In More Depth: Instruction Set Styles

For the next two exercises, your task is to compare the memory efficiency of four different styles of instruction sets for two code sequences. The architecture styles are the following:

- *Accumulator.*
- *Memory-memory:* All three operands of each instruction are in memory.
- *Stack:* All operations occur on top of the stack. Only push and pop access memory, and all other instructions remove their operands from the stack and replace them with the result. 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 have three operands per instruction. There are 16 general-purpose registers, and register specifiers are 4 bits long.

Consider the following C or Java code:

```
a = b + c;  # a, b, and c are variables in memory
```

Accumulator Instructions

The assignment would be translated into the following instructions in an accumulator instruction set:

```
load AddressB   # Acc = Memory[AddressB], or Acc = B
add AddressC    # Acc = B + Memory[AddressC],or Acc = B + C
store AddressA  # Memory[AddressA] = Acc, or A = B + C
```

All variables in a program are allocated to memory in accumulator machines, instead of normally to registers as we saw for MIPS. One way to think about this is that variables are always spilled to memory in this style of machine. As you may imagine, it takes many more instructions to execute a program with a single-accumulator architecture. (See Exercise 2.32 for another example.)

Memory-Memory Instructions

It would be translated into the following instructions in a memory-memory instruction set:

```
add AddressA,AddressB,AddressC
```

Stack Instructions

It would be translated into the following instructions in a stack instruction set:

```
push AddressC  # Top=Top+4;Stack[Top]=Memory[AddressC]
push AddressB  # Top=Top+4;Stack[Top]=Memory[AddressB]
add            # Stack[Top-4]=Stack[Top]
               # + Stack[Top-4];Top=Top-4;
pop AddressA   # Memory[AddressA]=Stack[Top];
               #   Top=Top-4;
```

To get the proper byte address, we adjust the stack by 4. The downside of stacks as compared to registers is that it is hard to reuse data that has been fetched or calculated without repeatedly going to memory.

For a given code sequence, we can calculate the instruction bytes fetched and the memory data bytes transferred using 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 optimizations to reduce memory traffic.

For example, a register load will require 4 instruction bytes (1 for the opcode, 1 for the register destination, and 2 for a memory address) to be fetched from memory along with 4 data bytes. A memory-memory add instruction will require 7 instruction bytes (1 for the opcode and 2 for each of the 3 memory addresses) to be fetched from memory and will result in 12 data bytes being transferred (8 from memory to the processor and 4 from the processor back to memory). The following table displays a summary of this information for each of the architectural styles for the code appearing above and in Section 2.19:

Style	Instructions for a = b + c	Code bytes	Data bytes
Accumulator	3	3 + 3 + 3	4 + 4 + 4
Memory-memory	1	7	12
Stack	4	3 + 3 + 1 + 3	4 + 4 + 0 + 4
Load-store	4	4 + 4 + 3 + 4	4 + 4 + 0 + 4

2.52 [20] <§2.18> For the following C code, write an equivalent assembly language program in each architectural style (assume all variables are initially in memory):

```
a = b + c;
b = a + c;
d = a - b;
```

For each code sequence, calculate the instruction bytes fetched and the memory data bytes transferred (read or written). Which architecture is most efficient as measured by code size? Which architecture is most efficient as measured by total memory bandwidth required (code + data)? If the answers are not the same, why are they different?

2.53 [20] <§2.18> Sometimes architectures are characterized according to the typical number of memory addresses per instruction. Commonly used terms are 0, 1, 2, and 3 addresses per instruction. Associate the names above with each category.