**CSE 434S: Assembly Review**

**Overview**

The purpose of this lab is to practice advanced static analysis and specifically, to practice 32-bit X86 assembly. Work on the following questions and write your answers below.

**Question 1:**

\_main:

push 3

push 2

call \_add\_a\_and\_b

add %esp, 8

ret

\_add\_a\_and\_b:

push %ebx

mov %eax, [%esp+8]

mov %ebx, [%esp+12]

add %eax, %ebx

pop %ebx

ret

Write C code that is equivalent to the assembly above (hint: include 2 functions, “main” and “add\_a\_and\_b”).

--------------------------------------------------------------------------------------------

int main() {

add\_a\_and\_b(2,3);

}

int add\_a\_and\_b (int x, int y) {

return x + y;

}

--------------------------------------------------------------------------------------------

**Question 2:**

\_a$ = 8

\_f PROC

mov ecx, DWORD PTR \_a$[esp-4] ; NOTE: \_a$[esp-4] = [\_a$ + esp – 4]

lea eax, DWORD PTR [ecx\*8]

sub eax, ecx

ret 0

\_f ENDP

The code above was optimized with msvc 2010. Write a function in C that is equivalent to this code. Hint: note that the usual function prologue/epilogue that saves/restores “ebp” is not present.

--------------------------------------------------------------------------------------------

Proc(int a) {

return (a\*8) - a

}

--------------------------------------------------------------------------------------------

**Question 3:**

\_TEXT SEGMENT

\_a$ = 8

\_b$ = 12

\_c$ = 16

\_f PROC

push ebp

mov ebp, esp

mov eax, DWORD PTR \_a$[ebp] ; NOTE: \_a$[ebp] = [ebp + \_a$]

imul eax, DWORD PTR \_b$[ebp]

add eax, DWORD PTR \_c$[ebp]

pop ebp

ret 0 ; NOTE: doesn’t really return 0! Return value still in eax.

\_f ENDP

\_main PROC

push ebp

mov ebp, esp

push 3

push 2

push 1

call \_f

add esp, 12

push eax

push OFFSET $SG2463 ; '%d', 0aH, 00H

call \_printf

add esp, 8

xor eax, eax

pop ebp

ret 0

\_main ENDP

The code above was optimized with msvc 2010. Write a function in C that is equivalent to the assembly above.

--------------------------------------------------------------------------------------------

f(int a, int b, int c) {

return a \* b + c;

}

main() {

printf(“%d”, f(1,2,3));

}

--------------------------------------------------------------------------------------------

**Question 4:**

Below is a c program followed by its assembly representation. Can you add

your notes next to each ‘;’ to indicate what each line does?

#include <stdlib.h>

#include <stdio.h>

int check(){

int x = 2;

int y = 1;

int z = x + y;

return z;

}

int main(){

int x = 0;

if (x == 12){

check();

}

else {

return 0;

}

}

\_check:

00001f40 push ebp

00001f41 mov ebp, esp

00001f43 sub esp, 0xc

00001f46 mov dword [ss:ebp+var\_4], 0x2 ; x = 2

00001f4d mov dword [ss:ebp+var\_8], 0x1 ; y = 1

00001f54 mov eax, dword [ss:ebp+var\_4] ; eax = x

00001f57 add eax, dword [ss:ebp+var\_8] ; eax = x + y

00001f5a mov dword [ss:ebp+var\_C], eax ; ebp + 12 = eax

00001f5d mov eax, dword [ss:ebp+var\_C] ; eax = ebp + 12

00001f60 add esp, 0xc

00001f63 pop ebp

00001f64 ret

\_main:

00001f70 push ebp

00001f71 mov ebp, esp ; ebp = esp

00001f73 sub esp, 0x18 ; give room for local variables

00001f76 mov dword [ss:ebp+var\_4], 0x0 ; ebp + 4 = 0

00001f7d mov dword [ss:ebp+var\_8], 0x0 ; ebp + 8 = 0

00001f84 cmp dword [ss:ebp+var\_8], 0xc ; if x == 12

00001f88 jne 0x1f9b ; not equal, goto 1f9b

00001f8e call \_check ; jump to check function

00001f93 mov dword [ss:ebp+var\_C], eax

00001f96 jmp 0x1fa2

00001f9b mov dword [ss:ebp+var\_4], 0x0 ; ebp + 4 = 0

00001fa2 mov eax, dword [ss:ebp+var\_4] ; eax = ebp + 4

00001fa5 add esp, 0x18

00001fa8 pop ebp ; move ebp

00001fa9 ret ; return

**Question 5:**

start:

PUSH EBP

MOV EBP, ESP

MOV EDI, [EBP+arg\_0]

XOR EAX, EAX

MOV ECX, 0xFFFFFFFF

REPNE SCASB

NEG ECX

MOV EAX, ECX

MOV ESI, [EBP+arg\_0]

MOV EDI, [EBP+arg\_4]

REP MOVSB

MOV ESP, EBP

POP EBP

RETN

5.1) In a few sentences, explain what this function does.

--------------------------------------------------------------------------------------------

The function receives two arguments. First, get the length of the first parameter. And move bytes from source variable to destination variable.

--------------------------------------------------------------------------------------------

5.2) Write a function in C that is equivalent to the assembly above.

--------------------------------------------------------------------------------------------

start(char \*text1, char \*text2)

{

/\* Copy text1 to text2 character by character \*/

Int i=0;

while(text1[i] != '\0')

{

i ++;

}

Int j = 0;

While(j < i ) {

text2[j] = text1[j];

j++;

}

}

--------------------------------------------------------------------------------------------

5.4) Let arg\_0 be a pointer to the null-terminated string “C:\Windows\System32\” and let arg\_4 be a pointer to an empty buffer.

What is the value of the buffer pointed to by arg\_4 when the function completes? What value does the function return?

--------------------------------------------------------------------------------------------

The buffer has the same string as arg\_0.

--------------------------------------------------------------------------------------------