Compiler Design Memory Protocol

- Wintersemester 2022/23
- Exam in presence
- 60min, no tools allowed
- 50 points in total

1.) Syntax and Semantics [11p]

SAXPY Grammar for Basic Linear Algebra Subprogram (BLAS) given
Code example:

```
for i in [0..n]:
        y[i] = a * x[i] + y[i];
```

Grammar:

```
G -> Ss
Ss -> S Ss | epsilon
S -> A; | L
L -> for id in [E..E]: B
B -> {S} | S
A -> V = E
E -> E + T
T -> T * F
F -> V | num
V -> id[E] | id
```

- a) Extend the Grammar for the operator `+=`. It should accept both `y[i] = a * x[i] + y[i];` and `y[i] += a * x[i]`
- b) Does line 2 of the code example need to be closed by `;` ? Justify your answer.
- c) Eliminate the left recursion for the Non-Terminals `E` and `T`
- d) What parser do and don't work with the initial and your final grammar ?

2.) Intermediate Representation [12p]

Code example given

```
n = ...;
i = 0;
while (i < n){
        y[i] = a * x[i] + y[i];
        i = i + 1;
}
return y;
```

- a) Construct the Control Flow Graph
- b) Construct the Dominator Tree
- c) Calculate the Dominance Frontier for line 3, 4 and 5
- d) Rewrite the Control Flow Graph into Single Assingment Form. Use the ϕ operator only for the variable `i`

3.) Loop Unrolling [10p]

Code Example

```
for (int i=0; i<n; i++){
        y[i] = a * x[i] + y[i];
}
```

- a) Do a loop unrolling of the code example above by the factor 4
- b) Explain briefly what is necessary if `n` is not evenly dividable by 4

4.) Instruction Selection, Instruction Scheduling [12p]

Assembly Code of modified NVIDIA PTX is given

```
ld.u32 r1, [@i]    // load value of i into r1
ld.f32 r2, [@a]    // load value of a into r2
ld.u64 r3, @x      // load address of x into r3
ld.u64 r4, @y
shl.b64 r5, r1, 2  // r5 = i * 4
add.u64 r6, r3, r5
add.u64 r2, r4, r5
ld.f32 r8, [r6]
ld.f32 r9, [r7]
mul.f32 r10, r2, r8  // r10 = a * r8
```

```
add.f32 r11, r10, r9  // r11 = r10 + r9
st.f32 [r7], r11
```

- a) Construct the Precedense Graph
- b) There hardware, which allows performing an `add` and `mul` simultaneously. It can be called with the operator `fma.f32 r, x, y, z` which is equivilant to `r = x * y + z`. Explain why a compiler whould choose this operand over a single `add` and `mul`
- c) Show that the code above can be rewritten with `fma.f32`
  (Comment: I ask during the exam that the `fma.f32` is only defined for Floating Points)