**Universität des Saarlandes**
**FR 6.2 - Informatik**
Prof. Dr. W.J. Paul
Dipl.-Ing. Christoph Baumann

# Exercise Sheet 7
# Computer Architecture II

(Due: Dec 10th, 2013)

**Exercise 1: (Significand Normalization Shifter) (10 Points)**

Figure 1 depicts the implementation of the significand normalization shifter in the FPU rounding unit. The significand $f' = f_r/2 > 0$ is put in a cyclical left shifter with shift amount $\langle sh[5:0]\rangle$ where $\sigma = [sh[12:0]] \leq 56$ is the desired left shift distance. Additionally masks $v, w \in \mathbb{B}^{64}$ are computed and anded to the shifter output $fs$ to obtain the normalized significand. In the analysis of the MASK circuit we already derived that $v = \overline{u}$ and $\forall i \in [0:63].\ w_i = u_i \wedge sh[12]$ with:

$$u[0:63] = \begin{cases} 0^{64-\sigma}1^{\sigma} & : \quad 0 \leq \sigma \\ 1^{|\sigma|}0^{64-|\sigma|} & : \quad -63 \leq \sigma \leq -1 \\ 1^{64} & : \quad \sigma < -63 \end{cases}$$

Let $f_n = \langle fn[0].fn[1:127]\rangle$. To be proven:

$$f_n =_p 2^{\sigma} \cdot f'$$

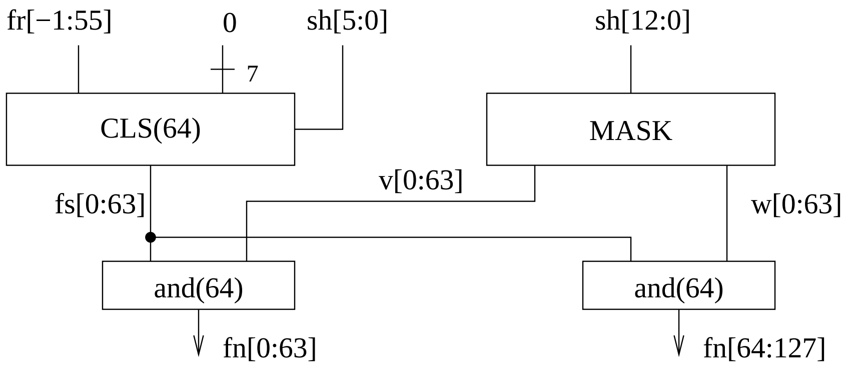

Figure 1: Significand Normalization Shifter

**Exercise 2: (Improved Exponent Normalization Shifter) (6+2 Points)**

Figure 2 shows the exponent normalization shifter from the FPU rounding unit. One could speed up this circuit by getting rid of the 3/2 Adder. This is done by combining two of the inputs.

a) Show how this can be achieved and that your construction works.

b) How does this modification affect the overall cost and delay of the circuit $ExpNorm$?

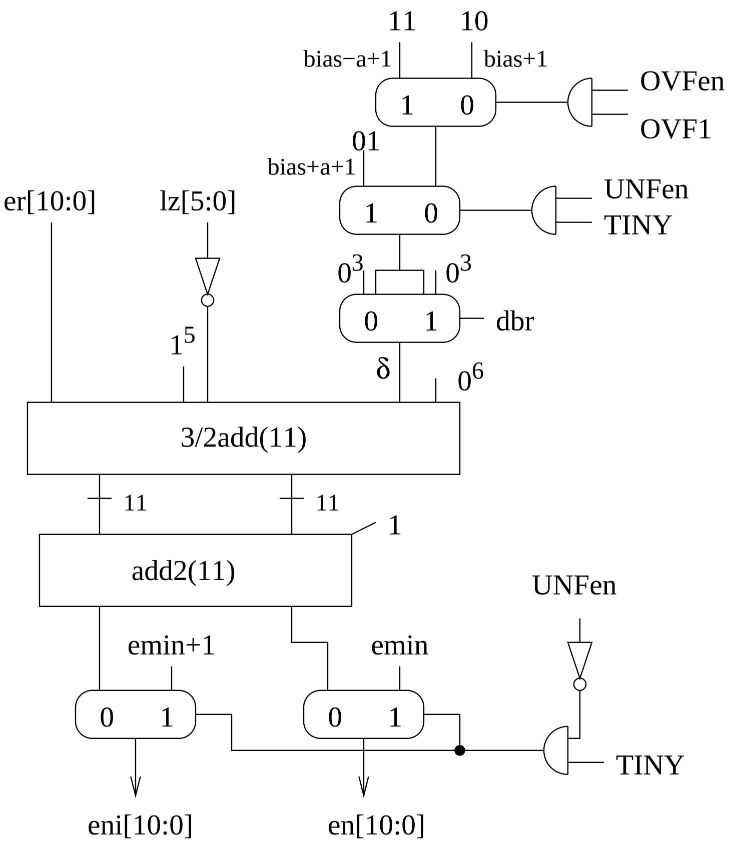

Figure 2: Exponent Normalization Shifter

**Exercise 3: (Fast and Cheap Incrementers)** **(3+3+3+8+5 Points)**

In the significand rounder we need a large (53-bit) incrementer. An $n$-bit incrementer is a circuit with inputs $a \in \mathbb{B}^n$, $inc \in \mathbb{B}$ and outputs $s \in \mathbb{B}^n$, $c \in \mathbb{B}$ such that the property

$$\langle c\ s[n-1:0]\rangle = \langle a[n-1:0]\rangle + inc$$

holds. A fast incrementer with logarithmic delay and linear-logarithmic cost can be implemented in a way similar to the Conditional Sum Adder.

a) Construct an $n$-bit Conditional Sum Incrementer for $n = 2^k$!

b) Prove the correctness of your construction!

c) How would you optimize your construction for $n \neq 2^k$?

d) Give the exact cost and delay of your Incrementer for $n = 2^k$ in non-recursive formulas! Prove their correctness!

e) Consider a Carry Lookahead implementation of an incrementer! How does it compare to the Conditional Sum Incrementer concerning cost and delay?