Homework 2: GCC conventions

Answer the following questions. In all of these questions, you have to write assembly code for `x86` 32-bit architecture.

1. Write the assembly code corresponding to `foo` and `bar` routines.

   ```
   int foo () {
       int a, b, c, d, e, f;
       a = 0, b = 1, c = 2, d = 3;
       e = 5, f = 6;
       bar(b, c, d, f);
       return a + b + c;
   }
   int bar(int a, int b, int c, int d) {
       int x, y, z, w;
       x = a, y = b, z = c, w = d;
       return x + y + z + w;
   }

   Register allocation was done as follows:
   foo: a(%eax), b(%ebx), c(%ecx), d(%edx), e(%esi), f(%edi)
   bar: x(%eax), y(%ebx), z(%ecx), w(%edx)
   ```

   You have the follow the `GCC` calling convention as discussed in the class. [0.5]

2. Write the assembly code corresponding to `foo` and `bar` routines.

   ```
   void foo (int a, int b) {
       bar(&a, &b);
   }
   int bar(int *x, int *y) {
       *x = 0;
       *y = 1;
   }
   ```

   You have to use `lea` instruction to obtain the addresses of `a` and `b`. [0.5]

3. Consider the following code.

```
int bar(int x, int y);

int foo (int a, int b) {
    return bar(a, b);
}
```

   Compile the above code using `gcc -m32 -c foo.c`. Disassemble using `objdump -dx foo.o`. Write the disassembly (with bytecode) corresponding to `foo` routine. Let us assume, at runtime, the address of the first instruction in `foo` is `x`, what will be the value of return address pushed on the stack when `bar` is called from `foo`. [0.5]

4. Consider the following code.

```
int x;
void foo() {
    x = 0;
}
```

   Assuming that the compiler has reserved four-byte space just before the `foo` routine for variable `x`, write an assembly code corresponding to `foo`. If you want to know the length of an instruction, you can write the instruction in a file (say temp.s), and generate the executable `a.out`, by running ``` ``as --32 temp.s'' ```. You can disassemble `a.out` using ``` ``objdump -dx a.out'' ``` and inspect the bytecode to find the length of a given instruction. [0.5]

5. Suppose we want to implement a function that accepts a variable number of arguments. E.g., if `foo` accepts variable number of arguments, then foo(1, 4), foo(1, 0, 6), foo(2, 5, 9, 0), etc., are valid function calls. What are the challenges in generating code for such a function? [0.5]

How to submit

Submit your handwritten homework in the submission box placed at the old academic building (2nd floor). The box will be placed on days when the homework is due.