# Computer Architecture: unassessed tutorial exercises

## Exercise 2.1

(a) Design a circuit which increments a number, in two's complement representation, by one, without using fulladders. Ignore overflow detection.

(b) Show how the circuit in (a) can be included, with minimum modification, in the ALU design described in the lecture. The extended ALU should be able to compute $a + b + 1$ and $a - b + 1$, where $a$ and $b$ are the two $n$-bit inputs to the ALU.

## Exercise 2.2

For this exercise, your task is to compare the memory efficiency of four different styles of instruction sets. The architecture styles are:

- *Accumulator*: all operations use the accumulator which is the only register available. `load` and `store` operations transfer data between the accumulator and memory, other operations, such as `add` and `neg`, use the accumulator as one of the source operands (if there are more than one) and as the destination operand.
- *Memory-Memory*: all three operands of each instruction are in memory. Arithmetic instructions include `add` and `sub`.
- *Stack*: all operations occur on top of the stack. Only `push` and `pop` access memory, and all other instructions, such as `add` and `neg`, remove their operands from the stack and replace them with the result. The instruction `dup` replicates the top element. The implementation uses a stack for the top two entries; accesses that use other stack positions are memory references.
- *Load/Store*: all operations occur in registers, and register-to-register instructions, such as `add` and `sub`, have three operands per instruction. There are 16 general-purpose registers, and register specifiers are 4 bits long.

To measure memory efficiency, make the following assumptions about all four instruction sets:

The opcode is always 1 byte (8 bits).
All memory addresses are 2 bytes (16 bits).
All data operands are 4 bytes (32 bits).
All instructions are an integral number of bytes in length.

There are no other optimisations to reduce memory traffic, and all variables will be placed initially in memory. For example, a register load will require 4 instruction bytes from memory: one for the opcode, one for the register destination, and two for a memory address; there will also be 4 data bytes for the operand. Similarly, a memory-memory add instruction will require 7 instruction bytes from memory: one for the opcode and two for each of the 3 memory address, and there will be 8 data bytes from memory to processor and 4 data bytes from processor to memory.

For each of the four architecture styles, write the instruction code sequence for the following program fragment:

$$\begin{aligned} a &= b + c; \\ b &= a + c; \\ d &= a - b; \end{aligned}$$

For each code sequence, calculate the instruction bytes fetched and the memory-data bytes transferred (read or written).

(a) Which architecture is most efficient as measured by code size?

(b) Which architecture is most efficient as measured by total memory bandwidth required (code plus data)?

(c) If the answers are not the same, why are they different?