Jon Dis 4 Notes

Monday, September 17, 2018 12:05 AM

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)
```

MEM(s0+0) → t0 arr[0] → t0
MEM(s0+8) → t1 arr[2] → t1
t0+t1 → t2
t2 → MEM(s0+4) t2 → arr[1]

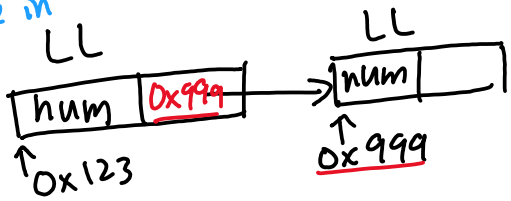
arr[1] = arr[0] + arr[2]

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:
```

s1 = Linked List

s1=0 → end
M(s1+0) → t0
t0+1 → t0
t0 → M(s1+0)
M(s1+4) → s1 (load next node in)
jump

incr all elems of the list, ending w/ null pointer



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:
```

0 → t0
t0 < 6?
no → end
Calc offset, t0 elms dow
offset + base
elem → t4
t4 = 0 - t4
put back
t0 = t0 + 1

first 6 elems negated, arr has 6 elems
→ negates all of arr's elems

2 RISC-V Calling Conventions

2.1 How do we pass arguments into functions?

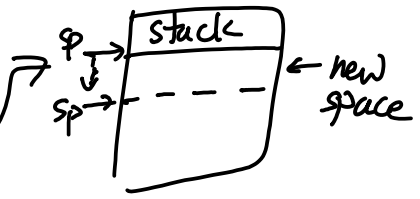
a0-a7 ← look here in func

2.2 How are values returned by functions?

a0, a1 registers ← look here for rtn

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

sp = stack pointer, store vars to save them: decrement sp → add to space created



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

a0-a7, t0-t6, ra — will change w/ jal

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

sp, gp (global pt), tp (thread pt), s0-s11

not important

saved them from your func

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.

a0 argument
a0 return

```
sumSquare: addi sp, sp, -12
           sw   ra, 0(sp)
           sw   s0, 4(sp)
           sw   s1, 8(sp)
           add  s0, a0, x0      # let s0 be n
           add  s1, x0, x0      # s1 be tot sum
loop:      bge  x0, s0, end     # input of square
           add  a0, s0, x0
           jal  ra, square      # output of square
           add  s1, s1, a0
           addi s0, s0, -1
           jal  x0, loop
end:       add  a0, s1, x0      # sumSquare output
           lw   ra, 0(sp)
           lw   s0, 4(sp)
           lw   s1, 8(sp)
           addi sp, sp, 12
           jr   ra
```

prologue

sumSquare input

epilogue

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;`
	beq s0,x0, Ret0
	addi t2, x0, 1
	beq s0,t2, Ret1
	addi s0, s0, -2
	Loop: beq s0, x0, RetF
	add s1, t0, t1
	addi t1, t0, 0
	addi t0, s1, 0
	addi s0, s0, -1
	jal x0, Loop
	Ret0: addi a0, x0, 0
	jal x0, Done
	Ret1: addi a0, x0, 1
	jal x0, Done
	RetF: add a0, x0, s1
	Done: ~

prologue
└ what to save?
s0, s1

epilogue