1.5 Tutorial 5

1.5.1 Exercise 1:

1. For the following C statement, write the corresponding RISC-V assembly code. Assume that the C variables f, g, and h, have already been placed in registers x5, x6, and x7 respectively. Use a minimal number of RISC-V assembly instructions.

   f = g + (h − 5);

2. Write a single C statement that corresponds to the two RISC-V assembly instructions below.

   add f, g, h
   add f, i, f

3. For the following C statement, write the corresponding RISC-V assembly code. Assume that the variables i and j are assigned to registers x28 and x29, respectively. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively.

   B[8] = A[i−j];

4. For the RISC-V assembly instructions below, what is the corresponding C statement? Assume that the variables f and g are assigned to registers x5 and x6, respectively. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively.

   slli x30, x5, 3
   add x30, x10, x30
   slli x31, x6, 3
   add x31, x11, x31
   lw x5, 0(x30)
   addi x12, x30, 8
   lw x30, 0(x12)
   add x30, x30, x5
   sw x30, 0(x31)

5. Show how the value 0xabcdef12 would be arranged in memory of a little-endian and a big-endian machine. Assume the data are stored starting at address 0 and that the word size is 4 bytes.

6. Translate the following C code to RISC-V. Assume that the variables i and j are assigned to registers x28 and x29, respectively. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively. Assume that the elements of the arrays A and B are 8-byte words:

   B[8] = A[i] + A[j];

7. Translate the following RISC-V code to C. Assume that the variable f is assigned to registers x5. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively.

   addi x30, x10, 8

addi x31, x10, 0
sw x31, 0(x30)
lw x30, 0(x30)
add x5, x30, x31

1.5.2 Exercise 2:

Translate to RISC-V:

1. // s0 -> a, s1 -> b
   // s2 -> c, s3 -> z
   int a = 4, b = 5, c = 6, z;
   z = a + b + c + 10;

2. // s0 -> int * p = intArr;
   // s1 -> a;
   *p = 0;
   int a = 2;
   p[1] = p[a] = a;

1.5.3 Exercise 3:

Assume we have an array in memory that contains int *arr = 1,2,3,4,5,6,0. Let register s0 hold the address of the element at index 0 in arr. You may assume integers are four bytes and our values are word-aligned. What do the snippets of RISC-V code do? Assume that all the instructions are run one after the other in the same context.

1. lw t0, 12(s0)

2. sw t0, 16(s0)

3. slli t1, t0, 2
   add t2, s0, t1
   lw t3, 0(t2)
   addi t3, t3, 1
   sw t3, 0(t2)

4. lw t0, 0(s0)
   xori t0, t0, 0xFFF
   addi t0, t0, 1

1.5.4 Exercise 4:

Using the given instructions and the sample memory array, what will happen when the RISC-V code is executed?

1. What value does each register hold after the code is executed?

```
li t0 0x00FF0000
lw t1 0(t0)
addi t0 t0 4
lh t2 2(t0)
lw s0 0(t1)
lb s1 3(t2)
```

0xFFFFFFFF	
	...
0x00FF0004	0x000C561C
0x00FF0000	36
	...
0x00000036	0xFDFDFDFD
0x00000024	0xDEADB33F
	...
0x0000000C	0xC5161C00
	...
0x00000000	

2. Update the memory array with its new values after the code is executed. Some memory addresses may not have been labeled for you yet.

```
li t0 0xABADCAFE
li t1 0xF9120504
li t2 0xBEEFCACE
sw t0 0(t1)
addi t1 t1 4
addi t0 t0 4
sh t1 2(t0)
sb t2 1(t2)
sb t2 3(t1)
sb t2 3(t0)
```

0xFFFFFFFF	
0xF9120504	
0xABADCAFE	
0x00000004	
0x00000000	0x00000000