7. For this question consider the following MIPS assembly code.

```
1        lui  $s0, 0x2000   # Initialize addresses
2        ori  $s0, 0x0000   # First vector
3        lui  $s1, 0x1001
4        ori  $s1, 0x5000   # Second vector
5        lui  $s2, 0x1001
6        ori  $s2, 0x6000   # End address
7        addi $s3, $0, 0    # initialize
8   loop: lw   $t0, 0($s0)   # Read vector 1
9        lw   $t1, 0($s1)   # Read vector 2
10       add  $t2, $t0, $t1 # Add both to $t2
11       add  $s3, $s3, $t2 # Accumulate at $s3
12       addi $s0, $s0, 4   # increment vector 1
13       addi $s1, $s1, 4   # increment vector 2
14       beq  $s1, $s2, end # end reached?
15       j    loop          # no, loop.
16  end:
```

This code is being executed on a MIPS processor that executes all instructions in a single cycle, except for memory accesses (i.e. `lw` and `sw` instructions) which take 20 clock cycles. The processor will be waiting during this time.

(a) (2 points) How many clock cycles (approximately) does it take to execute the above mentioned program?

Solution: There are 7 instructions before the loop. The loop is executed 1024 times. There are 6 instructions that take 1 clock cycle and 2 `lw` instructions that take 20 cycles so the total is:

```
Total = 7 + 1024 * (2*20 + 6)
      = 47111
      ~ 45000 - 50000 cycles
```

(b) (4 points) We want to speed up the operation by using a cache memory. By using a cache with a capacity of 256 words, we can achieve 1 clock cycle memory accesses for data that is within the cache.

How fast would the above program execute if a direct mapped cache with block size of one word is used?

Solution:

In principle, all memory accesses would be cache misses, no data is read twice (no temporal locality), and block size is one word so no spatial locality can be exploited. In fact each `lw` instruction will be longer, as now we have one cycle for the cache miss (21 instead of 20 cycles).

```
Total = 7 + 1024 * (2*21 + 6)
      = 49159
      ~ 50000 cycles
```

(c) (4 points) What would be the best possible cache configuration with a capacity of 256 words for running this program?

Solution:

There are two independent memory locations being read. So the cache needs to be 2-way set associative so that the consecutive `lw` accesses on `$s0` and `$s1` do not invalidate the data.

Since there is no temporal locality (data is not read again in this program), spatial locality is the only way to speed up. The block size should be as large as possible. Since we have 2-ways, the largest block size we can have is 128 words.

In this case we would have 2 cache misses for each way every 128 memory accesses that will require 21 cycles each. All other accesses will have a single cycle access time.

```
Total = 7 + 8 * (2 * 21 +6) + 1016 * 8
      = 8519
      ~ 8000-9000 cycles
```