CPSC 313
06W Term 2
Problem Set #3 - Solution

1. (a)
```
int gcd(int a, int b)
{
    if (a == b)
        return a;
    else if (a > b)
        return gcd(a - b, b);
    else
        return gcd(a, b - a);
}
```

```
        .file   "gcdrec.c"
        .text
        .p2align 4,,15
.globl gcd
        .type   gcd, @function
gcd:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %edx
        movl    12(%ebp), %eax
        cmpl    %eax, %edx
        jne     .L9
        jmp     .L3
        .p2align 4,,7
.L11:
        subl    %eax, %edx
        cmpl    %eax, %edx
        je      .L3
.L9:
        cmpl    %eax, %edx
        .p2align 4,,5
        jg      .L11
        subl    %edx, %eax
        cmpl    %eax, %edx
        jne     .L9
.L3:
        popl    %ebp
        .p2align 4,,4
        ret
        .size   gcd, .-gcd
        .ident  "GCC: (GNU) 4.1.0 (SUSE Linux)"
        .section        .note.GNU-stack,"",@progbits
```

```
int gcd(int a, int b)
{
    for (; a != b; a > b ? (a -= b) : (b -= a)) ;
    return a;
}
```

```
        .file   "gcdfor.c"
        .text
        .p2align 4,,15
.globl gcd
        .type   gcd, @function
gcd:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %eax
        movl    12(%ebp), %edx
        cmpl    %eax, %edx
        jne     .L8
        jmp     .L2
        .p2align 4,,7
.L11:
        subl    %edx, %eax
        cmpl    %edx, %eax
        je      .L2
.L8:
        cmpl    %edx, %eax
        .p2align 4,,5
        jg      .L11
        subl    %eax, %edx
        cmpl    %edx, %eax
        jne     .L8
.L2:
        popl    %ebp
        .p2align 4,,4
        ret
        .size   gcd, .-gcd
        .ident  "GCC: (GNU) 4.1.0 (SUSE Linux)"
        .section        .note.GNU-stack,"",@progbits
```

```
int gcd(int a, int b)
{
    while (a != b) {
        if (a > b) {
            a -= b;
        } else {
            b -= a;
        }
    }
    return a;
}
```

```
        .file   "gcdwhile.c"
        .text
        .p2align 4,,15
.globl gcd
        .type   gcd, @function
gcd:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %eax
        movl    12(%ebp), %edx
        cmpl    %eax, %edx
        jne     .L8
        jmp     .L2
        .p2align 4,,7
.L11:
        subl    %edx, %eax
        cmpl    %edx, %eax
        je      .L2
.L8:
        cmpl    %edx, %eax
        .p2align 4,,5
        jg      .L11
        subl    %eax, %edx
        cmpl    %edx, %eax
        jne     .L8
.L2:
        popl    %ebp
        .p2align 4,,4
        ret
        .size   gcd, .-gcd
        .ident  "GCC: (GNU) 4.1.0 (SUSE Linux)"
        .section        .note.GNU-stack,"",@progbits
```

```
int gcd(int a, int b)
{
    do {
        if (a > b) {
            a -= b;
        } else if (b > a) {
            b -= a;
        }
    } while (a != b);
    return a;
}
```

```
        .file   "gcddo.c"
        .text
        .p2align 4,,15
.globl gcd
        .type   gcd, @function
gcd:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %eax
        movl    12(%ebp), %edx
        jmp     .L3
        .p2align 4,,7
.L14:
        subl    %edx, %eax
.L6:
        cmpl    %edx, %eax
        je      .L13
.L3:
        cmpl    %edx, %eax
        jg      .L14
        .p2align 4,,5
        jge     .L6
        subl    %eax, %edx
        cmpl    %edx, %eax
        .p2align 4,,5
        jne     .L3
        .p2align 4,,7
.L13:
        popl    %ebp
        .p2align 4,,6
        ret
        .size   gcd, .-gcd
        .ident  "GCC: (GNU) 4.1.0 (SUSE Linux)"
        .section        .note.GNU-stack,"",@progbits
```

(b) In my solution, the recursive, for and while versions generate essentially identical code. The do version is somewhat different, but still uses the same number of registers, and (within one at least) the same number of branches.

2.
```
        .file   "isFib.c"
        .text
        .p2align 4,,15
.globl isFib
        .type   isFib, @function
isFib:
        pushl   %ebp
        movl    %esp, %ebp
        xorl    %eax, %eax
        movl    8(%ebp), %ecx
        cmpl    $100, %ecx
        ja      .L2
        jmp     *.L4(,%ecx,4)
        .section        .rodata
        .align 4
.L4:
        .long   .L2
        .long   .L3
        .long   .L3
        .long   .L3
        .long   .L2
        .long   .L3
        .long   .L2
        .long   .L2
        .long   .L3
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L3
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L3
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
```

```
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L3
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L3
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
.long   .L2
```

```
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L3
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .long   .L2
        .text
        .p2align 4,,7
.L3:
        movl    $1, %eax
.L2:
        popl    %ebp
        ret
        .size   isFib, .-isFib
        .ident  "GCC: (GNU) 4.1.0 (SUSE Linux)"
        .section        .note.GNU-stack,"",@progbits
```

Test program:

```
#include <stdio.h>

extern int isFib(int n);

struct problem {
        int n;
        int ans;
} problems[] = {
        -23222, 0,
        -1, 0,
        0, 0,
        1, 1,
        2, 1,
        3, 1,
        5, 1,
        8, 1,
        13, 1,
        21, 1,
        34, 1,
        55, 1,
        89, 1,
        100, 0,
        454243, 0,
};

int main(int c, char **v)
{
        struct problem *p,
                *limit = &problems[sizeof(problems) / sizeof(struct problem)];
        for (p = problems; p < limit; ++p) {
                int ans = isFib(p->n);
                if (ans != p->ans) {
                        printf("isFib failed on %d, expected %d, but found %d\n",
                                p->n, p->ans, ans);
                } else {
                        printf("isFib correct on %d\n",
                                p->n);
                }
        }
}
```

3. (a)

```
        .file   "power.c"
        .text
        .p2align 4,,15
.globl power
        .type   power, @function
power:
        pushl   %ebp            # prologue
        movl    %esp, %ebp
        pushl   %ebx            # save callee save %ebx
        subl    $4, %esp        # allocate space for local variable
        movl    8(%ebp), %eax   # np in %eax
        movl    12(%ebp), %ebx  # base in %ebx
        movl    (%eax), %eax    # *np in %eax
        testl   %eax, %eax      # check for 0
        jne     .L2             # if not branch around
        movl    $1, -8(%ebp)    # n = 1
        jmp     .L3             # jump to epilogue
        .p2align 4,,7
.L2:
        subl    $1, %eax        # compute *np - 1
        movl    %eax, -8(%ebp)  # n = *np - 1
        pushl   %ebx            # push second arg (base) first
        leal    -8(%ebp), %eax  # compute &n
        pushl   %eax            # push first arg (&n) last
        call    power           # make the call
        addl    $8, %esp        # pop the arguments
        imull   %ebx, %eax      # multiply by base
        movl    %eax, -8(%ebp)  # set n
.L3:    movl    -8(%ebp), %eax  # return n
        addl    $4, %esp        # deallocate space for n
        popl    %ebx            # restore callee save reg %ebx
        popl    %ebp            # epilogue
        ret
        .size   power, .-power
```

(b) There are two options here. One is to keep track of the stack growth and adjust the offsets to find the arguments to power and its local variable (n) each time the size of the stack changes. The other is to allocate all the space necessary all at once and then the offsets of everything are fixed just like they were from the frame pointer, but they are bigger since esp points to the other end of the stack frame.

The first one:

```
        .file   "power.c"
        .text
        .p2align 4,,15
.globl power
        .type   power, @function
power:
                                # base offset of 1st argument is 4
        pushl   %ebx            # -4
        subl    $4, %esp        # -4
        movl    12(%esp), %eax
        movl    16(%esp), %ebx
        movl    (%eax), %eax
        testl   %eax, %eax
        jne     .L2
        movl    $1, (%esp)
        jmp     .L3
        .p2align 4,,7
.L2:
        subl    $1, %eax
        movl    %eax, (%esp)
        pushl   %ebx            # -4
        leal    4(%esp), %eax
        pushl   %eax            # -4
        call    power
        addl    $8, %esp        # +8
        imull   %ebx, %eax
        movl    %eax, (%esp)
.L3:    movl    (%esp), %eax
        addl    $4, %esp        # +4
        popl    %ebx            # +4
        ret
        .size   power, .-power
```

The second one:

```
        .file   "power.c"
        .text
        .p2align 4,,15
.globl power
        .type   power, @function
power:
        pushl   %ebx
        subl    $12, %esp               # 4 for n, 8 for args
                                        # esp never changes again
                                        # n is at 8(%esp)
                                        # saved ebx is at 12(%esp)
                                        # return addr is at 16(%esp)
                                        # my arg1 is at 20(%esp)
                                        # my arg2 is at 24(%esp)
                                        # when I call myself, arg1 is at 4(%esp)
                                        # when I call myself, arg2 is at (%esp)
        movl    20(%esp), %eax
        movl    24(%esp), %ebx
        movl    (%eax), %eax
        testl   %eax, %eax
        jne     .L2
        movl    $1, 8(%esp)
        jmp     .L3
        .p2align 4,,7
.L2:
        subl    $1, %eax
        movl    %eax, 8(%esp)
        movl    %ebx, 4(%esp)
        leal    8(%esp), %eax
        movl    %eax, (%esp)
        call    power
        imull   %ebx, %eax
        movl    %eax, 8(%esp)
.L3:    movl    8(%esp), %eax
        addl    $12, %esp
        popl    %ebx
        ret
        .size   power, .-power
```

4. (a) i. If A puts x in a caller-save register then A needs to save it each time A calls B, so 10 times.

   ii. If A puts x in a callee-save register, then A saves it only on entry, and since B never uses that register it never saves it, so 1 time.

   (b) i. If C puts x in a caller-save register and D puts y in (the same) caller-save register, then C needs to save/restore it each time C calls D (10 times) and for each time D is called, it needs to save/restore it when D calls E, (10 x 20 times), or 210 times.

   ii. If C puts x in a caller-save register and D puts y in a callee-save register, then C needs to save/restore it each time C calls D (10 times) and for each time D is called, it needs to save/restore it once, (10 x 1 times), or 20 times.

   iii. If C puts x in a callee-save register and D puts y in a caller-save register, then C needs to save/restore it once and for each time D is called, it needs to save/restore it when D calls E, (10 x 20 times), or 201 times.

   iv. If C puts x in a callee-save register and D puts y in (the same) callee-save register, then C needs to save/restore it once and for each time D is called, it needs to save/restore it once (10 x 1 times), or 11 times.