Lab 6 Worksheet

Your task in lab is to write RISC-V assembly programs to implement several functions.

1. Load and Store

Consider this function in C, which swaps the values at indices 1 and 3 in an array of ints:

```
void swap(int* arr) {
   int temp = arr[1];
   arr[1] = arr[3];
   arr[3] = temp;
}
```

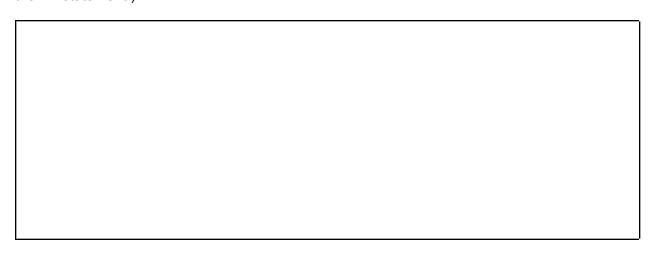
Assume that the arr pointer is in register x1. Also, don't worry about out-of-bounds access: assume that we allocated enough space for the arr array. Write the RISC-V assembly code to implement this swap. (Hint: this problem can be solved using only the lw and sw instructions.)

2. Conditional Control Flow

Consider this code with a simple if-statement:

```
if (x < y)
    y = (x - y) * 2;
else
    y--;</pre>
```

Assume that register x16 holds x and x17 holds y. You may use all other registers to store temporary values if you like. Write a RISC-V assembly program to implement this code. (Hint: you will want to begin with a bge instruction that compares the values of the x16 and x17 registers. It's also recommended to create labels for the else branch and the exit point after the if-statement.)



3. Loops

Consider this for loop in C:

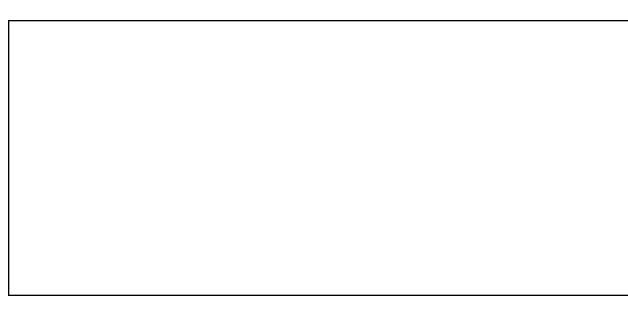
```
for (int i = 0; i < y; i++) {
  x = x + 2;
}
return x;</pre>
```

Assume that x and i start at 0, and that we use these register mappings:

- y is in register a0
- x is in register a1
- i is in register t0

Which of these RISC-V assembly translations are correct? For the incorrect translations, write a brief explanation of why they are incorrect. (Hint: there are two correct translations. To figure out which is correct, we recommend thinking about the no. of loop iterations that could be executed in each option, and whether too many/too few iterations are being executed.)

```
Option 1:
                         Option 2:
                                                   Option 3:
                                                   bge x0, a0, end
for:
                         for:
blt t0, a0, end
                         beq t0, a0, end
                                                   for:
body:
                        addi a1, a1, 2
                                                   bge t0, a0, end
                        addi t0, t0, 1
addi a1, a1, 2
                                                   addi a1, a1, 2
addi t0, t0, 1
                        beq x0, x0, for
                                                   addi t0, t0, 1
beq x0, x0, for
                        end:
                                                   beg x0, x0, for
end:
                                                   end:
             Option 4:
                                                    Option 5:
        bge x0, a0, end
                                              ble a0, x0, end
        for:
                                              for:
        bge t0, a0, end
                                              addi a1, a1, 2
        body:
                                              addi t0, t0, 1
        addi a1, a1, 2
                                              blt t0, a0, for
        addi t0, t0, 1
                                              end:
        end:
```



4. Putting Everything Together

Finally, let's translate the following C program that calculates the product of an array:

```
void product(int* arr, int size) {
  int product = 1;
  // --- START HERE ---
  for (int i = 0; i < size; i++) {
    product *= arr[i];
  }
  // --- END HERE ---
  printf("The product is %d\n", product);
}</pre>
```

Translate the indicated section of code (just the loop) to RISC-V assembly. Assume x1 holds arr pointer, x2 holds size, x3 holds product, and x4 will hold i. x3 is already initialized to 1 (outside of your code), and x4 is *uninitialized*. Feel free to use any other registers. (Hint: we recommend reviewing the <u>lecture notes on Control Flow</u> in RISC-V, specifically the sub-section titled "Implementing Loops" at the bottom of the page.)