lea.schoenberger [☺] tu-dortmund.de
christian.erdmann [☺] tu-dortmund.de
nils.hoelscher [☺] tu-dortmund.de
jan.pham [☺] tu-dortmund.de

Exercises for
Embedded Systems
Wintersemester 19/20

Exercise Sheet 10 (Practice)

(10 Points)

Please note: Submitting written solutions to this exercise sheet is not necessary. Discussion: 06.-10.01.2020.

Open a terminal and execute the command **ssh -X youraccount@ls12pc5.cs.tu-dortmund.de**, whereat you need to substitute *youraccount* with your user name. Download the archive `wcet2` from the course website using **wget**. If you are not familiar with the command line, please ask Google for help.

Preparation

Whenever you start a new session, you have to re-register some components. For this reason, enter the following command into a terminal window:

- **cd wcet**
- **./env.sh**
- **m+ tricore-gcc**
- **m+ ait**

1 Step 3: Scratchpad Allocation and Function Outlining (5 Points)

Execute the command `cd ~/wcet/step3` to open the directory used for this assignment. In the file `test.c`, some preparations have been made to transform the inner loops of the functions `Initialize` and `Sum` into separate functions via function outlining. In this manner, it is possible to swap out the inner functions to the scratchpad memory (SPM) individually.

Perform the function outlining of the inner loops of the aforementioned both functions.

Thereon, compile the program by executing the command **tricore-gcc -o test.elf -g -T ../tc1796.lds test.c** and load it into the `a3tricore` analyzer as explained in the previous exercise sheet. If you start an aiT analysis, some errors will be reported.

Correct the errors in the file `a.ais`, so that the analysis can be performed properly.

How did the WCET change compared to the previous version (complete functions in the SPM)?

Hint: For the previous version, a WCET of 85117 cycles was computed.

2 Step 4: Integer Linear Programming (5 Points)

aiT performs the WCET analysis by analyzing each basic block of a program separately, constructing an ILP (integer linear program) based on these results and solving it.

In the directory `step4`, more precisely, in the file `example.lp`, an exemplary formulation of an ILP can be found. Open this file with a text editor of your choice and solve it with an ILP solver via `lp_solve example.lp`.

Another example is given in the file `example2.lp`, where the control flow graph of a simple program is modeled. Solve this problem via `lp_solve example2.lp`.

Explain the result.

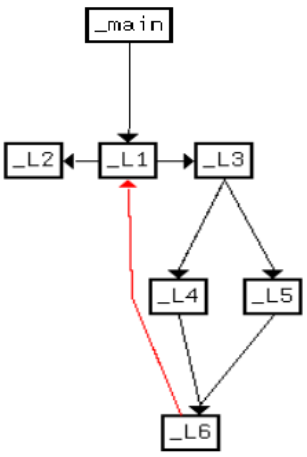
How can we model a certain basic block with execution time of more than one cycle in the ILP?

Here, the following program containing a loop is given:

```
int main()
{
  int i, j = 0;

  _Pragma( "loopbound min
                100 max 100" );
  for ( i = 0; i < 100; i++ ) {
    if ( i < 50 )
      j += i;
    else
      j += ( i * 13 ) % 42;
  }

  return j;
}
```



Block	Cycles
main	21
L1	27
L2	20
L3	2
L4	2
L5	20
L6	13

Abbildung 1: Example program with loop.

The basic structure of this program is already modeled in the file `ipet.lp`.

Add the number of cycles for each basic block.

If you try to solve the ILP via `lp_solve ipet.lp`, the solver will abort with the error message *This problem is unbounded*.

Why is it impossible for the ILP solver to find a solution?

Add the missing constraint for edge h.

If you try again to solve the corrected file with lp_solve, you should obtain a result.
Why does the solver compute a solution in which L4 is never executed?