National Yang Ming Chiao Tung University
Computer Architecture & System Lab

Lecture 4-1: Fibonacci Numbers in RISC-V

CS10014 Computer Organization

Department of Computer Science
Tsung Tai Yeh
Thursday: 1:20 pm– 3:10 pm
Classroom: EC-022

Acknowledgements and Disclaimer

- Slides were developed in the reference with
  - CS 61C at UC Berkeley
    - https://inst.eecs.berkeley.edu/~cs61c/sp23/
  - CS 252 at UC Berkeley
    - https://people.eecs.berkeley.edu/~culler/courses/cs252-s05/
  - CSCE 513 at University of South Carolina
    - https://passlab.github.io/CSCE513/

Outline

- Translate Fibonacci Number in RISC-V

Steps for Making a Procedure Call

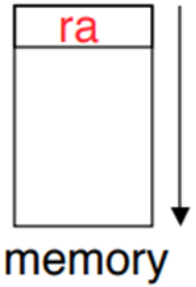
(1) Save necessary values onto the stack
(2) Assign argument(s), if any
(3) jal call
(4) Restore values from stack

Basic Structure of a Function

Prologue

```
entry_label:
addi $sp,$sp, -framesize
sw $ra, framesize-4($sp)   # save $ra
save other regs if need be
```

Body ··· **(call other functions…)**



Epilogue

```
restore other regs if need be
lw $ra, framesize-4($sp)   # restore $ra
addi $sp,$sp, framesize
jr $ra
```

Fibonacci Numbers (1/7)

- The Fibonacci numbers are defined as follows
  - F(n) = F(n - 1) + F(n - 2)
  - F(0) and F(1) are defined to be 0 and 1, respectively
- Rewriting this in C, we have:

```
int fib(int n) {
  if(n == 0) { return 0; }
  if(n == 1) { return 1; }
  return (fib(n - 1) + fib(n - 2));

}
```

Fibonacci Numbers (2/7)

- Now, let's translate this to RISC-V
- You will need space for three words on the stack
- The function will use one $s register, $s0
- Write the Prologue

```
fib:
addi $sp, $sp, -12      # Space for three words
sw $ra, 8($sp)          # Save the return address
sw $s0, 4($sp)          # Save $s0
```

Fibonacci Numbers (3/7)

- Now, write the Epilogue

```
fin:
lw $s0, 4($sp)          # Restore $s0
lw $ra, 8($sp)          # Restore return address
addi $sp, $sp, 12       # Pop the stack frame

jr $ra                  # Return to caller
```

Fibonacci Numbers (4/7)

- Finally, write the body
  - Start to translate the C code the lines indicated in the comments

```
int fib (int n) {
  if (n == 0) { return  0; } // Translate Me!
   if (n == 1) { return 1;} // Translate Me!
   ..... }
```

```
addi   $a0,  $zero,  0      # $a0 = 0, a0 is return value
beq    $a2,  $zero, fin     # if (n == 0)
addi   $a0,  $zero,  1      # $a0 = 1
beq    $a2,  $t0,    fin    # if (n == 1)
```

Fibonacci Numbers (5/7)

- Almost there

```
int fib (int n) {
    ……
    return (fib(n - 1) + fib (n - 2));
}
```

```
addi   $a2,  $a2,  -1        # $a2 = n - 1
sw             $a2,  0($sp)  # Need $a2 after jal
jal            fib           # fib(n - 1)
lw             $a2,  0($sp)  # Restore $a0
addi   $a2,  $a2,  -1        # $a2 = n - 2
```

Fibonacci Numbers (6/7)

- Remember the $a0 is caller saved!

```
int fib (int n) {
    ……
    return (fib(n - 1) + fib (n - 2));
}
```

```
add    $s0,  $a0,  $x0      # Place fib(n - 1) somewhere
                            # it won't get clobbered
jal    fib                  # fib (n - 2)
add    $a0, $a0,   $s0      # $a0 = fib(n-1) + fib(n-2)
```

Fibonacci Numbers (7/7)

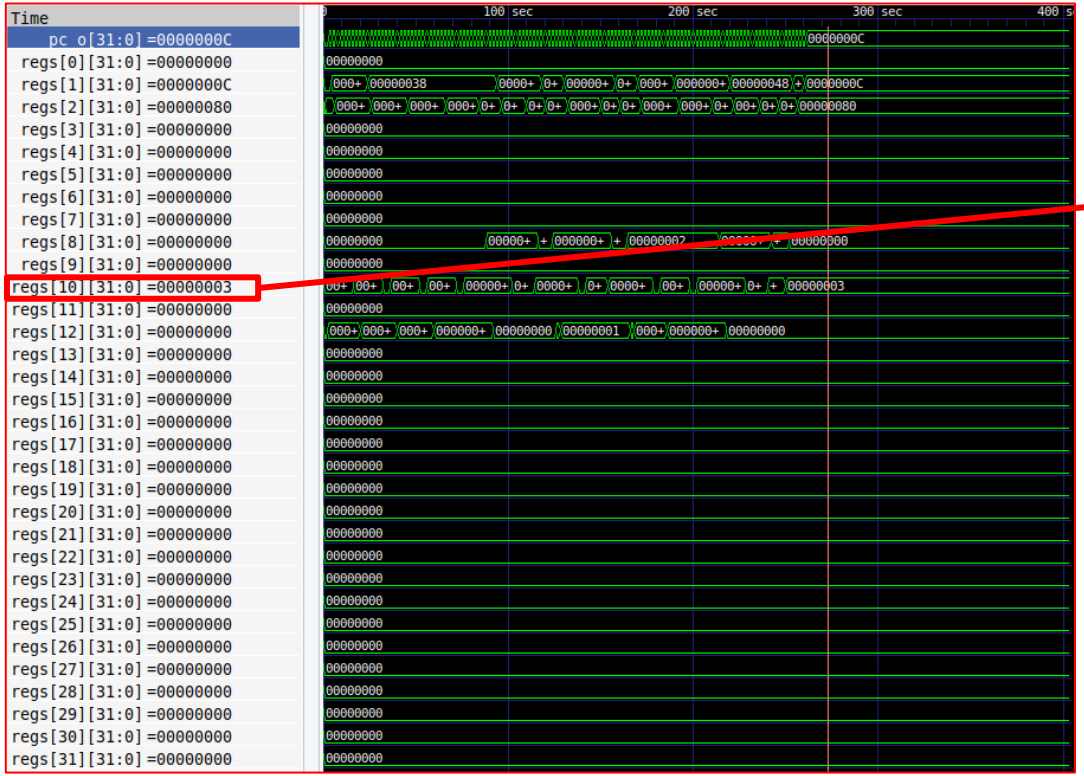
- Here is the complete code for reference

```
entry:
   addi a2, zero, 3
   jal  fib
   ret
fib:
   addi sp, sp, -12
   sw   ra, 8(sp)
   sw   s0, 4(sp)
```

```
fib_if_zero_or_one:
   addi a0, zero, 0
   beq  a2, zero, fib_fin
   addi a0, a0, 1
   beq  a2, a0, fib_fin
fib_call_n_1:
   addi a2, a2, -1
   sw   a2, 0(sp)
   jal  fib
   lw   a2, 0(sp)
   addi a2, a2, -1
```

```
fib_call_n_2:
   add  s0, a0, zero
   jal  fib
   add  a0, a0, s0
fib_fin:
   lw   s0, 4(sp)
   lw   ra, 8(sp)
   addi sp, sp, 12
   jr   ra
```

Simulation result



x0	zero	0x00000000
x1	ra	0x0000000c
x2	sp	0x7ffffff0
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000000
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0	0x00000000
x9	s1	0x00000000
x10	a0	0x00000003
x11	a1	0x00000000
x12	a2	0x00000000
x13	a3	0x00000000
x14	a4	0x00000000
x15	a5	0x00000000
x16	a6	0x00000000
x17	a7	0x00000000
x18	s2	0x00000000
x19	s3	0x00000000
x20	s4	0x00000000
x21	s5	0x00000000
x22	s6	0x00000000
x23	s7	0x00000000
x24	s8	0x00000000
x25	s9	0x00000000
x26	s10	0x00000000