

Assignment#7 Part 2: x86 assembly

CS232 Spring 2021

Due: Monday, April 26 at 11:59:59am

Notes

- Please choose “File”-> “Make a copy” to create a copy of this google document under your google account, and fill in your answers in your own copy, because you do not have permission to edit this document in place.
 - All the answer text areas are already set in blue. Please try to keep the blue setting for your answer text. Thanks for your collaboration in helping me with streaming grading.
 - Once you are ready to submit your homework, choose “File” -> “Download” -> “PDF Document” to save your homework locally as a pdf file.
 - In exams you have no access to compiler explorer, so you are recommended to solve the problems without it. You can use the compiler to verify your solution afterwards but please try not to rely on it while doing your homework.
-

1. **[8 points]** Write C code for `func()` based on the following assembly code that was generated.

func:

```
    pushl    %ebp
    movl    %esp, %ebp
    subl    $16, %esp
    movl    8(%ebp), %eax
    cmpl    12(%ebp), %eax
    jle     .L2
    movl    8(%ebp), %eax
    movl    %eax, -4(%ebp)
    jmp     .L3
```

.L2:

```
    movl    12(%ebp), %eax
    movl    %eax, -4(%ebp)
```

.L3:

```
    movl    -4(%ebp), %eax
    leave
    ret
```

main:

```
    pushl    %ebp
    movl    %esp, %ebp
    subl    $16, %esp
```

```

    pushl    $20
    pushl    $10
    call     func
    addl     $8, %esp
    movl     %eax, -4(%ebp)
    movl     $0, %eax
    leave
    ret
int func(int x, int y) {
//write your code here
int a;
if (x > y) {
a = x;
} else {
a = y;
}
return a;
}

int main(){
    int rc = func(10, 20);
    return 0;
}

```

2. [8 points] A function with prototype

```
int decode2(int x, int y, int z);
```

is compiled into 32bit x86 assembly code. The body of the code is as follows:

NOTE: x at %ebp+8, y at %ebp+12, z at %ebp+16

```

movl 12(%ebp), %edx
subl 16(%ebp), %edx
movl %edx, %eax
sall $31, %eax
sarl $31, %eax
imull 8(%ebp), %edx
xorl %edx, %eax

```

Parameters x, y, and z are stored at memory locations with offsets 8, 12, and 16 relative to the address in register %ebp. The code stores the return value in register %eax. The shl or sal instruction is used to shift the bits of the operand destination to the left, by the number of bits specified in the count operand

Write C code for `decode2` that will have an effect equivalent to our assembly Code.

```
int decode2(int x, int y, int z) {
//write your code here:
int t1 = y - z;
int t2 = x * t1;
int t3 = (t1 << 31) << 31;
int t4 = t3 ^ t2;
}
```

3. Consider the following assembly code for a C **for** loop: [10 points]

```
loop:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%ecx
    movl 12(%ebp),%edx
    xorl %eax,%eax
    cmpl %edx,%ecx
    jle .L4
.L6:
    decl %ecx
    incl %edx
    incl %eax
    cmpl %edx,%ecx
    jg .L6
.L4:
    incl %eax
    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 the symbolic variables **x**, **y**, and **result** in your expressions below --- **do NOT use register names**.)

```
int loop(int x, int y)
{
    int result;

    for ( _result=0 ; x>y ; result++ ) {

        x-- ;
    }
```

```

    __y++_____;
}

__result++_____;

return result;
}

```

4. **[8 points]** Match the following C functions (C1, C2, C3 and C4) with their corresponding assembly functions (A1, A2, A3, and A4). Write your answers in the spaces provided at the end of the questions.

C program	Assembly Program
<p>C1</p> <pre> int func(int x, int y) { int result = x && y; return result; } </pre>	<p>A1</p> <pre> func: pushl %ebp movl %esp, %ebp subl \$16, %esp movl 8(%ebp), %eax orl 12(%ebp), %eax movl %eax, -4(%ebp) movl -4(%ebp), %eax leave ret </pre>
<p>C2</p> <pre> int func(int x, int y) { int result = x y; return result; } </pre>	<p>A2</p> <pre> func: pushl %ebp movl %esp, %ebp subl \$16, %esp cmpl \$0, 8(%ebp) je .L2 cmpl \$0, 12(%ebp) je .L2 movl \$1, %eax jmp .L3 .L2: movl \$0, %eax .L3: movl %eax, -4(%ebp) movl -4(%ebp), %eax leave </pre>

	ret
<p style="text-align: center;">C3</p> <pre>int func(int x, int y) { int result = x & y; return result; }</pre>	<p style="text-align: center;">A3</p> <pre>func: pushl %ebp movl %esp, %ebp subl \$16, %esp movl 8(%ebp), %eax andl 12(%ebp), %eax movl %eax, -4(%ebp) movl -4(%ebp), %eax leave ret</pre>
<p style="text-align: center;">C4</p> <pre>int func(int x, int y) { int result = x y; return result; }</pre>	<p style="text-align: center;">A4</p> <pre>func: pushl %ebp movl %esp, %ebp subl \$16, %esp cmpl \$0, 8(%ebp) jne .L2 cmpl \$0, 12(%ebp) je .L3 .L2: movl \$1, %eax jmp .L4 .L3: movl \$0, %eax .L4: movl %eax, -4(%ebp) movl -4(%ebp), %eax leave ret</pre>

Write your answers below: (If C1 matches with A4, write A4 in the space next to C1)

C1 - A2

C2 - A4

C3 - A3

C4 - A1

5. [16 points] Consider the following recursive factorial function in C and Assembly language.

```
int rfact(int n)
{
    int result;
    if (n <= 1)
        result = 1;
    else
        result = n * rfact(n-1);
    return result;
}
```

Line# Assembly Code

```
1.  rfact:
2.      pushl %ebp
3.      movl %esp, %ebp
4.      pushl %ebx
5.      subl $4, %esp
6.      movl 8(%ebp), %ebx
7.      movl $1, %eax
8.      cmpl $1, %ebx
9.      jle .L53
10.     leal -1(%ebx), %eax
11.     movl %eax, (%esp)
12.     call rfact
13.     imull %ebx, %eax
14.     .L53:
15.     addl $4, %esp
16.     popl %ebx
17.     popl %ebp
18.     ret
```

Questions:

1. Why do we push the %ebx register's value on the stack frame of rfact?

(Refer: Line# 4 in assembly code - pushl %ebx)

Your answer: To save a copy of the caller's value of %ebx since %ebx is a callee-saved register.

2. What is the purpose of the following 2 statements?
- subl \$4, %esp (Line number 5)

Your answer:

Allocate 4 bytes of memory on the stack to save the argument n for the following recursive call of the function rfact.

b. `addl $4, %esp` (Line number 15)

Your answer:

Deallocate 4 bytes of memory on the stack that are used to store argument n in caller.

3. For every invocation of the function rfact() which register is used to store the value of its input argument?

Your answer:

`%ebx`

4. What is the purpose of the following line of assembly code (Line number 10)?

```
leal -1(%ebx), %eax
```

Your answer: To calculate the value of n (n1) for the next recursive call.

5. Why are the following 2 lines (Line numbers 2 - 3) needed in rfact() function?

```
pushl %ebp
```

```
movl %esp, %ebp
```

Your answer: These 2 lines are stack setup code found in almost all of the functions. Sets the frame pointer to the beginning of the active callee's stack frame.