

CSE 598

Secure Microkernel Design

Introduction to the fundamentals of computer architecture using the RISC-V ISA

Prof. Michel A. Kinsy



A Browser-based RISC-V Simulator

Adaptive and Secure Computing Systems (ASCS)
Laboratory

BRISC-V Simulator

- BRISC-V Simulator let's you:
 - Run RISC-V assembly code in the browser
 - Debug hand-written assembly
 - Run code until completion or until it hits a breakpoint
 - Step through execution, instruction-by-instruction
 - View the state of memory and registers at each step
 - View how each instruction is constructed – opcodes, registers, immediate values etc.

<https://ascslab.org/research/briscv/simulator/simulator.html>

Let's get Familiar with the GUI

BRISC-V Home
BRISC-V Simulator
Manual & Examples



Adaptive and Secure Computing Systems Lab • Boston University



ADAPTIVE & SECURE
COMPUTING SYSTEMS
LABORATORY

C source
⬆ ⚙

```

1 int fib(int n) {
2   if (n <= 1) {
3     return n;
4   } else {
5     return fib(n-1)+fib(n-2);
6   }
7 }
8
9 int return_function (int result) {
10  return result;
11 }
12
13 int main(){
14   int n = 9;
15   int result = return_function (fib(n));
16   return result;
17 }
```

C Source Code Pane

Not interesting for this class

Let's you compile C to RISC-V assembly

RISC-V Assembly
⬆ ▶ ⏮ ⏭

```

0      addi    zero,zero,0
kernel:
1      addi    sp,zero,1536
2      call    main
3      addi    zero,zero,0
4      mv      s1,a0
5      addi    zero,zero,0
6      addi    zero,zero,0
7      auipc   ra,0x0
8      jalr    ra,0(ra)
9      addi    zero,zero,0
10     addi    zero,zero,0
11
12     .file    "gcd.c"
13     .option  nopic
14     .text
15     .align   2
16     .globl   gcd
17     .type    gcd, @function
gcd:
18     addi     sp,sp,-48
19     sw       ra,44(sp)
20     sw       s0,40(sp)
21     addi     s0,sp,48
22     sw       a0,-36(s0)
23     sw       a1,-40(s0)
24     lw       a4,-36(s0)
25     lw       a5,-40(s0)
26     bne      a4,a5,.L2
27     lw       a5,-36(s0)
28     sw       a5,-20(s0)
29     j        .L3
.L2:
30     lw       a4,-36(s0)
31     lw       a5,-40(s0)
```

**RISC-V
Assembly
Pane**

Let's you run
assembly step-
by-step

Registers
Memory

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 0	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Register & Memory Pane

Shows the state of
registers and memory

Console
⬆

```

***** Parser Output *****
Parsing successful!
```

Console Pane

Shows messages from the compiler and the simulator

Also used for system calls – let's you input and print values

Instruction breakdown
⬆

31	20	19	15	14	12	11	7	6	0
imm	rs1	funct3	rd	opcode					
000000000000	00000	000	00000	0010011					

Instruction Breakdown Pane

Don't have valid RISC-V assembly code to start with?

BRISC-V Home BRISC-V Simulator Documentation and example code here! [Manual & Examples](#)

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

C source

```

1 int fib(int n) {
2     if (n <= 1) {
3         return n;
4     } else {
5         return fib(n-1)+fib(n-2);
6     }
7 }
8
9 int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

RISC-V Assembly

Load an assembly (*.s) file

Example assembly files:

- Greatest common divisor
- Fibonacci
- Binary search

Click the load button to open a drop down menu for loading examples

Load your code here!

Examples available here!

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 0	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Console

Instruction breakdown

Don't have valid RISC-V assembly code to start with?

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

C source

```

1 int fib(int n) {
2     if (n <= 1) {
3         return n;
4     } else {
5         return fib(n-1)+fib(n-2);
6     }
7 }
8
9 int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

RISC-V Assembly

Load an assembly (*.s) file

Greatest common divisor

Fibonacci

Binary search

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 0	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Console

Instruction breakdown

Load the GCD example

Kernel and User Instructions

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

ASCS ADAPTIVE & SECURE
COMPUTING SYSTEMS
LABORATORY

C source

```

1  int fib(int n) {
2      if (n <= 1) {
3          return n;
4      } else {
5          return fib(n-1)+fib(n-2);
6      }
7  }
8
9  int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

RISC-V Assembly

```

0      addi    zero,zero,0
kernel:
1      addi    sp,zero,1536
2      call    main
3      addi    zero,zero,0
4      mv      s1,a0
5      addi    zero,zero,0
6      addi    zero,zero,0
7      auipc   ra,0x0
8      jalr    ra,0(ra)
9      addi    zero,zero,0
10     addi    zero,zero,0
.file   "gcd.c"
.option nopic
.text
.align 2
.globl gcd
.type    gcd, @function
gcd:
11     addi    sp,sp,-48
12     sw      ra,44(sp)
13     sw      s0,40(sp)
14     addi    s0,sp,48
15     sw      a0,-36(s0)
16     sw      a1,-40(s0)
17     lw      a4,-36(s0)
18     lw      a5,-40(s0)
19     bne     a4,a5,.L2
20     lw      a5,-36(s0)
21     sw      a5,-20(s0)
22     j       .L3
.L2:
23     lw      a4,-36(s0)
24     lw      a5,-40(s0)

```

Our code got loaded!

Console

```

***** Parser Output *****
Parsing successful!

```

The console says that it parsed the file without problems
If there were problems, they would pop up here

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 0	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Instruction breakdown									
31	20	19	15	14	12	11	7	6	0
imm		rs1		funct3		rd		opcode	
000000000000		00000		000		00000		0010011	

Kernel and User Instructions

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

C source

```

1 int fib(int n) {
2     if (n <= 1) {
3         return n;
4     } else {
5         return fib(n-1)+fib(n-2);
6     }
7 }
8
9 int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

RISC-V Assembly

```

1 addi zero,zero,0
2
3 kernel:
4     addi sp,zero,1536
5     call main
6     addi zero,zero,0
7     mv s1,a0
8     addi zero,zero,0
9     addi zero,zero,0
10    addi zero,zero,0
11
12 .file "gcd.c"
13 .option nopic
14 .text
15 .align 2
16 .globl gcd
17 .type gcd, @function
18
19 gcd:
20     addi sp,sp,-48
21     sw ra,44(sp)
22     sw s0,40(sp)
23     addi s0,sp,48
24     sw a0,-36(s0)
25     sw a1,-40(s0)
26     lw a4,-36(s0)
27     lw a5,-40(s0)
28     bne a4,a5,.L2
29     lw a5,-36(s0)
30     sw a5,-20(s0)
31     j .L3
32
33 .L2:
34     lw a4,-36(s0)
35     lw a5,-40(s0)
36     bne a4,a5,.L2
37
38 .L3:
39     ret

```

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 0	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Console

```

***** Parser Output *****
Parsing successful!

```

Instruction breakdown

31	20	19	15	14	12	11	7	6	0
imm		rs1		funct3		rd		opcode	
000000000000		00000		000		00000		0010011	

Grey instructions are kernel instructions

They setup some registers like the stack pointer, and jump to label "main"

Kernel and User Instructions

BRISC-V Home

BRISC-V Simulator

Manual & Examples



C source

```

1  int fib(int n) {
2      if (n <= 1) {
3          return n;
4      } else {
5          return fib(n-1)+fib(n-2);
6      }
7  }
8
9  int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

RISC-V Assembly

```

0      addi    zero,zero,0
kernel:
1      addi    sp,zero,1536
2      call    main
3      addi    zero,zero,0
4      mv      s1,a0
5      addi    zero,zero,0
6      addi    zero,zero,0
7      auipc   ra,0x0
8      jalr    ra,0(ra)
9      addi    zero,zero,0
10     addi    zero,zero,0
11     .file    "gcd.c"
12     .option  nopie
13     .text
14     .align   2
15     .globl   gcd
16     .type    gcd, @function
gcd:
17     addi     sp,sp,-48
18     sw       ra,44(sp)
19     sw       s0,40(sp)
20     addi     s0,sp,48
21     sw       a0,-36(s0)
22     sw       a1,-40(s0)
23     lw       a4,-36(s0)
24     lw       a5,-40(s0)
25     bne      a4,a5,.L2
26     lw       a5,-36(s0)
27     sw       a5,-20(s0)
28     j        .L3
.L2:
29     lw       a4,-36(s0)
30     lw       a5,-40(s0)
31     bne      a4,a5,.L2

```

Registers

Memory

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 0	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Console

```

***** Parser Output *****
Parsing successful!

```

Instruction breakdown

31	20	19	15	14	12	11	7	6	0
imm		rs1		funct3		rd		opcode	
000000000000		00000		000		00000		0010011	

White instructions are user instructions

All the assembly you write will be here

Simulator Controls

BRISC-V Home
BRISC-V Simulator
Manual & Examples



Adaptive and Secure Computing Systems Lab • Boston University

Here are some simulator controls:



ADAPTIVE & SECURE
COMPUTING SYSTEMS
LABORATORY

C source

```

1 int fib(int n) {
2   if (n <= 1) {
3     return n;
4   } else {
5     return fib(n-1)+fib(n-2);
6   }
7 }
8
9 int return_function (int result) {
10  return result;
11 }
12
13 int main(){
14   int n = 9;
15   int result = return_function (fib(n));
16   return result;
17 }
```

RISC-V Assembly

```

0      addi    zero,zero,0
1      addi    sp,zero,1536
2      call    main
3      addi    zero,zero,0
4      mv      s1,a0
5      addi    zero,zero,0
6      addi    zero,zero,0
7      auipc   ra,0x0
8      jalr    ra,a0(ra)
9      addi    zero,zero,0
10     addi    zero,zero,0
11
12     .file    "gcd.c"
13     .option  nopie
14     .text
15     .align   2
16     .globl   gcd
17     .type    gcd, @function
18
19 gcd:
20     addi    sp,sp,-48
21     sw      ra,44(sp)
22     sw      s0,40(sp)
23     addi    s0,sp,48
24     sw      a0,-36(s0)
25     sw      a1,-40(s0)
26     lw      a4,-36(s0)
27     lw      a5,-40(s0)
28     bne     a4,a5,.L2
29     lw      a5,-36(s0)
30     sw      a5,-20(s0)
31     j       .L3
32
33 .L2:
34     lw      a4,-36(s0)
35     lw      a5,-40(s0)
36     bne     a4,a5,.L2
37     ret
```

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 0	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Console

```
***** Parser Output *****
Parsing successful!
```

Instruction breakdown

31	20	19	15	14	12	11	7	6	0
imm		rs1		funct3		rd		opcode	
000000000000		00000		000		00000		0010011	

Stepping Through a Program

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

C source

```

1 int fib(int n) {
2     if (n <= 1) {
3         return n;
4     } else {
5         return fib(n-1)+fib(n-2);
6     }
7 }
8
9 int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

RISC-V Assembly

```

0 addi zero,zero,0
1 kernel:
2 addi sp,zero,1536
3 call main
4 addi zero,zero,0
5 mv s1,a0
6 addi zero,zero,0
7 auipc ra,0x0
8 jalr ra,0(ra)
9 addi zero,zero,0
10 addi zero,zero,0
11
12 .file "gcd.c"
13 .option nopic
14 .text
15 .align 2
16 .globl gcd
17 .type gcd, @function
18 gcd:
19 addi sp,sp,-48
20 sw ra,44(sp)
21 sw s0,40(sp)
22 addi s0,sp,48
23 sw a0,-36(s0)
24 sw a1,-40(s0)
25 lw a4,-36(s0)
26 lw a5,-40(s0)
27 bne a4,a5,.L2
28 lw a5,-36(s0)
29 sw a5,-20(s0)
30 j .L3
31 .L2:
32 lw a4,-36(s0)
33 lw a5,-40(s0)
34 blt a4,a5,.L2

```

Let's click the step button twice

The blue line shows which instruction will be executed next

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 0	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Console

```

***** Parser Output *****
Parsing successful!

```

Instruction breakdown

31	20	19	15	14	12	11	7	6	0
imm		rs1		funct3		rd		opcode	
000000000000		00000		000		00000		0010011	

Stepping Through a Program

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

C source

```

1 int fib(int n) {
2     if (n <= 1) {
3         return n;
4     } else {
5         return fib(n-1)+fib(n-2);
6     }
7 }
8
9 int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }
    
```

RISC-V Assembly

```

0      addi    zero,zero,0
kernel:
1      addi    sp,zero,1536
2      call    main
3      addi    zero,zero,0
4      mv      s1,a0
5      addi    zero,zero,0
6      addi    zero,zero,0
7      auipc   ra,0x0
8      jalr    ra,0(ra)
9      addi    zero,zero,0
10     addi    zero,zero,0
    
```

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 0	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Instruction breakdown

31	20	19	15	14	12	11	7	6	0
imm		rs1		funct3		rd		opcode	
011000000000		00000		000		00010		0010011	

Click the step button again

These instructions did not do anything, but the next one will

The blue line moved two lines down

The instruction breakdown pane shows how the instruction is stored in memory

The top row represents the bit ranges, the middle row shows range names, the bottom row shows binary values for the specific instruction

Console
***** Parser Output *****
Parsing successful!

addi Changed the Register File

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

ASCS ADAPTIVE & SECURE
COMPUTING SYSTEMS
LABORATORY

C source

```

1 int fib(int n) {
2     if (n <= 1) {
3         return n;
4     } else {
5         return fib(n-1)+fib(n-2);
6     }
7 }
8
9 int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

RISC-V Assembly

```

0      addi    zero,zero,0
1      addi    sp,zero,1536
2      call    main
3      addi    zero,zero,0
4      mv      s1,a0
5      addi    zero,zero,0
6      addi    zero,zero,0
7      auipc   ra,0x0
8      jalr    ra,0(ra)
9      addi    zero,zero,0
10     addi    zero,zero,0

```

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 1536	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/t0	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

**addi sp, zero, 1536
got executed**

**It summed 0 and 1536,
and put it in register **sp****

Register **sp is
highlighted!**

**sp stands for stack
pointer**

Console

```

***** Parser Output *****
Parsing successful!

```

Instruction breakdown

Pseudo-Instructions don't have a breakdown

Setting Breakpoints

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

C source

```

1 int fib(int n) {
2     if (n <= 1) {
3         return n;
4     } else {
5         return fib(n-1)+fib(n-2);
6     }
7 }
8
9 int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

RISC-V Assembly

```

0      addi    zero,zero,0
kernel:
1      addi    sp,zero,1536
2      call    main
3      addi    zero,zero,0
4      mv      s1,a0
5      addi    zero,zero,0
6      addi    zero,zero,0
7      auipc   ra,0x0
8      jalr    ra,0(ra)
9      addi    zero,zero,0
10     addi    zero,zero,0

.file   "gcd.c"
.option nopic
.text
.align 2
.globl gcd
.type   gcd, @function

gcd:
11     addi    sp,sp,-48
12     sw      ra,44(sp)
13     sw      s0,40(sp)
14     addi    s0,s0,4
15     sw      a0,-36(s0)
16     sw      a1,-40(s0)
17     lw      a4,-36(s0)
18     lw      a5,-40(s0)
19     bne     a4,a5,12
20     lw      a5,-36(s0)
21     sw      a5,-20(s0)
22     j       .L2
.L2:
23     lw      a4,-36(s0)
24     lw      a5,-40(s0)
25     bne     a4,a5,12

```

Right-clicking on an instruction opens a menu

Let's click on the first item – Add breakpoint

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 1536	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/tp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Console

```

***** Parser Output *****
Parsing successful!

```

Instruction breakdown

Pseudo-Instructions don't have a breakdown

Setting Breakpoints

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

C source

```

1 int fib(int n) {
2     if (n <= 1) {
3         return n;
4     } else {
5         return fib(n-1)+fib(n-2);
6     }
7 }
8
9 int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

Now if we click run, simulator will keep executing instructions until it hits a breakpoint

RISC-V Assembly

```

0 addi zero,zero,0
kernel:
1 addi sp,zero,1536
2 call main
3 addi zero,zero,0
4 mv s1,a0
5 addi zero,zero,0
6 addi zero,zero,0
7 auipc ra,0x0
8 jalr ra,0(ra)
9 addi zero,zero,0
10 addi zero,zero,0
11
12 .file "gcd.c"
13 .option nopic
14 .text
15 .align 2
16 .globl gcd
17 .type gcd, @function
gcd:
18 addi sp,sp,-48
19 sw ra,44(sp)
20 sw s0,40(sp)
21 addi s0,sp,18
22 sw a0,-36(s0)
23 sw a1,-40(s0)
24 lw a4,-36(s0)
25 lw a5,-40(s0)
26 bne a4,a5,.L2
27 lw a5,-36(s0)
28 sw a5,-20(s0)
29 j .L3
.L2:
30 lw a4,-36(s0)
31 lw a5,-40(s0)
32 bne a4,a5,.L2

```

A breakpoint got created!

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 0
sp	[2] 1536	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/fp	[8] 0	s1	[9] 0
a0	[10] 0	a1	[11] 0
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 0
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Console

```

***** Parser Output *****
Parsing successful!

```

asclab.org/research/briscv/index.html

Instruction breakdown

Pseudo-Instructions don't have a breakdown

Running Code until a Breakpoint

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

C source

```

1 int fib(int n) {
2     if (n <= 1) {
3         return n;
4     } else {
5         return fib(n-1)+fib(n-2);
6     }
7 }
8
9 int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

RISC-V Assembly

```

0      addi    zero,zero,0
kernel:
1      addi    sp,zero,1536
2      call    main
3      addi    zero,zero,0
4      mv      s1,a0
5      addi    zero,zero,0
6      addi    zero,zero,0
7      auipc   ra,0x0
8      jalr    ra,0(ra)
9      addi    zero,zero,0
10     addi    zero,zero,0
11
12     .file    "gcd.c"
13     .option  nopie
14     .text
15     .align  2
16     .globl  gcd
17     .type   gcd, @function
gcd:
18     addi    sp,sp,-48
19     sw      ra,44(sp)
20     sw      s0,40(sp)
21     addi    s0,sp,48
22     sw      a0,-36(s0)
23     sw      a1,-40(s0)
24     lw      a4,-36(s0)
25     lw      a5,-40(s0)
26     bne     a4,a5,.L2
27     lw      a5,-36(s0)
28     sw      a5,-20(s0)
29     j       .L3
.L2:
30     lw      a4,-36(s0)
31     lw      a5,-40(s0)
32     bne     a4,a5,.L2

```

Registers

Register	Value	Register	Value
zero	[0] 0	ra	[1] 73
sp	[2] 1456	gp	[3] 0
tp	[4] 0	t0	[5] 0
t1	[6] 0	t2	[7] 0
s0/tp	[8] 1536	s1	[9] 0
a0	[10] 64	a1	[11] 48
a2	[12] 0	a3	[13] 0
a4	[14] 0	a5	[15] 48
a6	[16] 0	a7	[17] 0
s2	[18] 0	s3	[19] 0
s4	[20] 0	s5	[21] 0
s6	[22] 0	s7	[23] 0
s8	[24] 0	s9	[25] 0
s10	[26] 0	s11	[27] 0
t3	[28] 0	t4	[29] 0
t5	[30] 0	t6	[31] 0

Console

```

***** Parser Output *****
Parsing successful!

```

Instruction breakdown

31	20	19	15	14	12	11	7	6	0
imm		rs1		funct3		rd		opcode	
000000110000		00010		000		01000		0010011	

The instruction pointer moved to the breakpoint!

A lot of registers changed values!

Text and Data Sections

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

C source

```

1 int fib(int n) {
2     if (n <= 1) {
3         return n;
4     } else {
5         return fib(n-1)+fib(n-2);
6     }
7 }
8
9 int return_function(int n) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

RISC-V Assembly

```

22 ecall
   # let's statically allocate a string "HELLO!"
   # we start this by creating a read-only data section
   .rodata
   .HELLO:
   # strings should end with the null terminator \0
   # the null terminator's binary value is 0!
   # we split HELLO!\0 into two 32bit words:
   # HELL and 0!00 - note that thats an "0!" and 2 zeros
   # we write HELL in ascii:
   # H - 0x48
   # E - 0x45
   # L - 0x4C
   # since this is a little-endian architecture, we
   # write HELL in reverse - LEHH
   .word 0x4C4C4548
   # now we write the second part 0!00
   .word 0x0000214F
   # this section is parsed when you load the program -
   # not when the instruction pointer runs over it.
   # as soon as you loaded the program, you should see
   # this string in the memory pane's data section,
   # somewhere close to the bottom of it.
   #
   # now we can go back to a text section that has code
   .text
   # print the string "HELLO!\n"
23 addi t0, zero, 3      # this is the string printing syscall
24 lui a0, %hi(.HELLO)   # this loads the top 20 bits
                        # of .HELLO address into a0
25 addi a0, a0, %lo(.HELLO) # this loads the bottom 12 bits
26 addi a1, zero, 7      # length of the string
27 ecall
   # print characters '!', '\n', '-'

```

Registers **Memory**

HEX DEC BINARY

0x00000130: 00 00 00 00
0x00000134: 00 00 00 00
0x00000138: 00 00 00 00
0x0000013c: 00 00 00 00
0x00000140: 00 00 00 00
0x00000144: 00 00 00 00
0x00000148: 00 00 00 00
0x0000014c: 00 00 00 00
0x00000150: 00 00 00 00
0x00000154: 00 00 00 00
0x00000158: 00 00 00 00
0x0000015c: 00 00 00 00
0x00000160: 00 00 00 00
0x00000164: 00 00 00 00
0x00000168: 00 00 00 00
0x0000016c: 00 00 00 00
0x00000170: 00 00 00 00
0x00000174: 00 00 00 00
0x00000178: 00 00 00 00
0x0000017c: 00 00 00 00
0x00000180: 00 00 00 00
0x00000184: 00 00 00 00
0x00000188: 00 00 00 00
0x0000018c: 00 00 00 00
0x00000190: 00 00 00 00
0x00000194: 00 00 00 00
0x00000198: 00 00 00 00
0x0000019c: 00 00 00 00
0x000001a0: 00 00 00 00
0x000001a4: 00 00 00 00
0x000001a8: 00 00 00 00
0x000001ac: 00 00 00 00
0x000001b0: 00 00 00 00
0x000001b4: 00 00 00 00
0x000001b8: 00 00 00 00
0x000001bc: 00 00 00 00
0x000001c0: 00 00 00 00
0x000001c4: 00 00 00 00
0x000001c8: 00 00 00 00
0x000001cc: 00 00 00 00
0x000001d0: 00 00 00 00
0x000001d4: 00 00 00 00
0x000001d8: 00 00 00 00
0x000001dc: 00 00 00 00
0x000001e0: 00 00 00 00
0x000001e4: 00 00 00 00
0x000001e8: 00 00 00 00
0x000001ec: 00 00 00 00
0x000001f0: 00 00 00 00
0x000001f4: 00 00 00 00
0x000001f8: 00 00 00 00
0x000001fc: 00 00 00 00
0x00000200: 00 00 00 00
0x00000204: 00 00 00 00
0x00000208: 00 00 00 00
0x0000020c: 00 00 00 00
0x00000210: 00 00 00 00
0x00000214: 00 00 00 00
0x00000218: 00 00 00 00
0x0000021c: 00 00 00 00
0x00000220: 00 00 00 00
0x00000224: 00 00 00 00
0x00000228: 00 00 00 00
0x0000022c: 00 00 00 00
0x00000230: 00 00 00 00
0x00000234: 00 00 00 00
0x00000238: 00 00 00 00
0x0000023c: 00 00 00 00
0x00000240: 00 00 00 00
0x00000244: 00 00 00 00
0x00000248: 00 00 00 00
0x0000024c: 00 00 00 00
0x00000250: 00 00 00 00
0x00000254: 00 00 00 00
0x00000258: 00 00 00 00
0x0000025c: 00 00 00 00
0x00000260: 00 00 00 00
0x00000264: 00 00 00 00
0x00000268: 00 00 00 00
0x0000026c: 00 00 00 00
0x00000270: 00 00 00 00
0x00000274: 00 00 00 00
0x00000278: 00 00 00 00
0x0000027c: 00 00 00 00
0x00000280: 00 00 00 00
0x00000284: 00 00 00 00
0x00000288: 00 00 00 00
0x0000028c: 00 00 00 00
0x00000290: 00 00 00 00
0x00000294: 00 00 00 00
0x00000298: 00 00 00 00
0x0000029c: 00 00 00 00
0x000002a0: 00 00 00 00
0x000002a4: 00 00 00 00
0x000002a8: 00 00 00 00
0x000002ac: 00 00 00 00
0x000002b0: 00 00 00 00
0x000002b4: 00 00 00 00
0x000002b8: 00 00 00 00
0x000002bc: 00 00 00 00
0x000002c0: 00 00 00 00
0x000002c4: 00 00 00 00
0x000002c8: 00 00 00 00
0x000002cc: 00 00 00 00
0x000002d0: 00 00 00 00
0x000002d4: 00 00 00 00
0x000002d8: 00 00 00 00
0x000002dc: 00 00 00 00
0x000002e0: 00 00 00 00
0x000002e4: 00 00 00 00
0x000002e8: 00 00 00 00
0x000002ec: 00 00 00 00
0x000002f0: 00 00 00 00
0x000002f4: 00 00 00 00
0x000002f8: 00 00 00 00
0x000002fc: 00 00 00 00
0x00000300: 00 00 00 00
0x00000304: 00 00 00 00
0x00000308: 00 00 00 00
0x0000030c: 00 00 00 00
0x00000310: 00 00 00 00
0x00000314: 00 00 00 00
0x00000318: 00 00 00 00
0x0000031c: 00 00 00 00
0x00000320: 00 00 00 00
0x00000324: 00 00 00 00
0x00000328: 00 00 00 00
0x0000032c: 00 00 00 00
0x00000330: 00 00 00 00
0x00000334: 00 00 00 00
0x00000338: 00 00 00 00
0x0000033c: 00 00 00 00
0x00000340: 00 00 00 00
0x00000344: 00 00 00 00
0x00000348: 00 00 00 00
0x0000034c: 00 00 00 00
0x00000350: 00 00 00 00
0x00000354: 00 00 00 00
0x00000358: 00 00 00 00
0x0000035c: 00 00 00 00
0x00000360: 00 00 00 00
0x00000364: 00 00 00 00
0x00000368: 00 00 00 00
0x0000036c: 00 00 00 00
0x00000370: 00 00 00 00
0x00000374: 00 00 00 00
0x00000378: 00 00 00 00
0x0000037c: 00 00 00 00
0x00000380: 00 00 00 00
0x00000384: 00 00 00 00
0x00000388: 00 00 00 00
0x0000038c: 00 00 00 00
0x00000390: 00 00 00 00
0x00000394: 00 00 00 00
0x00000398: 00 00 00 00
0x0000039c: 00 00 00 00
0x000003a0: 00 00 00 00
0x000003a4: 00 00 00 00
0x000003a8: 00 00 00 00
0x000003ac: 00 00 00 00
0x000003b0: 00 00 00 00
0x000003b4: 00 00 00 00
0x000003b8: 00 00 00 00
0x000003bc: 00 00 00 00
0x000003c0: 00 00 00 00
0x000003c4: 00 00 00 00
0x000003c8: 00 00 00 00
0x000003cc: 00 00 00 00
0x000003d0: 00 00 00 00
0x000003d4: 00 00 00 00
0x000003d8: 00 00 00 00
0x000003dc: 00 00 00 00
0x000003e0: 00 00 00 00
0x000003e4: 00 00 00 00
0x000003e8: 00 00 00 00
0x000003ec: 00 00 00 00
0x000003f0: 00 00 00 00
0x000003f4: 00 00 00 00
0x000003f8: 00 00 00 00
0x000003fc: 00 00 00 00
0x00000400: 00 00 00 00
0x00000404: 00 00 00 00
0x00000408: 00 00 00 00
0x0000040c: 00 00 00 00
0x00000410: 00 00 00 00
0x00000414: 00 00 00 00
0x00000418: 00 00 00 00
0x0000041c: 00 00 00 00
0x00000420: 00 00 00 00
0x00000424: 00 00 00 00
0x00000428: 00 00 00 00
0x0000042c: 00 00 00 00
0x00000430: 00 00 00 00
0x00000434: 00 00 00 00
0x00000438: 00 00 00 00
0x0000043c: 00 00 00 00
0x00000440: 00 00 00 00
0x00000444: 00 00 00 00
0x00000448: 00 00 00 00
0x0000044c: 00 00 00 00
0x00000450: 00 00 00 00
0x00000454: 00 00 00 00
0x00000458: 00 00 00 00
0x0000045c: 00 00 00 00
0x00000460: 00 00 00 00
0x00000464: 00 00 00 00
0x00000468: 00 00 00 00
0x0000046c: 00 00 00 00
0x00000470: 00 00 00 00
0x00000474: 00 00 00 00
0x00000478: 00 00 00 00
0x0000047c: 00 00 00 00
0x00000480: 00 00 00 00
0x00000484: 00 00 00 00
0x00000488: 00 00 00 00
0x0000048c: 00 00 00 00
0x00000490: 00 00 00 00
0x00000494: 00 00 00 00
0x00000498: 00 00 00 00
0x0000049c: 00 00 00 00
0x000004a0: 00 00 00 00
0x000004a4: 00 00 00 00
0x000004a8: 00 00 00 00
0x000004ac: 00 00 00 00
0x000004b0: 00 00 00 00
0x000004b4: 00 00 00 00
0x000004b8: 00 00 00 00
0x000004bc: 00 00 00 00
0x000004c0: 00 00 00 00
0x000004c4: 00 00 00 00
0x000004c8: 00 00 00 00
0x000004cc: 00 00 00 00
0x000004d0: 00 00 00 00
0x000004d4: 00 00 00 00
0x000004d8: 00 00 00 00
0x000004dc: 00 00 00 00
0x000004e0: 00 00 00 00
0x000004e4: 00 00 00 00
0x000004e8: 00 00 00 00
0x000004ec: 00 00 00 00
0x000004f0: 00 00 00 00
0x000004f4: 00 00 00 00
0x000004f8: 00 00 00 00
0x000004fc: 00 00 00 00
0x00000500: 00 00 00 00
0x00000504: 00 00 00 00
0x00000508: 00 00 00 00
0x0000050c: 00 00 00 00
0x00000510: 00 00 00 00
0x00000514: 00 00 00 00
0x00000518: 00 00 00 00
0x0000051c: 00 00 00 00
0x00000520: 00 00 00 00
0x00000524: 00 00 00 00
0x00000528: 00 00 00 00
0x0000052c: 00 00 00 00
0x00000530: 00 00 00 00
0x00000534: 00 00 00 00
0x00000538: 00 00 00 00
0x0000053c: 00 00 00 00
0x00000540: 00 00 00 00
0x00000544: 00 00 00 00
0x00000548: 00 00 00 00
0x0000054c: 00 00 00 00
0x00000550: 00 00 00 00
0x00000554: 00 00 00 00
0x00000558: 00 00 00 00
0x0000055c: 00 00 00 00
0x00000560: 00 00 00 00
0x00000564: 00 00 00 00
0x00000568: 00 00 00 00
0x0000056c: 00 00 00 00
0x00000570: 00 00 00 00
0x00000574: 00 00 00 00
0x00000578: 00 00 00 00
0x0000057c: 00 00 00 00
0x00000580: 00 00 00 00
0x00000584: 00 00 00 00
0x00000588: 00 00 00 00
0x0000058c: 00 00 00 00
0x00000590: 00 00 00 00
0x00000594: 00 00 00 00
0x00000598: 00 00 00 00
0x0000059c: 00 00 00 00
0x000005a0: 00 00 00 00
0x000005a4: 00 00 00 00
0x000005a8: 00 00 00 00
0x000005ac: 00 00 00 00
0x000005b0: 00 00 00 00
0x000005b4: 00 00 00 00
0x000005b8: 00 00 00 00
0x000005bc: 00 00 00 00
0x000005c0: 00 00 00 00
0x000005c4: 00 00 00 00
0x000005c8: 00 00 00 00
0x000005cc: 00 00 00 00
0x000005d0: 00 00 00 00
0x000005d4: 00 00 00 00
0x000005d8: 00 00 00 00
0x000005dc: 00 00 00 00
0x000005e0: 00 00 00 00
0x000005e4: 00 00 00 00
0x000005e8: 00 00 00 00
0x000005ec: 00 00 00 00
0x000005f0: 00 00 00 00
0x000005f4: 00 00 00 00
0x000005f8: 00 00 00 00
0x000005fc: 00 00 00 00
0x00000600: 00 00 00 00
0x00000604: 00 00 00 00
0x00000608: 00 00 00 00
0x0000060c: 00 00 00 00
0x00000610: 00 00 00 00
0x00000614: 00 00 00 00
0x00000618: 00 00 00 00
0x0000061c: 00 00 00 00
0x00000620: 00 00 00 00
0x00000624: 00 00 00 00
0x00000628: 00 00 00 00
0x0000062c: 00 00 00 00
0x00000630: 00 00 00 00
0x00000634: 00 00 00 00
0x00000638: 00 00 00 00
0x0000063c: 00 00 00 00
0x00000640: 00 00 00 00
0x00000644: 00 00 00 00
0x00000648: 00 00 00 00
0x0000064c: 00 00 00 00
0x00000650: 00 00 00 00
0x00000654: 00 00 00 00
0x00000658: 00 00 00 00
0x0000065c: 00 00 00 00
0x00000660: 00 00 00 00
0x00000664: 00 00 00 00
0x00000668: 00 00 00 00
0x0000066c: 00 00 00 00
0x00000670: 00 00 00 00
0x00000674: 00 00 00 00
0x00000678: 00 00 00 00
0x0000067c: 00 00 00 00
0x00000680: 00 00 00 00
0x00000684: 00 00 00 00
0x00000688: 00 00 00 00
0x0000068c: 00 00 00 00
0x00000690: 00 00 00 00
0x00000694: 00 00 00 00
0x00000698: 00 00 00 00
0x0000069c: 00 00 00 00
0x000006a0: 00 00 00 00
0x000006a4: 00 00 00 00
0x000006a8: 00 00 00 00
0x000006ac: 00 00 00 00
0x000006b0: 00 00 00 00
0x000006b4: 00 00 00 00
0x000006b8: 00 00 00 00
0x000006bc: 00 00 00 00
0x000006c0: 00 00 00 00
0x000006c4: 00 00 00 00
0x000006c8: 00 00 00 00
0x000006cc: 00 00 00 00
0x000006d0: 00 00 00 00
0x000006d4: 00 00 00 00
0x000006d8: 00 00 00 00
0x000006dc: 00 00 00 00
0x000006e0: 00 00 00 00
0x000006e4: 00 00 00 00
0x000006e8: 00 00 00 00
0x000006ec: 00 00 00 00
0x000006f0: 00 00 00 00
0x000006f4: 00 00 00 00
0x000006f8: 00 00 00 00
0x000006fc: 00 00 00 00
0x00000700: 00 00 00 00
0x00000704: 00 00 00 00
0x00000708: 00 00 00 00
0x0000070c: 00 00 00 00
0x00000710: 00 00 00 00
0x00000714: 00 00 00 00
0x00000718: 00 00 00 00
0x0000071c: 00 00 00 00
0x00000720: 00 00 00 00
0x00000724: 00 00 00 00
0x00000728: 00 00 00 00
0x0000072c: 00 00 00 00
0x00000730: 00 00 00 00
0x00000734: 00 00 00 00
0x00000738: 00 00 00 00
0x0000073c: 00 00 00 00
0x00000740: 00 00 00 00
0x00000744: 00 00 00 00
0x00000748: 00 00 00 00
0x0000074c: 00 00 00 00
0x00000750: 00 00 00 00
0x00000754: 00 00 00 00
0x00000758: 00 00 00 00
0x0000075c: 00 00 00 00
0x00000760: 00 00 00 00
0x00000764: 00 00 00 00
0x00000768: 00 00 00 00
0x0000076c: 00 00 00 00
0x00000770: 00 00 00 00
0x00000774: 00 00 00 00
0x00000778: 00 00 00 00
0x0000077c: 00 00 00 00
0x00000780: 00 00 00 00
0x00000784: 00 00 00 00
0x00000788: 00 00 00 00
0x0000078c: 00 00 00 00
0x00000790: 00 00 00 00
0x00000794: 00 00 00 00
0x00000798: 00 00 00 00
0x0000079c: 00 00 00 00
0x000007a0: 00 00 00 00
0x000007a4: 00 00 00 00
0x000007a8: 00 00 00 00
0x000007ac: 00 00 00 00
0x000007b0: 00 00 00 00
0x000007b4: 00 00 00 00
0x000007b8: 00 00 00 00
0x000007bc: 00 00 00 00
0x000007c0: 00 00 00 00
0x000007c4: 00 00 00 00
0x000007c8: 00 00 00 00
0x000007cc: 00 00 00 00
0x000007d0: 00 00 00 00
0x000007d4: 00 00 00 00
0x000007d8: 00 00 00 00
0x000007dc: 00 00 00 00
0x000007e0: 00 00 00 00
0x000007e4: 00 00 00 00
0x000007e8: 00 00 00 00
0x000007ec: 00 00 00 00
0x000007f0: 00 00 00 00
0x000007f4: 00 00 00 00
0x000007f8: 00 00 00 00
0x000007fc: 00 00 00 00
0x00000800: 00 00 00 00
0x00000804: 00 00 00 00
0x00000808: 00 00 00 00
0x0000080c: 00 00 00 00
0x00000810: 00 00 00 00
0x00000814: 00 00 00 00
0x00000818: 00 00 00 00
0x0000081c: 00 00 00 00
0x00000820: 00 00 00 00
0x00000824: 0

Text and Data Sections

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

C source

```

1 int fib(int n) {
2     if (n <= 1) {
3         return n;
4     } else {
5         return fib(n-1)+fib(n-2);
6     }
7 }
8
9 int main() {
10     int n = 9;
11     int result = return_function(fib(n));
12     return result;
13 }
14
15 int main(){
16     int n = 9;
17     int result = return_function(fib(n));
18     return result;
19 }

```

Labels in data sections point to memory statically allocated after them

Here we allocate the string "HELLO!" by into two 32bit words – one with "HELL" and one with "O!00"

RISC-V Assembly

```

22 ecall
    # let's statically allocate a string "HELLO!"
    # we start this by creating a read-only data section
    .rodata
    .HELLO:
    # strings should end with the null terminator \0
    # the null terminator's binary value is 0!
    # we split HELLO!\0 into two 32bit words:
    # HELL and O!00 - note that thats an "O!" and 2 zeros
    # we write HELL in ascii:
    # H - 0x48
    # E - 0x45
    # L - 0x4C
    # since this is a little-endian architecture, we
    # write HELL in reverse - LEHH
    .word 0x4C4C4548
    # now we write the second part O!00
    .word 0x0000214F
    # this section is parsed when you load the program -
    # not when the instruction pointer runs over it.
    # as soon as you loaded the program, you should see
    # this string in the memory pane's data section,
    # somewhere close to the bottom of it.
    #
    # now we can go back to a text section that has code
    .text
    # print the string "HELLO!\n"
    addi t0, zero, 3      # this is the string printing syscall
    lui a0, %hi(.HELLO)   # this loads the top 20 bits
                          # of .HELLO address into a0
    addi a0, a0, %lo(.HELLO) # this loads the bottom 12 bits
    addi a1, zero, 7      # length of the string
    ecall
    # print characters '!', '\n', '-'

```

Registers **Memory**

HEX DEC BINARY

0x00000130: 00 00 00 00
0x00000134: 00 00 00 00
0x00000138: 00 00 00 00
0x0000013c: 00 00 00 00
0x00000140: 00 00 00 00
0x00000144: 00 00 00 00
0x00000148: 00 00 00 00
0x0000014c: 00 00 00 00
0x00000150: 00 00 00 00
0x00000154: 00 00 00 00
0x00000158: 00 00 00 00
0x0000015c: 00 00 00 00
0x00000160: 00 00 00 00
0x00000164: 00 00 00 00
0x00000168: 00 00 00 00
0x0000016c: 00 00 00 00
0x00000170: 00 00 00 00
0x00000174: 00 00 00 00
0x00000178: 00 00 00 00
0x0000017c: 00 00 00 00
0x00000180: 00 00 00 00
0x00000184: 00 00 00 00
0x00000188: 00 00 00 00
0x0000018c: 00 00 00 00
0x00000190: 00 00 00 00
0x00000194: 00 00 00 00
0x00000198: 00 00 00 00
0x0000019c: 00 00 00 00
0x000001a0: 00 00 00 00
0x000001a4: 00 00 00 00
0x000001a8: 00 00 00 00
0x000001ac: 00 00 00 00
0x000001b0: 00 00 00 00
0x000001b4: 00 00 00 00
0x000001b8: 00 00 00 00
0x000001bc: 00 00 00 00
0x000001c0: 00 00 00 00
0x000001c4: 00 00 00 00
0x000001c8: 00 00 00 00
0x000001cc: 00 00 00 00
0x000001d0: 00 00 00 00
0x000001d4: 00 00 00 00
0x000001d8: 00 00 00 00
0x000001dc: 00 00 00 00
0x000001e0: 00 00 00 00
0x000001e4: 00 00 00 00
0x000001e8: 00 00 00 00
0x000001ec: 00 00 00 00
0x000001f0: 00 00 00 00
0x000001f4: 00 00 00 00
0x000001f8: 00 00 00 00
0x000001fc: 00 00 00 00
0x00000200: 00 00 00 00
0x00000204: 00 00 00 00
0x00000208: 00 00 00 00
0x0000020c: 00 00 00 00
0x00000210: 00 00 00 00
0x00000214: 00 00 00 00
0x00000218: 00 00 00 00
0x0000021c: 00 00 00 00
0x00000220: 00 00 00 00
0x00000224: 00 00 00 00
0x00000228: 00 00 00 00
0x0000022c: 00 00 00 00
0x00000230: 00 00 00 00
0x00000234: 00 00 00 00
0x00000238: 00 00 00 00
0x0000023c: 00 00 00 00
0x00000240: 00 00 00 00
0x00000244: 00 00 00 00
0x00000248: 00 00 00 00
0x0000024c: 00 00 00 00
0x00000250: 00 00 00 00
0x00000254: 00 00 00 00
0x00000258: 00 00 00 00
0x0000025c: 00 00 00 00
0x00000260: 00 00 00 00
0x00000264: 00 00 00 00
0x00000268: 00 00 00 00
0x0000026c: 00 00 00 00
0x00000270: 00 00 00 00
0x00000274: 00 00 00 00
0x00000278: 00 00 00 00
0x0000027c: 00 00 00 00
0x00000280: 00 00 00 00
0x00000284: 00 00 00 00
0x00000288: 00 00 00 00
0x0000028c: 00 00 00 00
0x00000290: 00 00 00 00
0x00000294: 00 00 00 00
0x00000298: 00 00 00 00
0x0000029c: 00 00 00 00
0x000002a0: 00 00 00 00
0x000002a4: 00 00 00 00
0x000002a8: 00 00 00 00
0x000002ac: 00 00 00 00
0x000002b0: 00 00 00 00
0x000002b4: 00 00 00 00
0x000002b8: 00 00 00 00
0x000002bc: 00 00 00 00
0x000002c0: 00 00 00 00
0x000002c4: 00 00 00 00
0x000002c8: 00 00 00 00
0x000002cc: 00 00 00 00
0x000002d0: 00 00 00 00
0x000002d4: 00 00 00 00
0x000002d8: 00 00 00 00
0x000002dc: 00 00 00 00
0x000002e0: 00 00 00 00
0x000002e4: 00 00 00 00
0x000002e8: 00 00 00 00
0x000002ec: 00 00 00 00
0x000002f0: 00 00 00 00
0x000002f4: 00 00 00 00
0x000002f8: 00 00 00 00
0x000002fc: 00 00 00 00
0x00000300: 00 00 00 00
0x00000304: 00 00 00 00
0x00000308: 00 00 00 00
0x0000030c: 00 00 00 00
0x00000310: 00 00 00 00
0x00000314: 00 00 00 00
0x00000318: 00 00 00 00
0x0000031c: 00 00 00 00
0x00000320: 00 00 00 00
0x00000324: 00 00 00 00
0x00000328: 00 00 00 00
0x0000032c: 00 00 00 00
0x00000330: 00 00 00 00
0x00000334: 00 00 00 00
0x00000338: 00 00 00 00
0x0000033c: 00 00 00 00
0x00000340: 00 00 00 00
0x00000344: 00 00 00 00
0x00000348: 00 00 00 00
0x0000034c: 00 00 00 00
0x00000350: 00 00 00 00
0x00000354: 00 00 00 00
0x00000358: 00 00 00 00
0x0000035c: 00 00 00 00
0x00000360: 00 00 00 00
0x00000364: 00 00 00 00
0x00000368: 00 00 00 00
0x0000036c: 00 00 00 00
0x00000370: 00 00 00 00
0x00000374: 00 00 00 00
0x00000378: 00 00 00 00
0x0000037c: 00 00 00 00
0x00000380: 00 00 00 00
0x00000384: 00 00 00 00
0x00000388: 00 00 00 00
0x0000038c: 00 00 00 00
0x00000390: 00 00 00 00
0x00000394: 00 00 00 00
0x00000398: 00 00 00 00
0x0000039c: 00 00 00 00
0x000003a0: 00 00 00 00
0x000003a4: 00 00 00 00
0x000003a8: 00 00 00 00
0x000003ac: 00 00 00 00
0x000003b0: 00 00 00 00
0x000003b4: 00 00 00 00
0x000003b8: 00 00 00 00
0x000003bc: 00 00 00 00
0x000003c0: 00 00 00 00
0x000003c4: 00 00 00 00
0x000003c8: 00 00 00 00
0x000003cc: 00 00 00 00
0x000003d0: 00 00 00 00
0x000003d4: 00 00 00 00
0x000003d8: 00 00 00 00
0x000003dc: 00 00 00 00
0x000003e0: 00 00 00 00
0x000003e4: 00 00 00 00
0x000003e8: 00 00 00 00
0x000003ec: 00 00 00 00
0x000003f0: 00 00 00 00
0x000003f4: 00 00 00 00
0x000003f8: 00 00 00 00
0x000003fc: 00 00 00 00
0x00000400: 00 00 00 00
0x00000404: 00 00 00 00
0x00000408: 00 00 00 00
0x0000040c: 00 00 00 00
0x00000410: 00 00 00 00
0x00000414: 00 00 00 00
0x00000418: 00 00 00 00
0x0000041c: 00 00 00 00
0x00000420: 00 00 00 00
0x00000424: 00 00 00 00
0x00000428: 00 00 00 00
0x0000042c: 00 00 00 00
0x00000430: 00 00 00 00
0x00000434: 00 00 00 00
0x00000438: 00 00 00 00
0x0000043c: 00 00 00 00
0x00000440: 00 00 00 00
0x00000444: 00 00 00 00
0x00000448: 00 00 00 00
0x0000044c: 00 00 00 00
0x00000450: 00 00 00 00
0x00000454: 00 00 00 00
0x00000458: 00 00 00 00
0x0000045c: 00 00 00 00
0x00000460: 00 00 00 00
0x00000464: 00 00 00 00
0x00000468: 00 00 00 00
0x0000046c: 00 00 00 00
0x00000470: 00 00 00 00
0x00000474: 00 00 00 00
0x00000478: 00 00 00 00
0x0000047c: 00 00 00 00
0x00000480: 00 00 00 00
0x00000484: 00 00 00 00
0x00000488: 00 00 00 00
0x0000048c: 00 00 00 00
0x00000490: 00 00 00 00
0x00000494: 00 00 00 00
0x00000498: 00 00 00 00
0x0000049c: 00 00 00 00
0x000004a0: 00 00 00 00
0x000004a4: 00 00 00 00
0x000004a8: 00 00 00 00
0x000004ac: 00 00 00 00
0x000004b0: 00 00 00 00
0x000004b4: 00 00 00 00
0x000004b8: 00 00 00 00
0x000004bc: 00 00 00 00
0x000004c0: 00 00 00 00
0x000004c4: 00 00 00 00
0x000004c8: 00 00 00 00
0x000004cc: 00 00 00 00
0x000004d0: 00 00 00 00
0x000004d4: 00 00 00 00
0x000004d8: 00 00 00 00
0x000004dc: 00 00 00 00
0x000004e0: 00 00 00 00
0x000004e4: 00 00 00 00
0x000004e8: 00 00 00 00
0x000004ec: 00 00 00 00
0x000004f0: 00 00 00 00
0x000004f4: 00 00 00 00
0x000004f8: 00 00 00 00
0x000004fc: 00 00 00 00
0x00000500: 00 00 00 00
0x00000504: 00 00 00 00
0x00000508: 00 00 00 00
0x0000050c: 00 00 00 00
0x00000510: 00 00 00 00
0x00000514: 00 00 00 00
0x00000518: 00 00 00 00
0x0000051c: 00 00 00 00
0x00000520: 00 00 00 00
0x00000524: 00 00 00 00
0x00000528: 00 00 00 00
0x0000052c: 00 00 00 00
0x00000530: 00 00 00 00
0x00000534: 00 00 00 00
0x00000538: 00 00 00 00
0x0000053c: 00 00 00 00
0x00000540: 00 00 00 00
0x00000544: 00 00 00 00
0x00000548: 00 00 00 00
0x0000054c: 00 00 00 00
0x00000550: 00 00 00 00
0x00000554: 00 00 00 00
0x00000558: 00 00 00 00
0x0000055c: 00 00 00 00
0x00000560: 00 00 00 00
0x00000564: 00 00 00 00
0x00000568: 00 00 00 00
0x0000056c: 00 00 00 00
0x00000570: 00 00 00 00
0x00000574: 00 00 00 00
0x00000578: 00 00 00 00
0x0000057c: 00 00 00 00
0x00000580: 00 00 00 00
0x00000584: 00 00 00 00
0x00000588: 00 00 00 00
0x0000058c: 00 00 00 00
0x00000590: 00 00 00 00
0x00000594: 00 00 00 00
0x00000598: 00 00 00 00
0x0000059c: 00 00 00 00
0x000005a0: 00 00 00 00
0x000005a4: 00 00 00 00
0x000005a8: 00 00 00 00
0x000005ac: 00 00 00 00
0x000005b0: 00 00 00 00
0x000005b4: 00 00 00 00
0x000005b8: 00 00 00 00
0x000005bc: 00 00 00 00
0x000005c0: 00 00 00 00
0x000005c4: 00 00 00 00
0x000005c8: 00 00 00 00
0x000005cc: 00 00 00 00
0x000005d0: 00 00 00 00
0x000005d4: 00 00 00 00
0x000005d8: 00 00 00 00
0x000005dc: 00 00 00 00
0x000005e0: 00 00 00 00
0x000005e4: 00 00 00 00
0x000005e8: 00 00 00 00
0x000005ec: 00 00 00 00
0x000005f0: 00 00 00 00
0x000005f4: 00 00 00 00
0x000005f8: 00 00 00 00
0x000005fc: 00 00 00 00
0x00000600: 00 00 00 00
0x00000604: 00 00 00 00
0x00000608: 00 00 00 00
0x0000060c: 00 00 00 00
0x00000610: 00 00 00 00
0x00000614: 00 00 00 00
0x00000618: 00 00 00 00
0x0000061c: 00 00 00 00
0x00000620: 00 00 00 00
0x00000624: 00 00 00 00
0x00000628: 00 00 00 00
0x0000062c: 00 00 00 00
0x00000630: 00 00 00 00
0x00000634: 00 00 00 00
0x00000638: 00 00 00 00
0x0000063c: 00 00 00 00
0x00000640: 00 00 00 00
0x00000644: 00 00 00 00
0x00000648: 00 00 00 00
0x0000064c: 00 00 00 00
0x00000650: 00 00 00 00
0x00000654: 00 00 00 00
0x00000658: 00 00 00 00
0x0000065c: 00 00 00 00
0x00000660: 00 00 00 00
0x00000664: 00 00 00 00
0x00000668: 00 00 00 00
0x0000066c: 00 00 00 00
0x00000670: 00 00 00 00
0x00000674: 00 00 00 00
0x00000678: 00 00 00 00
0x0000067c: 00 00 00 00
0x00000680: 00 00 00 00
0x00000684: 00 00 00 00
0x00000688: 00 00 00 00
0x0000068c: 00 00 00 00
0x00000690: 00 00 00 00
0x00000694: 00 00 00 00
0x00000698: 00 00 00 00
0x0000069c: 00 00 00 00
0x000006a0: 00 00 00 00
0x000006a4: 00 00 00 00
0x000006a8: 00 00 00 00
0x000006ac: 00 00 00 00
0x000006b0: 00 00 00 00
0x000006b4: 00 00 00 00
0x000006b8: 00 00 00 00
0x000006bc: 00 00 00 00
0x000006c0: 00 00 00 00
0x000006c4: 00 00 00 00
0x000006c8: 00 00 00 00
0x000006cc: 00 00 00 00
0x000006d0: 00 00 00 00
0x000006d4: 00 00 00 00
0x000006d8: 00 00 00 00
0x000006dc: 00 00 00 00
0x000006e0: 00 00 00 00
0x000006e4: 00 00 00 00
0x000006e8: 00 00 00 00
0x000006ec: 00 00 00 00
0x000006f0: 00 00 00 00
0x000006f4: 00 00 00 00
0x000006f8: 00 00 00 00
0x000006fc: 00 00 00 00
0x00000700: 00 00 00 00
0x00000704: 00 00 00 00
0x00000708: 00 00 00 00
0x0000070c: 00 00 00 00
0x00000710: 00 00 00 00
0x00000714: 00 00 00 00
0x00000718: 00 00 00 00
0x0000071c: 00 00 00 00
0x00000720: 00 00 00 00
0x00000724: 00 00 00 00
0x00000728: 00 00 00 00
0x0000072c: 00 00 00 00
0x00000730: 00 00 00 00
0x00000734: 00 00 00 00
0x00000738: 00 00 00 00
0x0000073c: 00 00 00 00
0x00000740: 00 00 00 00
0x00000744: 00 00 00 00
0x00000748: 00 00 00 00
0x0000074c: 00 00 00 00
0x00000750: 00 00 00 00
0x00000754: 00 00 00 00
0x00000758: 00 00 00 00
0x0000075c: 00 00 00 00
0x00000760: 00 00 00 00
0x00000764: 00 00 00 00
0x00000768: 00 00 00 00
0x0000076c: 00 00 00 00
0x00000770: 00 00 00 00
0x00000774: 00 00 00 00
0x00000778: 00 00 00 00
0x0000077c: 00 00 00 00
0x00000780: 00 00 00 00
0x00000784: 00 00 00 00
0x00000788: 00 00 00 00
0x0000078c: 00 00 00 00
0x00000790: 00 00 00 00
0x00000794: 00 00 00 00
0x00000798: 00 00 00 00
0x0000079c: 00 00 00 00
0x000007a0: 00 00 00 00
0x000007a4: 00 00 00 00
0x000007a8: 00 00 00 00
0x000007ac: 00 00 00 00
0x000007b0: 00 00 00 00
0x000007b4: 00 00 00 00
0x000007b8: 00 00 00 00
0x000007bc: 00 00 00 00
0x000007c0: 00 00 00 00
0x000007c4: 00 00 00 00
0x000007c8: 00 00 00 00
0x000007cc: 00 00 00 00
0x000007d0: 00 00 00 00
0x000007d4: 00 00 00 00
0x000007d8: 00 00 00 00
0x000007dc: 00 00 00 00
0x000007e0: 00 00 00 00
0x000007e4: 00 00 00 00
0x000007e8: 00 00 00 00
0x000007ec: 00 00 00 00
0x000007f0: 00 00 00 00
0x000007f4: 00 00 00 00
0x000007f8: 00 00 00 00
0x000007fc: 00 00 00 00
0x00000800: 00 00 00 00
0x00000804: 00 00 00 00
0x00000808: 00 00 00 00
0x0000080c: 00 00 00 00
0x00000810: 00 00 00 00
0x00000814: 00 00 00 00
0x00000818: 00 00 00 00

Text and Data Sections

BRISC-V Home BRISC-V Simulator Manual & Examples

BRISC-V Simulator
Adaptive and Secure Computing Systems Lab • Boston University

C source

```

1  int fib(int n) {
2      if (n <= 1) {
3          return n;
4      } else {
5          return fib(n-1)+fib(n-2);
6      }
7  }
8
9  int return_function (int result) {
10     return result;
11 }
12
13 int main(){
14     int n = 9;
15     int result = return_function (fib(n));
16     return result;
17 }

```

RISC-V Assembly

```

22  ecall
    # let's statically allocate a string "HELLO!"
    # we start this by creating a read-only data section
    .rodata
    .HELLO:
    # strings should end with the null terminator \0
    # the null terminator's binary value is 0!
    # we split HELLO!\0 into two 32bit words:
    # HELL and 0!00 - note that thats an "0!" and 2 zeros
    # we write HELL in ascii:
    # H - 0x48
    # E - 0x45
    # L - 0x4C
    # since this is a little-endian architecture, we
    # write HELL in reverse - LEHH
    .word 0x4C4C4548
    # now we write the second part 0!00
    .word 0x0000214F
    # this section is parsed when you load the program -
    # not when the instruction pointer runs over it.
    # as soon as you loaded the program, you should see
    # this string in the memory pane's data section,
    # somewhere close to the bottom of it.
    #
    # now we can go back to a text section that has code
    .text
    # print the string "HELLO!\n"
23  addi t0, zero, 3      # this is the string printing syscall
24  lui a0, %hi(.HELLO)   # this loads the top 20 bits
                        # of .HELLO address into a0
25  addi a0, a0, %lo(.HELLO) # this loads the bottom 12 bits
26  addi a1, zero, 7      # length of the string
27  ecall
    # print characters '!', '\n', '-'

```

Registers **Memory**

HEX DEC BINARY

0x00000130: 00 00 00 00
0x00000134: 00 00 00 00
0x00000138: 00 00 00 00
0x0000013c: 00 00 00 00
0x00000140: 00 00 00 00
0x00000144: 00 00 00 00
0x00000148: 00 00 00 00
0x0000014c: 00 00 00 00
0x00000150: 00 00 00 00
0x00000154: 00 00 00 00
0x00000158: 00 00 00 00
0x0000015c: 00 00 00 00
0x00000160: 00 00 00 00
0x00000164: 00 00 00 00
0x00000168: 00 00 00 00
0x0000016c: 00 00 00 00
0x00000170: 00 00 00 00
0x00000174: 00 00 00 00
0x00000178: 00 00 00 00
0x0000017c: 00 00 00 00
0x00000180: 00 00 00 00
0x00000184: 00 00 00 00
0x00000188: 00 00 00 00
0x0000018c: 00 00 00 00
0x00000190: 00 00 00 00
0x00000194: 00 00 00 00
0x00000198: 00 00 00 00
0x0000019c: 00 00 00 00
0x000001a0: 00 00 00 00
0x000001a4: 00 00 00 00
0x000001a8: 00 00 00 00
0x000001ac: 00 00 00 00
0x000001b0: 00 00 00 00
0x000001b4: 00 00 00 00
0x000001b8: 00 00 00 00
0x000001bc: 00 00 00 00
0x000001c0: 00 00 00 00
0x000001c4: 00 00 00 00
0x000001c8: 00 00 00 00
0x000001cc: 00 00 00 00
0x000001d0: 00 00 00 00
0x000001d4: 00 00 00 00
0x000001d8: 00 00 00 00
0x000001dc: 00 00 00 00
0x000001e0: 00 00 00 00
0x000001e4: 00 00 00 00
0x000001e8: 00 00 00 00
0x000001ec: 00 00 00 00
0x000001f0: 00 00 00 00
0x000001f4: 00 00 00 00
0x000001f8: 00 00 00 00
0x000001fc: 00 00 00 00
0x00000200: 00 00 00 00
0x00000204: 00 00 00 00
0x00000208: 00 00 00 00
0x0000020c: 00 00 00 00
0x00000210: 00 00 00 00
0x00000214: 00 00 00 00
0x00000218: 00 00 00 00
0x0000021c: 00 00 00 00
0x00000220: 00 00 00 00
0x00000224: 00 00 00 00
0x00000228: 00 00 00 00
0x0000022c: 00 00 00 00
0x00000230: 00 00 00 00
0x00000234: 00 00 00 00
0x00000238: 00 00 00 00
0x0000023c: 00 00 00 00
0x00000240: 00 00 00 00
0x00000244: 00 00 00 00
0x00000248: 00 00 00 00
0x0000024c: 00 00 00 00
0x00000250: 00 00 00 00
0x00000254: 00 00 00 00
0x00000258: 00 00 00 00
0x0000025c: 00 00 00 00
0x00000260: 00 00 00 00
0x00000264: 00 00 00 00
0x00000268: 00 00 00 00
0x0000026c: 00 00 00 00
0x00000270: 00 00 00 00
0x00000274: 00 00 00 00
0x00000278: 00 00 00 00
0x0000027c: 00 00 00 00
0x00000280: 00 00 00 00
0x00000284: 00 00 00 00
0x00000288: 00 00 00 00
0x0000028c: 00 00 00 00
0x00000290: 00 00 00 00
0x00000294: 00 00 00 00
0x00000298: 00 00 00 00
0x0000029c: 00 00 00 00
0x000002a0: 00 00 00 00
0x000002a4: 00 00 00 00
0x000002a8: 00 00 00 00
0x000002ac: 00 00 00 00
0x000002b0: 00 00 00 00
0x000002b4: 00 00 00 00
0x000002b8: 00 00 00 00
0x000002bc: 00 00 00 00
0x000002c0: 00 00 00 00
0x000002c4: 00 00 00 00
0x000002c8: 00 00 00 00
0x000002cc: 00 00 00 00
0x000002d0: 00 00 00 00
0x000002d4: 00 00 00 00
0x000002d8: 00 00 00 00
0x000002dc: 00 00 00 00
0x000002e0: 00 00 00 00
0x000002e4: 00 00 00 00
0x000002e8: 00 00 00 00
0x000002ec: 00 00 00 00
0x000002f0: 00 00 00 00
0x000002f4: 00 00 00 00
0x000002f8: 00 00 00 00
0x000002fc: 00 00 00 00
0x00000300: 00 00 00 00
0x00000304: 00 00 00 00
0x00000308: 00 00 00 00
0x0000030c: 00 00 00 00
0x00000310: 00 00 00 00
0x00000314: 00 00 00 00
0x00000318: 00 00 00 00
0x0000031c: 00 00 00 00
0x00000320: 00 00 00 00
0x00000324: 00 00 00 00
0x00000328: 00 00 00 00
0x0000032c: 00 00 00 00
0x00000330: 00 00 00 00
0x00000334: 00 00 00 00
0x00000338: 00 00 00 00
0x0000033c: 00 00 00 00
0x00000340: 00 00 00 00
0x00000344: 00 00 00 00
0x00000348: 00 00 00 00
0x0000034c: 00 00 00 00
0x00000350: 00 00 00 00
0x00000354: 00 00 00 00
0x00000358: 00 00 00 00
0x0000035c: 00 00 00 00
0x00000360: 00 00 00 00
0x00000364: 00 00 00 00
0x00000368: 00 00 00 00
0x0000036c: 00 00 00 00
0x00000370: 00 00 00 00
0x00000374: 00 00 00 00
0x00000378: 00 00 00 00
0x0000037c: 00 00 00 00
0x00000380: 00 00 00 00
0x00000384: 00 00 00 00
0x00000388: 00 00 00 00
0x0000038c: 00 00 00 00
0x00000390: 00 00 00 00
0x00000394: 00 00 00 00
0x00000398: 00 00 00 00
0x0000039c: 00 00 00 00
0x000003a0: 00 00 00 00
0x000003a4: 00 00 00 00
0x000003a8: 00 00 00 00
0x000003ac: 00 00 00 00
0x000003b0: 00 00 00 00
0x000003b4: 00 00 00 00
0x000003b8: 00 00 00 00
0x000003bc: 00 00 00 00
0x000003c0: 00 00 00 00
0x000003c4: 00 00 00 00
0x000003c8: 00 00 00 00
0x000003cc: 00 00 00 00
0x000003d0: 00 00 00 00
0x000003d4: 00 00 00 00
0x000003d8: 00 00 00 00
0x000003dc: 00 00 00 00
0x000003e0: 00 00 00 00
0x000003e4: 00 00 00 00
0x000003e8: 00 00 00 00
0x000003ec: 00 00 00 00
0x000003f0: 00 00 00 00
0x000003f4: 00 00 00 00
0x000003f8: 00 00 00 00
0x000003fc: 00 00 00 00
0x00000400: 00 00 00 00
0x00000404: 00 00 00 00
0x00000408: 00 00 00 00
0x0000040c: 00 00 00 00
0x00000410: 00 00 00 00
0x00000414: 00 00 00 00
0x00000418: 00 00 00 00
0x0000041c: 00 00 00 00
0x00000420: 00 00 00 00
0x00000424: 00 00 00 00
0x00000428: 00 00 00 00
0x0000042c: 00 00 00 00
0x00000430: 00 00 00 00
0x00000434: 00 00 00 00
0x00000438: 00 00 00 00
0x0000043c: 00 00 00 00
0x00000440: 00 00 00 00
0x00000444: 00 00 00 00
0x00000448: 00 00 00 00
0x0000044c: 00 00 00 00
0x00000450: 00 00 00 00
0x00000454: 00 00 00 00
0x00000458: 00 00 00 00
0x0000045c: 00 00 00 00
0x00000460: 00 00 00 00
0x00000464: 00 00 00 00
0x00000468: 00 00 00 00
0x0000046c: 00 00 00 00
0x00000470: 00 00 00 00
0x00000474: 00 00 00 00
0x00000478: 00 00 00 00
0x0000047c: 00 00 00 00
0x00000480: 00 00 00 00
0x00000484: 00 00 00 00
0x00000488: 00 00 00 00
0x0000048c: 00 00 00 00
0x00000490: 00 00 00 00
0x00000494: 00 00 00 00
0x00000498: 00 00 00 00
0x0000049c: 00 00 00 00
0x000004a0: 00 00 00 00
0x000004a4: 00 00 00 00
0x000004a8: 00 00 00 00
0x000004ac: 00 00 00 00
0x000004b0: 00 00 00 00
0x000004b4: 00 00 00 00
0x000004b8: 00 00 00 00
0x000004bc: 00 00 00 00
0x000004c0: 00 00 00 00
0x000004c4: 00 00 00 00
0x000004c8: 00 00 00 00
0x000004cc: 00 00 00 00
0x000004d0: 00 00 00 00
0x000004d4: 00 00 00 00
0x000004d8: 00 00 00 00
0x000004dc: 00 00 00 00
0x000004e0: 00 00 00 00
0x000004e4: 00 00 00 00
0x000004e8: 00 00 00 00
0x000004ec: 00 00 00 00
0x000004f0: 00 00 00 00
0x000004f4: 00 00 00 00
0x000004f8: 00 00 00 00
0x000004fc: 00 00 00 00
0x00000500: 00 00 00 00
0x00000504: 00 00 00 00
0x00000508: 00 00 00 00
0x0000050c: 00 00 00 00
0x00000510: 00 00 00 00
0x00000514: 00 00 00 00
0x00000518: 00 00 00 00
0x0000051c: 00 00 00 00
0x00000520: 00 00 00 00
0x00000524: 00 00 00 00
0x00000528: 00 00 00 00
0x0000052c: 00 00 00 00
0x00000530: 00 00 00 00
0x00000534: 00 00 00 00
0x00000538: 00 00 00 00
0x0000053c: 00 00 00 00
0x00000540: 00 00 00 00
0x00000544: 00 00 00 00
0x00000548: 00 00 00 00
0x0000054c: 00 00 00 00
0x00000550: 00 00 00 00
0x00000554: 00 00 00 00
0x00000558: 00 00 00 00
0x0000055c: 00 00 00 00
0x00000560: 00 00 00 00
0x00000564: 00 00 00 00
0x00000568: 00 00 00 00
0x0000056c: 00 00 00 00
0x00000570: 00 00 00 00
0x00000574: 00 00 00 00
0x00000578: 00 00 00 00
0x0000057c: 00 00 00 00
0x00000580: 00 00 00 00
0x00000584: 00 00 00 00
0x00000588: 00 00 00 00
0x0000058c: 00 00 00 00
0x00000590: 00 00 00 00
0x00000594: 00 00 00 00
0x00000598: 00 00 00 00
0x0000059c: 00 00 00 00
0x000005a0: 00 00 00 00
0x000005a4: 00 00 00 00
0x000005a8: 00 00 00 00
0x000005ac: 00 00 00 00
0x000005b0: 00 00 00 00
0x000005b4: 00 00 00 00
0x000005b8: 00 00 00 00
0x000005bc: 00 00 00 00
0x000005c0: 00 00 00 00
0x000005c4: 00 00 00 00
0x000005c8: 00 00 00 00
0x000005cc: 00 00 00 00
0x000005d0: 00 00 00 00
0x000005d4: 00 00 00 00
0x000005d8: 00 00 00 00
0x000005dc: 00 00 00 00
0x000005e0: 00 00 00 00
0x000005e4: 00 00 00 00
0x000005e8: 00 00 00 00
0x000005ec: 00 00 00 00
0x000005f0: 00 00 00 00
0x000005f4: 00 00 00 00
0x000005f8: 00 00 00 00
0x000005fc: 00 00 00 00
0x00000600: 00 00 00 00
0x00000604: 00 00 00 00
0x00000608: 00 00 00 00
0x0000060c: 00 00 00 00
0x00000610: 00 00 00 00
0x00000614: 00 00 00 00
0x00000618: 00 00 00 00
0x0000061c: 00 00 00 00
0x00000620: 00 00 00 00
0x00000624: 00 00 00 00
0x00000628: 00 00 00 00
0x0000062c: 00 00 00 00
0x00000630: 00 00 00 00
0x00000634: 00 00 00 00
0x00000638: 00 00 00 00
0x0000063c: 00 00 00 00
0x00000640: 00 00 00 00
0x00000644: 00 00 00 00
0x00000648: 00 00 00 00
0x0000064c: 00 00 00 00
0x00000650: 00 00 00 00
0x00000654: 00 00 00 00
0x00000658: 00 00 00 00
0x0000065c: 00 00 00 00
0x00000660: 00 00 00 00
0x00000664: 00 00 00 00
0x00000668: 00 00 00 00
0x0000066c: 00 00 00 00
0x00000670: 00 00 00 00
0x00000674: 00 00 00 00
0x00000678: 00 00 00 00
0x0000067c: 00 00 00 00
0x00000680: 00 00 00 00
0x00000684: 00 00 00 00
0x00000688: 00 00 00 00
0x0000068c: 00 00 00 00
0x00000690: 00 00 00 00
0x00000694: 00 00 00 00
0x00000698: 00 00 00 00
0x0000069c: 00 00 00 00
0x000006a0: 00 00 00 00
0x000006a4: 00 00 00 00
0x000006a8: 00 00 00 00
0x000006ac: 00 00 00 00
0x000006b0: 00 00 00 00
0x000006b4: 00 00 00 00
0x000006b8: 00 00 00 00
0x000006bc: 00 00 00 00
0x000006c0: 00 00 00 00
0x000006c4: 00 00 00 00
0x000006c8: 00 00 00 00
0x000006cc: 00 00 00 00
0x000006d0: 00 00 00 00
0x000006d4: 00 00 00 00
0x000006d8: 00 00 00 00
0x000006dc: 00 00 00 00
0x000006e0: 00 00 00 00
0x000006e4: 00 00 00 00
0x000006e8: 00 00 00 00
0x000006ec: 00 00 00 00
0x000006f0: 00 00 00 00
0x000006f4: 00 00 00 00
0x000006f8: 00 00 00 00
0x000006fc: 00 00 00 00
0x00000700: 00 00 00 00
0x00000704: 00 00 00 00
0x00000708: 00 00 00 00
0x0000070c: 00 00 00 00
0x00000710: 00 00 00 00
0x00000714: 00 00 00 00
0x00000718: 00 00 00 00
0x0000071c: 00 00 00 00
0x00000720: 00 00 00 00
0x00000724: 00 00 00 00
0x00000728: 00 00 00 00
0x0000072c: 00 00 00 00
0x00000730: 00 00 00 00
0x00000734: 00 00 00 00
0x00000738: 00 00 00 00
0x0000073c: 00 00 00 00
0x00000740: 00 00 00 00
0x00000744: 00 00 00 00
0x00000748: 00 00 00 00
0x0000074c: 00 00 00 00
0x00000750: 00 00 00 00
0x00000754: 00 00 00 00
0x00000758: 00 00 00 00
0x0000075c: 00 00 00 00
0x00000760: 00 00 00 00
0x00000764: 00 00 00 00
0x00000768: 00 00 00 00
0x0000076c: 00 00 00 00
0x00000770: 00 00 00 00
0x00000774: 00 00 00 00
0x00000778: 00 00 00 00
0x0000077c: 00 00 00 00
0x00000780: 00 00 00 00
0x00000784: 00 00 00 00
0x00000788: 00 00 00 00
0x0000078c: 00 00 00 00
0x00000790: 00 00 00 00
0x00000794: 00 00 00 00
0x00000798: 00 00 00 00
0x0000079c: 00 00 00 00
0x000007a0: 00 00 00 00
0x000007a4: 00 00 00 00
0x000007a8: 00 00 00 00
0x000007ac: 00 00 00 00
0x000007b0: 00 00 00 00
0x000007b4: 00 00 00 00
0x000007b8: 00 00 00 00
0x000007bc: 00 00 00 00
0x000007c0: 00 00 00 00
0x000007c4: 00 00 00 00
0x000007c8: 00 00 00 00
0x000007cc: 00 00 00 00
0x000007d0: 00 00 00 00
0x000007d4: 00 00 00 00
0x000007d8: 00 00 00 00
0x000007dc: 00 00 00 00
0x000007e0: 00 00 00 00
0x000007e4: 00 00 00 00
0x000007e8: 00 00 00 00
0x000007ec: 00 00 00 00
0x000007f0: 00 00 00 00
0x000007f4: 00 00 00 00
0x000007f8: 00 00 00 00
0x000007fc: 00 00 00 00
0x00000800: 00 00 00 00
0x00000804: 00 00 00 00
0x00000808: 00 00 00 00
0x0000080c: 00 00 00 00
0x00000810: 00 00 00 00
0x00000814: 00 00 00 00
0x00000818: 00 00 00 00
0x0000081c: 00 00 00 00
0x00000820: 00 00 00 00
0x0

System Calls

- We also provide some simple system calls
- System calls are used for functionalities provided by the operating system
 - Think file systems, IO, etc.
- In RISC-V, system calls look something like:
 - Put the type of system call you want in register t0
 - More about that on the next slide
 - Put any arguments you may have in a0 and a1
 - Call instruction ECALL
 - If the system call has return values, they will be in a0
- To really get familiar with syscalls, try running the example syscall file in the simulator

Supported Syscalls

Syscall	Syscall ID (put this in t0)	Description
Print integer	1	Print integer value in a0 to console
Print char	2	Print ascii value in a0 to console
Print string	3	Print string with address in a0 and length in a1 to console
Read integer	4	Read integer from console into a0
Read char	5	Read character from console into a0 as an ascii value
Read string	6	Read string of length given in a1 from console and store it at address in a0
SBRK	7	Dynamically allocate the amount of bytes specified in a0. The pointer to the beginning of the newly allocated memory will be stored in a0. The value in a0 can be negative, if you want to deallocate some memory!

The screenshot shows a code editor with assembly code for a GCD function. The code is written in MIPS assembly with C comments. The register file on the right shows the state of the registers.

```

1  addi sp,zero,1536
2  call main
3  addi sp,zero,0
4  mv s0,a0
5  addi zero,zero,0
6  addi zero,zero,0
7  .type ra,0
8  .align 2
9  .globl gcd
10 .file "gcd.c"
    .option nopic
    .text
    .align 2
    .globl gcd
    .type gcd, @function
11 gcd:
12     addi sp,sp,-16
13     sw ra,44(sp)
14     sw s0,40(sp)
15     addi s0,sp,48
16     sw a0,-36(s0)
17     sw a1,-40(s0)
18     sw a2,-44(s0)
19     bne a4,a5,.L2
20     lw a5,-36(s0)
21     sw a5,-20(s0)
22     j .L3
23     lw a5,-40(s0)
24     sw a5,-24(s0)
25     lw a5,-44(s0)
26     bne a4,a5,.L2

```

Register File:

sp	[2]	1456
tp	[4]	0
t1		
fp		1536
a0	[10]	0
a2	[12]	0
a4	[14]	0
a6	[16]	0
s2	[18]	0
s4	[20]	0
s6	[22]	0
s8	[24]	0
s10	[26]	0
t3	[28]	0
t5	[30]	0

Instruction breakdown:

```

1  addi sp,zero,1536
2  call main
3  addi sp,zero,0
4  mv s0,a0
5  addi zero,zero,0
6  addi zero,zero,0
7  .type ra,0
8  .align 2
9  .globl gcd
10 .file "gcd.c"
    .option nopic
    .text
    .align 2
    .globl gcd
    .type gcd, @function
11 gcd:
12     addi sp,sp,-16
13     sw ra,44(sp)
14     sw s0,40(sp)
15     addi s0,sp,48
16     sw a0,-36(s0)
17     sw a1,-40(s0)
18     sw a2,-44(s0)
19     bne a4,a5,.L2
20     lw a5,-36(s0)
21     sw a5,-20(s0)
22     j .L3
23     lw a5,-40(s0)
24     sw a5,-24(s0)
25     lw a5,-44(s0)
26     bne a4,a5,.L2

```

Bugs and Features

- Found a bug? Have an issue? Want a feature/RISC-V extension?
 - Send an email to briscv.feedback@gmail.com
- Thanks!