University of Washington – Computer Science & Engineering
Winter 2017    Instructor: Justin Hsia    2017-02-10

CSE410 MIDTERM

Last Name: Perfect
First Name: Perry
Student ID Number: 1234567
Name of person to your Left | Right Samantha Student | Larry Learner

All work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CSE410 who haven’t taken it yet. (please sign)

Do not turn the page until 2:30.

Instructions
- This exam contains 7 pages, including this cover page. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- The last page is a reference sheet. Please detach it from the rest of the exam.
- The exam is closed book (no laptops, tablets, wearable devices, or calculators). You are allowed one page (US letter, double-sided) of handwritten notes.
- Please silence and put away all cell phones and other mobile or noise-making devices. Remove all hats, headphones, and watches.
- You have 50 minutes to complete this exam.

Advice
- Read questions carefully before starting. Skip questions that are taking a long time.
- Read all questions first and start where you feel the most confident.
- Relax. You are here to learn.

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Points</td>
<td>12</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td>46</td>
</tr>
</tbody>
</table>
Question 1: Number Representation [12 pts]

(A) What is the value of the unsigned char 0x58 in decimal? [1 pt]

0x58 = 5x16 + 8 = 88. Alternatively, 0x58 = 2^5 + 2^4 + 2^3 = 88

(B) What is the value of signed char z = (0x8B >> 4) in decimal? [2 pt]

z is signed, so arithmetic right shift. z = 0x8B >> 4 = 0xF8.
-z = 0x08 = 8, so z = -8.

(C) For signed char m = 0x20, find the smallest positive integer n (in decimal) such that m+n<0. [2 pt]

char is 8 bits, so Tmax = 127. Overflow occurs at 128, so n = 128 – 0x20 = 128 – 32 = 96.

For the rest of this problem we are working with a floating point representation that follows the same conventions as IEEE 754 except using 10 bits split into the following fields:

<table>
<thead>
<tr>
<th>Sign (1)</th>
<th>Exponent (5)</th>
<th>Mantissa (4)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>(D)</th>
<th>What is the magnitude of the bias of this new representation? [1 pt]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bias = 2^5 - 1 = 15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(E)</th>
<th>Translate the floating point number 0b 1101000001 into decimal. [3 pt]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S = 1, E = 10100_2, M = 0001_2. E is not a special case.</td>
</tr>
<tr>
<td></td>
<td>Exp = 2^4 + 2^2 - 15 = 5, Man = 1.0001_2.</td>
</tr>
<tr>
<td></td>
<td>(-1)^1 x 1.0001_2 x 2^5 = -100010_2 = -(2^5 + 2^1) = -34.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(F)</th>
<th>Assume the variable f stores a number encoded in the 10-bit floating point representation described above and variable b stores the bias (your answer to part (D)). Using C syntax, complete the line below that returns the signed value of the exponent. [3 pt]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>int exponent = _((f &amp; 0x1F0) &gt;&gt; 4) - b _________________________;</td>
</tr>
<tr>
<td>Also accepted:</td>
<td>((f &gt;&gt; 4) &amp; 0x1F) - b</td>
</tr>
</tbody>
</table>

SID: 1234567
Question 2: Pointers & Memory  [10 pts]

For this problem we are using a 64-bit x86-64 machine (little endian). The current state of memory (values in hex) is shown below:

<table>
<thead>
<tr>
<th>Word Addr</th>
<th>+0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>+6</th>
<th>+7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>AC</td>
<td>AB</td>
<td>03</td>
<td>01</td>
<td>BA</td>
<td>5E</td>
<td>BA</td>
<td>11</td>
</tr>
<tr>
<td>0x08</td>
<td>5E</td>
<td>00</td>
<td>AB</td>
<td>0C</td>
<td>BE</td>
<td>A7</td>
<td>CE</td>
<td>FA</td>
</tr>
<tr>
<td>0x10</td>
<td>1D</td>
<td>B0</td>
<td>99</td>
<td>DE</td>
<td>AD</td>
<td>60</td>
<td>BB</td>
<td>40</td>
</tr>
<tr>
<td>0x18</td>
<td>14</td>
<td>CD</td>
<td>FA</td>
<td>1D</td>
<td>D0</td>
<td>41</td>
<td>ED</td>
<td>77</td>
</tr>
<tr>
<td>0x20</td>
<td>BA</td>
<td>B0</td>
<td>FF</td>
<td>20</td>
<td>80</td>
<td>AA</td>
<td>BE</td>
<td>EF</td>
</tr>
</tbody>
</table>

char* cp = 0x13
short* sp = 0x26
int* ip = 0x4

(A) Using the values shown above, complete the C code below to fulfill the behaviors described in the comments using pointer arithmetic. [4 pt]

```c
char v1 = cp[___5___]; // set v1 = 0x14
int* v2 = ip + ___10___; // set v2 = 44
```

v1: Byte 0x14 is at address 0x18. 0x18 – cp = 5.
v2: No dereferencing; just pointer arithmetic (scaled by sizeof(int)=4).
i$p=4$. To get to 44, need to add 40 (10 by pointer arithmetic).

(B) What are the values (in hex) stored in each register shown after the following x86 instructions are executed? Remember to use the appropriate bit widths. [6 pt]

<table>
<thead>
<tr>
<th>Register</th>
<th>Data (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>0x0000 0000 0000 000C</td>
</tr>
<tr>
<td>%rsi</td>
<td>0x0000 0000 0000 0001</td>
</tr>
<tr>
<td>%al</td>
<td>0x0D</td>
</tr>
<tr>
<td>%bx</td>
<td>0xFACE</td>
</tr>
<tr>
<td>%rcx</td>
<td>0xFFFF FFFF FFFF FFAD</td>
</tr>
</tbody>
</table>

movw instruction pulls 2 bytes from memory starting at address 0xC+2 = 0xE.
Remember little-endian!
movsbq instruction pulls byte at memory address 0xC+8 = 0x14, which is 0xAD.
Then sign extend out to 64 bits.
Question 3: C & Assembly  [12 pts]

We are writing the function `copyDelim`, which takes two char pointers and copies the characters from `src` to `dst` until either the **null terminator** or a specified **delimiter** (`delim`) are found. **Example:** For pointer `p`, executing `copyDelim("410 ROCKSLIDE", p, 'L')` results in `p` pointing to "410 ROCKS".

(A) Fill in the blank below in the C function below:  [2 pt]

```c
void copyDelim (char* src, char* dst, char delim) {
    while( __*src____ & ( *src != delim ) ) {
        *dst++ = *src++; // *dst = *src; dst++; src++;
    }
    *dst = '\0'; // add null terminator (value 0)
}
```

(B) Fill in the blanks in the x86-64 code below with the correct instructions and operands. **Remember to use the proper size suffixes and correctly-sized register names!**  [10 pt]

```assembly
.copyDelim(char*, char*, char):
    .Loop:
    1    movb    (%rdi),  %al  # get *src
    2    testb   %al,      %al  # conditional
    3    je      .Exit     # conditional jump
    4    cmpb    %dl,      %al  # conditional
    5    je      .Exit     # conditional jump
    6    addq    $1,       %rdi  # increment src
    7    movb    %al,      (%rsi) # copy char
    8    leaq    1(%rsi),  %rsi # increment dst
    9    jmp     .Loop     # unconditional jump
    .Exit:
    10   movb    $0,       (%rsi) # terminate dst
    11   ret      # return
```
Grading Notes for Question 3:

**Line 1**: Must be dereference, must be 64-bit register name, `src` is first argument (%rdi).

**Line 4**: Must be `b` width specifier because source is %dl. Comparing `delim(%dl)` against `*src` (stored in %al).

**Line 5**: Checking for opposite of “Test” as exit condition, so `je` (jump if equal to zero) to `.Exit`.

**Line 6**: Must be `q` width specifier because destination is %rdi.

**Line 7**: Must be `b` width specifier because source is %al. Storing into memory at dst.

**Line 8**: Tricky alternate way to add 1 (compilers do produce this) to %rsi.

**Line 10**: Null terminator is just the character with value 0 (as hinted in the comments in the C code above).

**Line 11**: `retq` also accepted.
Question 4: Procedures & The Stack  [12 pts]

The recursive power function `power()` calculates `base^pow` and its x86-64 disassembly is shown below:

```c
int power(int base, unsigned int pow) {
    if (pow)
        return base * power(base, pow-1);
    return 1;
}
```

(A) How much space (in bytes) does this function take up in our final executable?  [1 pt]

Count all bytes (middle columns) or subtract address of next instruction (0x4005bc) from 0x4005a0.  

28 B

(B) Circle one: The label `power` will show up in which table(s) in the object file?  [2 pt]

Symbol Table  Relocation Table  Both Tables  Neither Table

`power` is called in this file (recursive) and can be called by external files, so in both.

(C) Which register is being saved on the stack?  [1 pt]

See `pushq` instruction (0x4005a4).

%rbx

(D) What is the return address to `power` that gets stored on the stack? Answer in hex.  [1 pt]

The address of the instruction after `call`.

0x4005af
(E) Assume main calls power(3,4). How many times will the return address to power be stored on the stack during the execution of this call? [2 pt]

4

The return address to power will be pushed onto the stack every time power makes a recursive call.
main → power(3,4) → power(3,3) → power(3,2) → power(3,1) → power(3,0)

(F) Assume main calls power(4,10) and we find that the return address to main is stored on the stack at address 0x7fffeca3f748. What data (answer in hex) will be stored on the stack at address 0x7fffeca3f720? You may use the provided stack diagram, but you will only be graded on the answer box to the right. [3 pt]

0x4

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7fffeca3f748</td>
<td>&lt;ret addr to main&gt;</td>
</tr>
<tr>
<td>0x7fffeca3f740</td>
<td>&lt;original rbx&gt;</td>
</tr>
<tr>
<td>0x7fffeca3f738</td>
<td>0x4005af</td>
</tr>
<tr>
<td>0x7fffeca3f730</td>
<td>0x4 &lt;base&gt;</td>
</tr>
<tr>
<td>0x7fffeca3f728</td>
<td>0x4005af</td>
</tr>
<tr>
<td>0x7fffeca3f720</td>
<td>0x4 &lt;base&gt;</td>
</tr>
</tbody>
</table>

(G) Harry the Husky claims that we didn’t need to push a register onto the stack in this case. Is our intrepid school’s mascot correct or not? Briefly explain. [2 pt]

Harry is correct! base doesn’t change between recursive calls and power doesn’t call other procedures, so there is no need to save %rdi in %rbx.

In fact, if you compile the C function with an optimization flag of -O2, it doesn’t push %rbx onto the stack!