PART A:

Solutions 1.4: (6 points)

1.4.1
Class A: \(10^5\) instr.
Class B: \(2 \times 10^5\) instr.
Class C: \(5 \times 10^5\) instr.
Class D: \(2 \times 10^5\) instr.

Time = \text{No. instr.} \times \text{CPI/clock rate}

P1: Time class A = \(0.66 \times 10^{-4}\)
Time class B = \(2.66 \times 10^{-4}\)
Time class C = \(10 \times 10^{-4}\)
Time class D = \(5.33 \times 10^{-4}\)
Total time P1 = \(18.65 \times 10^{-4}\)

P2: Time class A = \(10^{-4}\)
Time class B = \(2 \times 10^{-4}\)
Time class C = \(5 \times 10^{-4}\)
Time class D = \(2 \times 10^{-4}\)
Total time P2 = \(10 \times 10^{-4}\)

1.4.2 CPI = \text{time} \times \text{clock rate}/\text{No. instr.}
CPI(P1) = \(18.65 \times 10^{-4 \times 1.5 \times 10^9}/10^6 = 2.79\)
CPI(P2) = \(10 \times 10^{-4 \times 2 \times 10^9}/10^6 = 2.0\)

1.4.3

clock cycles(P1) = \(10^5 \times 1 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 3 + 2 \times 10^5 \times 4 = 28 \times 10^5\)

clock cycles(P2) = \(10^5 \times 2 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 2 + 2 \times 10^5 \times 2 = 20 \times 10^5\)

1.4.4

\((500 \times 1 + 50 \times 5 + 100 \times 5 + 50 \times 2) / (2 \times 10^9) = 675\) ns

1.4.5 CPI = \text{time} \times \text{clock rate}/\text{No. instr.}
CPI = \(675 \times 10^{-9 \times 2 \times 10^9}/700 = 1.92\)

1.4.6

Time = \((500 \times 1 + 50 \times 5 + 50 \times 5 + 50 \times 2) / (2 \times 10^9) = 550\) ns

Speed-up = \(675\) ns/\(550\) ns = \(1.22\)

CPI = \(550 \times 10^{-9 \times 2 \times 10^9}/650 = 1.69\)

Solutions 1.15: (6 points)

1.15.1
a. New FP time = \(T_{fp} = 35 \times 0.8 = 28\) s,
New total time = \(T_p = 28 + 85 + 50 + 30 = 193\) s. Reduction: 3.5%

1.15.2
a. New total time = \(T_p = 200 \times 0.8 = 160\) s,
\(T_{fp} + T_{ls} + T_{branch} = 115\) s,
\(T_{func} = T_p - 115 = 45\) s.
Reduction time INT: 47%

1.15.3
a. \(T_p = 200 \times 0.8 = 160\) s,
But, \(T_{fp} + T_{ls} + T_{branch} = 170\) s.
Since Sum all operations except branch is greater than 160, answer is NO.

1.15.4
Clock cycles = \(CPI_{fp} \times \text{No. FP instr.} + CPI_{ls} \times \text{No. INT instr.} + CPI_{branch} \times \text{No. L/S instr.} + CPI_{branch} \times \text{No. branch instr.}\)
\(T_{cpu} = \text{clock cycles/clock rate} = \text{clock cycles}/2 \times 10^9\)
a. 1 processor: clock cycles = \(8192\); \(T_{cpu} = 4.096\) s
To half the number of clock cycles by improving the CPI of FP instructions:

\[
\text{CPI}_{\text{improved FP}} \times \text{No. FP instr.} + \text{CPI}_{\text{int}} \times \text{No. INT instr.} + \text{CPI}_{\text{l/s}} \times \text{No. L/S instr.} + \text{CPI}_{\text{branch}} \times \text{No. branch instr.} = \text{clock cycles/2}
\]

\[
\text{CPI}_{\text{improved FP}} = \frac{\text{clock cycles/2} - (\text{CPI}_{\text{int}} \times \text{No. INT instr.} + \text{CPI}_{\text{l/s}} \times \text{No. L/S instr.} + \text{CPI}_{\text{branch}} \times \text{No. branch instr.})}{\text{No. FP instr.}}
\]

a. \( \text{CPI}_{\text{improved FP}} = \frac{4096 - 7632}{560} < 0 \Rightarrow \text{not possible} \)

1.15.5 Using the clock cycle data from 1.15.4 and the same procedure as above,

a. \( \text{CPI}_{\text{improved l/s}} = \frac{4096 - 3072}{1280} = 0.8 \)

1.15.6

\( \text{CPI}_{\text{int}} = 0.6 \times 1 = 0.6; \text{CPI}_{\text{fp}} = 0.6 \times 1 = 0.6; \text{CPI}_{\text{l/s}} = 0.7 \times 4 = 2.8; \text{CPI}_{\text{branch}} = 0.7 \times 2 = 1.4 \)

a. \( T_{\text{cpu(before improv.)}} = 4.096 \text{ s}; T_{\text{cpu(after improv.)}} = 2.739 \text{ s} \)

\( \Rightarrow \text{33\% improvement} \)

**PART B:**

```c
int max(int a, int b)
{
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```

```c
int sum(int * numbers, int count)
{
    int s = 0;
    int i;
    for(i = 0; i < count; i++) {
        s += numbers[i];
    }
    return s;
}
```

1. Please use gcc to generate the assembly for these two functions without optimization (-O0).

   1.1 Identify each function in the output file, and then create an annotated version of the assembly that includes a brief description of what each instruction is doing (referencing statements in the source code where appropriate), and a count of the arithmetic operations and memory accesses required by each instruction (check the slides from Thursday for an example).

   To figure out what the assembly code does, use the four documents listed below. The first one should have most of what you need and includes common addressing modes as well, so start there. However, depending on which version of gcc your machine has installed and which particular processor you are running, you may encounter additional instructions. The other documents will be helpful in this case. Unfortunately, we have not found a good, concise listing of x86 instructions.

   1.2 How many arithmetic operations and memory accesses will \text{sum}(a, 10) require (assuming 'a' is an array of 10 integers)?

   1.3 How about \text{max}(4, 1)?

2. Repeat the previous problem, but compile the program with optimization level 1 (-O1).

3. Assuming arithmetic operations and memory accesses take the same amount of time, what is the speedup of the optimized version relative to the unoptimized version?
SOLUTIONS

1.1: (2 points)

```
.globl max
.type max, @function

max:
pushl %ebp 1 1
movl %esp, %ebp 1
movl 8(%ebp), %eax 2 1
cmpl 12(%ebp), %eax 2 1
jle .L2 1
movl 8(%ebp), %eax 2 1
jmp .L3 1
.L2:
movl 12(%ebp), %eax 0
.L3:
popl %ebp 1 1
ret 1 1
```

(2 points)

1.2 Sum(a,10) : Arithmetic - 178, Memory - 78

1.3: Max(4,1) : Arithmetic - 12, Memory - 6
2. Optimized version: (2 points)

<table>
<thead>
<tr>
<th></th>
<th>Arithmetic</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.globl max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.type max</td>
<td>max, @function</td>
<td></td>
</tr>
<tr>
<td>max:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pushl %ebp</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>movl %esp, %ebp</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>movl 8(%ebp), %edx</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>movl 12(%ebp), %eax</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>cmpl %edx, %eax</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>jge .L2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>movl %edx, %eax</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>.L2:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>popl %ebp</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ret</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>.size max</td>
<td>max, -max</td>
<td></td>
</tr>
<tr>
<td>.globl sum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.type sum</td>
<td>sum, @function</td>
<td></td>
</tr>
<tr>
<td>sum:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pushl %ebp</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>movl %esp, %ebp</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>pushl %ebx</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>movl 8(%ebp), %ebx</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>movl 12(%ebp), %ecx</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>movl $0, %edx</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>movl $0, %eax</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>testl %ecx, %ecx</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>jle .L6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>.L6:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>popl %ebx</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>popl %ebp</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ret</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

2.2: Sum(a,10) : Arithmetic - 74, Memory - 17
2.3: Max(4,1) : Arithmetic - 10, Memory - 5

3. (2 points)
Max(4,1) Speedup = (12+6)/(10+5) = 1.2
Sum(a,10 Speedup = (178+78)/(74+17) = 2.81