(A) Design the **best performing architecture under the constraint that the area is less than or equal to 60.** Use simulations both to guide your design and verify your results. Note that simulating all possible reasonable configurations would take several hundreds of simulations, which is not realistic. You need a smart mechanism to navigate the design space.

The benchmark I choose is ***compress*** and I use the profiling simulator sim-profile first to get the statistics about the distribution of different instruction classes using command:

sim-profile -all true ./compress95.ss < ./compress95.in

Key information can be listed as follows, based on which we will design the best architecture:

```
load                 17119094  21.28
store                11648133  14.48
uncond branch         5381845   6.69
cond branch           8979175  11.16
int computation      36916920  45.90
fp computation         387057   0.48
```

Assuming we try in-order execution first, the maximum possible area is 63.2, so I design the experiment 1. Compared to the default configuration, we can infer that in-order execution will not get a higher performance.

sim-outorder -config default.cfg compress95.ss < compress95.in

	width	iALU	iMUL	fALU	fMUL	o-ord	RUU	LSQ	area	IPC
def	4	4	1	4	1	Yes	16	8	66.88	1.637
exp1	4	4	4	3	4	No	N/A	N/A	59.2	0.764
exp2	4	4	1	1	1	Yes	32	16	62.08	1.777
exp3	**4**	**4**	**1**	**1**	**1**	**Yes**	**32**	**8**	**58.08**	**1.718**
exp4	4	4	1	1	1	Yes	16	16	54.08	1.637

Based on the observation that integer multiplication, floating point computation only take up a small percentage of all the instructions, it should be safe to keep those numbers to 1.

```
add.s     D,S,T              0   0.00
add.d     D,S,T          16256   0.02
sub.s     D,S,T              0   0.00
sub.d     D,S,T              0   0.00
mul.s     D,S,T              0   0.00
mul.d     D,S,T              0   0.00
div.s     D,S,T              0   0.00
div.d     D,S,T          41383   0.05
```

After a few trials (experiments designed are also listed in the table), we figure out the potentially optimal setting satisfying the constraint. (which is highlighted in the table)

(B) Now assume that the power consumption for your architecture is proportional to the total area. For example, the power consumption for the processor with 39.04 unit area is 39.04 watt. Therefore, performance per watt for this architecture will be IPC/watt. **Design the architecture with best performance per watt value (higher is better) under the constraint that the area is less than or equal to 60.**

Considering that IPC is relatively a small value and watt/area is relatively a big value, therefore, maximizing IPC/watt is equivalent to minimizing watt/IPC or area/IPC.

width	iALU	iMUL	fALU	fMUL	o-ord	RUU	LSQ	area	IPC	a/I
4	4	1	4	1	Yes	16	8	66.88	1.637	40.86
4	4	1	1	1	Yes	32	16	62.08	1.777	34.94
4	3	1	1	1	Yes	32	16	59.28	1.721	34.45
4	2	1	1	1	Yes	32	16	56.48	1.505	37.53

We analyze the efficiency of integer ALU first. It's undoubtedly that IPC will increase when we add iALU to the architecture, but what we want to prove is that the extra iALU is EFFICIENT, which is measured by area/IPC. Instructions "addu" and "addiu" take up **11.09%** and **15.48%** of all the instructions, so more iALU *should* be useful which has also been supported by the data.
Then let's focus on the optimal value of RUU and LSQ:

width	iALU	iMUL	fALU	fMUL	o-ord	RUU	LSQ	area	IPC	a/I
4	4	1	1	1	Yes	32	16	62.08	1.777	34.94
4	4	1	1	1	Yes	32	8	58.08	1.718	33.81
4	4	1	1	1	Yes	32	4	56.08	1.445	38.81
4	4	1	1	1	Yes	16	16	54.08	1.637	33.04
4	**4**	**1**	**1**	**1**	**Yes**	**16**	**8**	**50.08**	**1.637**	**30.59**
4	4	1	1	1	Yes	16	4	48.08	1.435	33.51
4	4	1	1	1	Yes	8	16	50.08	1.353	37.01

RUU: to facilitate out-of-order execution while ensuring the architectural state of the machine (i.e., the programmer-visible registers and memory) is updated in the correct program order.
LSQ: specifically deals with memory operations — loads (reads) and stores (writes).

Apparently, RUU and LSQ cannot be too short in number while too many of these units might seem redundant (16 RUU and 8 LSQ could be an ideal setting). Finally, Let's check on in-order execution to see if it could out-perform out-of-order execution.

width	iALU	iMUL	fALU	fMUL	o-ord	RUU	LSQ	area	IPC	a/I
4	**4**	**1**	**1**	**1**	**Yes**	**16**	**8**	**50.08**	**1.637**	**30.59**
4	4	1	1	1	No	N/A	N/A	27.20	0.764	35.60

As is seen above, in-order execution is not as efficient as out-of-order execution with certain parameters in terms of area/IPC. To sum up, architecture settings which give the highest performance per watt value are shown in red above. (watt/IPC equals ***30.59*** so the best IPC/watt is its reciprocal)

(C) After you get your optimal design point in terms of performance (as in Question (A)) you need configure the size of the L1 instruction and data caches. In the default.cfg you can see the parameter '–cache:dl1' and '–cache:il1' with the same configuration value '128:64:1:1'. (You could find the meaning of each number in the SimpleScalar user-guide.) Essentially the default configuration is 64-byte block size and 128 sets in a direct-mapped cache. We would like to keep the block size constant at 64-byte and associativity equal to 1, that is, direct-mapped cache. But our objective is to

coordinate L1 instruction cache size and L1 data cache size to improve the IPC. The choices for number of cache sets are given below in the table. Your goal is to find the optimal design point (best IPC) among these choices.

Choice number	1	2	3	4	5	6	7
L1 ICache # of sets	1024	512	256	128	64	32	16
L1 DCache # of sets	16	32	64	128	256	512	1024

Task C is quite straight forward. We just need to run seven experiments using the given seven sets of cache parameters. The results are shown in the following table:

Choice number	ICache of sets	DCache of sets	IPC
1	1024	16	1.4431
2	512	32	1.6052
3	256	64	1.6766

4	**128**	**128**	**1.7180**
5	64	256	1.3770
6	32	512	1.3153
7	16	1024	0.9607

[***Introduction***]

In the realm of computer architecture, achieving optimal performance often requires a delicate balance of various design parameters. This assignment delves into three critical tasks that epitomize this balance. Task A challenges us to design a high-performing architecture, constrained by a specific area limit, emphasizing the trade-offs between power and space. Task B shifts the focus to energy efficiency, prompting a quest for the best performance per watt, a metric increasingly vital in today's energy-conscious world. Finally, Task C hones in on cache configuration, a pivotal component in determining system performance. By exploring these tasks, we will navigate the intricate landscape of design choices, understanding the implications of each decision on the overall system performance.

[***Results***]

In the pursuit of designing an optimal architecture for the "compress" benchmark, several experiments were conducted. Task A focused on balancing performance and area constraints. Among the configurations tested, Experiment 2 stood out with an IPC of 1.777, albeit at an area of 62.08. However, Experiment 3 offered a slightly more area-efficient design with an IPC of 1.718 and an area of 58.08. For Task B, the emphasis was on performance per watt. A notable configuration with an area of 50.08, an IPC of 1.637, and an area-to-IPC ratio of 30.59 was tested, highlighting the importance of integer ALU efficiency and optimal values for RUU and LSQ in maximizing IPC/watt. Lastly, Task C delved into cache configurations. The experiments revealed varying IPC values for different ICache and DCache set combinations, with the configuration of ICache = 128 and DCache = 128 yielding the highest IPC of 1.7180.

[***Conclusion***]

The comprehensive experiments conducted for the "compress" benchmark illuminate the complexities inherent in architectural design. Task A showcased the challenges of balancing performance with area constraints, emphasizing that achieving peak performance might necessitate trade-offs in design space. Task B highlighted the significance of components like the integer ALU, RUU, and LSQ in optimizing performance per watt, suggesting that certain execution strategies might offer superior efficiency over others. Meanwhile, Task C underscored the pivotal role of cache configurations in influencing overall performance. In essence, the journey to an optimal processor architecture is multifaceted, demanding a nuanced understanding of performance, area, and energy considerations. These experiments serve as a testament to the iterative nature of design and the value of empirical analysis in driving informed architectural decisions.

[***Optimal Design Point***]

Optimal design point for each task has been highlighted in **red** in the corresponding table.

By closely monitoring the distribution of different instruction classes, in our experiments, we observed specific stages or components, such as the integer ALU or the Load-Store Queue (LSQ), that were frequently under or over-utilized. To address these **bottlenecks**, our design choices were informed by a combination of reducing, augmenting, or optimizing the components in question. For instance, adjusting the number of integer ALUs or modifying the size of the RUU can alleviate pressure points in the pipeline. By iteratively refining the architecture based on these insights, we were able to enhance throughput, minimize stalls, and ultimately, craft a more streamlined and efficient pipeline.