Lab 3

61C Summer 2023

RISC-V

- One step closer to machine.
- Unlike with C, things like stack and registers are explicitly managed by the programmer.

Registers

- Component on the CPU used to store 32 bits of data
  - Does not differentiate between integers, addresses, etc
- RISC-V has 32 registers from x0-x31
- Each register has names and specific functions

Reg	Name	Reg	Name	Reg	Name	Reg	Name
x0	zero	x8	s0	x16	a6	x24	s8
x1	ra	x9	s1	x17	a7	x25	s9
x2	sp	x10	a0	x18	s2	x26	s10
x3	gp	x11	a1	x19	s3	x27	s11
x4	tp	x12	a2	x20	s4	x28	t3
x5	t0	x13	a3	x21	s5	x29	t4
x6	t1	x14	a4	x22	s6	x30	t5
x7	t2	x15	a5	x23	s7	x31	t6

Instruction Types and Syntax

- Arithmetics:
  - Addition/Subtraction: add, sub, addi
  - Bitwise: and, or, xor, andi, ori, xori
  - Shifts: sll (Shift Left Logical), srl (Shift Right Logical), sra (Shift Right Arithmetic), slli, srli, srai
  - Set Less Than: slt, sltu, slti, sltiu

Instruction Types and Syntax

Note: These instruction types are not official and you can find the official types on the reference card

- Arithmetics:
  - Addition/Subtraction: add, sub, addi
  - Bitwise: and, or, xor, andi, ori, xori
  - Shifts: sll (Shift Left Logical), srl (Shift Right Logical), sra (Shift Right Arithmetic), slli, srli, srai
  - Set Less Than: slt, sltu, slti, sltiu

Ex.

```
a = a + b;          ------------>   add t0 t0 t1 #where t0 is a and t1 is b

a = 6;              ------------>   addi t0 x0 6
b = -1;                             addi t1 x0 -1
int c = a ^ b;                      xor t2 t0 t1

a = a * 4           ------------>   slli t0 t0 2
```

Instruction Types and Syntax

- Memory:
  - Loading: lw, lh, lb, lhu, lbu
  - Storing: sw, sh, sb

Instruction Types and Syntax

- Memory:
  - Loading: lw, lh, lb, lhu, lbu
  - Storing: sw, sh, sb

offset from the first byte of array in terms of bytes

Contains address of the start of array

Ex.

```
a = arr[0];
b = arr[1];
```

→

```
lw t0 0(s0)
lw t1 4(s0)
```

```
arr[2] = a + b;
arr[3] = 6;
```

→

```
add t2 t0 t1
sw t2 8(s0)
addi t2 x0 6
sw t2 12(s0)
```

Instruction Types and Syntax

- Controls:
  - Conditional Jumps: beq, bge, bgeu, blt, bltu, bne
  - Unconditional Jumps: jal, jalr, j, jr (j & jr are pseudoinstructions)

Instruction Types and Syntax

- Controls:
  - Conditional Jumps: beq, bge, bgeu, blt, bltu, bne
  - Unconditional Jumps: jal, jalr, j, jr (j & jr are pseudoinstructions)

Ex.

```
if(a>=b)
   a = 1;
else
   a = 2;
```

```
        blt t0 t1 false
        li t0 1
        j exit
false:  li t0 2
exit:
```

Labels are human readable identifiers for lines

Instruction Types and Syntax

- Controls:
  - Conditional Jumps: beq, bge, bgeu, blt, bltu, bne
  - Unconditional Jumps: jal, jalr, j, jr (j & jr are pseudoinstructions)

Ex.

```
if(a>=b)
  a = 1;
else
  a = 2;
```

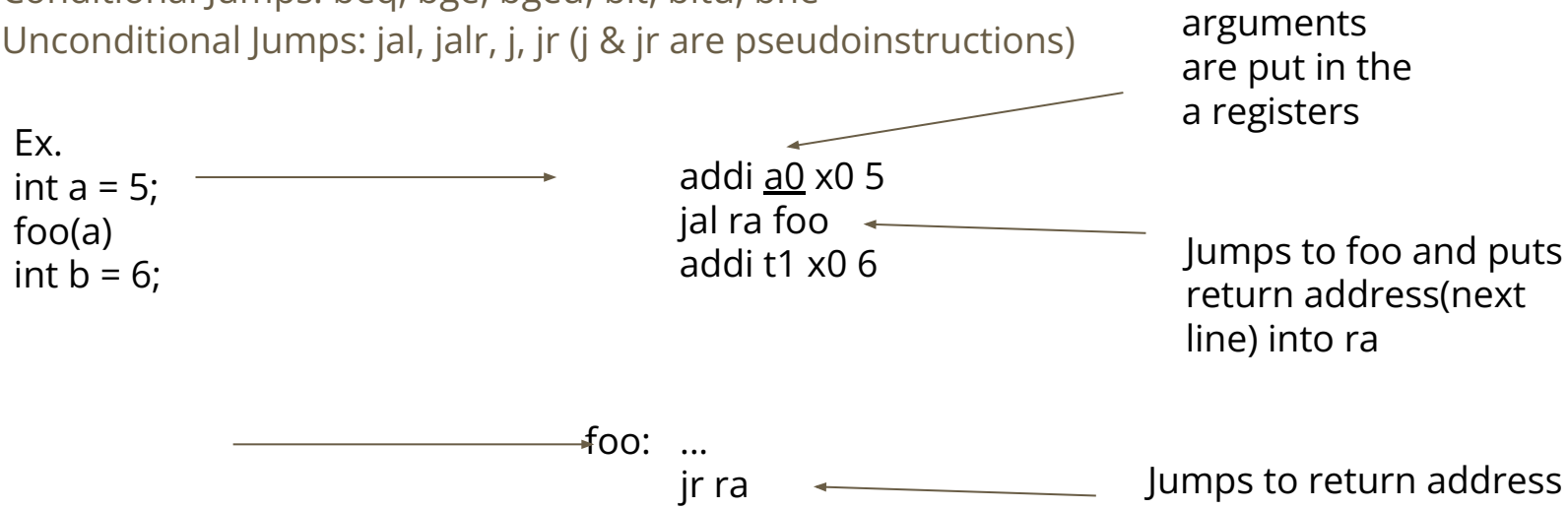
```
        blt t0 t1 false
        li t0 1
        j exit
false:  li t0 2
exit:
```

```
while (a>0)
  a = a - 1;
```

```
        beq t0 x0 exit2
        addi t0 t0 -1
exit2:
```

Instruction Types and Syntax

- Controls:
  - Conditional Jumps: beq, bge, bgeu, blt, bltu, bne
  - Unconditional Jumps: jal, jalr, j, jr (j & jr are pseudoinstructions)



Instruction Types and Syntax

- Look at reference card for specifics
- Miscellaneous:
  - lui, auipc: helpers for pseudoinstructions li and la
  - ebreak: sets break point in venus
- Extensions:
  - mul
- Pseudoinstructions:
  - j, jr, li, la, mv, neg, ret, not, nop, beqz, bnez

C to RISC-V(Fib)

```c
#include <stdio.h>

int n = 12;

// Function to find the nth Fibonacci number
int main(void) {
    int curr_fib = 0, next_fib = 1;
    int new_fib;
    for (int i = n; i > 0; i--) {
        new_fib = curr_fib + next_fib;
        curr_fib = next_fib;
        next_fib = new_fib;
    }
    printf("%d\n", curr_fib);
    return 0;
}
```

C to RISC-V(Fib)

```c
#include <stdio.h>

int n = 12;
```



```
.data
n: .word 12
```

Global variable indicated by .data
.word means size is 1 word

C to RISC-V(Fib)

```c
#include <stdio.h>

int n = 12;

// Function to find the nth Fibonacci number
int main(void) {
```

```
.data
n: .word 12
.text
main:
```

.text indicates the start of our code

C to RISC-V(Fib)

```c
#include <stdio.h>

int n = 12;

// Function to find the nth Fibonacci number
int main(void) {
    int curr_fib = 0, next_fib = 1;
    int new_fib;
```

```
.data
n: .word 12
.text
main:
    add t0, x0, x0 # curr_fib = 0
    addi t1, x0, 1 # next_fib = 1
```

C to RISC-V(Fib)

```c
#include <stdio.h>

int n = 12;

// Function to find the nth Fibonacci number
int main(void) {
    int curr_fib = 0, next_fib = 1;
    int new_fib;
    for (int i = n; i > 0; i--) {
```



```
.data
n: .word 12
.text
main:
    add t0, x0, x0 # curr_fib = 0
    addi t1, x0, 1 # next_fib = 1
    la t3, n
    lw t3, 0(t3)
```

n is not a register so we cannot use n directly in instructions

C to RISC-V(Fib)

```c
#include <stdio.h>

int n = 12;

// Function to find the nth Fibonacci number
int main(void) {
    int curr_fib = 0, next_fib = 1;
    int new_fib;
    for (int i = n; i > 0; i--) {
```

```
.data
n: .word 12
.text
main:
    add t0, x0, x0 # curr_fib = 0
    addi t1, x0, 1 # next_fib = 1
    la t3, n
    lw t3, 0(t3)
fib:
    beq t3, x0, finish # exit loop once
    ...
    ...
    addi t3, t3, -1 # decrement counter
    j fib # loop
finish:
```

C to RISC-V(Fib)

```c
#include <stdio.h>

int n = 12;

// Function to find the nth Fibonacci number
int main(void) {
    int curr_fib = 0, next_fib = 1;
    int new_fib;
    for (int i = n; i > 0; i--) {
        new_fib = curr_fib + next_fib;
        curr_fib = next_fib;
        next_fib = new_fib;
    }
```

```
.data
n: .word 12
.text
main:
    add t0, x0, x0 # curr_fib = 0
    addi t1, x0, 1 # next_fib = 1
    la t3, n
    lw t3, 0(t3)
fib:
    beq t3, x0, finish # exit loop once we have comp
    add t2, t1, t0 # new_fib = curr_fib + next_fib;
    mv t0, t1 # curr_fib = next_fib;
    mv t1, t2 # next_fib = new_fib;
    addi t3, t3, -1 # decrement counter
    j fib # loop
finish:
```

C to RISC-V(Fib)

```c
#include <stdio.h>

int n = 12;

// Function to find the nth Fibonacci number
int main(void) {
    int curr_fib = 0, next_fib = 1;
    int new_fib;
    for (int i = n; i > 0; i--) {
        new_fib = curr_fib + next_fib;
        curr_fib = next_fib;
        next_fib = new_fib;
    }
```

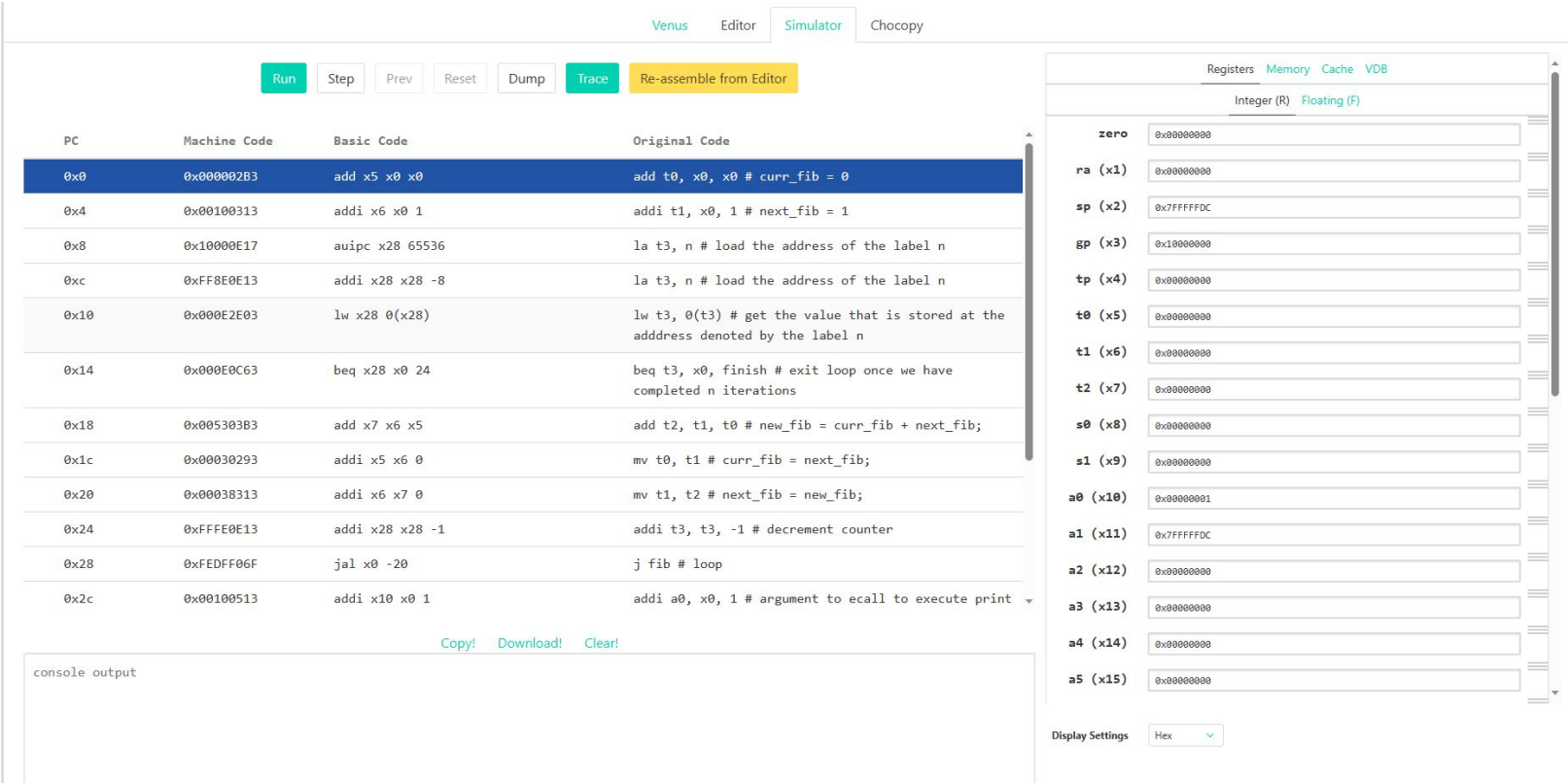
```
.data
n: .word 12

.text
main:
    add t0, x0, x0 # curr_fib = 0
    addi t1, x0, 1 # next_fib = 1
    la t3, n # load the address of the label n
    lw t3, 0(t3) # get the value that is stored at the adddress denote
fib:
    beq t3, x0, finish # exit loop once we have completed n iterations
    add t2, t1, t0 # new_fib = curr_fib + next_fib;
    mv t0, t1 # curr_fib = next_fib;
    mv t1, t2 # next_fib = new_fib;
    addi t3, t3, -1 # decrement counter
    j fib # loop
finish:
    addi a0, x0, 1 # argument to ecall to execute print integer
    addi a1, t0, 0 # argument to ecall, the value to be printed
    ecall # print integer ecall
    addi a0, x0, 10 # argument to ecall to terminate
    ecall # terminate ecall
```

Finally, we use ecall to print and terminate

Venus(debugging)

Check out the Venus reference for more information
cs61c.org/su23/resources/venus-reference/



Venus(debugging)

- Code portion displays program counter, and your code in various forms as you step through the program

PC	Machine Code	Basic Code	Original Code
0x0	0x000002B3	add x5 x0 x0	add t0, x0, x0 # curr_fib = 0
0x4	0x00100313	addi x6 x0 1	addi t1, x0, 1 # next_fib = 1
0x8	0x10000E17	auipc x28 65536	la t3, n # load the address of the label n
0xc	0xFF8E0E13	addi x28 x28 -8	la t3, n # load the address of the label n
0x10	0x000E2E03	lw x28 0(x28)	lw t3, 0(t3) # get the value that is stored at the adddress denoted by the label n
0x14	0x000E0C63	beq x28 x0 24	beq t3, x0, finish # exit loop once we have completed n iterations
0x18	0x005303B3	add x7 x6 x5	add t2, t1, t0 # new_fib = curr_fib + next_fib;
0x1c	0x00030293	addi x5 x6 0	mv t0, t1 # curr_fib = next_fib;
0x20	0x00038313	addi x6 x7 0	mv t1, t2 # next_fib = new_fib;
0x24	0xFFFE0E13	addi x28 x28 -1	addi t3, t3, -1 # decrement counter
0x28	0xFEDFF06F	jal x0 -20	j fib # loop
0x2c	0x00100513	addi x10 x0 1	addi a0, x0, 1 # argument to ecall to execute print

Venus(debugging)

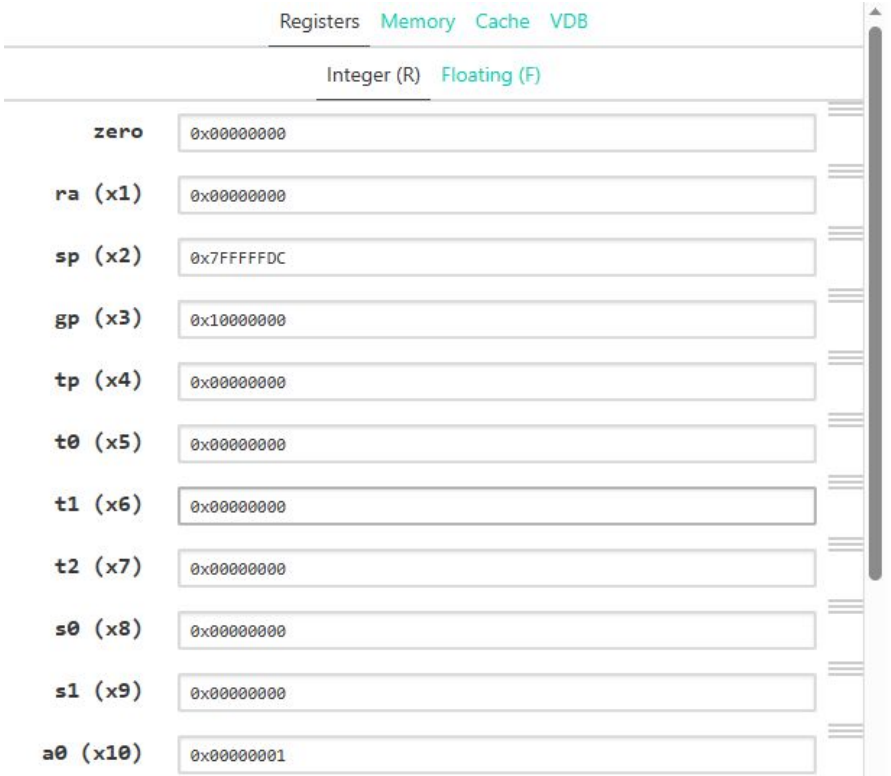
- Set breakpoints by clicking on the row

	PC	Machine Code	Basic Code	Original Code
	0x0	0x000002B3	add x5 x0 x0	add t0, x0, x0 # curr_fib = 0
	0x4	0x00100313	addi x6 x0 1	addi t1, x0, 1 # next_fib = 1
	0x8	0x10000E17	auipc x28 65536	la t3, n # load the address of the label n
	0xc	0xFF8E0E13	addi x28 x28 -8	la t3, n # load the address of the label n
	0x10	0x000E2E03	lw x28 0(x28)	lw t3, 0(t3) # get the value that is stored at the adddress denoted by the label n
●	0x14	0x000E0C63	beq x28 x0 24	beq t3, x0, finish # exit loop once we have completed n iterations
	0x18	0x005303B3	add x7 x6 x5	add t2, t1, t0 # new_fib = curr_fib + next_fib;
	0x1c	0x00030293	addi x5 x6 0	mv t0, t1 # curr_fib = next_fib;
	0x20	0x00038313	addi x6 x7 0	mv t1, t2 # next_fib = new_fib;
	0x24	0xFFFE0E13	addi x28 x28 -1	addi t3, t3, -1 # decrement counter
	0x28	0xFEDFF06F	jal x0 -20	j fib # loop
	0x2c	0x00100513	addi x10 x0 1	addi a0, x0, 1 # argument to ecall to execute

Copy! Download! Clear!

Venus(debugging)

- Register tab allows you to view the contents of all 32 registers as you step through the code



Venus(debugging)

- Memory tab allows you to look at memory content in any location