CS 61C Fall 2018

RISC-V Control Flow

Discussion 4: September 17, 2018

1 RISC-V with Arrays and Lists

Comment each snippet with what the snippet does. Assume that there is an array, `int arr[6] = {3, 1, 4, 1, 5, 9}`, which is starts at memory address `0xBFFFFF00`, and a linked list struct (as defined below), `struct ll* lst;`, whose first element is located at address `0xABCD0000`. `s0` then contains `arr`'s address, `0xBFFFFF00`, and `s1` contains `lst`'s address, `0xABCD0000`. You may assume integers and pointers are 4 bytes and that structs are tightly packed.

```
struct ll {
    int val;
    struct ll* next;
}
```

1.1
```
lw  t0, 0(s0)
lw  t1, 8(s0)
add t2, t0, t1
sw  t2, 4(s0)
```

1.2
```
loop: beq  s1, x0, end
      lw   t0, 0(s1)
      addi t0, t0, 1
      sw   t0, 0(s1)
      lw   s1, 4(s1)
      jal  x0, loop
 end:
```

1.3
```
       add  t0, x0, x0
loop:  slti t1, t0, 6
       beq  t1, x0, end
       slli t2, t0, 2
       add  t3, s0, t2
       lw   t4, 0(t3)
       sub  t4, x0, t4
       sw   t4, 0(t3)
       addi t0, t0, 1
       jal  x0, loop
 end:
```

2 RISC-V Calling Conventions

2.1 How do we pass arguments into functions?

2.2 How are values returned by functions?

2.3 What is `sp` and how should it be used in the context of RISC-V functions?

2.4 Which values need to saved by the caller, before jumping to a function using `jal`?

2.5 Which values need to be restored by the callee, before using `jalr` to return from a function?

3 Writing RISC-V Functions

3.1 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 an integer and returns its square. Implement `sumSquare` using `square` as a subroutine.

4 More Translating between C and RISC-V

4.1 Translate between the C and RISC-V code. You may want to use the RISC-V Green Card as a reference. We show you how the different variables map to registers – you don't have to worry about the stack or any memory-related issues.

C	RISC-V

`// Nth_Fibonacci(n):`
`// s0 -> n, s1 -> fib`
`// t0 -> i, t1 -> j`
`// Assume fib, i, j init'd to:`
`int fib = 1, i = 1, j = 1;`
`if (n==0)`
`  return 0;`
`else if (n==1)`
`  return 1;`
`n -= 2;`
`while (n != 0) {`
`  fib = i + j;`
`  j = i;`
`  i = fib;`
`  n--;`
`}`
`return fib;`	