Solution 6

1.

(a)

−128.625 → −10000000.101 = $-1.0000000101*2^{7}$ =
$(-1)^{\mathbf{1}}*2^{(\mathbf{134}-127)}*1.0000000101$ → **1 10000110 00000001010000000000000**.

(b)

0 00000000 10000000000000000000000 (underflow) = $(-1)^{\mathbf{0}}*2^{\mathbf{-126}}*0.1$
→ $0.5*2^{-126}$ = $\mathbf{5.877471754*10^{-39}}$.

(c)

0 01111111 00000000000000000000000 = $(-1)^{\mathbf{0}}*2^{(\mathbf{127}-127)}*1.0$
→ $1*2^{0}$ = **1**.

(d)

X = 1 10000011 **1**00101001111000000000000

−Y = 1 01111100 **1**10000000000000000000000

= 1 10000011 0000000**1**1000000000000000

X +(−Y) = 1 10000011 **1**00101100111000000000000

= −**1**.00101100111$*2^{4}$ → **−18.8046875**

(e)

−Y = 1100 0001 1100 0000 0000 0000 0000 0000

X+(−Y) = 1000 0011 0101 0100 1111 0000 0000 0000 → **−2,091,585,536**

2.

```
MOV    R2, R0               // R0 = R2
ADD    #4, R4, R4           // R4 = R4 + 4
NOP                         // Waiting for R0
NOP                         // Waiting for R0
ADD    R0, R2, R1           // R1 = R0 + R2
MOV    R4, R2               // R2 = R4
MOV    (R4), R6             // R6 = MEMORY[R4]
NOP                         // Waiting for R1
MOV    R3, (R1)             // MEMORY[R1] = R3
ADD    R0, R2, R3           // R3 = R0 + R2
ADD    R4, R6, R5           // R5 = R4 + R6
ADD    R2, R4, R1           // R1 = R2 + R4
```

3.

```
#include      <stdio.h>           /* Routines for input/output. */
#include      ”threads.h”         /* Routines for thread creation/synchronization. */

#define       N    100            /* Number of elements in each vector. */
#define       P    4              /* Number of processors for parallel execution. */

double        a[N], b[N];         /* Vectors for computing the dot product. */
double        dot_product;        /* The global sum of partial results computed by the threads. */
volatile int  thread_id_counter;  /* Used to ensure exclusive access to dot_product. */
                                  /* Note that the counter is declared as volatile. */

void          ParallelFunction (void)
{
              int my_id, i, start, end;
              double s;

              my_id = get_my_thread_id ();    /* Get unique identifier for this thread. */
              start = (N/P) * my_id;    /* Determine start/end using thread identifier. */
              end = (N/P) * (my_id + 1) − 1;    /* N is assumed to be evenly divisible by P. */
              s = 0.0;
              for (i = start; i <= end; i++)
                 s = s + a[i] * b[i];

              while (thread_id_counter != my_id);    /* Wait for permission to proceed. */
              dot_product = dot_product + s;    /* Update dot_product. */
              thread_id_counter = thread_id_counter + 1;    /* Give permission to next thread. */
}

void          main (void)
{
              int i;

              <Initialize vectors a[], b[] – details omitted.>
              dot_product = 0.0;    /* Initialize sum of partial results. */
              thread_id_counter = 0;    /* Initialize counter that ensures exclusive access. */
              for (i = 1; i < P; i++)    /* Create P − 1 additional threads. */
                 create_thread (ParallelFunction);
              ParallelFunction();    /* Main thread also joins parallel execution. */
              while (thread_id_counter != P);    /* Wait until last update to dot_product. */
              printf (”The dot product is %g\n”, dot_product);
}
```