Cole Edmonds
Doug Balish Project 2

1.

Translation of C to assembly:

With Labels:

```
matrix_opr: beq x5, x7, exit          // if i == 4, exit main loop
            slli x28, x5, 3           // i * 8 -> x28 for indexing
            sd x0, 128(x28)           // store zero in out[]
            ld x31, 128(x28)          // load zero from out[i] to x31
loop:       slli x29, x5, 2           // i * 4 -> x29 for indexing
            add x12, x6, x29          // x12 = j + i*4
            slli x12, x12, 3          // x12 = [i*4 + j] * 8
            ld x30, 0(x12)            // x30 = mat[i*4 + j]
            sll x30, x30, x5          // x30 = shift(value of x30, i )
            add x31, x31, x30         // x31 = x31 + x30
            sd x31, 128(x28)          // store x31 to out[i]
            addi x6, x6, 1            // j++
            beq x6, x7, reset         // if j == 4, jump to reset
            beq x0, x0, loop          // if j != 4, arrive here
reset:      addi x5, x5, 1            // if j == 4, arrive here, i++
            add x6, x0, x0            // force x6 (j) back to zero
            beq x0, x0, matrix_opr    // jump back to top
exit:
```

Cole Edmonds
Doug Balish   Project 2

Without labels (actual tested trace file):

```
beq x5, x7, 64

slli x28, x5, 3

sd x0, 128(x28)

ld x31, 128(x28)

slli x29, x5, 2

add x12, x6, x29

slli x12, x12, 3

ld x30, 0(x12)

sll x30, x30, x5

add x31, x31, x30

sd x31, 128(x28)

addi x6, x6, 1

beq x6, x7, 4

beq x0, x0, -16

addi x5, x5, 1

add x6, x0, x0

beq x0, x0, -32
```

2.

Data memory layout for out:

NOTE: mat[] memory (byte 0 – 127) **not** shown. Follows pattern shown in loop below, every 8th byte contains a value [0-15]

```
for(int i = 0; i < 16; i++) {
    core->data_mem[8*i] = i;
}
```

Below is the end result of the simulation shown for memory location 128 to 159 (the out[] array). Byte 0 of out[0] corresponds to location 128, and byte 0 of out[3] corresponds to location 152. For out[0] to out [2], the resulting value to be stored is less than 8 bits, so only the lowest (lest significant) bits/byte are used. Out[3] must store a 9 bit value of 432, so the first byte holds the first 8 bits of the value (176_{10}), and the second byte holds the most significant '1'. This configuration can handle up to 64 bit integers.

Byte:	0	1	2	3	4	5	6	7
Out[0]	6	0	0	0	0	0	0	0
Out[1]	44	0	0	0	0	0	0	0
Out[2]	152	0	0	0	0	0	0	0
Out[3]	176	1	0	0	0	0	0	0

Results and summary:

For this experiment, two C functions were provided to be translated into RISC-V asm, and the parser and core files were updated to reflect the addition of new types of instruction, as well as to handle storing values over 8 bits. For the translation, some optimizations were used to keep the parser and core file complexity down so that only sll, bne, and sd function needed to be added.

After running the simulation successfully, the output matched that of the given C functions, being 6, 44, 152, and 432.