CS 2021: Practice Exam 2 SOLUTION
Fall 2020
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.

### SOLUTION:

```assembly
.globl col_update
.col_update:

# YOUR CODE BELOW
movl 0(%rdi),%esi          # cur = info->cur
movl 4(%rdi),%edx          # step= info->step

cmp %0,%esi                # if(cur < 0)
jle .ERROR

## ODD CASE (fall through)
imull $3,%esi              # odd: cur *= 3
addl $1,%esi               # odd: cur += 1
jmp .RETURN                # jump over even

.EVEN:
sarl %1,%esi               # even: cur /= 2

.RETURN:

ERROR:
movl %esi,0(%rdi)          # info->cur = cur;
movl %edx,4(%rdi)          # info->step= step;
movl $0,%eax              # success
ret                        # return 0

ERROR:
movl $1,%eax              # error case
ret                        # return 1
```

```c
typedef struct{
    int cur;
    int step;
} colinfo_t;

int col_update(colinfo_t *info){
    // Updates current value and step in
    // colinfo_t pointed by param info. If
    // info->cur is invalid, makes no changes
    // and returns 1 to indicate an
    // error. Otherwise performs odd or even
    // update on cur and increments step
    // returning 0 for success.
    int cur = info->cur;
    int step = info->step;

    if(cur <= 0){
        return 1;
    }
    step++;  // step++;
    if(cur % 2 == 1){
        cur = cur*3+1;
    } else{  // even case
        cur = cur / 2;
    }
    info->cur = cur;
    info->step = step;
    return 0;
}
```

1A
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.

SOLUTION:

|-------+-------| |-------+-------| |-------+-------|
| REG | Value | | REG | Value | | REG | Value |
|-------+-------| |-------+-------| |-------+-------|
| rax | 10 | | rax | 310 | | rax | 560 |
| rdi | 20 | | rdi | 20 | | rdi | #3032 |
| rsi | 30 | | rsi | 50 | | rsi | 50 |
| rsp | #3032 | | rsp | #3024 | | rsp | #3024 |
|-------+-------| |-------+-------| |-------+-------|
| MEM | Value | | MEM | Value | | MEM | Value |
|-------+-------| |-------+-------| |-------+-------|
| #3032 | 250 | | #3032 | 250 | | #3032 | 250 |
| #3028 | 1 | | #3028 | 100 | | #3028 | 150 |
| #3024 | 2 | | #3024 | 300 | | #3024 | 300 |
| #3020 | 3 | | #3020 | 3 | | #3020 | 3 |

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

SOLUTION: The movq instruction at line 7 of compval writes 8 bytes. This is inappropriate as a 4-byte int is supposed to be written. Apparently the stack layout in main() has the variable actual at a memory address immediately below variable expect so that on writing 8 bytes, the low order 4 bytes correctly get written to actual but the high order 4 bytes (all 0’s for small values) overwrite the variable expect leaving it as 0. The fix for this is to use movl %eax, (%rdx) which will write 4 bytes, filling only actual.

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    expect = 7 * 2 + 5; // expected value
11    compval(7, 2, &actual); // actual result
12    printf("expect: %d\n",expect);
13    printf("actual: %d\n",actual);
14    expect = 5 * 9 + 5; // expected value
15    compval(5, 9, &actual); // actual result
16    printf("expect: %d\n",expect);
17    printf("actual: %d\n",actual);
18    return 0;
19 }
20 1 // compval_asm_corrected.s
2 .text
3 .global compval
4 compval:
5     imulq %rdi,%rsi
6     addq $5,%rsi
7     movl %esi,%rdx       # was movq %rsi, (%rdx)
8     ret       # now fixed