CS 2400 Midterm 1

Solutions

October 16, 2014

1

(a) True (b) True (c) False (d) True

2

Answer key:
1)– c)
2)– f)
3)– d)
4)– e)

3

since this is worth 18 points, give +3 to the 'x+2' expression blank, and +3 to last row's overflow blank. Give +2 to other 6 blanks.
y = -29
x = 30

Expression	Decimal Representation	Hex Representation	Overflow?
y	-29	0x23	No
x + 1	TMax	**0x1f**	No
x + 2	-32	0x20	Yes
x+y	1	0x01	**No**
x + TMax	**-3**	0x3d	Yes
TMin+y	**3**	**0x03**	**Yes**

4

a

```
0 x0804849b <+17>: movl $0xa,(%esp)
```

b

```
4 bytes for pushing \%ebp
4 bytes for pushing \%ebx
36 bytes for subtracting 0x24 from \%esp
4 bytes for pushing return address when calling 1st t()
Total bytes = 48
```

Grading:
+4 for 48 bytes
+2 for 20 bytes within 48
0 otherwise

c

Following the path of longest length:
within main:
4 pushing %ebp
variable # of bytes 0-15 for "and $0xfffffff0,%esp", which aligns stack pointer
on a multiple of 16. The stack pointer is likely at a multiple of 4,
so 0-12 bytes is the practical range, but we accepted up to 15.
32 subbing from %esp
4 return address for calling r()

within r:
4 pushing %ebp
4 pushing %ebx
36 subbing from %esp
%ignore call to t() since it will be less%
4 return address for calling s()

within s:
4 pushing %ebp
4 pushing %ebx
36 subbing from %esp
4 return address for calling t()
%only need one call since they are the same function%

within t:
4 pushing %ebp
16 subbing from %esp

Total bytes moved = 156 bytes if stack ptr aligned on multiple of 16,
156-171 OK if not aligned

We did not count the amount that the print statement may have increased the stack.

Grading:
+12 for [156 bytes, 171]
+10 for any answer in [136,156) or [172,192]
+8 for any answer in [116,136) or (192,212]
+6 for any answer in [96,116) or (212,232]
+4 for any answer in [56, 96) or (232,272]
+2 for any answer in [16, 56) or (272,312]
+0 otherwise

d

78

grading:
+8 for final value = 78
+6 for final value in [58,78) or (78,98]
+4 for final value in [38,58) or (98,118]
+2 for final value in [18,38) or (118,138]
+0 other

5

```
push   %ebp
mov    __%ESP__,%ebp
sub    $0x34,%esp
mov    0x8(%ebp),%eax
mov    %al,-0x34(%ebp)
movl   $0xf,-0x20(%ebp)
mov    0xc(__%EBP__),%eax
mov    %ax,-0x22(%ebp)
__JMP__    L1

.L4
__CMPL__   $0x0,-0x20(%ebp)
jns    L2
movzwl -0x22(%ebp),%eax
sub    $0x1,%eax
movsbl -0x34(%ebp),__%ECX__
movzwl -0x22(%ebp),%edx
__ADD__    %ecx,%edx
mov    %edx,-0x1c(%ebp,__%EAX__,4)
jmp    L3
```

```
.L2
__MOVZWL__ -0x22(%ebp),%eax          // note: some put movl here, but this is wrong
because it would move the lower 2 bytes from int y into the upper 2 bytes of eax
movsbl -0x34(%ebp),%edx
mov    __%EDX__,-0x1c(%ebp,%eax,4)  // note: some answers put 0x8(%ebp) here,
which is incorrect because an x86 assembly instruction cannot have both operands
referencing memory

.L3
movzwl -0x22(%ebp),%eax
mov    -0x1c(%ebp,%eax,4),%eax
add    %eax,-0x20(%ebp)
movzwl -0x22(%ebp),%eax
__SUB__    $0x1,%eax
mov    %ax,-0x22(%ebp)

.L1
cmpw   $0x0,-0x22(%ebp)
jne    .L4
mov    -0x20(%ebp),%eax
leave
ret
```