

CS 2021: Practice Exam 2

Spring 2022

University of Minnesota

Exam period: 20 minutes

Points available: 40

Problem 1 (15 pts): Nearby is a C function `col_update()` with associated data and documentation. **Re-implement this function in x86-64 assembly** according to the documentation given. Follow the same flow provided in the C implementation. The comments below the `colinfo_t` struct give information about how it lays out in memory and as a packed argument.

Indicate which registers correspond to which C variables.

```
.text
.globl      col_update
# YOUR CODE BELOW
col_update:
```

```
1 typedef struct{
2     int cur;
3     int step;
4 } colinfo_t;
5 // |         | Byte |   Byte | Packed |
6 // | Field | Size | Offset | Bits |
7 // |-----+-----+-----+-----|
8 // | cur   |   4 |    +0 | 0-31 |
9 // | step  |   4 |    +4 | 32-64 |
10
11 int col_update(colinfo_t *info){
12     // Updates current value and step in
13     // colinfo_t pointed by param info. If
14     // info->cur is invalid, makes no changes
15     // and returns 1 to indicate an
16     // error. Otherwise performs odd or even
17     // update on cur and increments step
18     // returning 0 for success.
19
20     int cur = info->cur;
21     int step = info->step;
22     if(cur <= 0){
23         return 1;
24     }
25     step++;
26     if(cur % 2 == 1){
27         cur = cur*3+1;
28     }
29     else{
30         cur = cur / 2;
31     }
32     info->cur = cur;
33     info->step = step;
34     return 0;
35 }
```

Problem 2 (15 pts): Below is an initial register/memory configuration along with snippets of assembly code. Each snippet is followed by a blank register/memory configuration which should be filled in with the values to reflect changes made by the preceding assembly. The code is continuous so that POS A is followed by POS B.

INITIAL

-----+-----		addl %edi, %esi		movq \$1, %rdi	
REG	Value	REG	Value	REG	Value
-----+-----		-----+-----		-----+-----	
rax	10	rax		rax	
rdi	20	rdi		rdi	
rsi	30	rsi		rsi	
rsp	#3032	rsp		rsp	
-----+-----		-----+-----		-----+-----	
MEM	Value	MEM	Value	MEM	Value
-----+-----		-----+-----		-----+-----	
#3032	250	#3032		#3032	
#3028	1	#3028		#3028	
#3024	2	#3024		#3024	
#3020	3	#3020		#3020	
-----+-----		-----+-----		-----+-----	

-----+-----		subq \$8, %rsp		movq \$1, %rdi	
-----+-----		movl \$100, 4(%rsp)		addl %esi, (%rsp,%rdi,4)	
-----+-----		movl \$300, 0(%rsp)		leaq 8(%rsp), %rdi	
-----+-----		addl (%rsp), %eax		addl (%rdi), %rax	
-----+-----		# POS A		# POS B	
-----+-----		-----+-----		-----+-----	

Problem 3 (10 pts): Rover Witer is writing an assembly function called `compval` which he will use in C programs. He writes a short C `main()` function to test `compval` but is shocked by the results which seem to defy the C and assembly code. Valgrind provides no insight for him. **Identify why** Rover's code is behaving so strangely and fix `compval` so it behaves correctly.

Sample Compile / Run:

```
> gcc compval_main.c compval_asm.s
> a.out
expect: 0
actual: 19
expect: 0
actual: 50
```

```
1 // compval_main.c
2 #include <stdio.h>
3
4 void compval(int x, int y, int *val);
5 // compute something based on x and y
6 // store result at int pointed to by val
7
8 int main(){
9     int expect, actual;
10
11     expect = 7 * 2 + 5;    // expected value
12     compval(7, 2, &actual); // actual result
13     printf("expect: %d\n", expect);
14     printf("actual: %d\n", actual);
15
16     expect = 5 * 9 + 5;    // expected value
17     compval(5, 9, &actual); // actual result
18     printf("expect: %d\n", expect);
19     printf("actual: %d\n", actual);
20
21     return 0;
22 }
```

```
1 # compval_asm.s
2 .text
3 .global compval
4 compval:
5     imulq    %rdi,%rsi
6     addq     $5,%rsi
7     movq     %rsi, (%rdx)
8     ret
```