Exercise Sheet 10

1 Register Allocation  


All graphs to be colored do not contain self-loops.

a) Consider the following code snippet, where the symbols $s_i$ denote symbolic registers:

\[
\begin{align*}
    s_1 &= 1 \\
    s_2 &= 4 \\
    s_3 &= s_1 + s_2 \\
    s_4 &= s_3 - s_1 \\
    s_5 &= s_2/7 - s_4 \\
    s_6 &= s_3 - s_2 \\
    \text{return } s_6 - s_5
\end{align*}
\]

Given the actual registers $r_i$ with $i \in \{1, 2, 3\}$:

- Provide the register interference graph.
- Assign actual registers to the symbolic registers by coloring the register interference graph using Chaitin’s local-colorability criterion and give the resulting code.

b) Give a register interference graph and a number $k$ of actual registers, such that Chaitin’s heuristic fails to find a non-interfering mapping from symbolic to actual registers although there exists such a mapping.
2 Code Scheduling

The Infineon TriCore™ processor features a dual-dispatch, out-of-order execution, in-order completion pipeline. This means that the core may dispatch two instructions at once. Independent of which instruction finishes first, the results of the execution of the instructions come into effect in the order the instructions appear in the program code.

Among others, the instruction set comprises the following instructions:  

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Semantics</th>
<th>Execution Unit</th>
<th>Execution Time (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add di,dj,dk</td>
<td>(d_i = d_j + d_k)</td>
<td>ALU</td>
<td>3</td>
</tr>
<tr>
<td>mul di,dj,dk</td>
<td>(d_i = d_j \times d_k)</td>
<td>ALU</td>
<td>5</td>
</tr>
<tr>
<td>div di,dj,dk</td>
<td>(d_i = d_j / d_k)</td>
<td>ALU</td>
<td>7</td>
</tr>
<tr>
<td>add.a ai,a[j],c</td>
<td>(a_i = a_j + c)</td>
<td>LSU</td>
<td>3</td>
</tr>
<tr>
<td>ld di,[a[j]]c</td>
<td>(d_i = \text{mem}[a_j + c])</td>
<td>LSU</td>
<td>4</td>
</tr>
<tr>
<td>st [a[j]]c,di</td>
<td>(\text{mem}[a_j + c] = d_i)</td>
<td>LSU</td>
<td>4</td>
</tr>
</tbody>
</table>

Each instruction reads its operands in the first cycle and writes to its destination in the last cycle. The remaining cycles are required to compute the result, access the data bus etc. In case an instruction reads from a registers, while another one writes to that register, the read will return the old value.

Given the following sequence of instructions:

1. add d1,d1,d2
2. mul d3,d1,d1
3. ld d2,[a2]0
4. adda a3,a2,4
5. div d1,d2,d3
6. st [a3]0,d2
7. ld d3,[a2]8

a) Provide the data dependency graph (DDG) for the given instruction sequence. Note that there can only be a dependency for two instructions \(a\) and \(b\), with \(a < b\).

b) Simulate the execution behavior of the TriCore™, by giving a schedule showing when (and how long) the instructions are executed by the corresponding execution unit\(^2\). How long does it take to execute the given piece of code (in terms of processor cycles)?

To clarify the dispatch behavior of the pipeline:

1) The processor only dispatches two instructions in parallel, if the first one is an ALU-instruction and the second one is executed by the LSU. Otherwise, the processor may only dispatch a single instruction.

\(^1\)\(a_n\) denotes an address register, \(d_n\) represents a data register. Address and data registers are completely independent from each other.

\(^2\)You may assume that the instructions to execute are directly available, i.e. there are no additional cycles required to fetch the instructions.
2) In case any of the two execution units is occupied, the processor *cannot* dispatch and has to wait until the instruction currently being executed is finished.

3) In case the processor has dispatched two instructions in parallel and the second one finishes earlier than the first one, the processor waits until the first instruction has completed execution. Both instructions then finish in the same cycle.

c) Exploit the fact that an ALU-instruction followed by an LSU-instruction causes the pipeline to dispatch the two instructions in parallel. Reorder the instructions such that the overall execution time is minimized. Mind the dependencies between the instructions, as determined in a).

d) Redo the simulation according to the rescheduling you have performed in c). How long does it take to execute the reordered instruction sequence?

e) Reconsider the original instruction sequence. How long does it take to execute the code snippet if the second dispatch constraint is omitted?