# HW04 - RISC-V

# **Q1** RISC-V Warmup

5 Points

REMINDER: Gradescope does not have submission history for in-browser assignments such as lecture questions and homeworks. If you press "save answer" or "submit" after the deadline, the autograder *will* mark it as late and you'll get 0 points in the next TCP rerun. When you review old homework and lecture assignments, do not click any buttons otherwise you risk getting a zero on that assignment.

Answer the following questions about RISC-V. Don't be afraid to take some RISCs.

RISC-V is a(n) _ language.	
✓ machine code	
high-level	
✓ assembly	
C-like	
numerical	
None of the above	
RISC-V follows the CPU	J design strategy.

CISC
✓ RISC
SISC
DISC
None of the above
Which of the following is true of the RISC philosophy? Select all that apply.
✓ RISC stands for Reduced Instruction Set Computing
RISC encourages the instruction set to be large and expansive
✓ In RISC, a simple instruction set helps to build faster hardware
✓ In RISC, complicated operations can be performed by composing simple instructions
None of the above
Which of the following is False about a processor register?
✓ A register is a small location of RAM memory.
✓ A register has access speed comparable to a hard drive.
✓ Registers comprise most of a computer's storage space.
✓ A computer typically has thousands of registers.
A register is a small amount of storage directly on the CPU.
None of the above

Because registers are so fast, we have a very limited number of them, since we must store them on the CPU directly.

RISC-V has \_\_#\_\_ integer registers.

32

For the RISC-V we learn in this class, a register can hold \_\_#\_\_ bytes.

4

Correct

Save Answer Last saved on Sep 30 at 4:53 PM

Q2 Instruction Types

9 Points

Classify each of the following RISC-V instructions by type.

addi t0, a0, 1

- OR
- **O** I
- Os
- O SB
- OU
- O UJ

#### **EXPLANATION**

This instruction is an I type, which signifies register-immediate operations. One way to tell is that it has two registers and one immediate as inputs.

O R
<b>⊙</b>
<b>O</b> s
O SB
OU
O M
EXPLANATION
This instruction is an I type. You can tell because there is a number (immediate) in the instruction itself.
beq a0, a1, loop
<b>O</b> R
O I
<b>O</b> s
<b>⊙</b> SB
ΟU
O NY
EXPLANATION  This instruction is a SB type. All branch instructions use the B-type instruction format in RISC-V.
sb t0, 12(s2)
O R
O I
<b>o</b> s
O SB
ΟU
O UJ

This instruction is a S type. All stores are encoded in the S-type format in RISC-V.	n
jal x0, label	
O R	
O I	
O s	
O SB	
OU	
<b>⊙</b> UJ	
EXPLANATION  This instruction is UJ type. The jump and link (JAL) instruction uses the Utype format in RISC-V.	JJ-
bne x10,x0,loop	
O R	
O I	
O s	
<b>⊙</b> SB	
O U	
O M	
EXPLANATION  This instruction is a SB type. All branch instructions use the B-type formations.  RISC-V.	at in

O R
<b>⊙</b> I
Os
O SB
OU
O NY
EXPLANATION  This instruction is an I type. JALR is different from JAL in that you must specify which register to store the return address (ra) in. Now that it requires two register inputs and an immediate, it is encoded as I-type.
slli x11, x10, 4
O R
<b>⊙</b> I
Os
O SB
OU
O NY
EXPLANATION  This instruction is an I type. You can tell because of the number (immediate)

provided as one of the arguments and the "i" in the function call.

sub t0, t1, zero

RISSBU

O UJ

#### **EXPLANATION**

This instruction is a R type. Note that "zero" is the name that refers to register x0, according to the RISC-V Green Sheet.



Save Answer

Last saved on Sep 30 at 6:01 PM

# **Q3** Misc Risc

4 Points

What does the following do:

```
xor x1, x1, x2
xor x2, x1, x2
xor x1, x1, x2
```

```
Sets x1 to 0
 Sets x1 to -1
 Sets x2 to 0
 Sets x2 to -1

✓ Swaps x1 and x2

 Sets x1 == x2
 Sets x1 and x2 to only bits which both do not share
 Sets x1 and x2 to only bits which both share
Given the following code:
  # x10 holds 0x34FF
  slli x12, x10, 0x10
  srli x12, x12, 0x08
  and x12, x12, x10
What is in x12?
Ox0
O 0x3400
O 0x4F0
O 0xFF00
O 0x34FF
O None of the other options
Given the following code sequence (assume x5 is somewhere on the stack):
  addi x11, x0, -1261
  sw x11, 0(x5)
```

```
sw x11, 2(x5)
lb x12, 1(x5)
lbu x13, 2(x5)
```

What is the 32bit value in x12? (Write your answer in hex)

```
0xFFFFFFB
```

What is the 32bit value in x13? (Write your answer in hex)

```
0x00000013
```



Save Answer

Last saved on Sep 30 at 5:24 PM

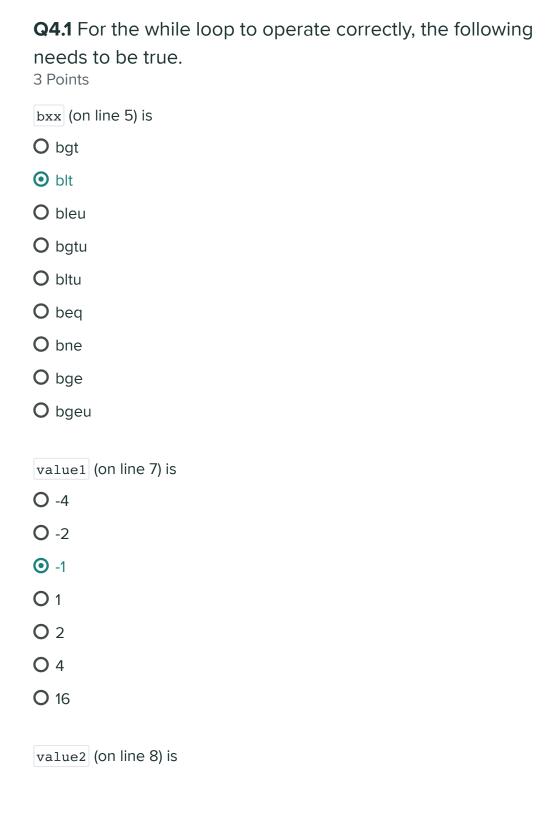
# Q4 Take a RISC for a WHILE

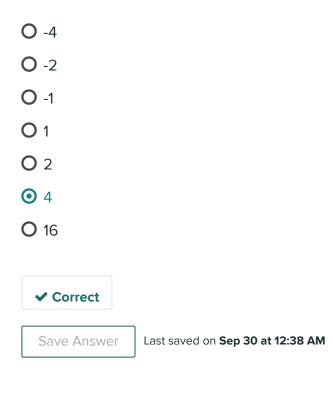
3 Points

C code:

```
1. int A[20];
2. int sum = 0;
3. int i = 19;
4. while ( i >= 0) {
5.    sum += A[i];
6.    i --;
7. }
```

Corresponds to the following RISC-V assembly code:





# Q5 RISC-V → Machine Code

5 Points

Convert each of the following RISC-V instructions to machine code.

Format your answer as 32-bit hexadecimal number, e.g. 0xABADCAFE or 0xabadcafe (the cafe just sucks). Leave any unused bits as 0.

Refer to your RISC-V Green Sheet if needed.

## Q5.1

1 Point

Convert:

0x40728433

```
sub s0, t0, t2
```

```
EXPLANATION
R-type:
• opcode: 011 0011
• rd: 8
• funct3: 000
• rs1: 5
• rs2: 7
• funct7: 010 0000
[funct7, rs2, rs1, funct3, rd, opcode]
= [0100000, 00111, 00101, 000, 01000, 0110011]
= [0100 0000 0111 0010 1000 0100 0011 0011]
= 0x40728433
✓ Correct
Save Answer
              Last saved on Sep 30 at 6:07 PM
```

## Q5.2

1 Point

Convert:

```
slli a0, t0, 3
```

0x00329513

Save Answer

Last saved on Sep 30 at 6:08 PM

# Q5.3

1 Point

Convert:

sw t0, 4(s0)

0x00542223

```
EXPLANATION
S-type:
• opcode: 010 0011
• imm[4:0]: 0 0100 (4)
• funct3: 010
• rs1: 8
• rs2: 5
• imm[11:5]: 0
[imm[11:5], rs2, rs1, funct3, imm[4:0], opcode]
[0000000, 00101, 01000, 010, 00100, 0100011]
= 0x00542223
✓ Correct
Save Answer
            Last saved on Sep 30 at 7:43 PM
```

## Q5.4

1 Point

Convert:

```
jal ra, fib
```

Assume  $\mathrm{PC} = 0\mathrm{x}00400014$ , and fib is located at  $0\mathrm{x}00400028$ .

0x014000EF

```
EXPLANATION
J type:
• opcode: 110 1111
• rd: 1
• jumpaddr = 0000 0000 0000 0001 0100
^ we are jumping 0x00400028 - 0x00400014 = 0x14 = 20 bytes
rearranged jumpaddr: 0000 0001 0100 0000 0000
Note: we have to rearrange it to fit the format specified on the Green Sheet.
[imm[20 | 10:1 | 11 | 19:12], rd, opcode]
[0000 0001 0100 0000 0000, 00001, 1101111]
= 0x014000EF
✓ Correct
Save Answer
            Last saved on Sep 30 at 7:46 PM
```

## Q5.5

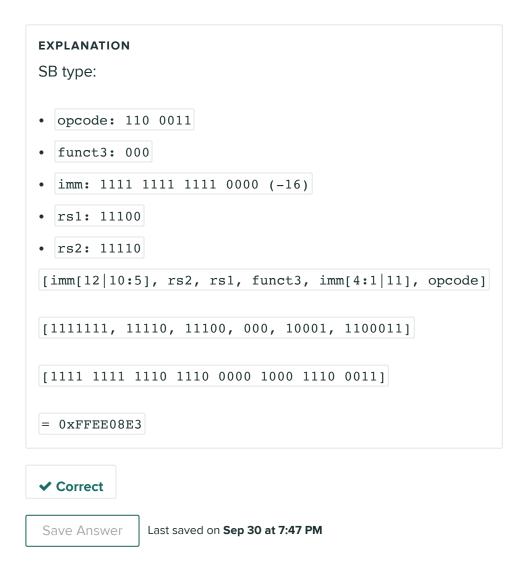
1 Point

Convert:

```
beq t3, t5, NEXT
```

Assume  $PC = 0 \mathrm{x} 0040011C$ , and NEXT is located at  $0 \mathrm{x} 0040010C$ .

0xFFEE08E3



# **Q6** Machine Code → RISC-V Instructions

4 Points

Convert each of the following 32-bit machine code instructions into a RISC-V instruction.

Formatting: use register names in your final answer, e.g. a0 instead of x10. The only exception is the zero register, which you should refer to as "x0" instead of "zero".

Example Format: "add t0, a0, x0" or "lw x0, 61(x5)"

For all questions, you can assume

 $PC = 0 \times 00400010$ 

and we have the following labels:

# func1: 0x00400000

## func2: 0x00400018

NOTE: Until Gradescope updates its checking, please make sure to follow the formatting above.

## Q6.1

1 Point

0x00598523

sb t0, 10(s3)

#### **EXPLANATION**

expanding gives: 0000 0000 0101 1001 1000 0101 0010 0011

opcode: 010 0011 -> sb (S type)

- imm[4:0]: 0 1010 (10)
- funct3: 000
- rs1: 1 0011 (19)
- rs2: 0 0101 (5)
- imm[11:5]: 000 0000 (O)

sb x5, 10(x19)

sb t0, 10(s3)



Save Answer

Last saved on Sep 30 at 8:07 PM

# Q6.2

1 Point

0x00b542b3

xor t0, a0, a1

#### **EXPLANATION**

expanding gives: 0000 0000 1011 0101 0100 0010 1011 0011

- opcode: 011 0011 -> R-type
- rd: 0 0101 (5)
- funct3: 100
- rs1: 0 1010 (10)
- rs2: 0 1011 (11)
- funct7: 000 0000

This gives:

xor x5, x10, x11

xor t0 a0 a1



Save Answer

Last saved on Sep 30 at 8:19 PM

## Q6.3

1 Point

0x00728463

beq t0, t2, func2

expanding gives: 0000 0000 0111 0010 1000 0100 0110 0011

- opcode: 110 0011 -> SB-Type
- imm[4:1|11]: 0100 0
- funct3: 000
- rs1: 0 0101 (5)
- rs2: 0 0111 (7)
- imm[12|10:5]: 0 000000

This means our immediate bits [12:1] is 0000 0000 0100, so the byte offset is: 0 0000 0000 1000 or 8 bytes. As the current PC is  $0 \times 00400010$ , we are branching to  $0 \times 00400018$ , which is the label func2. This gives us the following instruction.

beq t0, t1, func2



Save Answer

Last saved on Sep 30 at 8:22 PM

## Q6.4

1 Point

0xFF1FF06F

jal x0, func1

# **EXPLANATION** expanding gives: 1111 1111 0001 1111 1111 0000 0110 1111 opcode: 110 1111 -> UJ-Type • rd: 0 0000 (0) • imm[20|10:1|11|19:12]: 1 | 111 1111 000 | 1 | 1111 1111 Thus, our immediate bits [20:1] is 1111 1111 1111 1111 1000, so appending on the implicit zero gives us 1 1111 1111 1111 1111 0000. This number is equal to -16, so we are jumping to -16 bytes from our current PC. This results in the address 0x00400000, which is func1. Our instruction is then listen below. jal x0, func1 **✓** Correct Last saved on Sep 30 at 8:22 PM Save Answer

# Q7 C ↔ RISC-V

4 Points

In each of the following questions, you'll be given either C or RISC-V code, and asked to examine a translation into RISC-V or C, respectively.

The translation may have a bug or logic error. Your task is to identify the line number containing the error, or enter "none" if no problem exists.

Given this C code:

```
if (x < 5) {
    goto label;
}
```

Identify the buggy line in the following RISC-V translation. Select "None", if there is no error.

```
/* x = s0 */
1. addi t0 x0 5
2. slt t1 s0 t0
3. bne t1 x0 label
```

Line 1

Line 2

Line 3

✓ None

#### **EXPLANATION**

The code is correct.

Given this C code:

```
int x[2] = \{1, 2\}
int y = x[0] + x[1] + 2;
```

identify the buggy line in the following RISC-V translation or enter "none", if there is none:

```
/* x = s0, y = s1 */
1. lw t0 0(s0)
2. lw t1 1(s0)
3. add s1,t0,t1
4. addi s1,s1,2
```

Line 1	
✓ Line 2	
Line 3	
Line 4	
None	
EXPLANATION  The offset in lw is in bytes  Line2 should be lw t1, 4	s, and the pointer $x+1$ is 4 bytes away from $x$ .
Given this RISC-V code:	
sub s1 s2 s1 add s0 s0 s2 j end	
identify the buggy line in the none. (Hint: register order ma	following C translation or enter "none", if there is atters):
/* x = s0, y = s1, z =  1. y -= z;  2. x += z;  3. goto end;	: s2 */
✓ Line 1	
Line 2	
Line 3	
None	

The sub instruction of the RISC-V corresponds to y = z - y which is different from y-=z.

Given this RISC-V code:

```
slli s0 s0 4
xori s0 s0 3
sw s0 12(s1)
```

identify the buggy line in the following C translation or enter "none", if there is none:

```
1. int x; // x = s0
2. int y[5]; // y = s1
3. x *= math.pow(2, 4);
4. x ^= 3;
5. y[4] = x;
```

Line 1

Line 2

Line 3

Line 4

✓ Line 5

None

#### **EXPLANATION**

12(s1) means 12/4 = 3 words over, so the 5th line should be y[3] = xinstead.

**✓** Correct

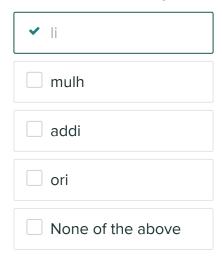
# **Q8** RISC-V Green Sheet questions

6 Points

Which of the following is the correct ordering of fields for R-type instruction from bits 31 to 0?

✓ funct7, rs2, rs1, funct3, rd, opcode
opcode, rs2, rs1, funct3, rd, funct7
unct7, rd, rs1, funct3, rs2, opcode
funct7, rs1, rs2, funct3, rd, opcode
☐ None of the above

Which of the following RISC-V instructions is actually a pseudo-instruction?



Which of the following fields is NOT used in I-Type instructions?

☐ rs1	
✓ rs2	
rd	
opcode	
None of the above	
	ers are preserved across a call? (i.e. should have after you make a RISC-V function call)
□ a0	
<b>✓</b> s0	
t4	

Which of the following RISC-V instructions includes a conditional statement in its operation?

**✓** x19

**∨** x9

\_\_ x31

☐ None of the above

<b>✓</b> beq	
<b>✓</b> slt	
lui	
<b>✓</b> sltu	
None of the above	
hich of the following ordering is correct fox0?	r memory allocation starting from
Text/Code, Static Data, Stack, Heap	
iextreode, Static Data, Stack, Fleap	
Static Data, Text/Code, Stack, Heap	
✓ Text/Code, Static Data, Heap, Stack	
None of the above	
<b>✓</b> Correct	
Save Answer Last saved on Sep 30 at 8:29 PM	
Save All Answers	Submit & View Submission 3