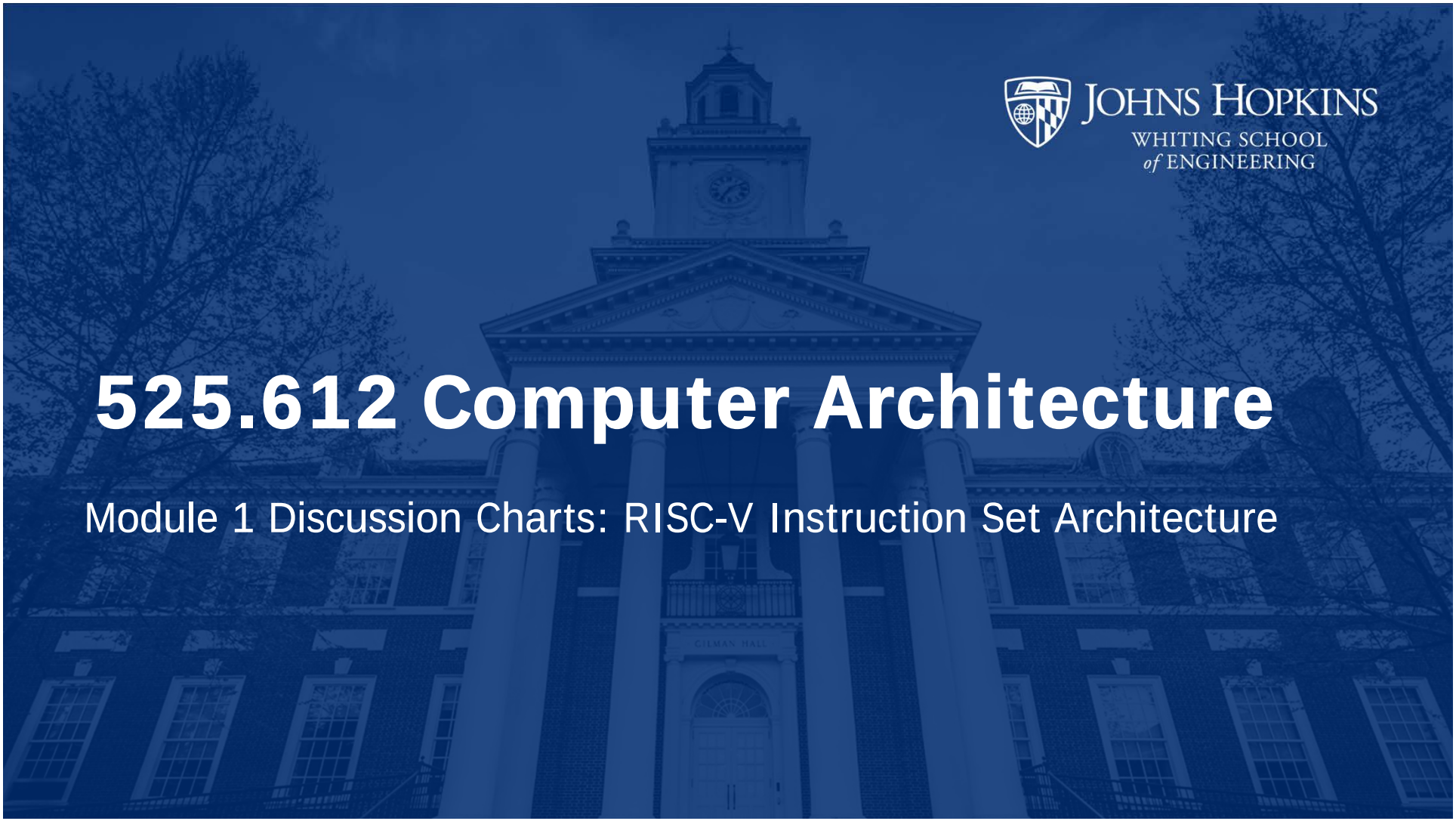


525.612 Computer Architecture

Module 1 Discussion Charts: RISC-V Instruction Set Architecture

Goals of this module

- RISC-V Architecture:
  - Review RISC-V Instructions
  - In later discussion forums we will cover Li style implementations:
    - single cycle CPU,
    - multi-cycle CPU,

RISC-V Organization

- An important difference between MIPS and RISC-V:
  - MIPS appears to be a monolithic set of instructions, covering integer and floating point
  - RISC-V is broken to 15 different classes
    - RV32I Base Integer Instructions – Integer operations not including multiply and divide. There are also 64 and 128 bit versions
    - RV32M Multiply Extension – Multiply and Divide. There is no need for HI/LO in RISC-V. There is also 64 bit versions
    - RV32A Atomic memory operations (load, store and read modify write)
    - RV32F Floating Point (there are double and quad precision as well)

RISC-V Assembly language program example

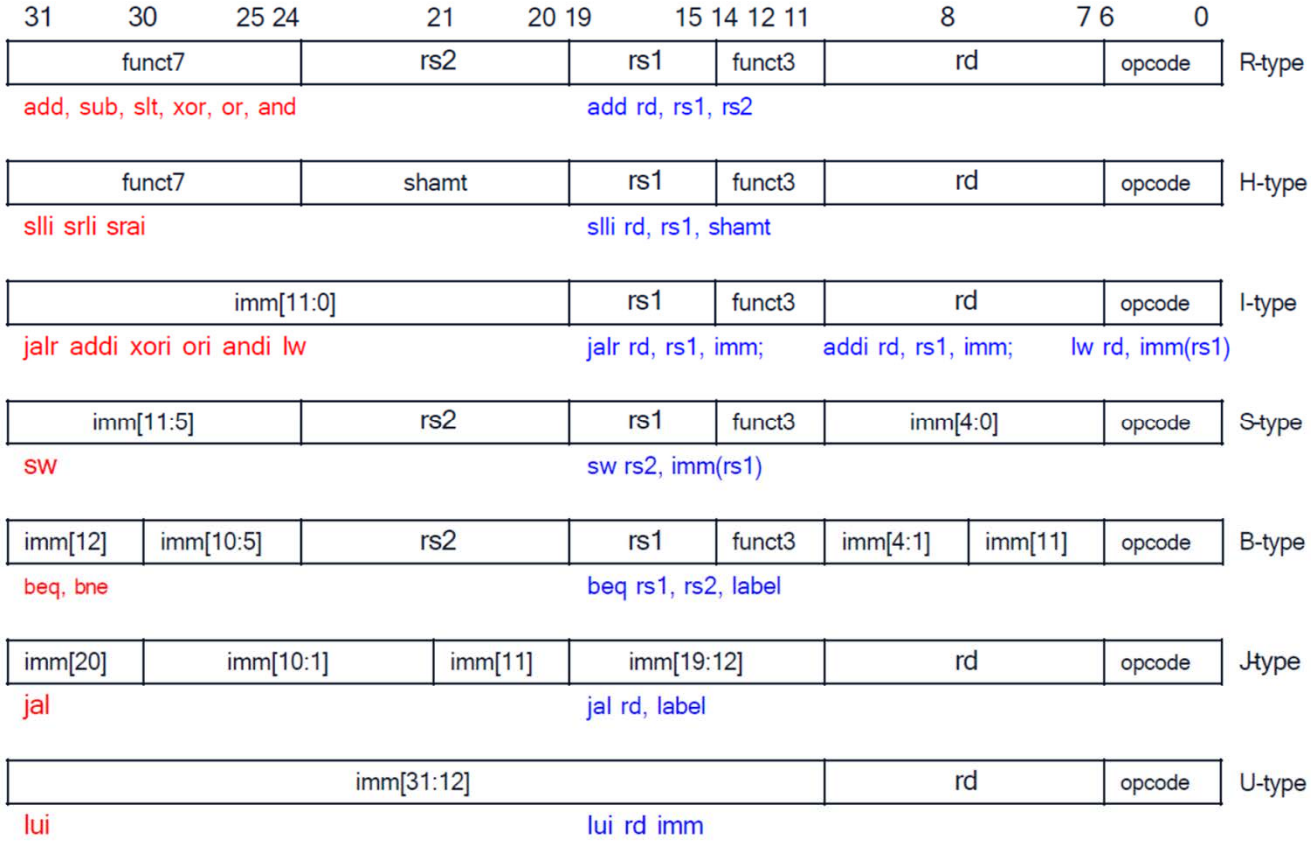
```
.text                          # code segment
main:                          # program entry
      addi x4, x0, 12          # reg[4] <= reg[0] + 12 = 12
      addi x5, x0, 13          # reg[5] <= reg[0] + 13 = 13
      add x6, x4, x5           # reg[6] <= reg[4] + reg[5]
.end                           # end of program
```

.text	code segment
# * * *	comment
main:	label (main: is the program entrance)
addi, add	RISC-V order
x0, x4, x5, x6	RISC-V register (register x0 is always 0)
.end	the end

RISC-V instruction example

```
1. add     rd, rs1, rs2          # rd <- rs1 + rs2
2. sub     rd, rs1, rs2          # rd <- rs1 - rs2
3. slt     rd, rs1, rs2          # rd <- rs1 < rs2 (signed)
4. xor     rd, rs1, rs2          # rd <- rs1 ^ rs2
5. or      rd, rs1, rs2          # rd <- rs1 | rs2
6. and     rd, rs1, rs2          # rd <- rs1 & rs2
7. slli rd, rs1, shamt           # rd <- rs1 << shamt
8. srli rd, rs1, shamt           # rd <- rs1 >> shamt
9. srai rd, rs1, shamt           # rd <- rs1 >>> shamt
10. jalr rd, rs1, imm            # rd <- pc+4; pc <- rs1+imm
11. addi rd, rs1, imm            # rd <- rs1 + imm
12. xori rd, rs1, imm            # rd <- rs1 ^ imm
13. ori rd, rs1, imm             # rd <- rs1 |
14. andi rd, rs1, imm            # rd <- rs1 & imm
15. lw     rd, imm(rs1)          # rd <- memory[rs1+imm]
16. sw     rs2, imm(rs1)         # memory[rs1+imm] <- rs2
17. beq    rs1, rs2, label # if (rs1==rs2) pc <- label rs1, rs2,
18. bne    label # if (rs1!=rs2) pc <- label rd,
19.jal             labels        # rd <- pc+4; pc <- label
20.lui     rd,     imm           # rd <- imm,000000000000
```

RISC-V Instruction Format



Register Definition

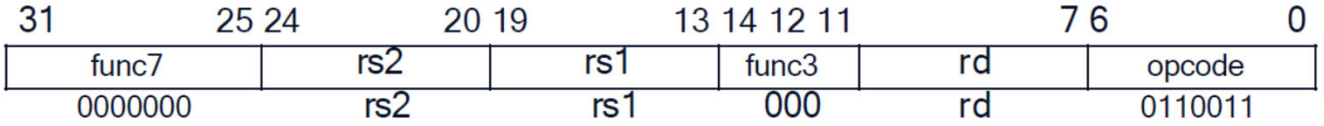
Register	ABI Name	Description
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary / alternate link register
x6-7	t1-2	Temporaries
x8	s0/fp	Saved register / frame pointer
x9	s1	Saved register
x10-11	a0-1	Function arguments / return
x12-17	a2-7	values
x18-27	s2-11	Saved registers
x28-31	t3-6	Temporaries

RV32I Base Instruction Set Encoding

0000000	rs2	rs1	000	rd	0110011
0100000	rs2	rs1	000	rd	0110011
0000000	rs2	rs1	010	rd	0110011
0000000	rs2	rs1	100	rd	0110011
0000000	rs2	rs1	110	rd	0110011
0000000	rs2	rs1	111	rd	0110011
0000000	shamt	rs1	001	rd	0010011
0000000	shamt	rs1	101	rd	0010011
0100000	shamt	rs1	101	rd	0010011
imm[11:0]		rs1	000	rd	1100111
imm[11:0]		rs1	000	rd	0010011
imm[11:0]		rs1	100	rd	0010011
imm[11:0]		rs1	110	rd	0010011
imm[11:0]		rs1	111	rd	0010011
imm[11:0]		rs1	010	rd	0000011
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011
imm[20|10:1|11|19:12]				rd	1101111
imm[31:12]				rd	0110111

1. add
2. sub
3. slt
4. xor
5. or
6. and
7. slli
8. srli
9. srai
10. jalr
11. addi
12. xori
13. ori
14. andi
15. lw
16. sw
17. beq
18. bne
19. jal
20. lui

1. add (Add)



add rd, rs1, rs2 # rd <- rs1 + rs2

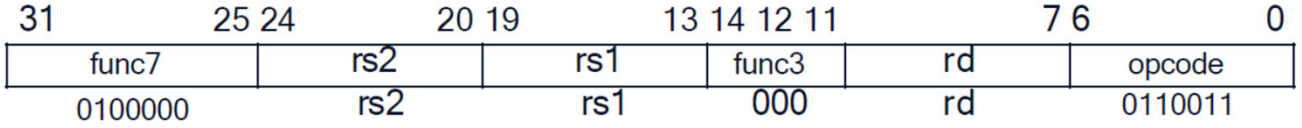
Instruction meaning: Add the values stored in register rs1and register rs2 and store the result in register rd for example:

```
add x8, x6, x7
```

For example,

if reg[6] = 2
reg[7] = 3
then reg[8] = 0000 0000 0000 0000 0000 0000 0000 0010 (2)
+ 0000 0000 0000 0000 0000 0000 0000 0011 = (3)
0000 0000 0000 0000 0000 0000 0000 0101 (5)

2.sub (Subtraction)



sub rd, rs1, rs2 # rd <- rs1 - rs2

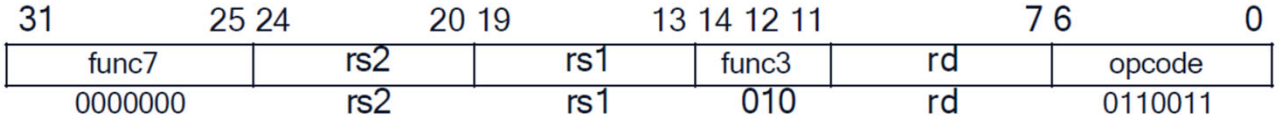
Instruction meaning: Subtract register rs1value from register value stored in rs2 and store the result in register rd For example:

```
sub x8, x6, x7
```

For example,

if reg[6] = 2
reg[7] = 3
then reg[8] = 0000 0000 0000 0000 0000 0000 0000 0010 (2)
- 0000 0000 0000 0000 0000 0000 0000 0011 = (3)
1111 1111 1111 1111 1111 1111 1111 1111 (-1)

3. slt (Set on Less Than)



slt rd, rs1, rs2 # rd <- rs1 < rs2 (signed)

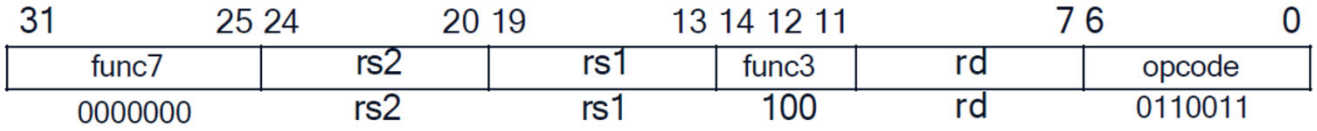
Instruction meaning: register rs1value and register value stored in rs2 are compared and if the value is less than, a 1 is stored in register rd. for example:

```
slt x8, x6, x7
```

For example,

if reg[6] = 2
reg[7] = 3
then reg[8] = 1 because 2 < 3 is true if
reg[6] = 3
reg[7] = 2
then reg[8] = 0 because 3 < 2 is false

4. xor (Exclusive Or)



xor rd, rs1, rs2 # rd <- rs1 ^ rs2

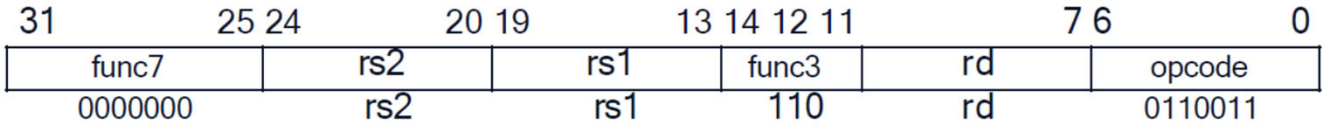
Instruction meaning: register rs1 values is XOR with register value stored in rs2 and the result is stored in register rd. For example:

```
xor x8, x6, x7
```

For example,

if reg[6] = 6
reg[7] = 3
then reg[8] = 0000 0000 0000 0000 0000 0000 0000 0110 (6)
^ 0000 0000 0000 0000 0000 0000 0000 0011 = (3)
0000 0000 0000 0000 0000 0000 0000 0101 (5)

5. or (Or)



or rd, rs1, rs2 # rd <- rs1 | rs2

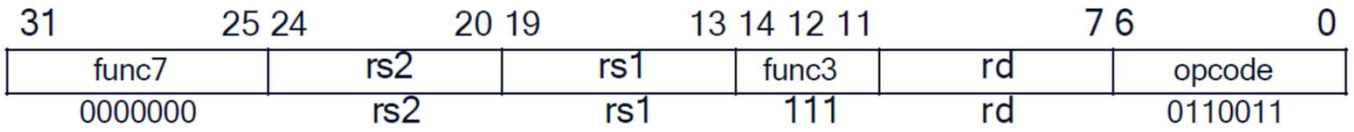
Instruction meaning:Contents of register rs1 and register rs2 are OR'd and the value is stored in rd For example:

```
or    x8, x6, x7
```

For example,

```
if      reg[6] = 6
        reg[7] = 3
then reg[8] = 0000 0000 0000 0000 0000 0000 0000 0110      (6)
            | 0000 0000 0000 0000 0000 0000 0000 0011 =    (3)
            0000 0000 0000 0000 0000 0000 0000 0111        (7)
```

6. and (And)



and rd, rs1, rs2 # rd <- rs1 & rs2

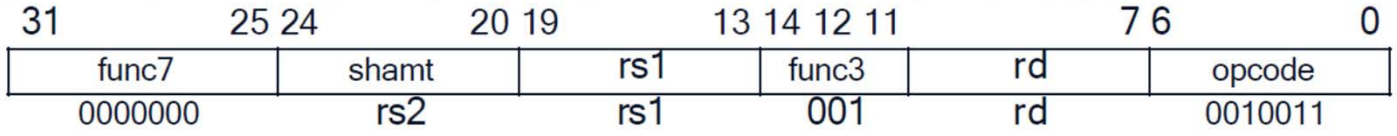
Instruction meaning: The value in register rs1 is AND'd with the value in registerrs2and stored in register rd for example:

and x8, x6, x7

For example,

if reg[6] = 6
reg[7] = 3
then reg[8] = 0000 0000 0000 0000 0000 0000 0000 0110 (6)
& 0000 0000 0000 0000 0000 0000 0000 0011 = (3)
0000 0000 0000 0000 0000 0000 0000 0010 (2)

7.slli (Shift Left Logical Immediate)



slli rd, rs1, shamt # rd <- rs1 << shamt

Instruction meaning: register value in rs1 is shifted left logical by the amount in shamt bits and the result is stored in register rd. For example:

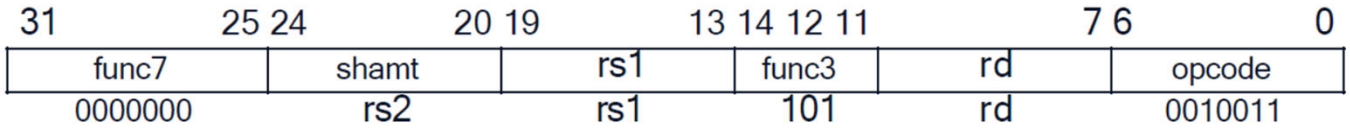
```
slli x8, x6, 4
```

For example,

if reg[6] = 0000 0000 0000 0000 0000 0000 0000 0010

then reg[8] = 0000 0000 0000 0000 0000 0000 0000 0010 << 4
= 0000 0000 0000 0000 0000 0000 0010 0000

8. srli (Shift Right Logical Immediate)



srli rd, rs1, shamt # rd <- rs1 >> shamt

Instruction meaning: The value in register rs1 is shifted right logical by shamt bits to the right and stored in register rd. For example:

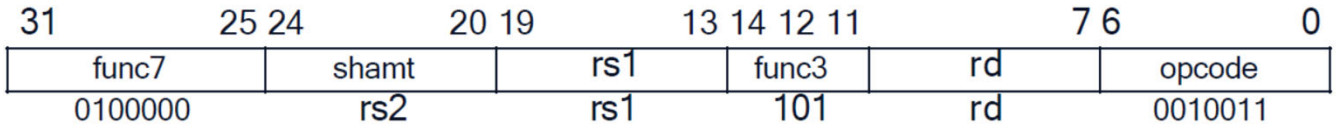
```
srli x8, x6, 4
```

For example,

if reg[6] = 1111 0000 0000 0000 0000 0000 0000 0010

then reg[8] = 1111 0000 0000 0000 0000 0000 0000 0010 >>4
= 0000 1111 0000 0000 0000 0000 0000 0000

9. srai (Shift Right Arithmetic Imm.)



srai rd, rs1, shamt # rd <- rs1 >>> shamt

Instruction meaning: The value in register rs1 is shifted right arithmetic by the amount shamt and stored the result in register rd. For example:

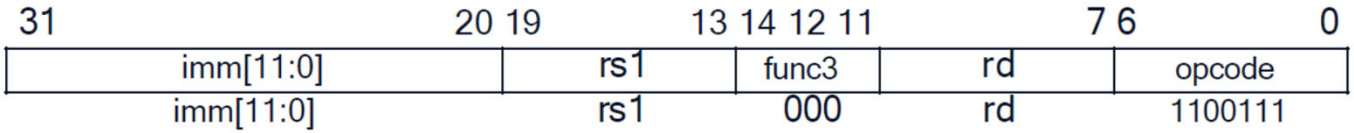
```
srai x8, x6, 4
```

For example,

if reg[6] = 1111 0000 0000 0000 0000 0000 0000 0010

then reg[8] = 1111 0000 0000 0000 0000 0000 0000 0010 >>> 4
= 1111 1111 0000 0000 0000 0000 0000 0000

10. jalr (Jump And Link Register)



jalr rd, rs1, imm # rd <- pc+4; pc <- (rs1+$signed(imm[11:0]))&0xfffffffe

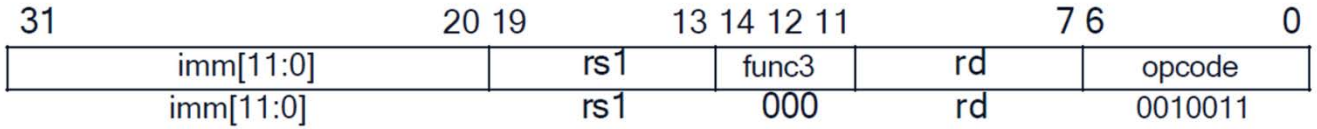
Instruction meaning: The value of pc+4 is stored in register rd and the value in register rs1 is added to the immediate value imm in two's complement form and the least significant bit is set to zero, and the result is used as the jump destination address. For example:

```
jalr x0, x1, 0          # = ret ra
```

For example,
if reg[1] = 0000 0000 0000 0000 0000 0000 1111 0001

then pc = 0000 0000 0000 0000 0000 0000 1111 0000

11. addi (Add Immediate)



addi rd, rs1, imm # rd <- rs1 + $signed(imm[11:0])

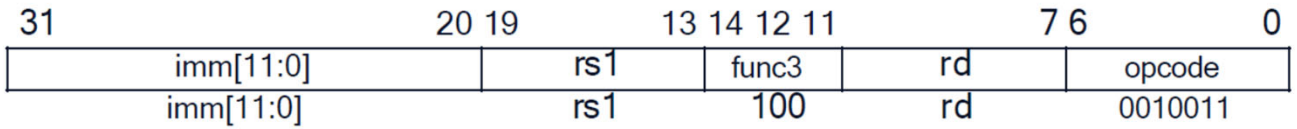
Instruction meaning: The value stored in register rs1 Is ADD'd by the immediate sign extended value imm represented in two's complement and stored in register rd.
For example:

addi x8, x6, -1

For example,
if reg[6] = 2

then reg[8] = 0000 0000 0000 0000 0000 0000 0000 0010 (2)
+ 1111 1111 1111 1111 1111 1111 1111 1111 = (-1)
0000 0000 0000 0000 0000 0000 0000 0001 (1)

12. xori (Exclusive Or Immediate)



xori rd, rs1, imm # rd <- rs1 ^ $signed(imm[11:0])

Instruction meaning: The value in register rs1 is XOR'd with the signed immediate value imm represented in two's complement form and stored in register rd. For example:

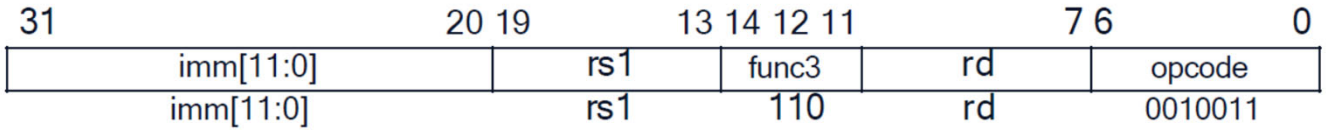
```
xori x8, x6, 0xf
```

For example,

```
if reg[6]    =   xxxx xxxx xxxx xxxx xxxx xxxx xxxx 1010

then reg[8]  =   xxxx xxxx xxxx xxxx xxxx xxxx xxxx 1010
             ^   0000 0000 0000 0000 0000 0000 0000 1111
             =   xxxx xxxx xxxx xxxx xxxx xxxx xxxx 0101
```

13. ori (Or Immediate)



ori rd, rs1, imm # rd <- rs1 | $signed(imm[11:0])

Instruction meaning: The value stored in register rs1 is OR'd with the signed immediate value imm represented in two's complement form and stored in the register rd. For example:

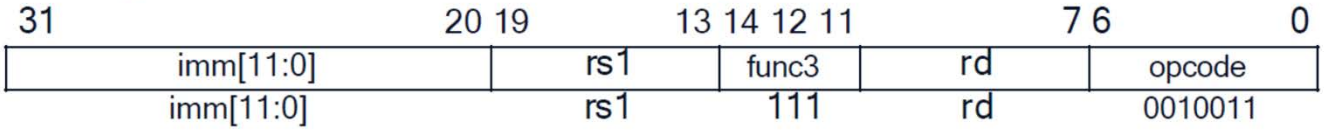
```
ori x8, x6, 0xf
```

For example,

```
if reg[6]   =  xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx

then reg[8] =  xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
               0000 0000 0000 0000 0000 0000 0000 1111 =
               xxxx xxxx xxxx xxxx xxxx xxxx xxxx 1111
```

14. andi (And Immediate)



andi rd, rs1, imm # rd <- rs1 & $signed(imm[11:0])

Instruction meaning: The value stored in register rs1 is AND'd by the signed immediate value imm represented in two's complement form and stored in the register rd. For example:

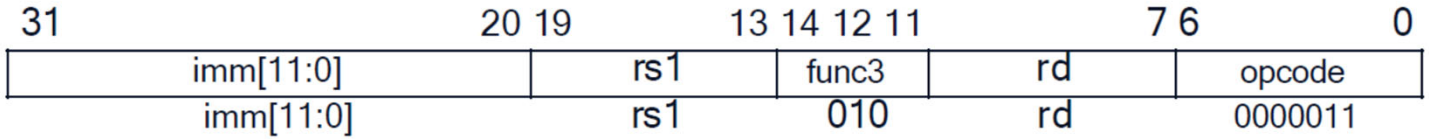
```
andi x8, x6, 0xf
```

For example,

```
if reg[6]    =  xxxx xxxx xxxx xxxx xxxx xxxx xxxx 1010

then reg[8]  =  xxxx xxxx xxxx xxxx xxxx xxxx xxxx 1010
             &  0000 0000 0000 0000 0000 0000 0000 1111 =
                0000 0000 0000 0000 0000 0000 0000 1010
```

15. lw (Load Word)



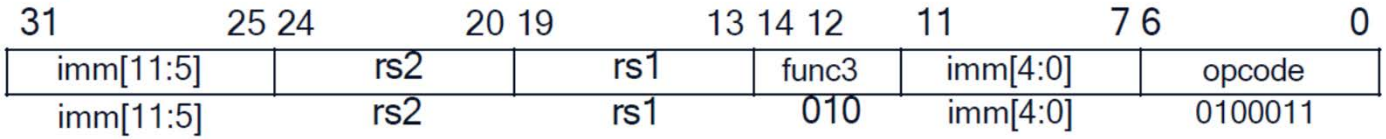
lw rd, imm(rs1) # rd <- memory[rs1 + $signed(imm[11:0])]

Instruction meaning: The register value stored in rs1 is added to the signed immediate two's complement value imm and represents the effective memory address, and the accessed memory data is stored in register rd For example:

```
lw    x8, 4(x6)
```

For example,
if reg[6] = 12 and mem[16] = 3
then reg[8] = mem[12+4] = mem[16] = 3

16. sw (Store Word)




```
sw    rs2, imm(rs1)      # memory[rs1 + $signed(imm[11:0])] <- rs2
```

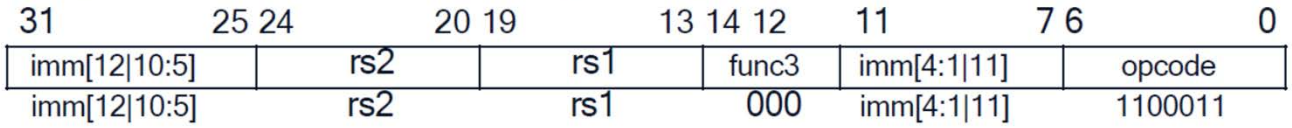
Instruction meaning: The register value stored in rs1 is added to the signed immediate two's complement value imm and represents the effective memory address, and the value in register rs2 is written to the memory location. For example:

```
sw    x8, 4(x6)
```

For example,

```
if      reg[6] = 12    reg[8] = 3
then mem[12+4] = mem[16] = reg[8] = 3
```

17. beq (Branch on Equal)



beq rs1, rs2, label # if (rs1 == rs2) pc <- pc + $signed({imm[12:1],0})

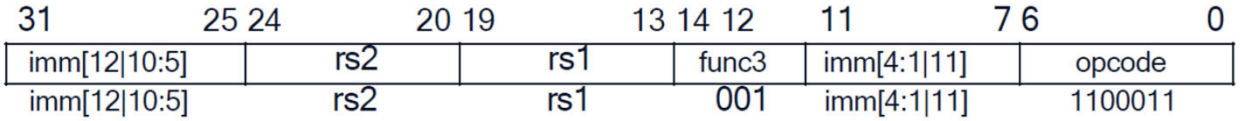
Instruction meaning: The register value rs1 and the register value rs2 are compared to see if they are equal. If equal the branch is taken. The branch destination address is pc + $signed({imm[12:1],0}) For example:

```
beq x8, x0, label
```

For example,

```
if reg[8] == 0 (equal, note: reg[0] = 0)
        then
            pc = pc + $signed({imm[12:1],0})
        else
            pc = pc + 4
```

18.bne (Branch on Not Equal)



bne rs1, rs2, label # if (rs1 != rs2) pc <- pc + $signed({imm[12:1],0})

Instruction meaning: The register value in rs1 and the register value in rs2 are compared to see if they are not equal. If not equal, the branch is taken. The branch destination address is pc + $signed({imm[12:1],0}) For example:

```
bne x8, x0, label
```

For example,

```
if reg[8] != 0 (equal, note: reg[0] = 0)
        then
                pc = pc + $signed({imm[12:1],0})
        else
                pc = pc + 4
```

19.jal (Jump And Link)

jal rd, label # rd <- pc + 4; pc <- pc + $signed({imm[20:1],0})

Instruction meaning: pc+4 store in Register rd as a return address. pc is modified by the immediate value represented in 2's complement imm and is used as the jump destination address.

example:

jal x1, subroutine # = call subroutine

For example,

```
reg[1] = pc + 4          # save return address, use ret x1 to return
pc = subroutine          # jump to subroutine
```

20. lui (Load Upper Immediate)

lui rd, imm # rd <- {imm[31:12],000000000000}

Instruction meaning: Register rd will hold the upper value:
immediate {imm[31:12],000000000000}

example:

```
lui x8, 0xfffff
```

For example,
reg[8] = 0xfffff000

20 RISC-V Command summary

```
1. add    rd, rs1, rs2          # rd <- rs1 + rs2
2. sub    rd, rs1, rs2          # rd <- rs1 - rs2
3. slt    rd, rs1, rs2          # rd <- rs1 < rs2 (signed)
4. xor    rd, rs1, rs2          # rd <- rs1 ^ rs2
5. or     rd, rs1, rs2          # rd <- rs1 | rs2
6. and    rd, rs1, rs2          # rd <- rs1 & rs2
7. slli rd, rs1, shamt          # rd <- rs1 << shamt
8. srli rd, rs1, shamt          # rd <- rs1 >> shamt
9. srai rd, rs1, shamt          # rd <- rs1 >>> shamt
10. jalr rd, rs1, imm           # rd <- pc+4; pc <- rs1+imm
11. addi rd, rs1, imm           # rd <- rs1 + imm
12. xori rd, rs1, imm           # rd <- rs1 ^ imm
13. ori rd, rs1, imm            # rd <- rs1 |
14. andi rd, rs1, imm           # rd <- rs1 & imm
15. lw    rd, imm(rs1)          # rd <- memory[rs1+imm]
16. sw    rs2, imm(rs1)         # memory[rs1+imm] <- rs2
17. beq   rs1, rs2, label # if (rs1==rs2) pc <- label rs1, rs2,
18. bne   label # if (rs1!=rs2) pc <- label rd,
19.jal          labels          # rd <- pc+4; pc <- label
20.lui    rd,   imm             # rd <- imm,000000000000
```

Example Program (sum.c)

emacs@localhost

File Edit Options Buffers Tools C Help

```
#include<stdio.h>
int main() {
    int a[] = {6, 5, -3};
    int i;
    int s = 0;
    for (i = 0; i < 3; i++) {
        s = s + a[i];
    }
    printf("sum = %d\n", s);
    return 0;
}
```

-:--- sum.c All L11 (C/l Abbrev)

Program Editor

sum.s

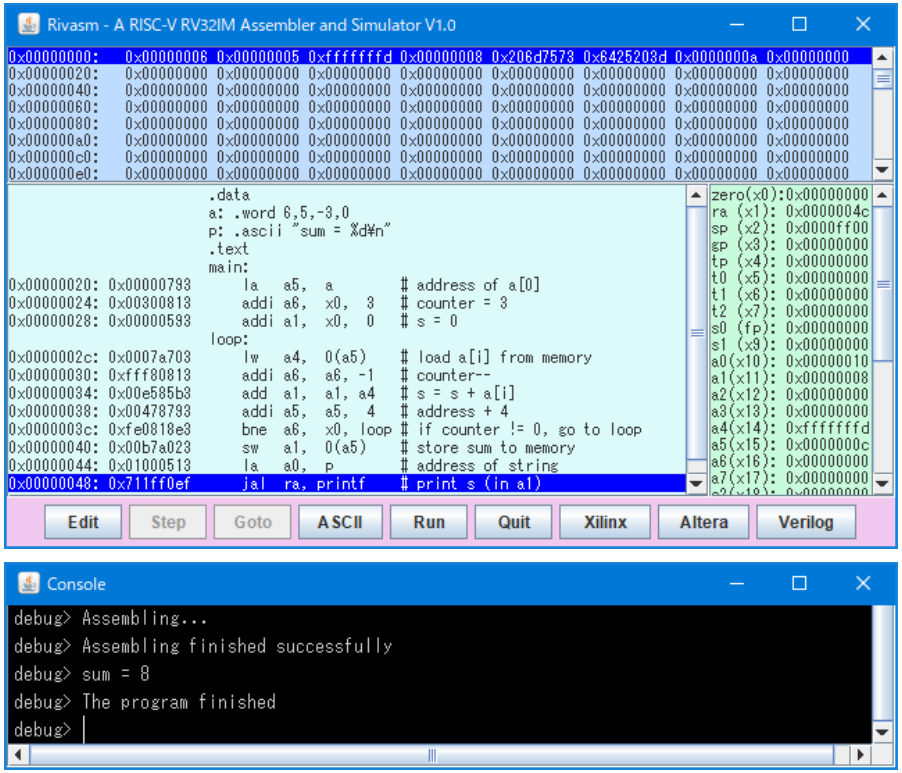
```
1  .data
2  a: .word 6,5,-3,0
3  p: .ascii "sum = %d\n"
4  .text
5  main:
6      la   a5,  a        # address of a[0]
7      addi a6,  x0,  3   # counter = 3
8      addi a1,  x0,  0   # s = 0
9  loop:
10     lw   a4,  0(a5)    # load a[i] from memory
11     addi a6,  a6, -1   # counter--
12     add  a1,  a1, a4   # s = s + a[i]
13     addi a5,  a5,  4   # address + 4
14     bne  a6,  x0, loop # if counter != 0, go to loop
15     sw   a1,  0(a5)    # store sum to memory
16     la   a0,  p        # address of string
17     jal  ra, printf    # print s (in a1)
18 .end
19
```

Open | Save | Assemble | Samples | Instructions

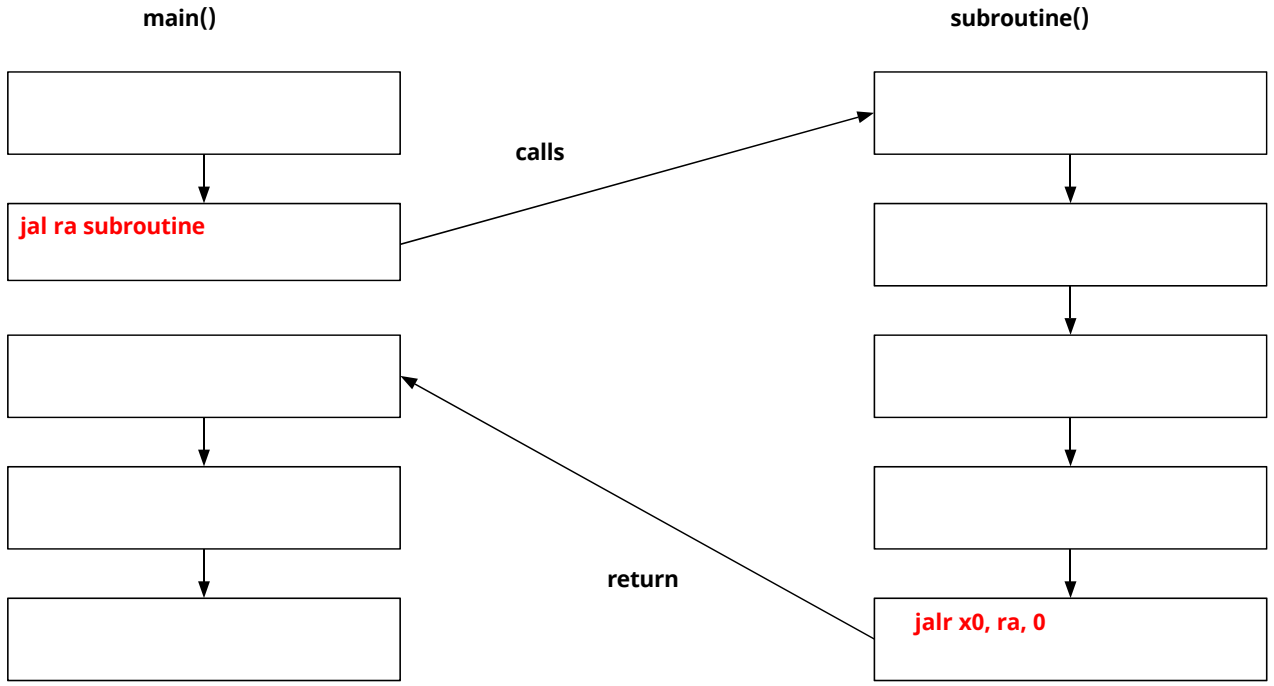
- For loops are implemented in RISC-V assembly
- Counter goes down from 3
- Address pointer incremented by 4

RISC-V Simulator (Rivasm)

- There are multiple RISC-V simulators available
- Li's Rivasm is a java based program
- Many additional simulators are available:
  - https://github.com/TheThirdOne/rars
  - https://github.com/ThaumicMekanism/venus



RISC-V Subroutine Call



Subroutine call instruction:jal

jal rd, label # rd <- pc+4; pc <- pc + $signed({imm[20:1],0})

Instruction meaning: rd will store the return address (pc+4) and the program counter will load pc with the immediate value in two's complement imm added to pc, and the result is used as the jump destination address.

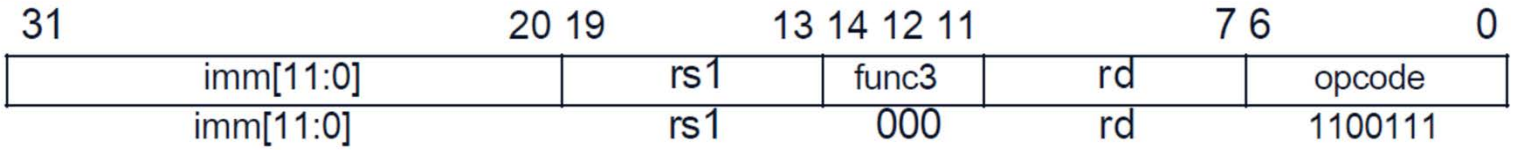
example:

```
jal x1, subroutine # = call subroutine
```

For example,

```
reg[1] = pc + 4          # save return address, use ret x1 to return
pc = subroutine          # jump to subroutine
```

Instructions to return from a subroutine: jalr



jalr rd, rs1, imm # rd <- pc+4; pc <- (rs1+$signed(imm[11:0]))&0xfffffffe

Instruction meaning: rd will store the next address (pc+4). The program counter will store register rs1 added to immediate value in two's complement imm. The least significant bit is set to zero, and the result is used as the jump destination address.

For example:

```
jalr x0, x1, 0          # = ret ra
```

For example,
if reg[1] = 0000 0000 0000 0000 0000 0000 1111 0001 then
pc = 0000 0000 0000 0000 0000 0000 1111 0000

Subroutine call (Example in C)

```
# include <stdio.h>
int sum (int a, int b) { // subroutine
      return a + b;
}
int main() {
      int x = 7, y = 5; int
      z;
      z = sum (x, y); // call subroutine
      printf("z = %d\n", z);
      return 0;
}
```

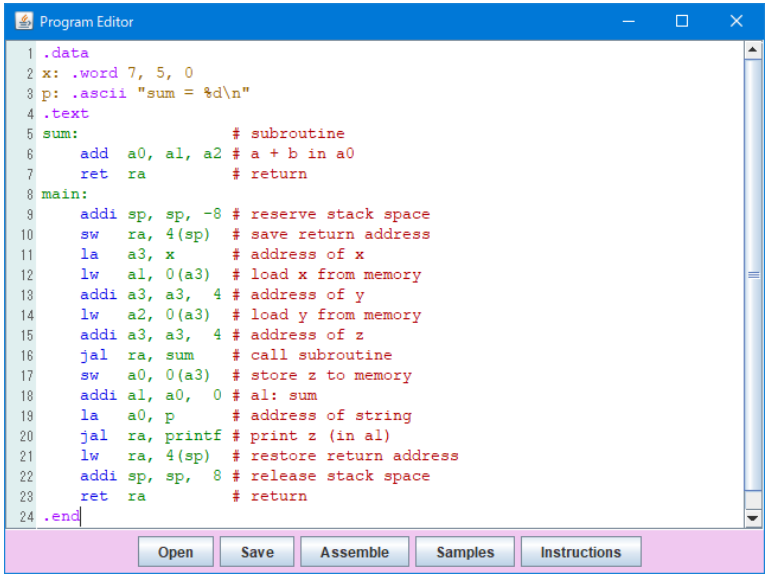
Compile and execute on PC:

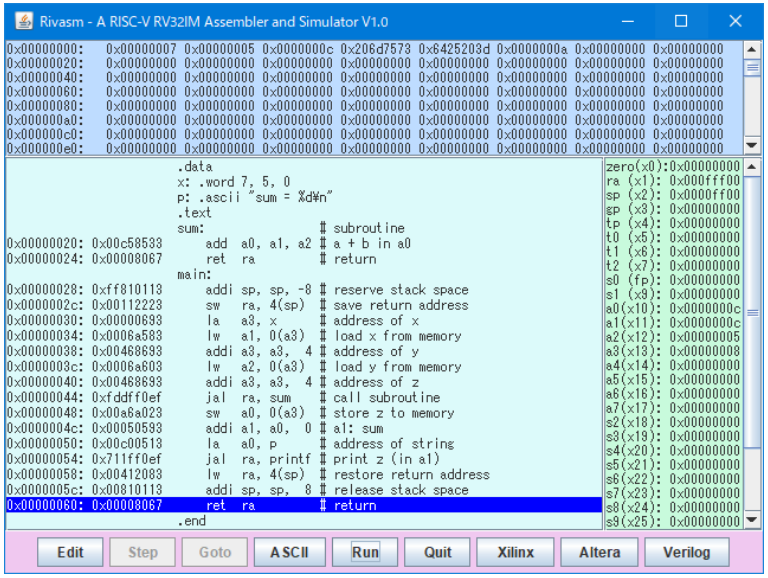
$ gcc subroutine.c -o subroutine
$ ./ subroutine
z=12

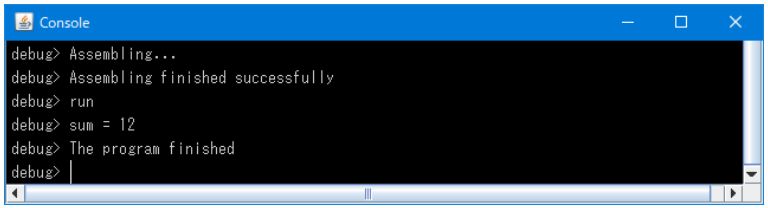
Subroutine call: RISC-V Example

```
.data
x: .word 7, 5, 0 p:
.ascii "sum =
%d\n" .text
sum
     add  a0, a1, a2 # a + b in a0 ra      #subroutines
:    ret                                   # return
main:
     addi sp, sp, -8                       # reserve stack space sw ra, 4(sp)
                                           # save return address
     la    a3, x                           # address of x
     lw    a1, 0(a3)                       # load x from memory
     addi a3, a3, 4                        # address of y
     lw a2, 0(a3)                          # load y from memory
     addi a3, a3, 4                        # address of z
     jal   ra sum                          # call subroutine
     sw    a0, 0(a3)                       # store z to memory 0
     addi a1, a0,                          # a1: sum
     la    a0,p                            # address of string
     jal   ra, printf                      # print z (in a1) ra, 4(sp) #
     lw    restore                         # return  address
     addi sp, sp, 8                        # release stack space
     ret ra                                # return
.end
```

Subroutine Call







References

- Material provided with permission from Yamin Li. Original art generated for his book on MIPS processing:
  - Li, Yamin, 2015, Computer Principles and Design in Verilog HDL, John Wiley & Sons

JOHNS HOPKINS

WHITING SCHOOL
of ENGINEERING