CAS CS 210 - Computer Systems

Fall 2009

Problem Set #2 (Intel Instructions)

Due: Friday, October 23, 4:00 pm

This problem set is to be completed **individually**. Explain how you got to your answers by providing clear justification, which includes commenting on what each assembly instruction does and showing the contents of the stack.

1. Consider a C function with the following prototype:

```
int foo(int x)
{
    return ______________________;
}
```

The function returns an arithmetic-logic expression in x. No other variables appear in this expression. The function is compiled into IA32 assembly code.

(a) Write C code for `foo` that will have an effect equivalent to the following assembly code:

```
foo:
     pushl %ebp
     movl %esp,%ebp
     movl 8(%ebp),%eax
     sall $4,%eax
     subl 8(%ebp),%eax
     movl %ebp,%esp
     popl %ebp
     ret
```

You can test your solution by compiling your C code with the -S flag. Your compiler may not generate identical assembly code, but it should be functionally equivalent.

(b) Write C code for `foo` that will have an effect equivalent to the following assembly code:

```
foo:
     pushl %ebp
     movl %esp,%ebp
     movl 8(%ebp),%eax
     testl %eax,%eax
     jge .L4
     addl $15,%eax
.L4:
     sarl $4,%eax
     movl %ebp,%esp
     popl %ebp
     ret
```

2. Consider the following C function with prototype:

```
int fun(int *ap, int *bp)
{
    int a;
    ________________;
    ________________;
    return _________;
}
```

Write C code for `fun` that will have an effect equivalent to the following IA32 assembly code:

```
pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%edx
movl 12(%ebp),%eax
movl %ebp,%esp
movl (%edx),%edx
addl %edx,(%eax)
movl %edx,%eax
popl %ebp
ret
```

Your C code can only use one local integer variable a, in addition to the two arguments—do *not* use register names. You can test your solution by compiling your C code with the -S flag. Your compiler may not generate identical assembly code, but it should be functionally equivalent.

3. Consider the following IA2 assembly code for a procedure `foo()`:

```
 foo:
      pushl %ebp
      movl %esp,%ebp
      movl 8(%ebp),%ecx
      movl 16(%ebp),%edx
      movl 12(%ebp),%eax
      decl %eax
      js .L3
 .L7:
      cmpl %edx,(%ecx,%eax,4)
      jne .L3
      decl %eax
      jns .L7
 .L3:
      movl %ebp,%esp
      popl %ebp
      ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. (Note: you may only use symbolic variables *a*, *n*, *val*, and *i* from the source code in your expressions below—do *not* use register names.)

```
int foo(int *a, int n, int val) {
  int i;

  for (i = _________; ____________________________ ; i =___________) {
     ;
  }
  return i;
}
```

4. Consider the source C code below, where `M` and `N` are constants declared with `#define`.

```
int mat1[M][N];
int mat2[N][M];

int copy_element(int i, int j)
{
    mat1[i][j] = mat2[j][i];
}
```

This generates the following IA32 assembly code:

```
copy_element:
        pushl %ebp
        movl %esp,%ebp
        pushl %ebx
        movl 8(%ebp),%ecx
        movl 12(%ebp),%ebx
        movl %ecx,%edx
        leal (%ebx,%ebx,8),%eax
        sall $4,%edx
        sall $2,%eax
        subl %ecx,%edx
        movl mat2(%eax,%ecx,4),%eax
        sall $2,%edx
        movl %eax,mat1(%edx,%ebx,4)
        movl -4(%ebp),%ebx
        movl %ebp,%esp
        popl %ebp
        ret
```

(a) What is the value of `M`?

(b) What is the value of `N`?

5. Consider the following recursive C function:

```
int silly(int n, int *p)
{
    int val, val2;

    if (n > 0)
        val2 = silly(n << 1, &val);
    else
        val = val2 = 0;

    *p = val + val2 + n;

    return val + val2;
}
```

This yields the following IA32 assembly code:

```
silly:
        pushl %ebp
        movl %esp,%ebp
        subl $20,%esp
        pushl %ebx
        movl 8(%ebp),%ebx
        testl %ebx,%ebx
        jle .L3
        addl $-8,%esp
        leal -4(%ebp),%eax
        pushl %eax
        leal (%ebx,%ebx),%eax
        pushl %eax
        call silly
        jmp .L4
        .p2align 4,,7
.L3:
        xorl %eax,%eax
        movl %eax,-4(%ebp)
.L4:
        movl -4(%ebp),%edx
        addl %eax,%edx
        movl 12(%ebp),%eax
        addl %edx,%ebx
        movl %ebx,(%eax)
        movl -24(%ebp),%ebx
        movl %edx,%eax
        movl %ebp,%esp
        popl %ebp
        ret
```

(a) 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?

(b) 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?

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

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