CS 61C Spring 2019

RISC-V Control Flow

Discussion 3: February 13, 2019

1 RISC-V with Arrays and Lists

Comment what each code block does. Each block runs in isolation. Assume that there is an array, `int arr[6] = {3, 1, 4, 1, 5, 9}`, which starts at memory address `0xBFFFFF00`, and a linked list struct (as defined below), `struct ll* lst`, whose first element is located at address `0xABCD0000`. Let `s0` contain `arr`'s address `0xBFFFFF00`, and let `s1` contain `lst`'s address `0xABCD0000`. You may assume integers and pointers are 4 bytes and that structs are tightly packed. Assume that `lst`'s last node's `next` is a NULL pointer to memory address `0x00000000`.

```
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)
```

Sets `arr[1]` to `arr[0]` + `arr[2]`

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:
```

Increments all values in the linked list by 1.

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:
```

Negates all elements in `arr`

2 RISC-V Calling Conventions

2.1 How do we pass arguments into functions?

Use the 8 arguments registers `a0` - `a7`

2.2 How are values returned by functions?

Use `a0` and `a1` as the return value registers as well

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

`sp` stands for stack pointer. We subtract from `sp` to create more space and add to free space. The stack is mainly used to save (and later restore) the value of registers that may be overwritten.

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

Registers `a0` - `a7`, `t0` - `t6`, and `ra`

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

Registers `sp`, `gp` (global pointer), `tp` (thread pointer), and `s0` - `s11`. Important to note that we don't really touch `gp` and `tp`

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 a single integer and returns its square. Implement `sumSquare` using `square` as a subroutine. Be sure to follow RISC-V caller/callee convention. (Hints: for `sumSquare`, 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`? What needs to go in `sumSquare`'s prologue and epilogue?)

```
sumSquare: addi sp, sp -12   # Make space for 3 words on the stack
           sw   ra, 0(sp)    # Store the return address
           sw   s0, 4(sp)    # Store register s0
           sw   s1, 8(sp)    # Store register s1
           add  s0, a0, x0   # Set s0 equal to the parameter n
           add  s1, x0, x0   # Set s1 (accumulator) equal to 0
     loop: bge  x0, s0, end  # Branch if s0 is not positive
           add  a0, s0, x0   # Set a0 to the value in s0, setting up
                             # args for call to function square
           jal  ra, square   # Call the function square
           add  s1, s1, a0   # Add the returned value into s1
           addi s0, s0, -1   # Decrement s0 by 1
           jal  x0, loop     # Jump back to the loop label
      end: add  a0, s1, x0   # Set a0 to s1 (desired return value)
           lw   ra, 0(sp)    # Restore ra
           lw   s0, 4(sp)    # Restore s0
           lw   s1, 8(sp)    # Restore s1
           addi sp, sp, 12   # Free space on the stack for the 3 words
           jr   ra           # Return to the caller
```

4 More Translating between C and RISC-V

4.1 Translate between the RISC-V code to C. You may want to use the RISC-V Green Card on the next page as a reference. What is this RISC-V function computing? Assume no stack or memory-related issues, and assume no negative inputs.

C	RISC-V

`// a0 -> x, a1 -> y,`
`// t0 -> result`
`// Function computes pow(x,y)`
`// Direct translation:`
`int power(int x, int y) {`
`  int result = 1;`
`  while (y & y != 0) {`
`  result *= x;`
`  y--;`
`  }`
`  return result;`
`}`
`// Note the loop condition could`
`// be simplified.`
	`Func: addi t0 x0 1`
	`Loop: and t1 a1 a1`
	`beq t1 x0 Done`
	`mul t0 t0 a0`
	`addi a1 a1 -1`
	`jal x0 Loop`
	`Done: add a0 t0 x0`
	`jr ra`