# Solution 6

**1.**

(a)

**−25.25** → −11001.01 = −1.100101*$2^4$ =
$(-1)^{\mathbf{1}}$*$2^{(\mathbf{131}-127)}$*1.100101 → **1 10000011 10010100000000000000000**.

(b)

**0 00000000 11000000000000000000000** (underflow) = $(-1)^{\mathbf{0}}$*$2^{\mathbf{-126}}$*0.11
→ 0.75*$2^{-126}$ ≈ **8.81620763*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**

**2.**

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

**3.**

Given **P = 8** and **Speedup = 5**, we need to solve **5 = 1/(1 − f + f/8)**, which yields **f = 0.91**, i.e., an application program must be 91% parallelizable.

**4.**

```
#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);
}
```