CS 61C Spring 2010 — Floats and Other Details

Scott Beamer (cs61c-ta)

Week 7 (3/2)

Float Introduction

- Floats were made to increase the range of values to include very small and large reals
- This added range comes at the cost of less precision
  - Remember: N bits represents only 2^N things, but no more
- Done with normalized binary numbers (only one nonzero bit left of binary point)
- IEEE 754 standard: $(-1)^{sign} \times 1.significand \times 2^{(exponent-127)}$

sign	exponent	significand
1 bit	8 bits	23 bits

- Notice: Implicit *1* to the left of binary point, significand is only to the right
- Exponent uses biased notation (Range of [-127, +128] shifted to [0, 255]). Why?
- Floats also use Sign & Magnitude. Why is this ok for floats?

Special Cases for Floats

Exponent	Significand	Meaning
0	0	zero
0	non-zero	denormalized
1-254 (MAX-1)	anything	float
255 (MAX)	0	+/- infinity
255 (MAX)	non-zero	NaN (Not a Number)

Denormalized Numbers

- Were made to fill in between 2^{-126} and 0
- *Denormalized* comes from number not being normalized (there is no longer a 1 before binary point)
- Has an implicit exponent of -126 and no longer has an implicit 1 in mantissa
- Thus they take the form: $(-1)^{sign} \times 0.significand \times 2^{-126}$

Doubles

- Were made to increase the precision and the range of floats
- Same format as floats just with more bits and and a MAX of 2047
- Double format: $(-1)^{sign} \times 1.significand \times 2^{(exponent-1023)}$

sign	exponent	significand
1 bit	11 bits	52 bits

Floating Point Questions

- What is the largest float < infinity? $1.11111111111111111111111 \times 2^{127}$ 0x7f7fffff
- What is the smallest positive float? $0.00000000000000000000001 \times 2^{-126}$ 0x00000001
- Say you wanted to make the float in $t0 8 times bigger, and all you had available were add, addi, and sll. How would you do it? (Assuming the float < 2^{124})

```
addi $t1, $0, 3
sll $t1, $t1, 23
add $t0, $t0, $t1
```

MAL vs. TAL

- *TAL (True Assembly Language)*: MIPS with no pseudo-instructions and strict enforcement of the ISA
  - TAL has a direct 1:1 mapping with raw bits, where each TAL instruction corresponds to exactly 1 instruction the CPU will execute
- *MAL (MIPS Assembly Language)*: MIPS where pseudo-instructions are allowed
  - An instruction with improper arguments is also a pseudo-instruction (add $t0, $t0, 2)
  - Some MAL instructions correspond to multiple instructions when assembled
  - Done to increase programmer (or compiler) productivity

Assembling Exercise

- Be the first pass of the assembler and convert the following MAL instructions to TAL:

MAL	TAL
li $s0, 0xdeadbeef	lui $s0, 0xdead ori $s0, $s0, 0xbeef
add $t2, $t3, 0xcafebebe	lui $at, 0xcafe ori $at, $at, 0xbebe add $t2, $t3, $at
bge $s2, -3, exit (exit is PC+8)	slti $at, $s2, -3 beq $at, $0, 1
swap $t0, $t1	add $at, $0, $t1 xor $t0, $t0, $t1 add $t1, $0, $t0 *or* xor $t1, $t0, $t1 add $t0, $0, $at xor $t0, $t0, $t1
lw $t0, $t1($t2)	add $at, $t1, $t2 lw $t0, 0($at)

- How would you implement the *inc* instruction? (inc $rt,imm → R[rt] = R[rt] + imm)

add $rt, $rt, imm

- How about the *freeze* instruction? Once the processor executes freeze, it will never execute anything else.

beq $0, $0, -1