Exam #1 Review

By sseshadr

Agenda

- Reminders
  - Test tomorrow!
    - One 8.5 x 11 sheet, two sides
  - Pick up your datalabs in OH
  - Cachelab comes out tomorrow
- Review
- Questions

[Subset of] What to Know

- Labs!
  - We try to reward people who did them well
- Assembly
  - Basics (what does `cmp` do, source vs. dest, operand order for add, etc.)?
  - What registers are special? Caller save vs. callee save?
  - Switch statements and jump tables?
  - Loops?
  - You **should be able to** trace through assembly. Practice it.
  - You **should be able to** write small amounts of assembly (like buflab).
- Data Representation
  - Two's compliment
  - Floating point
  - Endianness

[Subset of] What to Know

- Stack
  - What's different in 32- vs. 64-bit
  - You **should know** parameters, ebp values, return address, etc.
- Larger Structures
  - Structs and Unions
    - What's the difference?
    - Padding and alignment
  - Arrays
    - Multi-dimensional access
- Control
  - Loops in assembly?
  - Recursion?
- Memory
  - Heap vs. Stack
  - What is the L1 Cache?

Floating Point Review

- Basics
  - Sign, Mantissa, Exponent
  - Round to even
  - Bias
  - Infinity, +- zero, NaN
  - Normalized vs. Denormalized

1. Format A
   - There are $k = 3$ exponent bits. The exponent bias is 3.
   - There are $n = 5$ fraction bits.
2. Format B
   - There are $k = 5$ exponent bits. The exponent bias is 15.
   - There are $n = 3$ fraction bits.

Fill in the blanks in the table below by converting the given values in each format to the closest possible value in the other format. Express values as whole numbers (e.g., 17) or as fractions (e.g., 17/64). If necessary, you should apply the round-to-even rounding rule.

Format A		Format B	
Bits	Value	Bits	Value
`011 00000`	1	`01111 000`	1
110 11100	15	10010 111	15
100 10101	$\frac{53}{16}$	10000 101	13/4
111 00000	Infinity	`10100 110`	56
`000 00001`	1/128	01000 000	1/128

Round-to-even examples

- Represent 25/64 with 4 exponent bits, 3 fraction bits. Bias is $2^{(4-1)} - 1 = 7$
  - 0101 100: Rounded DOWN to value 3/8
- Represent 27/64 with 4 exponent bits, 3 fraction bits
  - 0101 110: Rounded UP to value 7/16
- Represent 51/128 with 4 exponent bits, 3 fraction bits
  - 0101 101: Rounded UP to value 13/32
  - Didn't use round-to-even on this… it wasn't a "tie"

Array Access

- Start with the C code
- Then look at the assembly
  - Work backwards!
- Easiest to just do an example

```
int array1[H][J];
int array2[J][H];

int copy_array(int x, int y) {
    array2[y][x] = array1[x][y];

    return 1;
}
```

*(array2 + yH + x)

*(array1 + xJ + y)

Remember though, multiply the offset by `sizeof(int)`

Suppose the above C code generates the following x86-64 assembly code:

```
# On entry:
#    %edi = x
#    %esi = y
#
copy_array:
    movslq  %edi,%rdi
    movslq  %esi,%rsi
    movq    %rdi, %rax
    leaq    (%rsi,%rsi,2), %rdx
    salq    $5, %rax
    subq    %rdi, %rax
    leaq    (%rdi,%rdx,2), %rdx
    addq    %rsi, %rax
    movl    array1(,%rax,4), %eax
    movl    %eax, array2(,%rdx,4)
    movl    $1, %eax
    ret
```

rax == x

rdx == 3y

rax == 32x

rax == 31x

rdx == 6y+x

rax == 31x + y

rax must have value Jx +y

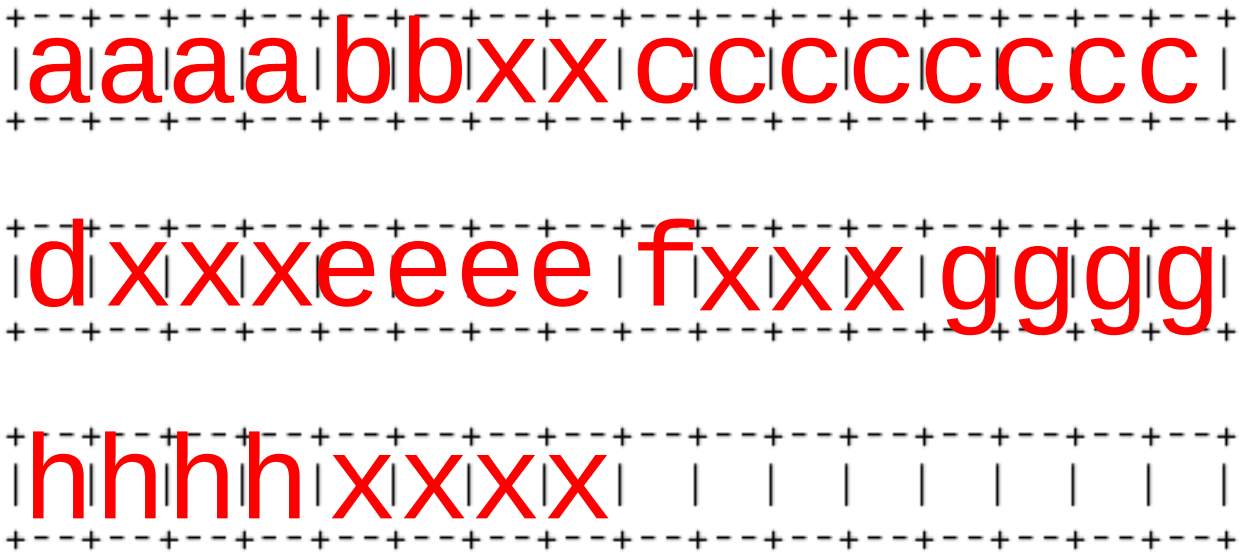
rdx must have value Hy +x

Structs

- What is a union again?
- How big are things?
  - If you can't remember, cheat sheet.
  - int, char, pointer (you should know these by now)
  - float, double, short
- Alignment rules
  - If you can't remember, cheat sheet.

```
struct {
    char *a;
    short b;
    double c;
    char d;
    float e;
    char f;
    long g;
    void *h;
} foo;
```

A. Show how the struct above would appear on a 32-bit Windows machine (primitives of size k are k-byte aligned). Label the bytes that belong to the various fields with their names and clearly mark the end of the struct. Use hatch marks to indicate bytes that are allocated in the struct but are not used.



Are we done???

NO!!