Assignment 1

Alex Clemmer
Student number: u0458675

Problem 1:

1.3.1: The clear winner is P2. The rightmost column is the result of the performance function (which is the middle column):

Processor	Performance Function	Performance
P1	$\frac{\text{I}\times 1.5 \text{ CPI}}{2\times 10^9} = \text{I} \cdot 500 \text{ ps} \cdot 1.5 \text{ CPI}$	$750 \cdot \text{I ps}$
P2	$\frac{\text{I}\times 1.0 \text{ CPI}}{1.5\times 10^9} \approx \text{I} \cdot 666.666... \text{ ps} \cdot 1.0 \text{ CPI}$	$666.666... \cdot \text{I ps}$
P3	$\frac{\text{I}\times 2.5 \text{ CPI}}{3\times 10^9} \approx \text{I} \cdot 833.333... \text{ ps} \cdot 1.0 \text{ CPI}$	$833.333... \cdot \text{I ps}$

1.3.2: Total cycles is equal to the number of instructions executed in a second times the number of seconds. For P1, $2 \times 10^9 \times 10$ seconds $= 2 \times 10^{10}$ cycles. Similarly, for P2, $1.5 \times 10^9 \times 10$ seconds $= 1.5 \times 10^{10}$ cycles, and for P3, $3 \times 10^9 \times 10$ seconds $= 3 \times 10^{10}$ cycles.

Given the total cycles for the process, we can also calculate the total instructions using the formula $\frac{\text{total cycles}}{\text{cycles per instruction}} = \text{instructions}$. That means $\text{P1}_{instr} = \frac{2\times 10^{10} cycles}{1.5 \text{ CPI}}$, $\text{P2}_{instr} = \frac{1.5\times 10^{10} cycles}{1.0 \text{ CPI}}$, and $\text{P3}_{instr} = \frac{3\times 10^{10} cycles}{2.5 \text{ CPI}}$, which is all summarized by the third column of this chart:

Clock Cycles and total instructions given 10-second execution time		
Processor	Cycles	Total Instructions Executed
P1	2×10^{10} cycles	$1.333... \times 10^{10}$ instructions
P2	1.5×10^{10} cycles	1.5×10^{10} instructions
P3	3×10^{10} cycles	1.2×10^{10} instructions

1.3.3: So CPU time $= \frac{\text{Instruction Count} \cdot \text{CPI}}{\text{Clock rate}}$. Since time is going down by 30%, CPU time will always be 7 seconds. Of course, since we're using picoseconds, we have to convert that 7 seconds into picoseconds. CPI is going up 20%, and will be reflected as follows:

$$P1 : 7 \times 10^{12}\text{picoseconds} = \frac{\text{I} \cdot 1.8 \text{ CPI}}{\text{Clock rate}} \approx 2.5714 GHz \quad (1)$$

$$P2 : 7 \times 10^{12}\text{picoseconds} = \frac{\text{I} \cdot 1.2 \text{ CPI}}{\text{Clock rate}} \approx 1.7143 GHz \quad (2)$$

$$P3 : 7 \times 10^{12}\text{picoseconds} = \frac{\text{I} \cdot 2.5 \text{ CPI}}{\text{Clock rate}} \approx 3.5714 GHz \quad (3)$$

Follow-up: Any number of features could affect this performance. One processor might be an FPU, for example, and optimized for floating-point calculations, while the others might be CPUs. Or one processor might have a lot more (or a lot fewer) registers, or registers placed poorly on the chip. Different instruction sets make a difference, as some are more efficient than others.

Problem 2:

1.11.1: To calculate yield, we must first find the die area. For (a): 90 dies per wafer $= \frac{\text{wafer area}}{\text{die area}} = \frac{(\frac{15}{2})^2\pi}{\text{die area}_{(a)}}; \therefore$ die area$_{(a)} = \frac{5\pi}{8}$. For (b): 140 dies per wafer $= \frac{(\frac{25}{2})^2\pi}{\text{die area}_{(b)}}; \therefore$ die area$_{(b)} = \frac{125\pi}{112}$.

So the yield will be as follows:

$$\text{Yield}_{(a)} = \frac{1}{(1 + .018\frac{\frac{5\pi}{8}}{2})^2} \approx .965572 \tag{4}$$

$$\text{Yield}_{(b)} = \frac{1}{(1 + .024\frac{\frac{125\pi}{112}}{2})^2} \approx .92087 \tag{5}$$

1.11.2: Cost per die $= \frac{\text{cost per wafer}}{\text{dies per wafer} \times \text{yield}}$, so the operations are pretty straightforward:

$$\text{Cost per die}_{(a)} = \frac{10}{90 \cdot .96557} \approx .115073 \tag{6}$$

$$\text{Cost per die}_{(b)} = \frac{20}{140 \cdot .92087} \approx .155131 \tag{7}$$

1.11.3: Given a 10% increase in dies per wafer, (a) now has 99 DPW and (b) now has 154 DPW. So our die area (using the formula from 1.11.1) will look like so:

$$99 \text{ dies per wafer} = \frac{(\frac{15}{2})^2\pi}{\text{die area}_{(a)}}; \therefore \text{die area}_{(a)} = \frac{25\pi}{44} \tag{8}$$

$$154 \text{ dies per wafer} = \frac{(\frac{25}{2})^2\pi}{\text{die area}_{(b)}}; \therefore \text{die area}_{(a)} = \frac{625\pi}{616} \tag{9}$$

Given a 15% increase in defects per area, (a) is now at .0207 DPA, while (b) is now at .0276 DPA.

$$\text{New Yield}_{(a)} = \frac{1}{(1 + .018\frac{\frac{5\pi}{8}}{2})^2} \approx .96405 \tag{10}$$

$$\text{New Yield}_{(b)} = \frac{1}{(1 + .024\frac{\frac{125\pi}{112}}{2})^2} \approx .917507 \tag{11}$$

Follow-up: There are lots of reasons that chip that takes up 80% of a wafer would be very hard to make. The first (and most severe) of which is, the chances of defects appearing somewhere on the chip are enormous.

Consider, a chip that takes up 80% of wafer with area $\pi \cdot 6^2$cm would take approximately $81cm^2$. Assuming a modest defect per area rate of .0187, yield $= \frac{1}{(1+(.0187\frac{81}{2})^2} \approx .323805$. So around 32% of chips would survive the process.

It may be an amazing bit of hardware, but it's really hard to believe any firm would consider making a chip with rates like those, especially since they

would have to reflect the losses in the consumer price. It just wouldn't be worth it.

Problem 3:

1.10.1: We can calculate the number of instructions required for a job by adding up the number of instructions from each sub-set of the work (e.g., instructions = arithmetic + branch + load/store).

	Total Processors	Total Instruction Count	Aggregated Instruction Count
a	1	$2560 + 1280 + 256 = \mathbf{4096}$	$4096 \cdot 1 = \mathbf{4096}$
	2	$1280 + 640 + 128 = \mathbf{2048}$	$2048 \cdot 2 = \mathbf{4096}$
	4	$640 + 320 + 64 = \mathbf{1024}$	$1024 \cdot 4 = \mathbf{4096}$
	8	$320 + 160 + 32 = \mathbf{512}$	$512 \cdot 4 = \mathbf{4096}$
b	1	$2560 + 1280 + 256 = \mathbf{4096}$	$4096 \cdot 1 = \mathbf{4096}$
	2	$1350 + 8000 + 128 = \mathbf{2278}$	$2278 \cdot 2 = \mathbf{4556}$
	4	$800 + 600 + 64 = \mathbf{1464}$	$1464 \cdot 4 = \mathbf{5856}$
	8	$600 + 500 + 32 = \mathbf{1132}$	$1132 \cdot 4 = \mathbf{9056}$

1.10.2: We can obtain the execution time by multiplying the cycles per instruction by the time it takes to execute one clock cycle: $\frac{1}{\text{length of cycle}} \cdot \text{CPI}$, where *length of cycle* $= 2 \cdot 10^9$ seconds. Thus,

	Total Processors	Total Execution Time
a	1	$\frac{1}{781250} + \frac{1}{390625} + \frac{1}{3906250} \approx .000004$
	2	$\frac{1}{1562500} + \frac{1}{781250} + \frac{1}{7812500} \approx .000002$
	4	$\frac{1}{3125000} + \frac{1}{1562500} + \frac{1}{15625000} \approx .000001$
	8	$\frac{1}{6250000} + \frac{1}{3125000} + \frac{1}{31250000} \approx 3.2 \times 10^{-8}$
b	1	$\frac{1}{781250} + \frac{429}{50000000} + \frac{1}{3906250} \approx .000004$
	2	$\frac{27}{40000000} + \frac{3}{1250000} + \frac{1}{7812500} \approx .000003203$
	4	$\frac{1}{2500000} + \frac{27}{10000000} + \frac{1}{15625000} \approx .000003164$
	8	$\frac{3}{10000000} + \frac{13}{4000000} + \frac{1}{31250000} \approx .000003582$

1.10.3: What happens when we double the arithmetic instructions? Let's find out:

Execution Time with Doubled Arithmetic Instructions		
	Total Processors	Total Execution Time
b	1	$\frac{1}{390625} + \frac{429}{50000000} + \frac{1}{3906250} \approx .000005396$
	2	$\frac{27}{20000000} + \frac{3}{1250000} + \frac{1}{7812500} \approx .000003878$
	4	$\frac{1}{1250000} + \frac{27}{10000000} + \frac{1}{15625000} \approx .000003564$
	8	$\frac{3}{10000000} + \frac{13}{4000000} + \frac{1}{31250000} \approx .000003582$

So when we compare the total execution time of this chart and the one from 1.10.2, we see that doubling the instruction count actually does cause a spike in execution time, at least in single-processor settings. But this disadvantage drops off precipitously as the number of processors goes up, which is what we would expect.

Follow-up: The additional instructions probably represents the synchronization overhead. I don't know of any program that is so parallel that there is literally no synchronization overhead. This is an unfortunate but necessary tradeoff, and it really highlights something important about parallelism/concurrency, which is that you need to be sure that this synchronization is not so prohibitive that it makes the program slower. That is to say, not every program should be parallel.

Problem 4:

- I don't know whether I should have converted 'u' to its int ascii value, but I interpreted my uID as **00458675**.
- The output is **91735**, which is the integer representation of my uID divided by 5.
- Since this program divides the input by 5, to output my uID, the program should receive my input multiplied by 5, or **2293375**.