```
00 int SumVector(int *v, int length){
01   int  accum = 0
02    for(i=0; i<length ; i++){
03      accum = accum + v[i];
04    }
05  return accum;
06 }
```

Figure 2: C code for a simple loop.

```
00 SumVector:
01       add  $v0, $0, $0
02       bgt  $a1, $0, cont
03       jr   $ra
04 cont: add  $t0, $a0, $0
05       sll  $t1, $a1, 2
06       add  $t2, $a0, $t1
07 loop: lw   $t3, 0($t0)
08       add  $v0, $v0, $t3
09       addi $t0, $t0, 4
10       blt  $t0, $t2, loop
11       jr   $ra
```

(a) Version A

```
00 SumVector:
01       add  $v0, $0, $0
02       add  $t0, $0, $0
03 loop: bge  $t0, $a1, done
04       sll  $t1, $t0, 2
05       add  $t2, $a0, $t1
06       lw   $t3, 0($t2)
07       add  $v0, $v0, $t3
08       addi $t0, $t0, 1
09       j    loop
10 done: jr   $ra
```

(b) Version B

```
00 SumVector:
01       add  $v0, $0, $0
02       bgt  $a1, $0, cont
03       jr   $ra
04 cont: add  $t0, $0, $0
05 loop: sll  $t1, $t0, 2
06       add  $t2, $a0, $t1
07       lw   $t3, 0($t2)
08       add  $v0, $v0, $t3
09       addi $t0, $t0, 1
10       blt  $t0, $a1, loop
11 done: jr   $ra
```

(c) Version C

Figure 3: Three versions of assembly code for the C code of Figure 2

Question 4 (20 points): Figure 2 shows a simple function written in C. Figure 3 shows three versions of MIPS assembly code that attempt to implement the function of Figure 2.

a. (**5 points** Do all three versions of the assembly code correctly implement the C code? If not, explain any incorrections.

   Yes, they are all correct.

b. (**5 points**) Using the letters A, B, and C, provide a sorted list of the three versions according to their efficiency. List first the least efficient and list last the most efficient version. Explain the criteria that you used for sorting.

   B, C, A: the criterium is simply the number of instructions executed inside the loop.

c. (**5 points**) Assume that the least efficient implementation is the baseline for comparison. For the other two, starting at the least efficient and moving to the most efficient version, briefly explain what the programmer, or compiler, did to the implementation to make it more efficient than the previous one.

   From B to C a guard branch is inserted before the start of the loop to eliminate the need for both a branch and a jump in the loop.

From C to A the loop index is replaced with a pointer to the memory address that contains the array element and the increment is thus replaced by a pointer increment.