1.6 [20] <§1.6> Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (class A, B, C, and D). P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3, and P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2.

Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 10% class A, 20% class B, 50% class C, and 20% class D, which implementation is faster?

a. What is the global CPI for each implementation?

b. Find the clock cycles required in both cases.

1.8 The Pentium 4 Prescott processor, released in 2004, had a clock rate of 3.6 GHz and voltage of 1.25 V. Assume that, on average, it consumed 10 W of static power and 90 W of dynamic power.

The Core i5 Ivy Bridge, released in 2012, had a clock rate of 3.4 GHz and voltage of 0.9 V. Assume that, on average, it consumed 30 W of static power and 40 W of dynamic power.

1.8.1 [5] <§1.7> For each processor find the average capacitive loads.

1.8.2 [5] <§1.7> Find the percentage of the total dissipated power comprised by static power and the ratio of static power to dynamic power for each technology.

1.8.3 [15] <§1.7> If the total dissipated power is to be reduced by 10%, how much should the voltage be reduced to maintain the same leakage current? Note: power is defined as the product of voltage and current.

1.9 Assume for arithmetic, load/store, and branch instructions, a processor has CPIs of 1, 12, and 5, respectively. Also assume that on a single processor a program requires the execution of 2.56E9 arithmetic instructions, 1.28E9 load/store instructions, and 256 million branch instructions. Assume that each processor has a 2 GHz clock frequency.

Assume that, as the program is parallelized to run over multiple cores, the number of arithmetic and load/store instructions per processor is divided by 0.7 x p (where p is the number of processors) but the number of branch instructions per processor remains the same.

1.9.1 [5] <§1.7> Find the total execution time for this program on 1, 2, 4, and 8 processors, and show the relative speedup of the 2, 4, and 8 processor result relative to the single processor result.

1.9.2 [10] <§§1.6, 1.8> If the CPI of the arithmetic instructions was doubled, what would the impact be on the execution time of the program on 1, 2, 4, or 8 processors?

1.9.3 [10] <§§1.6, 1.8> To what should the CPI of load/store instructions be reduced in order for a single processor to match the performance of four processors using the original CPI values?

1.15 [5] <§1.8> When a program is adapted to run on multiple processors in a multiprocessor system, the execution time on each processor is comprised of computing time and the overhead time required for locked critical sections and/or to send data from one processor to another.

Assume a program requires t = 100 s of execution time on one processor. When run *p* processors, each processor requires t/p s, as well as an additional 4 s of overhead, irrespective of the number of processors. Compute the per-processor execution time for 2, 4, 8, 16, 32, 64, and 128 processors. For each case, list the corresponding speedup relative to a single processor and the ratio between actual speedup versus ideal speedup (speedup if there was no overhead).

2.26 Consider the following MIPS loop:

```
LOOP: slt  $t2, $0,  $t1
      beq  $t2, $0,  DONE
      subi $t1, $t1, 1
      addi $s2, $s2, 2
      j    LOOP
DONE:
```

2.26.1 [5] <§2.7> Assume that the register `$t1` is initialized to the value 10. What is the value in register `$s2` assuming `$s2` is initially zero?

2.26.2 [5] <§2.7> For each of the loops above, write the equivalent C code routine. Assume that the registers `$s1`, `$s2`, `$t1`, and `$t2` are integers `A`, `B`, `i`, and temp, respectively.

2.26.3 [5] <§2.7> For the loops written in MIPS assembly above, assume that the register `$t1` is initialized to the value N. How many MIPS instructions are executed?

2.27 [5] <§2.7> Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of `a`, `b`, `i`, and `j` are in registers `$s0`, `$s1`, `$t0`, and `$t1`, respectively. Also, assume that register `$s2` holds the base address of the array D.

```
for(i=0; i<a; i++)
   for(j=0; j<b; j++)
      D[4*j] = i + j;
```

2.28 [5] <§2.7> How many MIPS instructions does it take to implement the C code from Exercise 2.27? If the variables a and b are initialized to 10 and 1 and all elements of D are initially 0, what is the total number of MIPS instructions that is executed to complete the loop?

2.31 [5] <§2.8> Implement the following C code in MIPS assembly. What is the total number of MIPS instructions needed to execute the function?

```
int fib(int n){
    if (n==0)
       return 0;
    else if (n == 1)
       return 1;
    else
       return fib(n-1) + fib(n-2);
```

2.43 [5] <§2.11> Write the MIPS assembly code to implement the following C code:

```
lock(lk);
shvar=max(shvar,x);
unlock(lk);
```

Assume that the address of the `lk` variable is in `$a0`, the address of the shvar variable is in `$a1`, and the value of variable `x` is in `$a2`. Your critical section should not contain any function calls. Use `ll/sc` instructions to implement the `lock()` operation, and the `unlock()` operation is simply an ordinary store instruction.

2.46 Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 500 million arithmetic instructions, 300 million load/store instructions, 100 million branch instructions.

2.46.1 [5] <§2.19> Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, and the cost of increasing the clock cycle time by only 10%. Is this a good design choice? Why?

2.46.2 [5] <§2.19> Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?