1.) Pseudo-instructions are not part of the MIPS instruction set but often appear in MIPS programs. The assembler then has to convert them into a small set of real MIPS instructions. For each of the following instructions, convert them to an equivalent (VERY short) set of real MIPS instructions. If you need a temporary variable, use $at (that’s what it is for). You will probably also want to read about the LUI (load upper immediate) instruction from Appendix A in the book.

<table>
<thead>
<tr>
<th>Pseudo-instruction</th>
<th>What it accomplishes</th>
</tr>
</thead>
<tbody>
<tr>
<td>li $t2, big</td>
<td>$t1 = big (note: solved below)</td>
</tr>
<tr>
<td>move $t1, $t2</td>
<td>$t1 = $t2</td>
</tr>
<tr>
<td>beq $t1, small, L</td>
<td>if ($t1 = small) go to L</td>
</tr>
<tr>
<td>ble $t3, $t4, L</td>
<td>if ($t3 &lt;= $t4) go to L</td>
</tr>
<tr>
<td>addi $t0, $t2, big</td>
<td>$t0 = $t2 + big</td>
</tr>
</tbody>
</table>

Note that “small” is a constant that fits in 16 bits (i.e. small enough for the immediate field of an instruction), and “big” is a constant that is 32 bits (i.e. too long for the immediate field). If you need to access a portion of the constant, you can do that. For example, to implement “li $t2, big”, you would write:

```
lui $at, big[31:16]       # $at’s top 16 bits set properly
ori $t2, $at, big[15:0]   # $t2’s bottom 16 bits also set
```

2.) Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the value $a, b, and D, are in registers $s0, $s1, and $s2 respectively, and D is an array of integers.

```c
while (a<10) {
    D[a] = b + a;
    a += 1;
}
```
3.) Convert the following code segment to machine code. Assume the code starts at address \( 2000_{10} \).

*Note that the SUBI is a pseudo-instruction – you’ll need to translate this into something that you actually can convert (should be quite easy to do).*

```assembly
LOOP:
  addi $t2, $0, 0xA  # hexadecimal constant.
LOOP2:
  addi $s2, $s2, 2
  subi $t2, $t2, 1
  bne $t2, $0, LOOP2
  subi $t1, $t1, 1
  bne $t1, $0, LOOP
DONE:
```

4.) Convert the following machine code program to assembly.

```
00011010010000000000000000000011
00100000000000000000000000000000
1010110001010110000000000000000000
1010110001010100000000000000000010
0010000100100100100000000000001000
```