Problem 1

Rewrite the code for prefixSum(Table 1) so that it does not need to repeatedly retrieve the value of p[i] from memory. You do not need to use loop unrolling. We measured the resulting code to have a CPE of 3.00, limited by the latency of floating-point addition.

```
/* compute prefix sum of vector a */
void prefixSum(float a[], float p[], long int n)
{
  long int i;
  p[0] = a[0];
  for(i = 1; i < n; i++)
          p [i] = p[i-1] + a[i];
}
```

Table 1

Problem 2

Suppose that we want to enhance the processor used for Web serving. The new processor is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement according to Amdahl's law?

Problem 3

1. Consider the following function with potential load-store interactions:

```
void foo (int *src, int *dest, int n)
{
      int i;
      for ( i = 0; i < n; i++ )
            dest[i] = src[i] + i;
}
```

Performance of arithmetic operations is list below:

Instruction	Latency	Cycles/Issue
Load / Store (Cache hit)	3	1
Load / Store (Cache miss)	20	1
Integer Multiply	4	1
Integer Divide	36	36
Double/Single FP Multiply	5	2
Double/Single FP Add	3	1

- Double/Single FP Divide 38 38

Suppose *a* is an array of length 500 and all data are always available in cache.

1) What is the expected CPE for call foo(*a*+1, *a* , 499) and foo(*a*, *a*+1 , 499) .
2) What is the expected CPE without cache?

Problem 4

Suppose we have a block of code, as follows in table 2. After running the code, you found that the performance is not very good. Try to optimize the function using any technologies you've learnt in the ICS class.

```
void test( ){
    int x = 1, y = 2;
    int a[100];
    for(int i = 0; i < 100; i++)
    {
        if(i % 2 == 0) a[i] = x;
        else
            a[i] = y;
    }
}
```

Table 2