Problem 1

There are many solutions to these problems. My solutions use as few stores and loads as possible, primarily to keep the code simple.

a)  
   lw $t0 y  
   lw $t1 x  
   sub $t0 $0 $t0 #make y = -1  
   sub $t0 $t0 $t1 #y - x  
   sw $t0 y

b) I'm assuming that $i is an array offset, and not already adjusted for word alignment.  
   lw $t0 i #get the values for mem offsets  
   addi $t1 $t0 -1  
   addi $t2 $t0 1  
   sll $t0 $t0 2 #insure word alignment  
   sll $t1 $t1 2  
   sll $t2 $t2 2  
   lw $t1 a($t1) #fetch the values  
   lw $t2 a($t2)  
   add $t1 $t1 $t2 #do the add  
   sw $t1 a($t0) #store the value

c) A clever coder could do this without the expensive branch, but it would make the code harder to read.  
   lw $t0 x  
   lw $t1 y  
   sgt $t2 $t0 $t1 #is x > y?  
   bne $t2 $0 else #looks like x is < y, so go to else case  
   sub $t0 $t0 $t1 #set x = x - y  
   j done #the new x is done, go to store it  
else: sub $t0 $t1 $t0 #set x = y - x  
   done: sw $t0 x #store the new x
d) Since there is no guarantee that the loop will execute even one time, \( i \) must be checked before executing any loop code.

\[
\begin{align*}
\text{lw} & \ $t0 \ i \quad \# \text{get } i \\
\text{j} & \ \text{test} \quad \# \text{jump to the test} \\
\text{while:} & \ \text{srl} \ $t0 \ $t0 \ 1 \quad \# \text{shift right 1} \\
\text{test:} & \ \text{andi} \ $t0 \ $t0 \ 1 \quad \# \text{and } i \text{ with 1} \\
& \ \text{beq} \ $t0 \ $0 \ \text{while} \quad \# \text{if test is met, run loop again} \\
& \ \text{sw} \ $t0 \ i \quad \# \text{store final value}
\end{align*}
\]

e) Since the loop will always run the first time, my code enters the loop with \( i \) equal to 0. Upon entering the loop structure, \( i \) is immediately incremented. For this reason, the loop runs from 1 to 9, since after the eighth time it will return to the top of the loop and begin the ninth and final run.

\[
\begin{align*}
\text{addi} & \ $t0 \ $0 \ $0 \quad \# \text{set 'i' to 0} \\
\text{for:} & \ \text{addi} \ $t0 \ $t0 \ 1 \quad \# \text{do the i++ part (the first time through, this will make i=1)} \\
& \ \text{sll} \ $t1 \ $t0 \ 2 \quad \# \text{multiply for word alignment} \\
& \ \text{add} \ $t2 \ $t0 \ $t0 \quad \# \text{set } i = i + i \\
& \ \text{addi} \ $t2 \ $t2 \ 1 \quad \# \text{set } i = i + 1 \\
& \ \text{sw} \ $t2 \ a($t1) \quad \# \text{store the new } i \text{ in memory, using word aligned value} \\
& \ \text{slti} \ $t1 \ $t0 \ 9 \quad \# \text{check if } i < 9 \text{ (because we should stop the 10th time)} \\
& \ \text{bne} \ $t1 \ $t0 \ 9 \quad \# \text{if } i \text{ is still < 9, do the loop again}
\end{align*}
\]

f) Note that this is a fairly complicated address redirection. For the purpose of this class, we will be assuming that the array is filled with indices and not memory pointers.

\[
\begin{align*}
\text{lw} & \ $t0 \ x \quad \# \text{get } x \\
\text{sll} & \ $t0 \ $t0 \ 2 \\
\text{lw} & \ $t1 \ a($t0) \quad \# \text{get value at } a[x] \\
\text{sll} & \ $t1 \ $t1 \ 2 \\
\text{lw} & \ $t1 \ a($t1) \quad \# \text{get value at } a[a[x]] \\
\text{sw} & \ $t1 \ a($t0) \quad \# \text{store the result back in } a[x]
\end{align*}
\]

**Problem 2**

a) \[
\begin{align*}
\text{add} & \ $t0 \ $t0 \ $0 \ $0 \\
\text{add} & \ $t1 \ $t1 \ $0 \ $0 \\
\text{add} & \ $t2 \ $t2 \ $0 \ $0 \\
\text{add} & \ $t3 \ $t3 \ $0 \ $0 \\
\text{add} & \ $t4 \ $t4 \ $0 \ $0 \\
\text{add} & \ $t5 \ $t5 \ $0 \ $0 \\
\text{add} & \ $t6 \ $t6 \ $0 \ $0 \\
\text{add} & \ $t7 \ $t7 \ $0 \ $0 \\
\text{add} & \ $t8 \ $t8 \ $0 \ $0
\end{align*}
\]
b)  
```
addi $t0 $0 0x100  #starting at address 0x100
loop: sw $0 0($t0)  #store 0 at the address
       addi $t0 $t0 4  #add 4 for the next address
       sge $t1 $t0 0x1fc #check if address is less than 0x1fc
       bne $t1 $0 loop  #if not greater, clear the next address
```

c)  
```
addi $t2 $t0 $0  #put value at $t0 in a temporary location
add $t0,$t1 $0  #copy value at $t1 to $t0
addi $t1 $t2 $0  #copy original $t0 value from temporary location to $t1
```

d)  
```
ad $t2 $0 $0  #set counter to 0
add $t0 $0 0x100  #starting at address 0x100
loop: lw $t1 0($t0)  #get value at the current address
       bne $t1 $0 nextaddr #if it is not 0, continue loop
       addi $t2 $t2 1  #since the value is 0, we should count it
nextaddr: addi $t0 $t0 4  #add 4 for the next address
       sge $t1 $t0 0x1fc #check if address is greater than 0x1fc
       bne $t1 $0 loop  #if not greater, clear the next address
```

**Problem 3**

a) This code is going to go through the value at $a0 and count the number of ‘1’s in its binary form.
```
add $t0,$0,$0  #start with a counter ($t0) equal to 0
loop: andi $t1,$a0,1 #and $a0 with 1, if odd, result is 1, if even result is 0
       add $t0,$t0,$t1 #increment the counter by 1 if $a0 was odd, or by 0 if $a0 was even
       srl $a0,$a0,1 #shift the value right (throw away least sig. digit)
       bne $a0,$0,loop #check if all bits have been shift out of $a0
end:  add $a0,$t0,$0  #set $a0 to the number of ‘1’s in its binary representation
```

b) In the case of numbers with 0 as their most significant bit, the result is the same as using `srl`. However, if the most significant bit is 1 (negative numbers, for example), the function would enter an infinite loop. Since the counter continues to increment, the value in $t0 would eventually overflow and generate an exception.

c) This code will begin at the first element in a list and shift all elements towards the beginning by one. The first element in the list is lost during the operation.
```
lw $t0,4($t1) #get the value at a[$t1+4] (next item) and store it in $t0
sw $t0,($t1) #store the fetched value at a[$t1] (destroying the original value)
addi $t1,$t1,4 #increment $t1 to move to the next element
bne $t0,$0,loop #continue on until a[$t1] is zero
```
d) Searches an array to find the first element with a value of zero and returns its index. Assuming the array is terminated by an element with the value of zero (for example, a string), this code would get the length of the array.

```assembly
la $t1,a  #gets a pointer to the array ‘a’
loop: lw $t0,($t1) #stores the value of a[$t1] into $t0
    addi $t1,$t1,4 #increments to the next element in the array
    bne $t0,$0,loop #if the element at a[$t1] is not a zero, keep looking
la $t0,a       #gets the address of the start of array ‘a’
sub $t0,$t1,$t0 #subtracts the address of ‘a’ from the index where the zero was found
sra $t0,$t0,2  #divides the result by 4
```

**Problem 4**

In order to reduce the complexity, this code starts at the last element in \(a\) and works towards the first. This allows for a simple check for the memory offset to be 0. To find the odd elements, each element is logically ‘and’ed with 1. Since the least significant bit in binary controls whether the value is odd, \(odd \& 1 = 1\) while \(even \& 1 = 0\).

\(a:\)

```
.word -3, 17, 12, 101, -51, 42, 16, 2, 17, -4
```

**main:**

```assembly
add $t0 $0 $0  #start with the total at zero
addi $t1 $0 10 #start at index 10
```

**loop:**

```assembly
addi $t1 $t1 -1  #decrement the offset counter
sll $t4 $t1 2   #align the value for memory access
lw $t2 a($t4)  #get the value at a[$t1]
andi $t3 $t2 1  #and with 1, the return should be 0 if even
bne $t3 $0 check #if the value is not 0 (not even), run loop again
add $t0 $t2 $t0 #keep a running sum of the even values
```

**check:**

```assembly
bne $t1 $0 loop  #if we are at the zeroth element, we’re done
j $31
```