CS4400 *Alex Clemmer, u0458675*

Lecture 4 Problem Set

1 Lecture 6

Please note the following information on your assignment:

1. Which function is equivalent to the IA32 assembly code shown here?

   This IA32 assembly clearly corresponds to `fun2`. We load `x` into `%eax` and `y` into `%edx` in lines 4 and 3, respectively; then, in line 5, we compare these values. On line 6, if `y >= x`, we *do not* alter `%eax` at all, and simply return, which means that we are returning `x`. However, in any other case (*i.e.*, if y ¡ x), we place `%edx` in `%eax`, which means that we are returning `y`.

2. **Problem 3.56** Complete the function:

   The function should look like this:

   ```
   int loop(int x, int n)
   {
       int result = -1;
       int mask;
       for (mask = 1; mask != 0; mask = mask << 1) {
           result ^= x & mask;
       }
   }
   ```

3. **Problem 3.59** Fill in the body of the switch statement with C code that will have the same behavior as the machine code:

```
int switch_prob(int x, int n)
{
    int result = x;

    switch (n) {
        case 50:
        case 52:
            result <<= 2;
            break;
        case 53:
            result >>= 2;
            break;
        case 54:
            result += 2;
        case 55:
            result *= result;
        default:
            result += 10;
    }

    return result;
}
```

2 Lecture 7

1. Is the variable `val` stored on the stack? If so, at what byte offset (relative to `%ebp`) is it stored, and why is it necessary to store it on the stack?

   In at least one critical case it absolutely is. In the line `val2 = silly(n << 1, &val);`, we are calling `silly` and supplying `&val` as an argument, and in order to do that, it must be in memory. We get the address using the instruction `leal -4(%ebp),%eax`, and then push that to stack with `pushl %eax`.

2. Is the variable `val2` stored on the stack? If so, at what byte offset (relative to `%ebp`) is it stored, and why is it necessary to store it on the stack?

   In at least one very important case it is. At `.L3` we set `%eax` to 0 using the classic `xor` trick, and then with `movl %eax,-4(%ebp)`, we move this value to the stack at the place where `val2` is stored (*e.g.*, `-4(%ebp)`). This is actually confirmed at `.L4`, where to implement the expression `*p = val + val2 + n;` we use the instruction `movl -4(%ebp),%edx` to load `val2` and then `addl %eax,%edx` to add it to `val`.

3. What (if anything) is stored at `-24(%ebp)`? If something is stored there, why is it necessary to store it?

At the point in the routine that 24 makes sense as an offset, that's where the registers are stored. In this case we are storing `n`, and it needs to be restored when we return.

4. What (if anything) is stored at `-8(%ebp)`? If something is stored there, why is it necessary to store it?

It's never used. I was told this might be for cache alignment, but that is not my area of expertise.