EE2 Computer Architecture

Solutions to Problem Sheet 2 & Laboratory 2

*1.

```
begin:   addi   $t0, $zero, 0      # clear $t0 (sum)
         addi   $t1, $zero, 1      # clear $t1 (index i)
loop:    slt    $t2, $a0, $t1      # if (n < i)
         bne    $t2, $zero, finish #    goto finish
         add    $t0, $t0, $t1      # sum = sum + i
         addi   $t1, $t1, 2        # i = i + 2
         j      loop               # loop back
finish:  add    $v0, $t0, $zero    # transfer sum to $v0
```

2. i) `a = b + 100;`

```
addi $t0,$t1,100 # register $t0 = $t1 + 100
```

ii) `x[10] = x[11] + c;`

The base address of x, is 4,000,000 in decimal and , in binary, is 0000 0000 0011 1101 0000 1001 0000 0000, which implies that we must use `lui`:

```
lui   $t1, 0000 0000 0011 1101
ori   $t1, $t1, 0000 1001 0000 0000
lw    $t2, 44($t1)
add   $t2, $t2, $t0
sw    $t2, 40($t1)
```

*3. The corrected program can be:

```
      addi  $v0,$zero,-1   # Initialize to avoid counting zero word
loop: lw    $v1,0($a0)     # Read next word from source
      addi  $v0,$v0,1      # Increment count words copied
      sw    $v1,0($a1)     # Write to destination
      addi  $a0,$a0,4      # Advance pointer to next source
      addi  $a1,$a1,4      # Advance pointer to next dest
      bne   $v1,$zero,loop # Loop if the word copied ¹ zero
```

Bugs:
1. Count ($v0) is not initialized.
2. Zero word is counted. (1 and 2 fixed by initializing $v0 to –1).
3. Source pointer ($a0) incremented by 1, not 4.
4. Destination pointer ($a1) incremented by 1, not 4.

*4.

Pseudoinstruction	What it accomplishes	Solution
`move $t5, $t3`	`$t5 = $t3`	`add  $t5, $t3, $zero`
`clear $t5`	`$t5 = 0`	`add  $t5, $zero, $zero`
`li $t5, small`	`$t5 = small`	`addi $t5, $zero, small`
`li $t5, big`	`$t5 = big`	`lui  $t5, upper_half(big)` `ori  $t5, $t5, lower_half(big)`

`lw $t5, big($t3)`	`$t5 = Memory[$t3 + big]`	`li   $at, big`
		`add  $at, $at, $t3`
		`lw   $t5, 0($at)`

`addi $t5, $t3, big`	`$t5 = $t3 + big`	`li   $at, big` `add  $t5, $t3, $at`
`beq $t5, small, L`	`if ($t5 = small) go to L`	`li   $at, small` `beq  $t5, $at, L`
`beq $t5, big, L`	`if ($t5 = big) go to L`	`li   $at, big` `beq  $at, $zero, L`
`ble $t5, $t3, L`	`if ($t5 <= $t3) go to L`	`slt  $at, $t3, $t5` `beq  $at, $zero, L`
`bgt $t5, $t3, L`	`if ($t5 > $t3) go to L`	`slt  $at, $t3, $t5` `bne  $at, $zero, L`
`bge $t5, $t3, L`	`if ($t5 >= $t3) go to L`	`slt  $at, $t5, $t3` `beq  $at, $zero, L`

Note: In the solutions, we make use of the li instruction, which should be implemented as shown in rows 3 and 4.