CS61C Summer 2025

Floating Point, RISC-V

Discussion 3

1 Floating Point

1.1 Convert the following single-precision floating point numbers from hexadecimal to decimal or from decimal to hexadecimal using the IEEE 754 Floating Point Standard. You may leave your answer as an expression.

a) `8.25` `0x41040000`

b) `39.5625` `0x421E4000`

c) `0x00000F00` `0.000457763671875`

d) `0` `0x00000000, 0x80000000`

e) `0xFF94BEEF` NaN

f) ∞ `0x7F800000`

g) `1/3` `0x3EAAAAAB`

2 More Floating Point

As we saw above, not every number can be represented perfectly using floating point. For this question, we will only look at positive numbers.

2.1 What is the next smallest number larger than 2 that can be represented completely?
0x40000001

2.2 What is the next smallest number larger than 4 that can be represented completely?
0x40800001

2.3 What is the largest odd number that we can represent? Hint: at what power can we only represent even numbers?
2^24 - 1,当阶数足够大时小数点会越过尾数的表示区间，尾数只有23位，当阶数超过23时每下一个数都是偶数，故最大奇数为2^24 - 1

3 RISC-V Instructions

3.1 Assume we have an array in memory that contains `int *arr = {1,2,3,4,5,6,0}`. Let register `s0` hold the address of the element at index 0 in `arr`. You may assume integers are four bytes and our values are word-aligned. What do the following snippets of RISC-V code do? Assume that all the instructions are run one after the other in the same context.

a) `lw t0, 12(s0)` 将arr[3]加载到t0中

b) `sw t0, 16(s0)` 将t0的值存到a[4]

c)
```
slli t1, t0, 2
add t2, s0, t1
 lw t3, 0(t2)
addi t3, t3, 1
sw t3, 0(t2)
```
t0的值左移两位传到t1
将t1+s0传到t2
读取t2+0位置的值到t3
t3 = t3 + 1
将t3存到t2+0处

d)
```
lw t0, 0(s0)
xori t0, t0, 0xFFF
addi t0, t0, 1
```
加载s0+0处的值到t0
t0 = t0 xor 0xFFF
t0 = t0 + 1

4 RISC-V Memory Access

Using the given instructions and the sample memory array, what will happen when the RISC-V code is executed? For load instructions (**`lw`**, **`lb`**, **`lh`**), write out what each register will store. For store instructions (**`sw`**, **`sh`**, **`sb`**), update the memory array accordingly. Recall that RISC-V is little-endian and byte addressable. For any unknown instructions, use the CS 61C reference card!

4.1

```
li t0 0x00FF0000
lw t1 0(t0)
addi t0 t0 4
lh t2 2(t0)
lw s0 0(t1)
lb s1 3(t2)
```

0xFFFFFFFF	
	...
0x00FF0004	0x000C561C
0x00FF0000	36
	...
0x00000036	0xFDFDFDFD
	...
0x00000024	0xDEADB33F
	...
0x0000000C	0xC5161C00
	...
0x00000000	

What value does each register hold after the code is executed?

`t0`: `0x00FF0004`

`t1`: 36

`t2`: `0x0000000C`

`s0`: `0xDEADB33F`

`s1`: `0xFFFFFFC5`

4.2 Update the memory array with its new values after the code is executed. Assume each byte in the memory array is initialized to zero.

```
li t0 0xABADCAF8
li t1 0xF9120504
li t2 0xBEEFDAB0
sw t0 0(t1)
addi t0 t0 4
sh t1 2(t0)
sh t2 0(t0)
lw t3 0(t1)
sb t1 1(t3)
sb t2 3(t3)
```

0xFFFFFFFF	0x00000000
	...
0xF9120504	0xABADCAF8
	...
0xBEEFDAB0	0x00000000
	...
0xABADCAFC	0x0504DAB0
0xABADCAF8	0xB0000400
	...
0x00000000	0x00000000

t0: 0xABADCAFC
t1: 0xF9120504
t2: 0xBEEFDAB0
t3: 0xABADCAF8

5 Lost in Translation

5.1 Translate the code verbatim between C and RISC-V. The comments above the code indicate which registers to store the variables.

C	RISC-V

`// s0 -> a`
`// s1 -> b`
`// s2 -> c`
`// s3 -> z`
`int a = 4, b = 5, c = 6;`
`int z = a + b + c + 10;`
	`addi s0 x0 4`
	`addi s1 x0 5`
	`addi s2 x0 6`
	`add s3 s0 s1`
	`add s3 s3 s2`
	`addi s3 s3 10`

`// int *p = intArr;`
`// s0 -> p;`
`// s1 -> a;`
`*p = 0;`
`int a = 2;`
`p[1] = p[a] = a;`
	`sw x0 0(s0)`
	`addi s1 x0 2`
	`slli t0 s1 2`
	`add t0 t0 s0`
	`sw s1 0(t0)`
	`sw s1 4(s0)`

`// s0 -> a,`
`// s1 -> b`
`int a = 5;`
`int b = 10;`
`if (a + a == b) {`
`  a = 0;`
`} else {`
`  b = a - 1;`
`}`
	`addi s0 x0 5`
	`addi s1 x0 10`
	`add t0 s0 s0`
	`beq t0 s1 br`
	`addi s1 s0 -1`
	`jal x0 exit`
	`br:`
	`addi s0 x0 0`
	`exit:`

`// Compute s1 = 2^30`
`int s0 = 0;`
`int s1 = 1;`
`for (; s0 != 30; s0 += 1) {`
`  s1 *= 2;`
`}`
	`addi s0 x0 0`
	`addi s1 x0 1`
	`addi t0 x0 30`
	`loop:`
	`beq s0 t0 exit`
	`slli s1 s1 1`
	`addi s0 x0 1`
	`jal x0 loop`
	`exit:`

C	**RISC-V**

`// s0 -> n`
`// s1 -> sum`
`for (int sum = 0; n > 0; n--)`
`{`
`  sum += n;`
`}`
	`addi s1 x0 0`
	`loop:`
	`bge x0 s0 exit`
	`add s1 s1 s0`
	`addi s0 x0 -1`
	`jal x0 loop`
	`exit:`