Question 4 (30 points): From a performance standpoint linked lists are not desirable because they create an artificial dependence between the elements of the list and prevent the parallel execution of instructions. However, a linked-list is an easy-to-implement unbounded data structure that is convenient while storing an unknown amount of data. Sometimes, after all elements are added to a linked list, it may be beneficial to convert the linked list into an array. In this question you will write MIPS assembly code to transform a linked list into an array.

You will write three MIPS functions to implement this conversion: `CountElements`, `CopyToArray`, and `ConvertListToArray`. The C language declaration for the structure used to create the linked list is as follows:

```
 struct item{
     int              ID;
     int              price;
     struct item      *next;
};
```

The list is represented by the address of the first element of the list. This address is zero when the list is empty. The last element of the list has the value zero in its `next` field. Assume that in this machine an integer is represented in 32 bits and a pointer is also represented in 32 bits.

In the implementation of all routines you must follow all the MIPS function-calling and parameter-passing conventions. Also, in your code you are not allowed to use pseudo instructions that manipulate constants that are larger than 16 bits.

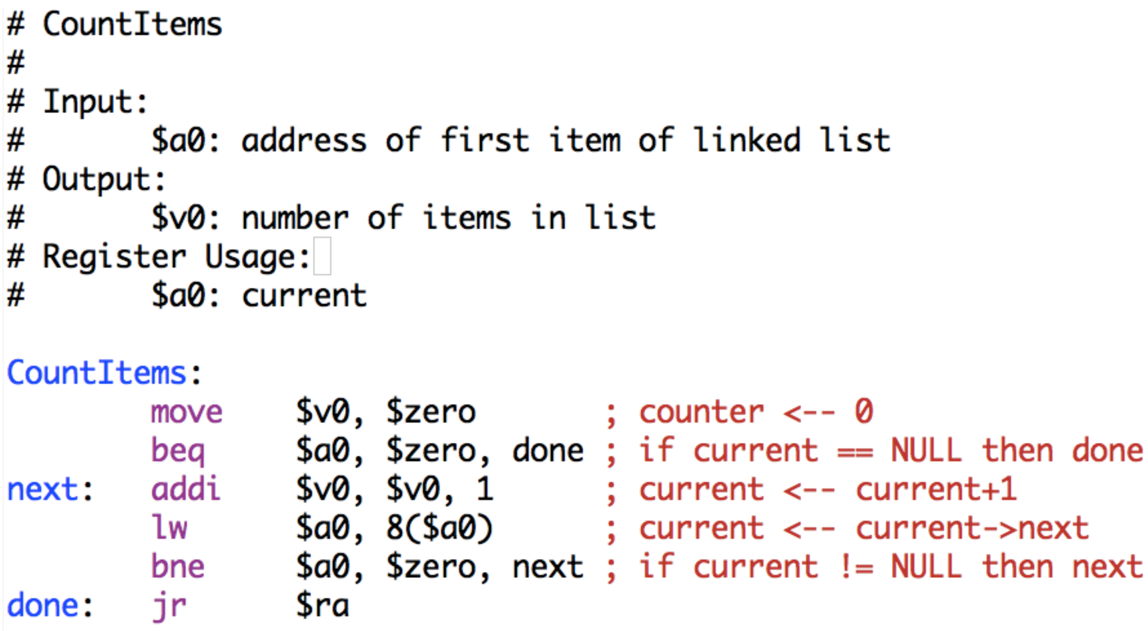
a. (**8 points**) Write MIPS assembly code for a function called `CountItems` that counts the number of items in the linked list. The interface for this routine is as follows:

Input:

`$a0`: address of first item of linked list

Output:

`$v0`: number of items in the list



```
# CountItems
#
# Input:
#       $a0: address of first item of linked list
# Output:
#       $v0: number of items in list
# Register Usage:
#       $a0: current

CountItems:
        move    $v0, $zero       ; counter <-- 0
        beq     $a0, $zero, done ; if current == NULL then done
next:   addi    $v0, $v0, 1      ; current <-- current+1
        lw      $a0, 8($a0)      ; current <-- current->next
        bne     $a0, $zero, next ; if current != NULL then next
done:   jr      $ra
```

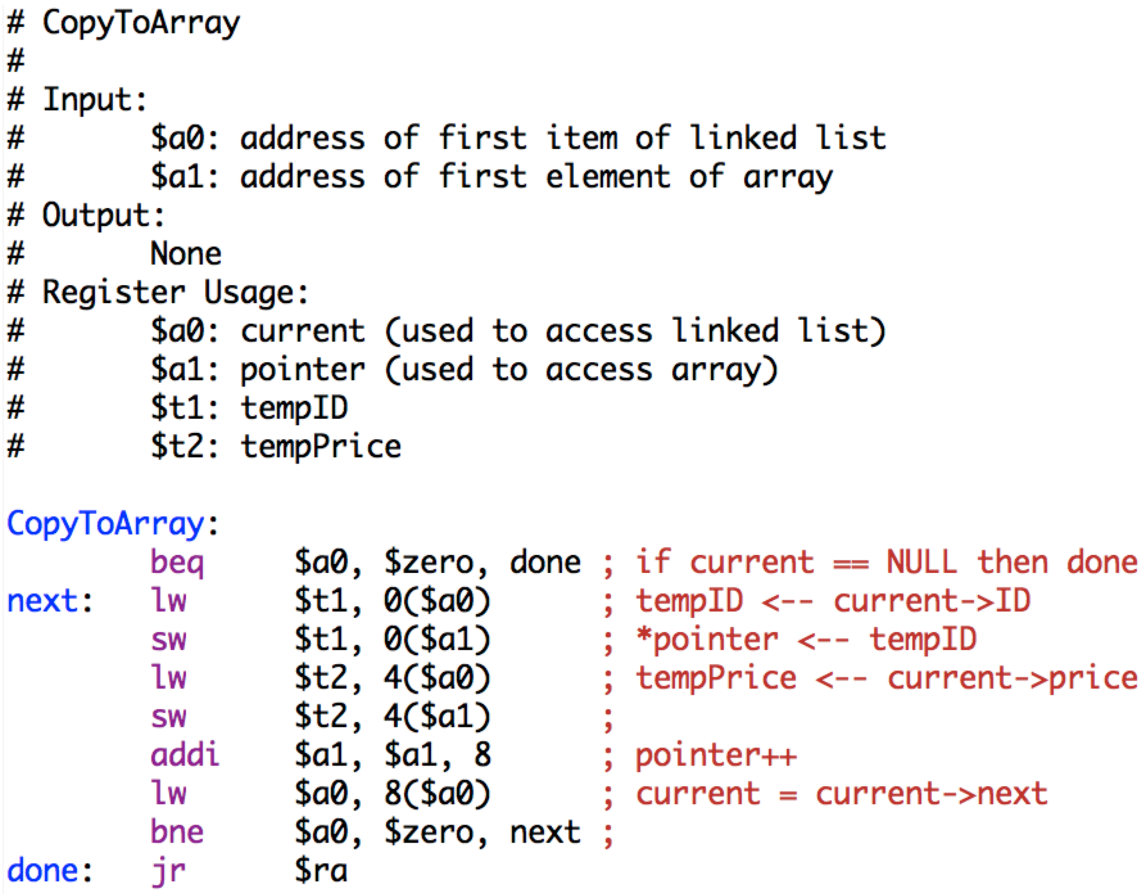
b. (**8 points**) Write MIPS assembly code for a function called `CopyToArray` that copies all the items of the linked list into an array. Unlike the linked-list items, the array elements do not need a next field. The interface for this routine is as follows:

Input:

`$a0`: address of first item of linked list
`$a1`: address of first element of array

Output:

none



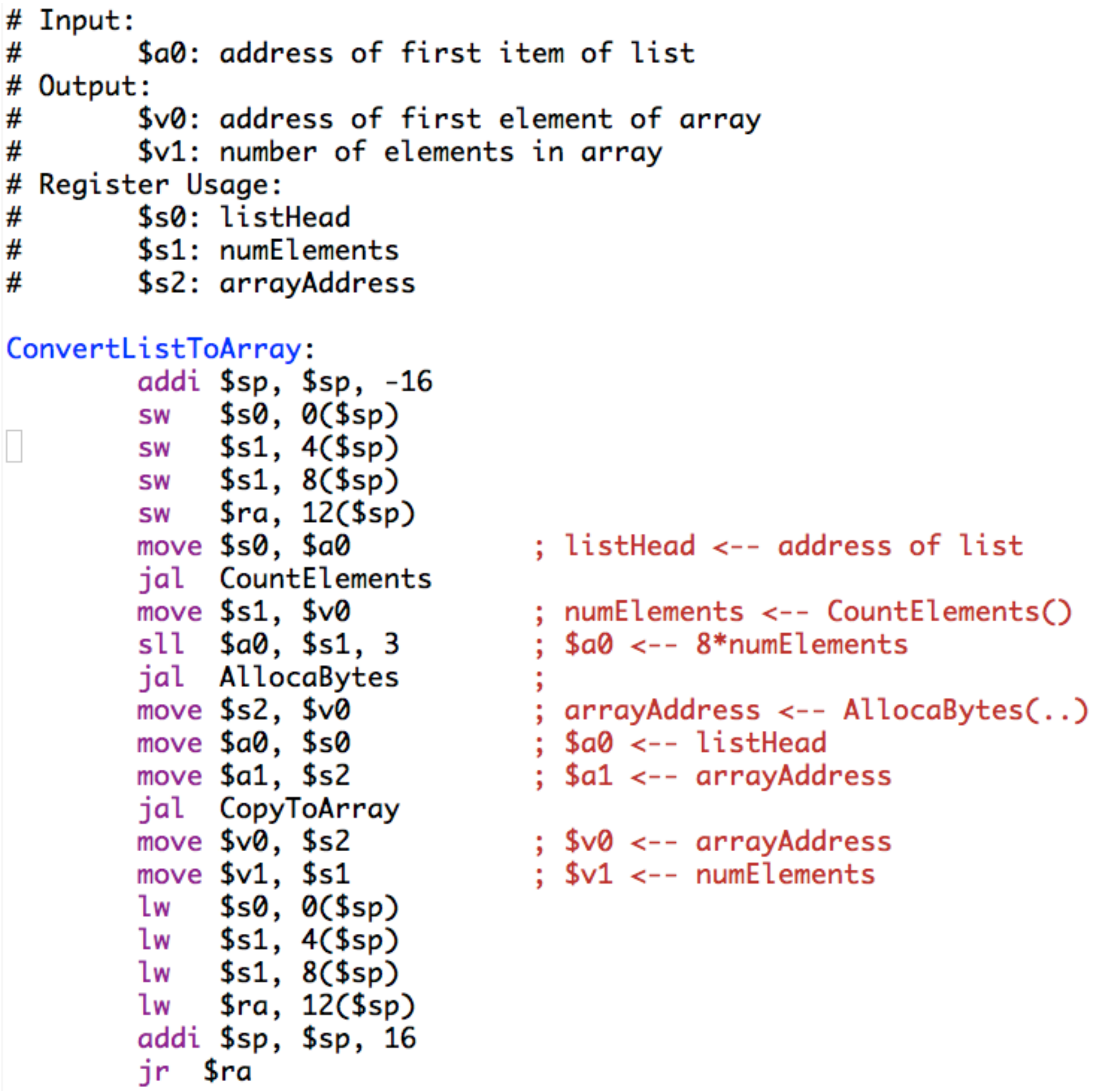
```
# CopyToArray
#
# Input:
#       $a0: address of first item of linked list
#       $a1: address of first element of array
# Output:
#       None
# Register Usage:
#       $a0: current (used to access linked list)
#       $a1: pointer (used to access array)
#       $t1: tempID
#       $t2: tempPrice

CopyToArray:
        beq     $a0, $zero, done ; if current == NULL then done
next:   lw      $t1, 0($a0)      ; tempID <-- current->ID
        sw      $t1, 0($a1)      ; *pointer <-- tempID
        lw      $t2, 4($a0)      ; tempPrice <-- current->price
        sw      $t2, 4($a1)      ;
        addi    $a1, $a1, 8      ; pointer++
        lw      $a0, 8($a0)      ; current = current->next
        bne     $a0, $zero, next ;
done:   jr      $ra
```

c. (**14 points**) Now you will use the `CountItems` and the `CopyToArray` routines specified above to implement the `ConvertListToArray` routine with the following interface:

Input:
`$a0`: address of first item of linked list

Output:
`$v0`: address of first element of array
`$v1`: number of elements in array

You also will need a routine to allocate memory for the array. You can call a function called `AllocaBytes` that receives in `$a0` the number of bytes to be allocated and returns in `$v0` the address of first allocated byte. `AllocaBytes` allocates an area of memory whose size, in bytes, was specified as its parameter and returns the address of the first byte of the allocated area. You **do not need to write the code for `AllocaBytes`**. You have to write the code for the routine `ConvertListToArray`.



```
# Input:
#       $a0: address of first item of list
# Output:
#       $v0: address of first element of array
#       $v1: number of elements in array
# Register Usage:
#       $s0: listHead
#       $s1: numElements
#       $s2: arrayAddress

ConvertListToArray:
        addi $sp, $sp, -16
        sw   $s0, 0($sp)
        sw   $s1, 4($sp)
        sw   $s1, 8($sp)
        sw   $ra, 12($sp)
        move $s0, $a0           ; listHead <-- address of list
        jal  CountElements
        move $s1, $v0           ; numElements <-- CountElements()
        sll  $a0, $s1, 3        ; $a0 <-- 8*numElements
        jal  AllocaBytes        ;
        move $s2, $v0           ; arrayAddress <-- AllocaBytes(..)
        move $a0, $s0           ; $a0 <-- listHead
        move $a1, $s2           ; $a1 <-- arrayAddress
        jal  CopyToArray
        move $v0, $s2           ; $v0 <-- arrayAddress
        move $v1, $s1           ; $v1 <-- numElements
        lw   $s0, 0($sp)
        lw   $s1, 4($sp)
        lw   $s1, 8($sp)
        lw   $ra, 12($sp)
        addi $sp, $sp, 16
        jr  $ra
```