1  RISC-V: A Rundown

RISC-V is an assembly language, which is comprised of simple instructions that each do a single task such as addition or storing a chunk of data to memory.

For example, on the left is a line of C code and on the right is a chunk of RISC-V code that accomplishes the same thing.

```c
int x = 5, y[2];
y[0] = x;
y[1] = x * x;
```

```risc-v
// x -> s0, &y -> s1
addi s0, x0, 5
sw s0, 0(s1)
mul t0, s0, s0
sw t0, 4(s1)
```

1.1 Can you figure out what each line in the RISC-V code is doing?

2  Registers

In RISC-V, we have two methods of storing data, one of them is main memory, the other is through registers. Registers are much faster than using main memory, but are very limited in space (32-bits)

<table>
<thead>
<tr>
<th>Register(s)</th>
<th>Alt.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>zero</td>
<td>The zero register, always zero</td>
</tr>
<tr>
<td>x1</td>
<td>ra</td>
<td>The return address register, stores where functions should return</td>
</tr>
<tr>
<td>x2</td>
<td>sp</td>
<td>The stack pointer, where the stack ends</td>
</tr>
<tr>
<td>x5-x7, x28-x31</td>
<td>t0-t6</td>
<td>The temporary registers</td>
</tr>
<tr>
<td>x8-x9, x18-x27</td>
<td>s0-s11</td>
<td>The saved registers</td>
</tr>
<tr>
<td>x10-x17</td>
<td>a0-a7</td>
<td>The argument registers, a0-a1 are also return value</td>
</tr>
</tbody>
</table>

2.1 Can you convert each instruction’s registers to the other form?

```
add s0, zero, a1 -->
or x18, x1, x30 -->
```

3  Basic Instructions

For your reference, here are a couple of the basic instructions for arithmetic operations and dealing with memory:
Basic Operations:

<table>
<thead>
<tr>
<th>[inst]</th>
<th>[destination register] [argument register 1] [argument register 2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Adds the two argument registers and stores in destination register</td>
</tr>
<tr>
<td>xor</td>
<td>Exclusive or’s the two argument registers and stores in destination register</td>
</tr>
<tr>
<td>mul</td>
<td>Multiplies the two argument registers and stores in destination register</td>
</tr>
<tr>
<td>sll</td>
<td>Logical left shifts AR1 by AR2 and stores in DR</td>
</tr>
<tr>
<td>srl</td>
<td>Logical right shifts AR1 by AR2 and stores in DR</td>
</tr>
<tr>
<td>sra</td>
<td>Arithmetic right shifts AR1 by AR2 and stores in DR</td>
</tr>
<tr>
<td>slt/u</td>
<td>If AR1 &lt; AR2, stores 1 in DR, otherwise stores 0, u does unsigned comparison</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[inst]</th>
<th>[register] [offset][[register with base address]]</th>
</tr>
</thead>
<tbody>
<tr>
<td>sw</td>
<td>Stores the contents of the register to the address+offset in memory</td>
</tr>
<tr>
<td>lw</td>
<td>Takes the contents of address+offset in memory and stores in the register</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[inst]</th>
<th>[argument register 1] [argument register 2] [label]</th>
</tr>
</thead>
<tbody>
<tr>
<td>beq</td>
<td>If AR1 == AR2, moves to label</td>
</tr>
<tr>
<td>bne</td>
<td>If AR1 != AR2, moves to label</td>
</tr>
<tr>
<td>jal</td>
<td>Stores the current instruction’s address into DR and moves to label</td>
</tr>
</tbody>
</table>

You may also see that there is an ”i” at the end of certain instructions, such as addi, slli, etc. This means that AR2 becomes an ”immediate” or an integer instead of using a register.

3.1 Assume we have an array in memory that contains `int* arr = {1,2,3,4,5,6,0}`. Let the values of arr be a multiple of 4 and stored in register `s0`. What do the snippets of RISC-V code do? Assume that all the instructions are run one after the other in the same context.

a) `lw t0, 12(s0) -->`

b) `slli t1, t0, 2
add t2, s0, t1
lw t3, 0(t2) -->
addi t3, t3, 1
sw t3, 0(t2)`

c) `lw t0, 0(s0)
xori t0, t0, 0xFFF -->
addi t0, t0, 1`

3.2 While only using the instructions (and their ”i” forms) given above, how can we branch on the following conditions:

\[ s0 < s1 \quad \quad s0 \geq s1 \quad \quad s0 > 1 \]
## 4 C to RISC-V

### 4.1 Translate between the C and RISC-V verbatim

<table>
<thead>
<tr>
<th>C</th>
<th>RISC-V</th>
</tr>
</thead>
</table>
| // s0 -> a, s1 -> b  
// s2 -> c, s3 -> z  
int a = 4, b = 5, c = 6, z;  
z = a + b + c + 10; | addi s0, x0, 0  
addi s1, x0, 1  
addi t0, x0, 30  
loop:  
  beq s0, t0, exit  
  add s1, s1, s1  
  addi s0, s0, 1  
  jal x0, loop  
exit: |
| // s0 -> int * p = intArr;  
// s1 -> a;  
*p = 0;  
int a = 2;  
p[1] = p[a] = a; | |
| // s0 -> a, s1 -> b  
int a = 5, b = 10;  
if(a + a == b) {  
a = 0;  
} else {  
b = a - 1;  
} | addi s0, x0, 0  
addi s1, x0, 1  
addi t0, x0, 30  
loop:  
  beq s0, t0, exit  
  add s1, s1, s1  
  addi s0, s0, 1  
  jal x0, loop  
exit: |
| // s0 -> n, s1 -> sum  
// assume n > 0 to start  
for(int sum = 0; n > 0; n--) {  
  sum += n;  
} | |