# CSCI 2021 Machine Architecture and Organization Fall 2017 Homework Assignment # 2 Instructions:

## This homework must be done individually.

* Posted Tuesday, October 10th and **due on moodle at 11:55 pm on Tuesday, October 24**

## Please submit as a PDF containing solutions

* There are five problems; we will go over them in discussion section after the due date.
* The textbook in this context is: R. Bryant, D. O'Hallaron, Computer Systems: A Programmer's Perspective. (3rd Edition)

**Problem 1 (While Loop) (20 points)**

Consider the following assembly code for a function with a while loop:

Prob1:

jmp .L2

.L5:

testb $1, %dil #%dil is the lower 8 bits of %rdi je .L3

leal 7(%rdi,%rdi,8), %eax addl $6, %esi

shll %edi jmp .L2

.L3:

.L2:

leal 1(,%rsi, 2), %eax addl $3, %esi

shll $2, %edi

cmpl %edi, %esi jb .L5

ret

## Based on the assembly code above, fill in the blanks below in its corresponding C source code. You may only use the source-level C variable names such as n, m and result. Don’t use register names!

unsigned prob2(unsigned n, unsigned m){ unsigned result;

while (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) { if (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ) {

result = \_\_\_\_\_\_\_\_\_\_\_\_\_;

m = \_\_\_\_\_\_\_\_\_\_\_\_\_; n = \_\_\_\_\_\_\_\_\_\_\_\_\_;

}

else{

result =

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

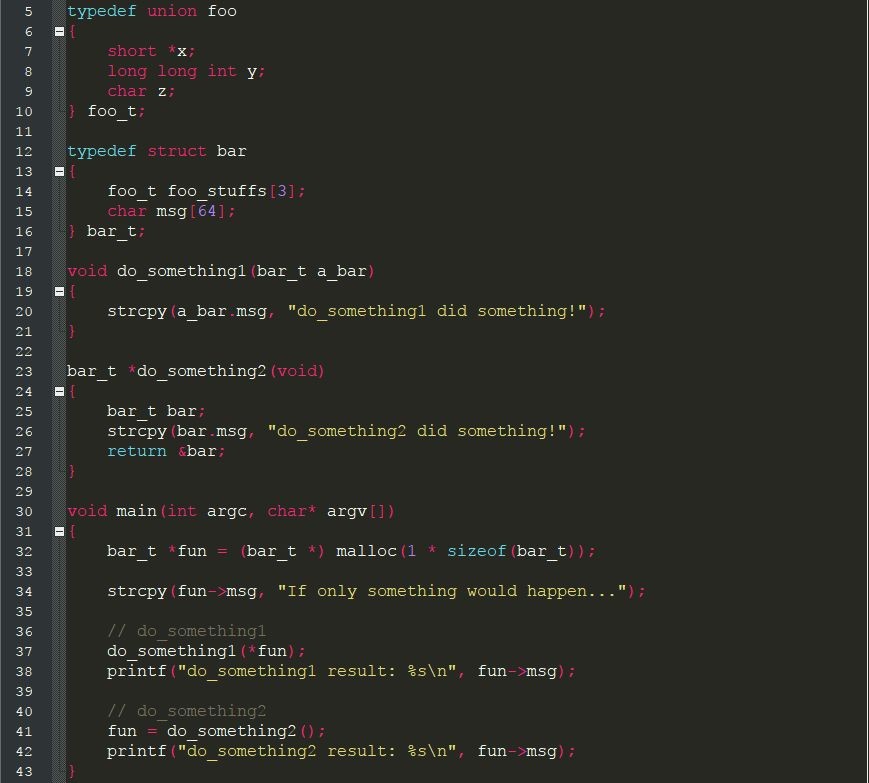
m = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_; n = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

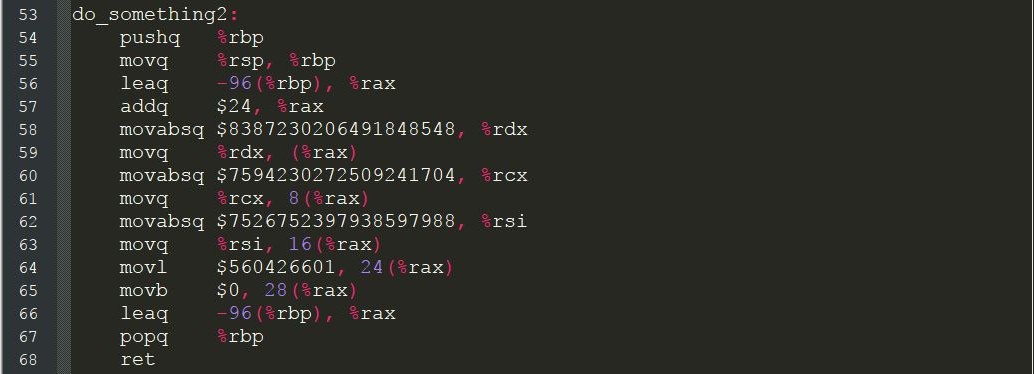
return result;

}

# Problem 2 (Structs and Unions) (20 points)

Matt wanted to flex his C programming skills by playing with structs and unions. He devised the following code:

1. What is the output of the two print statements in lines 38 and 42? Explain this result.
2. In Part A, if the expected result differed from the actual result, how might you change the existing code to work as intended?
3. Show the partitioning of the bar\_t struct (i.e., show the number of bytes dedicated to each field).
4. How many bytes does the bar\_t struct require in total?
5. Using the following assembly code compiled by GCC, draw the stack frame for do\_something2. You must show each member field for any composite structures.



# Problem 3 (Jump Table) (20 points)

## For a C function hw2\_switch with a general structure shown later, gcc generates assembly code including a jump table, shown as follows.

hw2\_switch:

cmpq $7, %rdx ja .L2

jmp \*.L4(,%rdx,8)

.L4:

.L3:

.L5:

.quad .L2

.quad .L3

.quad .L5

.quad .L2

.quad .L6

.quad .L2

.quad .L7

.quad .L5

addq %rsi, %rsi addq (%rdi), %rsi jmp .L8

leaq (%rsi,%rsi,2), %rax leaq (%rax,%rax), %rsi salq $7, %rsi

.L6:

.L7:

.L2:

.L8:

jmp .L8

leaq 20(%rsi), %rax movq %rax, (%rdi)

leaq (%rsi,%rsi,2), %rax leaq (%rax,%rax), %rsi addq (%rdi), %rsi

jmp .L8

addq $7, %rsi

movq %rsi, (%rdi) ret

## Based on the assembly code above, fill in the blanks below in its corresponding C source code. You may only use the source-level C variables x, m, result, and value: don’t use register names!

void hw2\_switch (long \*value, long x, long m) { long result;

switch (m) {

case \_\_\_\_\_\_\_: /\* Case A \*/

result = break;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

case \_\_\_\_\_\_\_: /\* Case B \*/

case \_\_\_\_\_\_\_: /\* Case C \*/

result = break;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

case \_\_\_\_\_\_\_: /\* Case D \*/

\*value = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

case

/\* fall through \*/

\_\_\_\_\_\_\_: /\* Case E \*/

result = break;

default:

result =

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

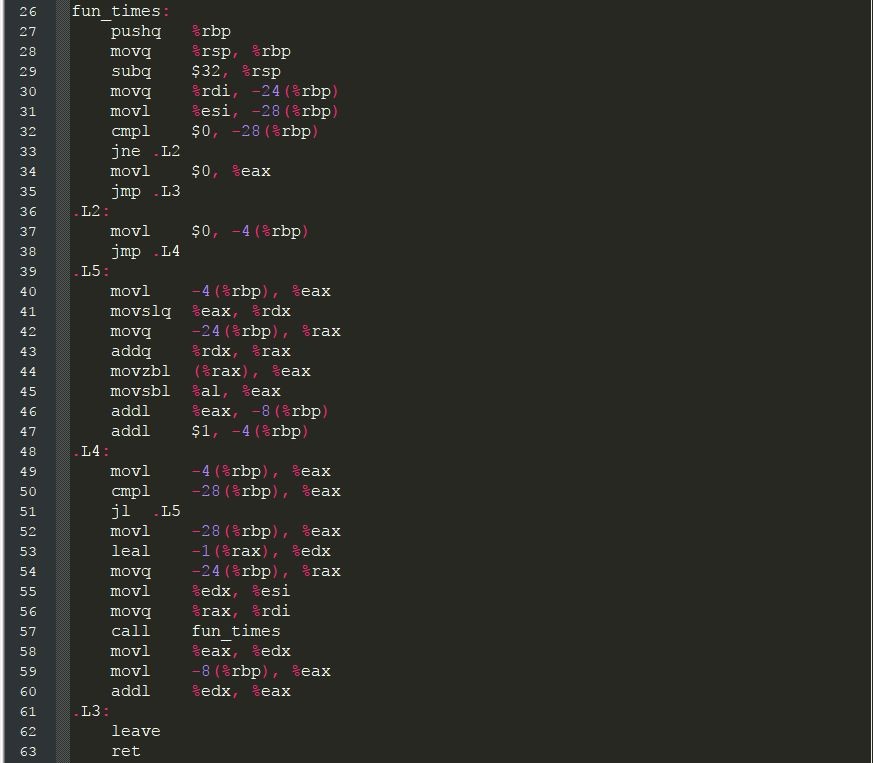
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

\*value = result;

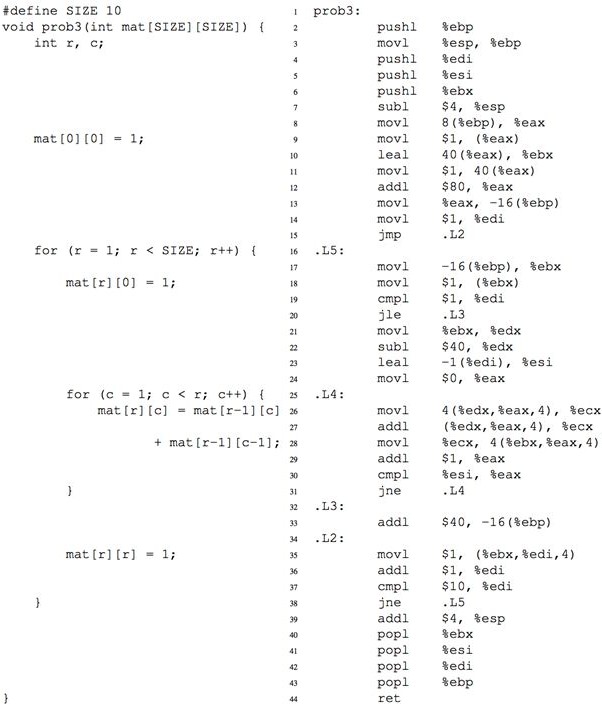
}

# Problem 4 (Recursion) (20 points)

## Amelia really likes recursive functions and has written a function fun\_times that produces the following assembly code when compiled using GCC:



1. What does fun\_times do? Demonstrate the logic either in C code or in pseudocode.
2. How many bytes is allocated on the stack with each call to fun\_times?
3. Where is the result (i.e., return value) stored?

**Problem 5 (For Loop and Array) (20 points)**

Because the compiler has optimized some of the accesses to the array, the registers don’t all correspond exactly to variables in the source code. (And the statements and instructions don’t line up exactly one-toone either, so don’t put too much significance in the way we’ve spaced the lines.) For each of the following registers, as it is used in a particular range of instructions

(shown by their assembly code line number), write a C expression that corresponds to the value in the register. Your expressions should be written using the C variables mat, r, and c, together with C operators and constants; don’t use register names.

|  |  |
| --- | --- |
| Register | C expression |
| %eax, lines 8-11 |  |
| %edi, lines 14-37 |  |
| %ebx, lines 10-35 |  |
| %edx, lines 22-27 |  |
| %esi, lines 23-30 |  |
| %eax, lines 23-30 |  |