Due: Monday, Feb 8, 2010. 11:59pm (no extensions).

What to submit: For parts 1, 2, and 4: an ASCII file with answers, submitted under id ‘ex3’. For part 3: a C file with your code, submitted under id ‘ex3part3’. Use the submit.pl script in cs3214/bin/submit.pl to submit from the command line, or use the submit website.

This exercise is intended to reinforce the content of lectures 3-5. As before, you’re asked to use gcc 4.1.2 for an IA32 target. Specify the –O switch unless the question asks otherwise.

1. Switch Statements. Lecture 3/4 points out that gcc chooses a strategy for emitting switch statements based on the values that occur in the case arms. For a small number of cases, it’ll generate if-else chains. For a larger number of cases, it’ll switch to using a jump table.

   (a) Assume the case values are 0, 1, 2, ... N for a chosen N. How large does N need to be for gcc to emit a jump table?

   (b) Let’s assume you’ve chosen N to be large enough such that 0, 1, ... N uses a jump table. Now assume the case values are M, M+1, M+2, ... M+N for a given M. How does gcc’s strategy change? Does it depend on the value of M?

   (c) Now assume the case values are M, M+4, M+2*4, M+3*4, ... M+N*4. What does gcc generate in this case?

2. Variations on Passing Arguments. Consider the following variation of the call_swap() function shown in Lecture.

   ```c
   void swap(int *xp, int *yp)
   {
       int t0 = *xp;
       int t1 = *yp;
       *yp = t0;
       *xp = t1;
   }
   
   int zip1 = 15213;
   int zip2 = 91125;
   
   void call_swap(void)
   {
       swap(&zip1, &zip2);
       swap(&zip1, &zip2);
   }
   ```

   Compile this code with the following flags (in this order):

   A gcc -S -O -mno-accumulate-outgoing-args swap.c
   B gcc -S -O -fno-defer-pop -mno-accumulate-outgoing-args swap.c
   C gcc -S -O swap.c
D gcc -S -O -mregparm=2 swap.c

For each variant, determine (count) how much stack space is allocated for the frame on procedure entry, how much stack space is used right before the first call, and how much stack space is used right before the second call to swap

<table>
<thead>
<tr>
<th>on procedure entry</th>
<th>before 1st</th>
<th>before 2nd</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Consider the difference $ebp - esp$ when determining the frame size.

For variant D (which is an ABI-breaking variant that passes arguments in registers), find out which registers hold the first and second argument, respectively.

3. **Variable Arguments.** When programming in C, it is important to understand how functions such as `printf()` work, which take a variable number of arguments. Read the man pages for `stdarg(3)` to learn how these work, then implement a simple function `sumargsuntilzero` that sum all nonzero arguments passed to it, up to not including a final zero argument. Implement your function portably. With your implementation, the following program should output ‘CS3214’:

```c
#include <stdio.h>
#include <stdarg.h>

/* Sum arguments passed to this function
 * All arguments except for the last argument must be non-zero
 */
int sumargsuntilzero(int first, ...)
{
    // your implementation goes here
}

int main(int ac, char *av[])
{
    printf("CS%d%d%d%d\n",
            sumargsuntilzero(1, 2, 0),
            sumargsuntilzero(1, 1, 0),
            sumargsuntilzero(2, -1, 0),
            sumargsuntilzero(5, -1, -1, 1, 0)
    );
}
```
4. **Nested vs. Multi-Arrays.** Consider the following example code:

```c
#include <stdio.h>

int nestedarray[4][5] = {
    { 1, 5, 2, 0, 6},
    { 1, 5, 2, 1, 3},
    { 1, 5, 2, 1, 7},
    { 1, 5, 2, 2, 1},
};

int * multiarray[4] = {
    nestedarray[0],
    nestedarray[1],
    nestedarray[2],
    nestedarray[3]
};

int get_nestedarray_elem(int nestedarray[4][5], int row, int col)
{
    return nestedarray[row][col];
}

int main(int ac, char *av[])
{
    printf("%d\n", get_nestedarray_elem(multiarray, 0, 3));
}
```

The compiler reports the following error message when compiling the program:

array.c: In function 'main':
array.c:24: warning: passing argument 1 of 'get_nestedarray_elem' from incompatible pointer type

Ignoring this warning and running the program, we obtain the following output:

```
$ ./array
134518300
```

The value 134518300 appears to be an address. What is it the address of? Give an equivalent C expression & (....).