| 3. | Fachgebiet Architektur eingebetteter Systeme<br>**Advanced Computer Architectures** | SoSe2017 |
|---|---|---|

Assignment start: 06.06.2017 Submission deadline: 26.06.2017

## Assignment 3 - SimpleScalar 25 Points

In the third assignment an advanced processor simulator is used, SimpleScalar. You will investigate the IPC improvements resulting from key advances in processor microarchitecture. The SimpleScalar simulator can simulate (single-core) processors with a variety of configurations. The type of branch predictor, number of issue slots, cache size, cache latency, and other architectural parameters can be freely configured. Documentation about the SimpleScalar simulator is available at `http://www.simplescalar.com/docs/users_guide_v2.pdf`. In this assignment we only use the *sim-outorder* version of the simulator. The simulator only runs on Linux (x86, 64-bit).

Five precompiled benchmark programs for the SimpleScalar simulator are available. The simulator and the benchmarks can be found in */afs/tu-berlin.de/units/Fak_IV/aes/aca/SimpleScalar* and archived on the ISIS page.

- *Fibonacci* - calculates the 20th instance of the Fibonacci sequence
- *Matmul* - matrix multiplication of size 50x50
- *Pi* - estimates the pi number
- *Whetstone* - a double precision benchmark
- *Memcopy* - a memory benchmark

In the first four exercises you have to produce bar graphs of the effect of improving a single feature. Put the benchmarks on the horizontal axis, and the IPC on the vertical axis.

1. (5 Points) Investigate the effect of the branch predicting modes offered by SimpleScalar. You can exclude the *taken* predictor option from the analysis. Due to a simulator bug it produces the same results as *nottaken*. Set the other parameters to configure the simulated processor as a single-issue, in-order processor, with one integer and floating point alu and mult unit. Manipulate the -fetch:ifqsize, -decode:width, -issue:width, -commit:width, -issue:inorder, -res:ialu, -res:imult, -res:fpalu and -res:fpmult options to achieve this. The number of memports should stay at the default value of 2.

   Perform the experiment for each benchmark and put the IPC results in a bar graph. For each benchmark, the result of the different branch predictors must be clustered in the following order: nottaken, bimod, 2lev, comb, perfect.

2. (5 Points) Investigate the effect of increasing the fetch, decode, issue, commit width and the number of resources (“-res:” options) for the options 1, 2, 4 and 8. Also investigate a fifth option with a processor width of 8, but using default amount of resources (4 ialu, 1 imult, 4 fpalu, 1 imult). The number of memports should stay at the default value of 2. Set the simulator to use the 2lev branch predictor.

   Perform the experiment for both in-order and out-of-order execution. This should result in two bar graphs with each five clustered bars for each benchmark with increasing processor width. Put the fifth experiment at the end of each cluster.

3. (5 Points) Investigate the effect of increasing the size of the register update unit (ruu) and load store queue (lsq). Set the processor width to 8 and use out-of-order execution (manipulate the

-fetch:ifqsize, -decode:width, -issue:width, -commit:width and -issue:inorder options). Use the combinations (ruu, lsq) of (16, 8), (32, 8), (32, 16), (64, 16) and (64, 32).The number of memports should stay at the default value of 2. The simulator should use the 2lev branch predictor.

Investigate this for the default amount of execution resources (4/1/4/1) and for a doubled amount (8/2/8/2), resulting in two bar graphs.

4. (5 Points) Write a small report ( 350 words) on your findings in the previous assignments. Explain what limits the IPC for the benchmarks initially and how branch prediction, processor width, and increasing the instruction window and load store queue overcome this. Also comment for benchmarks with the lowest improvements why the IPC is is hitting a wall.

In the last exercise you have to propose a cache architecture to mitigate the performance penalty of a higher memory latency. The default memory latency of SimpleScalar is 18 cycles for the initial memory chunk, and 2 cycles for each subsequent memory chunk. Each chunk has the same width as the memory bus width, i.e., each cache line transfer is transferred in multiple chunks. When we increase the initial latency to 200 cycles the performance is significantly lower. An initial latency of 200 cycles, however, is a more realistic number in current state-of-the-art processors. In this assignment we only use the *memcopy* benchmark, which has both large contiguous and more fine-grained memory access patterns. See the source file `memcpy.c` for more details.

5. (5 Points) First, simulate the IPC and calculate the AMAT before the increased memory latency and after the increased memory latency. For the AMAT calculation you can simplify by only considering the L2 cache for this, i.e, the AMAT in case you have an L1 miss.

   Second, propose a cache configuration that masks the performance penalty of the increased memory latency. You can increase the number of sets, set size (block/line size), associativity, of the unified L2 cache only. The default latency of the L2 cache is 6 cycles. For each doubling of the number of sets and/or associativity add 1 extra cycle of latency. Report the chosen cache configuration and resulting IPC and AMAT. Provide an explanation why you choose this cache configuration and explain how the additional memory latency is mitigated.

   For this assignment SimpleScalar has to be configured as follows: comb branch predictor, four-issue, out-of-order, default amount of resources, and a (ruu,lsq) of (32,16).

For submission, prepare one document with the results of all five exercises as the deliverable for the SimpleScalar assignment. Each exercise is worth 5 points for a total of 25 points.