## **CSC 373: Computer System I, 2019 Spring, Assignment #3**

Last revised 2019 May 8

### **Purpose:**

To:

1. Go over the basics of assembly language
2. Go over how to use a debugger
3. Go over the layout of an activation record

### **Assignment**

Please do the following:

1. Download the program called toAnalyzeCDM.zip from COL
2. Use an sftp program like [filezilla](http://filezilla-project.org/download.php?type=client) to upload it to a ctilinux machine (like 140.192.36.184) Do *not* bother unzipping it on your local machine.

On the same machine unzip it with:  
$ **unzip toAnalyzeCDM.zip**

1. Do chmod u+x toAnalyze to make tell Unix that it is an executable program

Analyze it with gdb: gdb toAnalyze. It has a structure like:  
int louie (/\* some number of args \*/)

{

/\* Some number of local vars \*/

return( /\* something \*/ );

}

int dewey (/\* some number of args \*/)

{

/\* Some number of local vars \*/

/\* Some loop \*/

/\* Some code, including call(s) to louie() \*/

return( /\* something \*/ );

}

int main ()

{

/\* Some number of local vars, including call(s) to dewey() \*/

return(0);

}



### **Answer the following:**

**(20 Points) Assembly language understanding (1):**The assembly language for louie() is:  
(gdb) **disass louie**

Dump of assembler code for function louie:

0x00000000004004cd <+0>: push %rbp

0x00000000004004ce <+1>: mov %rsp,%rbp

0x00000000004004d1 <+4>: mov %edi,-0x14(%rbp)

0x00000000004004d4 <+7>: mov %esi,-0x18(%rbp)

0x00000000004004d7 <+10>: mov -0x14(%rbp),%eax

0x00000000004004da <+13>: sub $0x2,%eax

0x00000000004004dd <+16>: mov %eax,-0x4(%rbp)

0x00000000004004e0 <+19>: mov -0x18(%rbp),%eax

0x00000000004004e3 <+22>: imul -0x4(%rbp),%eax

0x00000000004004e7 <+26>: pop %rbp

0x00000000004004e8 <+27>: retq

End of assembler dump.

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

|  |  |
| --- | --- |
| 1. **Instruction:** | 1. **Purpose:** |
| 1. push %rbp | 1. Pushes %rpb onto the stack to prepare function for use\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. mov %rsp,%rbp | 1. Move %rsp to top of stack (%rbp) to prepare louie function for use\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. mov %edi,-0x14(%rbp) | 1. Move the passed in value at %edi to 14 bytes less than %rbp in order to create a new local variable with that data\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. mov %esi,-0x18(%rbp) | 1. Move the passed in value at %esi to 18 bytes less than %rbp in order to create a new local variable with that data\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. mov -0x14(%rbp),%eax | 1. Move the first local variable to %eax for manipulation\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. sub 0x2,%eax | 1. Subtract 2 from %eax to create a new value that we will store soon\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. mov %eax,-0x4(%rbp) | 1. Store our new value in a new variable located 4 bytes below %rbp for use later on\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. mov -0x18(%rbp),%eax | 1. Move our second local variable to %eax for manipulation\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. imul -0x4(%rbp),%eax | 1. Multiple our newly saved variable that is 4 bytes below %rbp with our second local variable that we just moved to %eax with signage\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. pop %rbp | 1. Pop %rbp back to the previous function that called louie to let the compiler know we’re done with louie\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. retq | 1. Takes whatever value the function louie returns back to the address from which louie was called and resumes execution\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

1. **(10 Points) Assembly language understanding (2):**Write a C function that does what louie() 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).

int louie (first, second)

{

int third = first - 2;

return(second \* third);

}

**(20 Points) Activation Records (1):**Stop the program at its *second* call to louie(). When I did so I got the following:  
(gdb) **break louie**

Breakpoint 1 at 0x4004d1

(gdb) **run**

Starting program: /home/instructor/Documents/Academic/DePaul/Classes/CSC373/20189-3Spr/Assign3/toAnalyze

Breakpoint 1, 0x00000000004004d1 in louie ()

(gdb) **c**

Continuing.

Breakpoint 1, 0x00000000004004d1 in louie ()

(gdb) **stepi**

0x00000000004004d4 in louie ()

(gdb) **stepi**

0x00000000004004d7 in louie ()

(gdb) **stepi**

0x00000000004004da in louie ()

(gdb) **stepi**

0x00000000004004dd in louie ()

(gdb) **stepi**

0x00000000004004e0 in louie ()

(gdb) **stepi**

0x00000000004004e3 in louie ()

(gdb) **stepi**

0x00000000004004e7 in louie ()

(gdb) **info reg**

rax 0x0 0

rbx 0x0 0

rcx 0x400550 4195664

rdx 0x4 4

rsi 0x4 4

rdi 0x2 2

rbp 0x7fffffffdc28 0x7fffffffdc28

rsp 0x7fffffffdc28 0x7fffffffdc28

r8 0x7ffff7dd5e80 140737351868032

r9 0x0 0

r10 0x7fffffffd760 140737488344928

r11 0x7ffff7a302e0 140737348043488

r12 0x4003e0 4195296

r13 0x7fffffffdd50 140737488346448

r14 0x0 0

r15 0x0 0

rip 0x4004e7 0x4004e7 <louie+26>

eflags 0x206 [ PF IF ]

cs 0x33 51

ss 0x2b 43

ds 0x0 0

es 0x0 0

fs 0x0 0

gs 0x0 0

1. Write the activation record for louie() when %rip gets to 0x00000000004004e7.  
   Under **Value** put the numeric value held at that address.  
   Under **Purpose** put one of the following:
2. not part of louie()'s activation record
3. argument to louie()
4. the address inx dewey() to which rip should return
5. the stored rbp address for dewey()
6. local variable to louie()
7. in the activation record of louie(), but not used

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1. **Address:** | 1. **Value:** | 1. **Purpose:** |
|  | 1. rbp + 0x10 | 1. \_\_\_\_\_\_\_\_\_\_\_ | 1. not part of louie()'s activation record\_\_\_\_\_\_\_\_\_\_\_ |
|  | 1. rbp + 0xC | 1. \_\_\_\_\_\_\_\_\_\_\_ | 1. not part of louie()'s activation record\_\_\_\_\_\_\_\_\_\_\_ |
|  | 1. rbp + 0x8 | 1. Address of the function calling Louie 0x0040050f \_\_\_\_\_\_\_\_\_\_\_ | 1. the address inx dewey() to which rip should return\_\_\_\_\_\_\_\_\_\_\_ |
|  | 1. rbp + 0x4 | 1. Part of the address of the function calling Louie 0x00007fff\_\_\_\_\_\_\_\_\_\_\_ | 1. Hold the address inx dewey() to which rip should return\_\_\_\_\_\_\_\_\_\_\_ |
| 1. rbp --> | 1. rbp + 0x0 | 1. 0xffffdc28\_\_\_\_\_\_\_\_\_\_\_ | 1. the stored rbp address for dewey() 2. \_\_\_\_\_\_\_\_\_\_\_ |
|  | 1. rbp - 0x4 | 1. 0x0\_\_\_\_\_\_\_\_\_\_\_ | 1. local variable to louie()\_\_\_\_\_\_\_\_\_\_\_ |
|  | 1. rbp - 0x8 | 1. \_\_\_\_\_\_\_\_\_\_\_ | 1. in the activation record of louie(), but not used\_\_\_\_\_\_\_\_\_\_\_ |
|  | 1. rbp - 0xC | 1. \_\_\_\_\_\_\_\_\_\_\_ | 1. in the activation record of louie(), but not used 2. \_\_\_\_\_\_\_\_\_\_\_ |
|  | 1. rbp - 0x10 | 1. \_\_\_\_\_\_\_\_\_\_\_ | 1. in the activation record of louie(), but not used 2. \_\_\_\_\_\_\_\_\_\_\_ |
|  | 1. rbp - 0x14 | 1. 0x2\_\_\_\_\_\_\_\_\_\_\_ | 1. local variable to louie() but was an argument to louie() to begin with\_\_\_\_\_\_\_\_\_\_\_ |
|  | 1. rbp - 0x18 | 1. 0x4\_\_\_\_\_\_\_\_\_\_\_ | 1. local variable to louie() but was an argument to louie() to begin with\_\_\_\_\_\_\_\_\_\_\_ |
|  | 1. rbp - 0x1C | 1. \_\_\_\_\_\_\_\_\_\_\_ | 1. not part of louie()'s activation record\_\_\_\_\_\_\_\_\_\_\_ |

2. **(10 Points) Assembly language understanding (3):**What are the value(s) that dewey() obtains as arguments from main()?

6 and 0

In which registers are they passed?

esi and edi

1. **(10 Points) Assembly language understanding (4):**How many *local variables* does dewey() have?

4 local variables  
Where are they on the stack?

* -0x14(%rbp)
* -0x18(%rbp)
* -0x4(%rbp)
* -0x8(%rbp)

Give an offset from rbp from within dewey()'s activation record. (Include arguments passed in registers that are subsequently placed on the stack as local variables, too.)

1. **(20 Points) Debugger usage (1):**dewey() has a loop. Inside of dewey() what are the values of both its *arguments* and *local variables* the *first time*, *second time* and *third time* rip is 0x00400520? 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 deweyl 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*.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. **Call:** | 1. **rbp + -0x18\_\_\_\_\_** | 1. **rbp + -0x4\_\_\_\_\_** | 1. **rbp + -0x14\_\_\_\_\_** | 1. **rbp + -0x8\_\_\_\_\_** | 1. **rbp + \_\_\_\_\_** | 1. **rbp + \_\_\_\_\_** |
| 1. 1 | 1. 6\_\_\_\_\_\_\_\_ | 1. 0\_\_\_\_\_\_\_\_ | 1. 0\_\_\_\_\_\_\_\_ | 1. 0\_\_\_\_\_\_\_\_ | 1. \_\_\_\_\_\_\_\_ | 1. \_\_\_\_\_\_\_\_ |
| 1. 2 | 1. 4\_\_\_\_\_\_\_\_ | 1. -12\_\_\_\_\_\_\_\_ | 1. 2\_\_\_\_\_\_\_\_ | 1. -12\_\_\_\_\_\_\_\_ | 1. \_\_\_\_\_\_\_\_ | 1. \_\_\_\_\_\_\_\_ |
| 1. 3 | 2\_\_\_\_\_\_\_\_ | 1. -12\_\_\_\_\_\_\_\_ | 1. 4\_\_\_\_\_\_\_\_ | 1. 0\_\_\_\_\_\_\_\_ | 1. \_\_\_\_\_\_\_\_ | 1. \_\_\_\_\_\_\_\_ |

1. **(5 Points) Debugger usage (2):**What value does dewey() return to main()?

Dewey returns 0xfffffff4 in hex, 4294967284 in unsigned decimal, and most importantly, -12 in signed (2’s compiment) decimal, because that is the last value held in Dewey’s -0x4(%rbp)

**(5 Points) Assembly language understanding (5):**dewey() calls louie(). louie() starts at address 0x0040,04CD. If you look at the machine code for dewey()'s call to louie(), however, you'll see that the actual number in the function call is 0xFFFF,FFBE  
0x04004e9 <dewey>:

. . .

400508: 89 c7 mov %eax,%edi

40050a: e8 be ff ff ff callq 4004cd <louie>

40050f: 89 45 f8 mov %eax,-0x8(%rbp)

. . .

1. 1. What to what number did the CPU add with 0xFFFF,FFBE to get the address of louie(), 0x0040,04CD?

0040050F

* 1. Do this addition. Compute 0x0040,04CD.

FFFFFFBE + 0040050F = 1004004CD = 004004CD, because you drop the 1