Question 1 (20 points): In this question you will demonstrate that you understand operations involving arrays and pointers. In order to do that you will write the assembly code for the following function written in the C programming language. Assume that `i` is a global integer variable.

```
00   void UpdateY(int *A, int *X, int *Y, int *col) {
01       for(j = A[i] ; j < A[i+1] ; j++)
02            Y[j] =  Y[j] + X[col[j]];
03  }
```

a. (**10 points**) Using pseudo-code notation, write the low-level code that you will then use to generate the assembly code for the `UpdateY` function. This code must show all the address calculations for the array accesses. Also, loops should be coded with goto statements. In this pseudo-code, you should use the names of the variables and arrays from the C code where appropriate. Also, each pseudo-code line should correspond to an assembly instruction. Thus an statement such as `a ← b + c + d;` must be represented as `t1 ← c + d; a ← b + t1`. The following sample lines of pseudo-code are not semantically meaningful. They are intended only to illustrate the style of notation that you should use in your pseudo code:

```
       a ← b + c
L1:    Mem[x] ← y          # memory with address x receives value y
       t ← Mem[x]          # t receives value of memory with address x
       if(w ≥ z) goto L1
```

```
UpdateY:   t0 ← 4*i
           t0 ← A + t0
           j ← Mem[t0]           # j  ← A[i]
           t2 ← Mem[t0+4]        # t2 ← A[i+1]
loop_j:    if(j ≥ t2) goto done
           t3 ←  4*j
           t4 ←  col + t3
           t4 ←  Mem[t4]         # t4 ← col[j]
           t4 ←  4*t4
           t5 ←  X + t4
           t5 ←  Mem[t5]         # t5 ← X[col[j]]
           t6 ←  Y + t3
           t7 ←  Mem[t6]         # t7 ← Y[j]
           t7 ←  t7 + t5         # t7 ← Y[j] + X[col[j]]
           Mem[t6] ←  t7         # Y[j] ← Y[j] + X[col[j]]
           j ←  j+1
           goto loop_j
done:      return
```

b. (**10 points**) Following the MIPS calling conventions, write the MIPS assembly code for the `UpdateY` function using the pseudo-code that you wrote for part (a). Assume that the global variable `i` is stored in register `$s0` (this is a deviation from the standard MIPS calling convention).

```
UpdateY:    sll     $t0, $s0, 2      # $t0 ← 4*i
            add     $t0, $a0, $t0    # $t0 ← Address(A[i])
            lw      $t1, 0($t0)      # j ← A[i]
            lw      $t2, 4($t0)      # $t2 ← A[i+1]
loop_j:     bge     $t1, t2, done    # if(j ≥ t2) goto done
            sll     $t3, $t1, 2      # $t3 ← 4*j
            add     $t4, $a3, $t3    # $t4 ← Address(col[j])
            lw      $t4, 0($t4)      # $t4 ← col[j]
            sll     $t4, t4, 2       # $t4 ← 4*col[j]
            add     $t5, $a1, $t4    # $t5 ← Address(X[col[j])
            lw      $t5, 0($t5)      # $t5 ← X[col[j]]
            add     $t6, $a2, $t3    # $t6 ← Address(Y[j])
            lw      $t7, 0($t6)      # $t7 ← Y[j]
            add     $t7, $t7, $t5    # $t7 ← Y[j] + X[col[j]]
            sw      $t7, 0($t6)      # Y[j] ← Y[j] + X[col[j]]
            addi    $t1, $t1, 1      # j ← j+1
            j       $loop_j          # goto loop_j
done:       jr      $ra
```