Assignment 3

Student: Edward Lin

1. **(20 Points) Assembly language understanding (1):**

   The assembly language for `bar()` is:

```
(gdb) disass bar
Dump of assembler code for function bar:
   0x00000000004004cd <+0>:     push   %rbp
   0x00000000004004ce <+1>:     mov    %rsp,%rbp
   0x00000000004004d1 <+4>:     mov    %edi,-0x14(%rbp)
   0x00000000004004d4 <+7>:     mov    -0x14(%rbp),%eax
   0x00000000004004d7 <+10>:    add    %eax,%eax
   0x00000000004004d9 <+12>:    mov    %eax,-0x4(%rbp)
   0x00000000004004dc <+15>:    mov    -0x4(%rbp),%eax
   0x00000000004004df <+18>:    pop    %rbp
   0x00000000004004e0 <+19>:    retq
End of assembler dump.
```

   Give a 1-2 sentence description of the purpose of each instruction.
   I am more interested in the *why* than the *what*.

Instruction:	Purpose:
push % rbp	Store the previous rbp value into memory so that we can use rbp to point other address and we won't forget the previous rbp value.
mov %rsp,%rbp	Set pointer to the bottom of stack, allowing rbp to establish new stack frame.
mov %edi,-0x14(%rbp)	Store the value what function get into the address where rbp minus 14 bytes.
mov -0x14(%rbp),%eax	Store the value of the address where rbp minus 14 bytes into eax, general purpose register, so that we can calculate it.
add %eax,%eax	Add eax itself, like eax += eax;
mov %eax,-0x4(%rbp)	Store the value of eax register into -0x4(%rbp); Compiler does it anyway, I guess it's the result of no optimizing?
mov -0x4(%rbp),%eax	Store the value of -0x4(%rbp) into eax register; Compiler does it anyway, I guess it's the result of no optimizing?
pop %rbp	Restore rbp's previous content and put it into specified register.
retq	Return the result.

2. (10 Points) Assembly language understanding (2):

Write a C function that does what `bar()` does.
You won't be able to figure out the names of my parameters var(s) and local var(s); just make up your own name(s).

My answer:

```
int bar(int v){

v+=v;

return v;}
```

3. (20 Points) Activation Records (1):

Stop the program at its *second* call to `bar()`. When I did so I got the following:

```
(gdb) break bar
Breakpoint 1 at 0x4004d1
(gdb) run
Starting program:
/home/instructor/Documents/Academic/DePaul/Classes/CSC373/20178-
4SumI/Assign3/toAnalyze

Breakpoint 1, 0x00000000004004d1 in bar ()
Missing separate debuginfos, use: debuginfo-install glibc-2.17-
222.el7.x86_64
(gdb) c
Continuing.

Breakpoint 1, 0x00000000004004d1 in bar ()
(gdb) stepi
0x00000000004004d4 in bar ()
(gdb) stepi
0x00000000004004d7 in bar ()
(gdb) stepi
0x00000000004004d9 in bar ()
(gdb) stepi
0x00000000004004dc in bar ()
(gdb) stepi
0x00000000004004df in bar ()
(gdb) info reg
rax            0x4       4
rbx            0x0       0
rcx            0x400560  4195680
rdx            0x2       2
rsi            0x4       4
rdi            0x2       2
rbp            0x7fffffffdbd0   0x7fffffffdbd0
rsp            0x7fffffffdbd0   0x7fffffffdbd0
r8             0x7ffff7dd5e80   140737351868032
```

```
r9             0x0       0
r10            0x7fffffffd760    140737488344928
r11            0x7ffff7a30350    140737348043600
r12            0x4003e0  4195296
r13            0x7fffffffdd30    140737488346416
r14            0x0       0
r15            0x0       0
rip            0x4004df  0x4004df <bar+18>
eflags         0x202     [ IF ]
cs             0x33      51
ss             0x2b      43
ds             0x0       0
es             0x0       0
fs             0x0       0
gs             0x0       0
```

Write the activation record for `bar()` when `%rip` gets to 0x00000000004004df.
Under **Value** put the numeric value held at that address.
Under **Purpose** put one of the following:

a. not part of `bar()`'s activation record
b. argument to `bar()`
c. the address inx `foo()` to which `rip` should return
d. the stored `rbp` address for `foo()`
e. local variable to `bar()`
f. in the activation record of `bar()`, but not used

	Address:	Value:	Purpose:
	rbp + 0x10	0x00000000	f
	rbp + 0xC	0x00000000	c
	rbp + 0x8	0x00400515	c
	rbp + 0x4	0x00007fff	d
rbp -->	rbp + 0x0	0xffffe410	d
	rbp - 0x4	0x00000004	e
	rbp - 0x8	0x00000000	f
	rbp - 0xC	0x00000000	f
	rbp - 0x10	0x00000000	f
	rbp - 0x14	0x00000002	e
	rbp - 0x18	0x2f2f2f2f	a

4. **(10 Points) Assembly language understanding (3):**

What are the value(s) that `foo()` obtains as arguments from `main()`?

My answer: 0, 4

In which registers are they passed?

My answer: edi & esi

Give offset(s) from `rbp` from within `foo()`'s activation record *or* give the name(s) of the registers.

My answer: edi, esi

5. **(10 Points) Assembly language understanding (4):**

   How many *local variables* does `foo()` have?

   My answer: 5

   Where are they on the stack?

   My answer: -0x14(%rbp)、-0x18(%rbp)、 -0x4(%rbp) 、-0x8(%rbp) 、-0xc(%rbp)

   Give an offset from `rbp` from within `foo()`'s activation record.

   My answer: -0x4(%rbp)

   (I guess it means picking any one of where local variables lives?)

6. **(20 Points) Debugger usage (1):**

   `foo()` has a recursive call. Inside of `foo()` what are the values that both its *arguments* and *local variables* take on when rip is at address `0x00400518`? At the *top* of the table give the offset from `rbp` (the hexadecimal number added to `rbp` to get the address of the variable) of the parameter or local variable. (I may have tried to fool you the the number of variables.)

   In the body of the table write the values that that variable has when you hit address *local variables*.

Call:	rbp -0x4	rbp -0x8	rbp -0xc	rbp -0x14	rbp -0x18	rbp +_____
1	4	8	0	0	4	
2	2	4	0	2	4	

7. **(5 Points) Debugger usage (2):**

   What value does `foo()` return to `main()`?

   My answer: 12 (0x0000000c)

8. **(5 Points) Assembly language understanding (5):**

   `foo()` calls `bar()`. `bar()` starts at address 0x004004CD. If you look at the machine code for `foo()`'s call to `bar()`, however, you'll see that the actual number in the function call is 0xFFFF,FFB8

```
0x04004e1 <foo>:
  . . .
  40050e:89 c7                  mov    %eax,%edi
  400510:e8 b8 ff ff ff         callq  4004cd <bar>
  400515:89 45 f8               mov    %eax,-0x8(%rbp)
  . . .
```

   a. What to what number did the CPU add with 0xFFFF,FFB8 to get the address of `bar()`, 0x0040,04CD?

      My answer: 0x00400515

   b. Do this addition. Compute 0x0040,04CD.

      My answer: 0x00400485