TDTS08: Lab Report

Lab 3: Superscalar Processors

Name	PIN	Email
Alexander Yngve	930320-6651	aleyn573@student.liu.se
Pål Kastman	851212-7575	palka285@student.liu.se

Contents

1 Introduction

The purpose of this lab is to learn how Supersclar Processors work, and to try and modify a processor architecture to make it simpler, but it should still perform within 5% of the inital designs performance.

2 Method

We started out by investigating every part of the design individually, to see how they affected the performance of the design.

We then choose to simplify the parts that didn't affect the performance. We determined what parts we couldn't simplify due to that the performance would go further than 5% from the initial performance.

Now we looked at the parts of the design that we could modify, and at their traces.

3 Result

In order to establish a baseline performance the simulator was run with the default arguments and a trace was created, with the command shown in listing 1.

Listing 1 – Simulator command.

```
sim-outorder -config superscalar.cfg -ptrace trace.trc
    100000:+30 go.ss 3 8
```

The config file "superscalar.cfg" is supplied with the lab and contains all default settings. The most interesting of these are shown in table 1.

Table 1 – Default settings.

Setting	Value
res:ialu	4
res:imult	4
res:fpalu	2
res:fpmult	2
ruu:size	32
commit:width	8
issue:width	8
decode:width	16
fetch:speed	1

Running the simulator with the default settings gave us a simulation time of **50868222** cycles, with the 5% requirement from section 1 this results in a maximum of **53411633** simulation cycles.

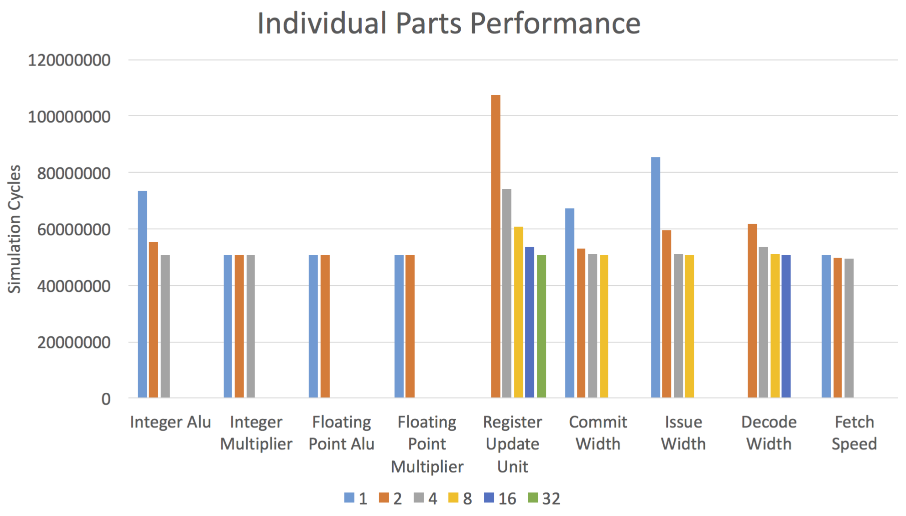

Figure 1 – How the individual parts perform.

An overview of the different setups can be seen in figure 1. It shows the default settings for all parameters except the current one which we adjusted.

3.1 Integer components

In table 2 we can see that reducing the number of integer multipliers to only one, no noticeable change in performance could be seen. Therefore we decided to reduce the amount to only one.

Table 2 – Integer settings.

Setting	Value	Simulation cycles
res:ialu	2	55360220
res:ialu	1	73570279
res:imult	2	50868222
res:imult	1	50868609

When observing the integer alu:s, we instead see in table 2, that the performance will reduce drastically when reducing the amount from 4, thus we decided not to change this in the design, as the performance will drop beneath 5% of the standard design performance.

3.2 Floating Point components

In table 3 we can see that by changing the floating point alu & multiplier, the system didn't perform any worse, thus we decided to simplify these as much as possible.

Table 3 – Floating point settings.

Setting	Value	Simulation cycles
res:fpalu	1	50868222
res:fpmult	1	50868222

3.3 Control components

The speed of the system was already at the lowest possible, which means by changing this value will increase the complexity of the design, and therefore we decided not to.

When changing the Register Update Unit wee can see in table 4, that when reducing only one step to 16 instead of 32 we get such a bad result that we would no longer be within 5% of the standard designs performance, therefore this is left unchanged.

Table 4 – Control components settings.

Setting	Value	Simulation cycles
ruu:size	16	53807041
ruu:size	8	60781481
ruu:size	4	74098784
ruu:size	2	107565825
commit:width	4	51076751
commit:width	2	53040135
commit:width	1	67227355
issue:width	4	51187526
issue:width	2	59378979
issue:width	1	85571238
decode:width	8	51165249
decode:width	4	53609877
decode:width	2	61683477
fetch:speed	4	49804290
fetch:speed	2	50868222

3.4 Simplified Design

Using the values in table 5 the simulation cycles increased to **51696860** which is within the +5% performance requirements.

Table 5 – Simplified settings.

Setting	**Default value**	**Modified value**
res:ialu	4	4
res:imult	4	1
res:fpalu	2	1
res:fpmult	2	1
ruu:size	32	32
commit:width	8	4
issue:width	8	4
decode:width	16	8
fetch:speed	1	1

3.5 Traces

In our modified design, we can see that the later parts of the pipeline (EX & WB) are utilized more effeciently than in the original, where IF seems to fetch more instructions than what it can execute. In our design we fetch them at a slower pace, hence having a more equal utilization of all stages.

4 Discussion

For floating point alu & multiplier we believe that these parts doesn't affect the performance due to that go.ss only uses integer values.

Using traces in this way seems hard because it's hard to compare since you might pick a place in the code where there are no differences even though the architecture is modified.