your object file you will type: nasm -f elf32 sort.asm -o sort.o
To create your executable you will type: gcc -m32 -g -o sort1.out sorting1.c sort.o

Matloff demonstrates sorting using a sort known as selection sort, which is less efficient than insertion sort. Here are the two algorithms:

Selection Sort: For each position, starting from the zeroth position
1. Find the smallest value in the subarray encompassing that position and all positions with larger indices.
2. Swap that smallest value with the entry in the current position.

Insertion Sort: For each position, starting from the [1] position
1. Store the entry in the current position in a temporary variable.
2. Working from the current position towards the zeroth position, shift right one position in the array all entries that are larger than that in the temporary variable.
3. When an entry that is smaller than the temporary value is encountered, then copy the temporary value into the position just to the right of the smaller value. Note that the entry originally in the destination position will have already been copied one position to its right.

Both algorithms rely on the fact that after each iteration, the subarray to the left of the current position is sorted. The selection sort is slower because it must always search through the remaining array for the smallest values, while the insertion sort continues shifting items only as long as they are larger than the current entry. This difference is most apparent when the algorithms are attempting to sort an array of N items that are already sorted. In such a case, selection sort will look through N items for the smallest value, then N-1 items, then N-2 items, and so on, for N(N+1)/2 comparisons. On the other hand, insertion sort will stop comparing after the first comparison of each position, for a total of N comparisons.

Lessons learned:
1. leal means "load effective address long". This operation copies an address to a register so that you can later add displacements to it.

2. Work little by little, by writing some code, compiling, and then running. At the beginning, you will just want the program to print out the words as they were originally ordered. Get your pushes and pops working properly before bothering writing your sorting code. Use the standard push ebp; mov ebp, esp; sub esp, x; to setup the frame pointer and make room for your local variable needing x bytes of room.

2) (10 points) Filename: sorting2.c Imbed your assembly code from #1 in the sort() function of sorting2.c. You will want to read http://www.reversing.be/article.php?story=20051203194931893. Once you have imbedded your code you can use gcc –masm=intel –S sorting2.c to make sure everything looks correct.

3) (10 points, extra credit) Filename: sort3.asm
   Write an Intel assembly language string sorting function, named sort(), for the nasm assembler that relies on insertion sort. The function takes a two dimensional array char[][25], and an int n for parameters, and is called from sort3.c.

4) (25 points) Filename: mult.mps
   Write a MIPS program that multiplies the values in an array of ten unsigned integers by the values contained in the in another array of ten unsigned integers, and print the results to the screen. The two arrays will start at 0xA00 and 0xA30. To print an integer, place it in $a0, set $v0 to 1, and make a syscall. To exit, set $v0 to 10, and make a syscall.
   You may assume that all of the integers are less than 0xFFFF so that the result will fit in one integer. You may not use the MULT, nor MULTU opcodes in your program. You must multiply by shifting and adding as in the example below. Your multiplication routine should be a subroutine called from your main program. You should use the appropriate registers, e.g. parameter passing registers, based on their role in the program. Note that hexadecimal numbers are prefaced with “0x” and not “$” in MIPS.

In C++, you would implement:

```c
unsigned int multiply(unsigned int x, unsigned int y) {
    unsigned int reg = 0;
    while (y != 0) {
        if (y & 1)
            reg += x;
        x <<= 1;
        y >>= 1;
    } // while y not zero
    return reg;
} // multiply()

int main() {
    unsigned int a[10], b[10];
    for(int i = 0; i < 10; i++) {
        unsigned int result = multiply(a[i], b[i]);
        cout << result << endl;
    } // for i
    exit(0);
} // main()
```

You will find sorting1.c, sorting2.c, sorting3.c, Makefile1 for sorting1.out, Makefile2 for sorting2.out, Makefile3 for sorting3.out, a00-40.obj, and many selection sort example files in ~ssdavis/50/p4.