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

3.5 The base address of x, 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)
```

3.9 The C loop is

```
while (save[i] == k)
      i = i + j;
```

with i, j, and k corresponding to registers $s3, $s4, and $s5 and the base of the array save in $s6. The assembly code given in the example is

Code before

```
Loop: add $t1, $s3, $s3    # Temp reg $t1 = 2 * i
      add $t1, $t1, $t1    # Temp reg $t1 = 4 * i
      add $t1, $t1, $s6    # $t1 = address of save [i]
      lw  $t0, 0($t1)      # Temp reg $t0 = save[i]
      bne $t0, $s5, Exit   # go to Exit if save[i] ≠ k
      add $s3, $s3, $s4    # i = i + j
      j Loop               # go to Loop
Exit:
```

Number of instructions executed if save[i + m * j] does not equal k for m = 10 and does equal k for $0 \le m \le 9$ is $10 \times 7 + 5 = 75$, which corresponds to 10 complete iterations of the loop plus a final pass that goes to Exit at the bne instruction before updating i. Straightforward rewriting to use at most one branch or jump in the loop yields

Code after

```
      add $t1, $s3, $s3    # Temp reg $t1 = 2 * i
      add $t1, $t1, $t1    # Temp reg $t1 = 4 * i
      add $t1, $t1, $s6    # $t1 = address of save[i]
      lw  $t0, 0($t1)      # Temp reg $t0 = save[i]
      bne $t0, $s5, Exit   # go to Exit if save[i] ≠ k
Loop: add $s3, $s3, $s4    # i = i + j
      add $t1, $s3, $s3    # Temp reg $t1 = 2 * i
      add $t1, $t1, $t1    # Temp reg $t1 = 4 * i
      add $t1, $t1, $s6    # $t1 = address of save[i]
      lw  $t0, 0($t1)      # Temp reg $t0 = save[i]
      beq $t0, $s5, Loop   # go to Loop if save[i] = k
Exit:
```

The number of instructions executed by this new form of the loop is $5 + 10 \times 6 = 65$. If $4 \times j$ is computed before the loop, then further saving in the loop body is possible.

Code after further improvement

```
      add $t2, $s4, $s4    # Temp reg $t2 = 2 * j
      add $t2, $t2, $t2    # Temp reg $t2 = 4 * j
      add $t1, $s3, $s3    # Temp reg $t1 = 2 * i
      add $t1, $t1, $t1    # Temp reg $t1 = 4 * i
```

```
        add $t1, $t1, $s6    # $t1 = address of save[i]
        lw  $t0, 0($t1)      # Temp reg $t0 = save[i]
        bne $t0, $s5, Exit   # go to Exit if save[i] ≠ k
Loop:   add $t1, $t1, $t2    # $t1 = address of save [i + m * j]
        lw  $t0, 0($t1)      # Temp reg $t0 = save[i]
        beq $t0, $s5, Loop   # go to Loop if save[i] = k
Exit:
```

The number of instructions executed is now $7 + 10 \times 3 = 37$.

3.12 The problem is that we are using PC-relative addressing, so if the address of `there` is too far away, we won't be able to use 16 bits to describe where it is relative to the PC. One simple solution would be

```
here:   bne $t1, $t2, skip
        j there
skip:
        ...
there:  add $t1, $t1, $t1
```

This will work as long as our program does not cross the 256-MB address boundary described in the elaboration on page 150.

3.14 From Figure 3.38, 33% of all instructions executed by spice are data access instructions. Thus, for every 100 instructions there are 100 + 33 = 133 memory accesses: one to read each instruction and 33 to access data.

a. The percentage of all memory accesses that are for data = 33/133 = 25%.

b. Assuming two-thirds of data transfers are loads, the percentage of all memory accesses that are reads = $\dfrac{100 + \left(33 \times \frac{2}{3}\right)}{133} = 92\%$.