Instruction Set Card SwissQM

René Müller, Michael Duller
{muellren, dullerm}@inf.ethz.ch
Department of Computer Science, ETH Zurich

	Instruction	Description
Stack	`dup`	$\alpha, x \Rightarrow \alpha, x, x$
	`dup_x1`	$\alpha, y, x \Rightarrow \alpha, x, y, x$
	`dup_x2`	$\alpha, z, y, x \Rightarrow \alpha, x, z, y, x$
	`dup2`	$\alpha, y, x \Rightarrow \alpha, y, x, y, x$
	`dup2_x1`	$\alpha, z, y, x \Rightarrow \alpha, y, x, z, y, x$
	`dup2_x2`	$\alpha, w, z, y, x \Rightarrow \alpha, y, x, w, z, y, x$
	`pop`	$\alpha, x \Rightarrow \alpha$
	`pop2`	$\alpha, y, x \Rightarrow \alpha$
	`swap`	$\alpha, y, x \Rightarrow \alpha, x, y$
	`iconst_0`	$\alpha \Rightarrow \alpha, 0$
	`iconst_1`	$\alpha \Rightarrow \alpha, 1$
	`iconst_2`	$\alpha \Rightarrow \alpha, 2$
	`iconst_4`	$\alpha \Rightarrow \alpha, 4$
	`iconst_m1`	$\alpha \Rightarrow \alpha, -1$
	`ipushb <int8>`	$\alpha \Rightarrow \alpha, \text{sign\_ext}(i)$
	`ipushw <int16>`	$\alpha \Rightarrow \alpha, i$
Arithmetic and Logic	`iadd`	$\alpha, y, x \Rightarrow \alpha, y + x$
	`isub`	$\alpha, y, x \Rightarrow \alpha, y - x$
	`imul`	$\alpha, y, x \Rightarrow \alpha, y * x$
	`idiv`	$\alpha, y, x \Rightarrow \alpha, \lfloor y/x \rfloor$
	`irem`	$\alpha, y, x \Rightarrow \alpha, y \bmod x$
	`ineg`	$\alpha, x \Rightarrow \alpha, -x$
	`iinc`	$\alpha, x \Rightarrow \alpha, x + 1$
	`idec`	$\alpha, x \Rightarrow \alpha, x - 1$
	`iand`	$\alpha, y, x \Rightarrow \alpha, y \& x$
	`ior`	$\alpha, y, x \Rightarrow \alpha, y \mid x$
	`inot`	$\alpha, x \Rightarrow \alpha, {}^{\sim}x$
Control	`if_icmpeq <int8>`	$\alpha, y, x \Rightarrow \alpha$, jump if $y = x$
	`if_icmpneq <int8>`	$\alpha, y, x \Rightarrow \alpha$, jump if $y \neq x$
	`if_icmplt <int8>`	$\alpha, y, x \Rightarrow \alpha$, jump if $y < x$
	`if_icmple <int8>`	$\alpha, y, x \Rightarrow \alpha$, jump if $y \leq x$
	`if_icmpgt <int8>`	$\alpha, y, x \Rightarrow \alpha$, jump if $y > x$
	`if_icmpge <int8>`	$\alpha, y, x \Rightarrow \alpha$, jump if $y \geq x$
	`ifeq <int8>`	$\alpha, x \Rightarrow \alpha$, jump if $x = 0$
	`ifneq <int8>`	$\alpha, x \Rightarrow \alpha$, jump if $x \neq 0$
	`iflt <int8>`	$\alpha, x \Rightarrow \alpha$, jump if $x < 0$
	`ifle <int8>`	$\alpha, x \Rightarrow \alpha$, jump if $x \leq 0$
	`ifgt <int8>`	$\alpha, x \Rightarrow \alpha$, jump if $x > 0$
	`ifge <int8>`	$\alpha, x \Rightarrow \alpha$, jump if $x \geq 0$
	`goto <int8>`	$\alpha \Rightarrow \alpha$, jump always
Buffers	`iload <int8>`	$\alpha \Rightarrow \alpha$, `buf[i]`
	`iload`	$\alpha, i \Rightarrow \alpha$, `buf[i]`
	`istore <int8>`	$\alpha, x \Rightarrow \alpha$ and `buf[i]` $= x$
	`istore`	$\alpha, x, i \Rightarrow \alpha$ and `buf[i]` $= x$
	`iload_sy <int8>`	$\alpha \Rightarrow \alpha$, `syn[i]`
	`iload_sy`	$\alpha, i \Rightarrow \alpha$, `syn[i]`
	`istore_sy <int8>`	$\alpha, x \Rightarrow \alpha$ and `syn[i]` $= x$
	`istore_sy`	$\alpha, x, i \Rightarrow \alpha$ and `syn[i]` $= x$
	`clear_sy`	clear synopsis
	`send_tb`	send transmission buffer
	`send_sy`	send synopsis
Sensors	`get_nodeid`	$\alpha \Rightarrow \alpha, nodeid$
	`get_parent`	$\alpha \Rightarrow \alpha, parent$
	`get_light`	$\alpha \Rightarrow \alpha, light$
	`get_lightpar`	$\alpha \Rightarrow \alpha, lightpar$
	`get_temp`	$\alpha \Rightarrow \alpha, temp$
	`get_humid`	$\alpha \Rightarrow \alpha, humid$
	`get_voltage`	$\alpha \Rightarrow \alpha, batteryvoltage$
Aggregation	`merge`	$\alpha, \text{agg}_m, \ldots, \text{agg}_1, m, n \Rightarrow \alpha$

code	agg
1	`COUNT`
2	`MAX`
3	`MIN`
4	`SUM`
5	`AVG`
6	`VARIANCE`

n: number of groups
m: number of aggregates
agg_i: code of aggregate i