CS61C Review

Midterm Spring 2008

Five Elements of a Computer

- Control
- Datapath
- Memory
- Input
- Output

Negative Numbers

- Sign/Magnitude
- One's Complement
- Two's Complement

- Pros, Cons of Each

C Topics

- Pointers!
  - malloc, free
  - Handles
- Pass by Value vs Pass by Reference
- Arrays
- Structs
- typedef

Memory Management

- Static
- The Stack
- The Heap

Memory Management Allocation Schemes

- Best-fit
- First-fit
- Next-fit

- Slab
- Buddy

MIPS

- R, I, J format instructions (on your green sheet!)
- MAL vs TAL

- MIPS to Binary, Binary to MIPS
- Difference between branches, jumps

Various Other Things

- Floats
- CALL (Compile, Assemble, Link, Load)

Garbage Collection

- Reference Counting
- Mark and Sweep
- Copying

- Pros and Cons

Socratic Method,
Understanding Floats

Float Cheat Sheet

S	Exponent	Significand
1 bit	E bits	F bits

Normalized Float:

$(-1)^{S} \cdot (1 + \text{Significand}) \times 2^{(\text{Exponent} - \text{Bias})}$

Denormalized Float:

$(-1)^{S} \cdot (\text{Significand}) \times 2^{(1 - \text{Bias})}$

Bias = $2^{(E - 1)} - 1$
(0 and all 1s)

Exponent	Significand	Value
0	0	0
0	nonzero	Denorm
$1 \sim 2^{E} - 2$	Anything	+/- fl. Pt #
$2^{E} - 1$ (all 1s)	0	+/- ∞
$2^{E} - 1$ (all 1s)	nonzero	NaN

Float Cheat Sheet

S	Exponent	Significand
1 bit	E bits	F bits

Just as in sign and magnitude, the sign bit encodes the sign of the number, 0 means positive, 1 means negative.

Float Cheat Sheet

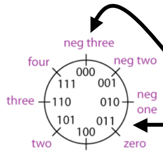
S	Exponent	Significand
1 bit	E bits	F bits

The significand is encoded as a fixed point unsigned number, such that the most significant bit has a value of $2^{(-1)}$. Accordingly, the significand always has a value < 1.

Float Cheat Sheet

S	Exponent	Significand
1 bit	E bits	F bits

The exponent is encoded as an unsigned integer with a bias. The bias rotates the number ring such that the value zero no longer corresponds with the bitpattern all 0s. Usually this is a bad thing, but here, it's what we want.



The bias here would be negative 3 (or 3), depending on which way you are going.

Warm Up

Turn these decimal numbers into binary:
22, 1.5, 5/64, 22/32

Now normalize them.

Now make them into (single precision) floats.

Warm Up

Turn these decimal numbers into binary:
22, 1.5, 5/64, 22/32
10110, 1.1, 0.00101, 0.10110
Now normalize them.
$1.011x^4$, $1.1x2^0$, $1.01x2^{-3}$, $1.011x2^{-1}$
Now make them into (single precision) floats.
[0|4+127|0b0110….0]
[0|0+127|0b10…0]
[0|-3+127|0b010…0]
[0|-1+127|0b0110…0]

Question:
Why bother with a bias? Can't we just use a Two's comp. exponent representation?

Related questions: Talk to your neighbor about these!
Which of the following two (single precision) floats is bigger?
0x7f00 0000 or 0x0080 0000

Which of the following two integers is bigger?
0x7f00 0000 or 0x0080 0000

Now assume we used a two's complement exponent instead, which of the two floats is bigger?
0x7f00 0000 or 0x0080 0000

What would zero encode as with a two's complement exponent?

Question:
Why bother with a bias? Can't we just use a Two's comp. exponent representation?

Sure, it works. But …
Biased exponent => existing integer hardware comparators still work!

Zero = 0x4000 0000 => kind of weird.

Most negative exponent = 0b1000 0000

Question:
Why is the bias $2^{(E-1)} - 1$ (0 and all 1s)?

Related questions:
What fraction (roughly) of the values positive floating point numbers represent are in the range [0,1)?

What about the range [1, infinity)?

Suppose we change the bias to 0, how would the answers above change?

How about using 2^E-1 as the bias?

What choice of bias would split the represented values such that half are in [0, 4), and half are in [4, infinity)?

Question:
Why is the bias $2^{(E-1)} - 1$ (0 and all 1s)?

It's a design choice.

$2^{(E-1)} - 1$ splits the representation about 1.0
Half the positive floats are < 1, Half are > 1

Question:
Why is the implicit denorm exponent (1-Bias)?

Related questions:
What is the smallest non-zero denorm?
What is the second smallest, third?
What is the step size for denormalized numbers?
How many positive denorms are there?
What is the value of the largest denorm?
How does this value relate to step size and # denorms?

What is the value of the smallest normalized float?
What is the step size b/w this smallest normal and its greater neighbor?
How far apart are the smallest normal and the largest denorm?

Suppose the denorm exponent were (0-Bias), as the normal pattern suggests, what would the step size be?
Considering the number of steps and the step size, what would the largest denorm's value be?

Question:
Why is the implicit denorm exponent (1-Bias)?

Related questions:
What is the smallest non-zero denorm? $(-1)^0(2^{-23})2^{-126} = 2^{-149}$
What is the second smallest, third? $2^{-148} = 2(2^{-149})$; $2^{-148} + 2^{-149} = 3(2^{-149})$
What is the step size for denormalized numbers? 2^{-149}
How many positive denorms are there? 23 significand bits -> 2^{23} denorms
What is the value of the largest denorm? $(2^{23}-1)(2^{-149}) = 2^{-126} - 2^{-149}$
How does this value relate to step size and # denorms? Stepsize*#denorms

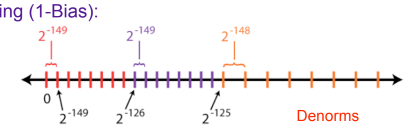
What is the value of the smallest normalized float? $(-1)^0(1 + 0)2^{-126} = 2^{-126}$
What is the step size b/w this smallest normal and its greater neighbor? 2^{-149}
How far apart are the smallest normal and the largest denorm? 2^{-149}

Suppose the denorm exponent were (0-Bias), as the normal pattern suggests, what would the step size be? 2^{-150}
Considering the number of steps and the step size, what would the largest denorm's value be? $(2^{23}-1)(2^{-150}) = 2^{-127} - 2^{-150}$

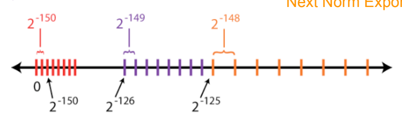
Question:
Why is the implicit denorm exponent (1-Bias)?

Implicit Exponent = (0-Bias) => Gaps in Representation

Using (1-Bias):



Using (0-Bias):



Denorms
Smallest Norm Exponent
Next Norm Exponent

Question:
Why is $2^{24} + 1.0 = 2^{24}$, but $(2^{24} + 2^1) + 1.0 = (2^{24} + 2^2)$?

Related questions:
Why are there floats X such that X+1.0 = X?
What is the smallest such number? (Hint: Think about lab 6.)

What do the bottom-most bits of 2^{24}'s significand look like?
What about $(2^{24} + 2^1)$'s significand?

What are the four rounding modes that floats use?
How would they round the following binary numbers to the nearest integer? 00.00, 00.01, 00.10, 00.11, 01.00, 01.01, 01.10, 01.11

Which patterns do the lower bits of the significands of 2^{24} and $(2^{24} + 2^1)$ match with? What about after you add 1.0?

What rounding modes could make the stated question possible?

Question:
Why is $2^{24} + 1.0 = 2^{24}$, but $(2^{24} + 2^1) + 1.0 = (2^{24} + 2^2)$?

Only 23 significand bits means 1.0 is just barely too small relative to 2^{24}'s implicit 1 to be saved.

Floating point unit has 2 guard bits used for intermediate computation.
The bits of the first computation look like this:
[1][00…000][10]
The bits of the second computation look like this:
[1][00…001][10]

Need to round to fit the guard bits in the significand.
Default (aka unbiased or round to even) rounding mode would round to [1][00…000] and [1][00…010] respectively.

Round towards +infinity would do the same.

MIPS, C, and You

Ropes

```
struct ropeNode {
   char * string; // 0x0 offset
   struct ropeNode * next; // 0x4 offset
}
typedef struct ropeNode* rope;

rope weave(char * str, rope r){
   // append str to the front of r (copy str)
}
void fray(rope r){
   // free all memory associated with r
}
```

Weave, C->MIPS

```
struct ropeNode {
   char * string; // 0x0 offset
   struct ropeNode * next; // 0x4 offset
}
rope weave(char * str, rope r){
   rope end = (rope) malloc (sizeof(struct ropeNode));
   end->string = (char *) malloc ((strlen(str) + 1)*sizeof(char));
   strcpy(end->string, str);
   end->next = r;
   return end;
}

# $s0 = str, $s1 = r, $s2 = end
weave:
   # FILL ME IN!
```

Weave, in MIPS

```
weave:
   #prologue
   addiu $sp, $sp, -16
   sw $ra, 0($sp)
   sw $s0, 4($sp)
   sw $s1, 8($sp)
   sw $s2, 12($sp)
   #body
   move $s0, $a0
   move $s1, $a1
   li $a0, 8
   jal malloc
   move $s2, $v0
   move $a0, $s0
   jal strlen
   addiu $a0, $v0, 1
   jal malloc
   sw $v0, 0($s2)
   move $a0, $v0
   move $a1, $s0
   jal strcpy
   sw $s1, 4($s2)
   move $v0, $s2
   #epilogue
   lw $ra, 0($sp)
   lw $s0, 4($sp)
   lw $s1, 8($sp)
   lw $s2, 12($sp)
   addiu $sp, $sp, 16
   jr $ra
```

Fray, C->MIPS

```
struct ropeNode {
      char * string; // 0x0 offset
      struct ropeNode * next; // 0x4 offset
}

void fray(rope r){
      if (r->next != NULL)
            fray(r->next);
      free(r->string);
      free(r);
}

fray:
      #fill me in!
```

Fray, in MIPS

```
fray:
   addiu $sp, $sp, -8
   sw $ra, 0($sp)
   sw $s0, 4($sp)

   move $s0, $a0
   lw $t0, 4($s0)
   beq $t0, $zero, done
   move $a0, $t0
   jal fray
done:  lw $a0, 0($s0)
       jal free
       move $a0, $s0
       jal free

       lw $ra, 0($sp)
       lw $s0, 4($sp)
       addiu $sp, $sp, 8
       jr $ra
```