Pseudo-instructions are instructions in assembly language that are not part of the ARM instruction set. Instead, the assembler converts them into a small set of real ARM instructions. For each of the following instructions, convert them to an equivalent (VERY short) set of real ARM instructions. If you need a temporary variable, use X16 (a register reserved for the assembler as a temporary). You will also want to read about the MOVZ and MOVK instructions in Chapter 2.10 in the book.

<table>
<thead>
<tr>
<th>Pseudo-instruction</th>
<th>What it accomplishes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LI X2, 32'big</td>
<td>X2 = big (note: solved below)</td>
</tr>
<tr>
<td>CLEAR X1</td>
<td>X1 = 0</td>
</tr>
<tr>
<td>BNE X1, 16'small, L</td>
<td>if (X1 != small) go to L</td>
</tr>
<tr>
<td>LDUR X0 [X1, 48'big]</td>
<td>X0 = Memory[X1 + big]</td>
</tr>
</tbody>
</table>

Note that constants have their bitwidths given as bitwidth’const. If you need to access a portion of the constant, you can do that. For example, to implement “LI X2, 32’big”, you would write:

```assembly
MOVZ X2, big[31:16], LSL 16 // zero X2, set [31:16] properly
MOVK X2, big[15:0], LSL 0   // set bottom 16 bits
```

```
CLEAR X1
ADD X1, X31, X31 //NOTE: Many others possible
BNE X1, 16'small, L
MOVZ X16, small, LSL 0 //x16 = small
CMP X1, X16 // compare X1 to small
B.NE L // if !=, go to L
LDUR X0, [X1, 48'big]
MOVZ X16, big[47:32], LSL 32
MOVK X16, big[31:16], LSL 16
MOVK X16, big[15:0], LSL 0 // x16 = big
ADD X16, X1, X16
LDUR X0, [X16, #0]
```
Convert the following code to ARM instructions. Assume a is in X0, b in X1, n in X2, and the result should go into X10. You may change the values in any of these registers. Implement the subroutine call by simply branching to the top of your code, but with the arguments updated appropriately. To end your program, call "BR X30" to return to the subroutine caller. Shorter and faster programs will be given more credit.

```c
int fib_iter(int a, int b, int n) {
    if (n == 0)
        return b;
    else
        return fib_iter(a+b, a, n-1);
}
```

// x0=a, x1=b, x2=n, x10=result

```
FIB_ITER:
    CBNZ  X2, ELSE-IF
    ADD  X10, X1, X31
    BR   X30
ELSE-IF:
    ADD  X0, X0, X1
    SUB  X1, X0, X1
    SUBI X2, X2, #1
    B    FIB_ITER
```

// Don't exit until n==0
// return b
// End subroutine

// x0=a+b
// x1=(a+b)-b = a
// x2=n-1