CS 61C Summer 2022 — RISC-V Discussion 4

1 Precheck

1.1 After calling a function and having that function return, the `t` registers may have been changed during the execution of the function, while `a` registers cannot.

1.2 Let `a0` point to the start of an array `x`. `lw s0, 4(a0)` will always load `x[1]` into `s0`.

1.3 Assuming no compiler or operating system protections, it is possible to have the code jump to data stored at `0(a0)` (offset 0 from the value in register `a0`) and execute instructions from there.

1.4 Assuming integers are 4 bytes, adding the ASCII character `'d'` to the address of an integer array would get you the element at index 25 of that array (assuming the array is large enough).

1.5 `jalr` is a shorthand expression for a `jal` that jumps to the specified label and does not store a return address anywhere.

1.6 Calling `j label` does the exact same thing as calling `jal label`.

2 Translation

RISC-V is an assembly language, which is comprised of simple instructions that each do a single task such as addition or storing a chunk of data to memory.

For example, on the left is a line of C code and on the right is a chunk of RISC-V code that accomplishes the same thing.

```
int x = 5;
y[2];
y[0] = x;
y[1] = x * x;
```

```
// x -> s0, &y -> s1
addi s0, x0, 5
sw   s0, 0(s1)
mul  t0, s0, s0
sw   t0, 4(s1)
```

2.1 Translate between the C and RISC-V verbatim.

C	RISC-V

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

`// s0 -> int * p = intArr;`
`// s1 -> a;`
`*p = 0;`
`int a = 2;`
`p[1] = p[a] = a;`	

`// s0 -> a, s1 -> b`
`int a = 5, b = 10;`
`if(a + a == b) {`
`    a = 0;`
`} else {`
`    b = a - 1;`
`}`	

	`addi s0, x0, 0`
	`addi s1, x0, 1`
	`addi t0, x0, 30`
	`loop:`
	`beq s0, t0, exit`
	`add s1, s1, s1`
	`addi s0, s0, 1`
	`jal x0, loop`
	`exit:`

`// s0 -> n, s1 -> sum`
`// assume n > 0 to start`
`for(int sum = 0; n > 0; n--) {`
`sum += n;`
`}`	

3 Q3

In RISC-V, we have two methods of storing data: main memory and registers. Registers are much faster than using main memory, but are very limited in space (32 bits each). You should ALWAYS use the names of registers, e.g. `s0` rather than `x8`; the one exception to this rule is the zero register `x0`, as it is often shorter to write `x0` than its name `zero`, and the purpose of the register is still easy to tell with either identifier. The below table of register names is reproduced from the RISC-V green card.

Register(s)	Alt.	Description
x0	zero	The zero register, always zero
x1	ra	The return address register, stores where functions should return
x2	sp	The stack pointer, where the stack ends
x5-x7, x28-x31	t0-t6	The temporary registers
x8-x9, x18-x27	s0-s11	The saved registers
x10-x17	a0-a7	The argument registers, a0-a1 are also return value

3.1 Can you convert each instruction's registers to the other form?

```
add s0, zero, a1     -->
or  x18, x1, x30     -->
```

4 Q4

4.1 What is the range of 32-bit instructions that can be reached from the current PC using a branch instruction?

4.2 What is the maximum range of 32-bit instructions that can be reached from the current PC using a jump instruction?

4.3 Given the following RISC-V code (and instruction addresses), fill in the blank fields for the following instructions (you'll need your RISC-V green card!).

```
0x002cff00: loop: add t1, t2, t0          |_______|_______|_______|_______|_______|__0x33__|
0x002cff04:       jal ra, foo             |_____________________________|_____________|__0x6F__|
0x002cff08:       bne t1, zero, loop      |_______|_______|_______|_______|_______|__0x63__|
...
0x002cff2c: foo:  jr ra                 ra = _____________________________
```

4.1 What is the range of 32-bit instructions that can be reached from the current PC using a branch instruction?

4.2 What is the maximum range of 32-bit instructions that can be reached from the current PC using a jump instruction?

4.3 Given the following RISC-V code (and instruction addresses), fill in the blank fields for the following instructions (you'll need your RISC-V green card!).

```
0x002cff00: loop: add t1, t2, t0          |_______|_______|_______|_______|_______|__0x33__|
0x002cff04:       jal ra, foo             |_____________________________|_____________|__0x6F__|
0x002cff08:       bne t1, zero, loop      |_______|_______|_______|_______|_______|__0x63__|
...
0x002cff2c: foo:  jr ra                 ra = _____________________________
```

5 Q5

5.1 Write a function `sumSquare` in RISC-V that, when given an integer `n`, returns the summation below. If `n` is not positive, then the function returns 0.

$$n^2 + (n-1)^2 + (n-2)^2 + \ldots + 1^2$$

For this problem, you are given a RISC-V function called `square` that takes in a single integer and returns its square.

First, let's implement the meat of the function: the squaring and summing. We will be abiding by the caller/callee convention, so in what register can we expect the parameter `n`? What registers should hold `square`'s parameter and return value? In what register should we place the return value of `sumSquare`?

5.2 Since `sumSquare` is the callee, we need to ensure that it is not overriding any registers that the caller may use. Given your implementation above, write a prologue and epilogue to account for the registers you used.