Do the following exercises of the book David A. Patterson and John L. Hennessy, “Computer organization and design ARM edition: the hardware software interface:”

2.23, 2.30, 2.32, 2.35, 2.37, 2.38, 2.41, 2.42,

2.23 Suppose the *program counter* (PC) is set to 0×2000 0000.

2.23.1 [5] <§2.10> What range of addresses can be reached using the LEGv8 *branch* (B) instruction? (In other words, what is the set of possible values for the PC after the branch instruction executes?).

2.23.2 [5] <§2.10> What range of addresses can be reached using the LEGv8 *conditional branch-on-equal* (CBZ) instruction? (In other words, what is the set of possible values for the PC after the branch instruction executes?)

2.30 [30] <§2.8> Implement the following C code in LEGv8 assembly. Hint: Remember that the stack pointer must remain aligned on a multiple of 16.

```
int fib(int n){
    if (n==0)
       return 0;
    else if (n == 1)
       return 1;
    else
       return fib(n-1) + fib(n-2);
}
```

2.32 [20] <§2.8> Translate function f into LEGv8 assembly language. If you need to use registers X10 through X27, use the lower-numbered registers first. Assume the function declaration for g is “int g(int a, int b)”. The code for function f is as follows:

```
int f(int a, int b, int c, int d){
  return g(g(a,b),c+d);
}
```

2.35 [30] <§2.9> Write a program in LEGv8 assembly to convert an ASCII string containing a positive or negative integer decimal string to an integer. Your program should expect register `X0` to hold the address of a null-terminated string containing an optional '+' or '−' followed by some combination of the digits 0 through 9. Your program should compute the integer value equivalent to this string of digits, then place the number in register `X0`. If a non-digit character appears anywhere in the string, your program should stop with the value −1 in register `X0`. For example, if register `X0` points to a sequence of three bytes 50_{ten}, 52_{ten}, 0_{ten} (the null-terminated string "24"), then when the program stops, register `X0` should contain the value 24_{ten}. The ARMv8 `MUL` instruction takes two registers as input. There is no "MULI" instruction. Thus, just store the constant 10 in a register.

2.37 [5] <§2.10> Write the LEGv8 assembly code that creates the 64-bit constant $0x1122334455667788_{two}$ and stores that value to register `X0`.

2.38 [10] <§2.11> Write the LEGv8 assembly code to implement the following C code:

```
lock(lk);
shvar=max(shvar,x);
unlock(lk);
```

Assume that the address of the `lk` variable is in `X0`, the address of the `shvar` variable is in `X1`, and the value of variable `x` is in `X2`. Your critical section should not contain any function calls. Use `LDXR/STXR` instructions to implement the `lock()` operation, the `unlock()` operation is simply an ordinary store instruction.

2.41 Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 500 million arithmetic instructions, 300 million load/store instructions, 100 million branch instructions.

2.41.1 [5] <§§1.6, 2.13> Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, while increasing the clock cycle time by only 10%. Is this a good design choice? Why?

2.41.2 [5] <§§1.6, 2.13> Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

2.42 Assume that for a given program 70% of the executed instructions are arithmetic, 10% are load/store, and 20% are branch.

2.42.1 [5] <§§1.6, 2.13> Given this instruction mix and the assumption that an arithmetic instruction requires two cycles, a load/store instruction takes six cycles, and a branch instruction takes three cycles, find the average CPI.

2.42.2 [5] <§§1.6, 2.13> For a 25% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

2.42.3 [5] <§§1.6, 2.13> For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?