SB-PRAM

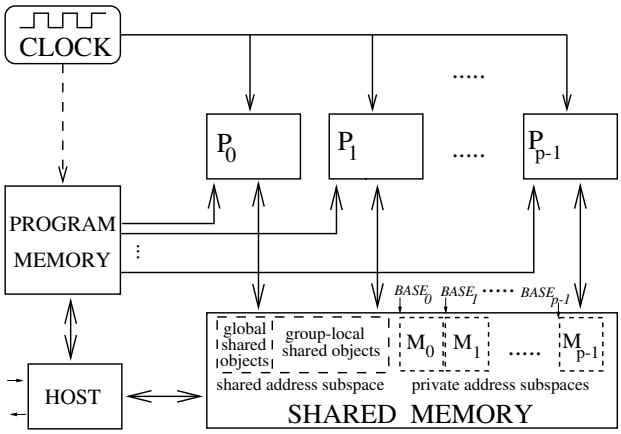
Instruction Set Simulator System Software

Quick Reference Card

for the Fork95 programmer

Christoph W. Keßler

Universität Trier, FB IV – Informatik
D-54286 Trier, Germany
kessler@psi.uni-trier.de



Current Hardware Status see the WWW homepage of the SB-PRAM project at Saarbrücken University, `http://www-wjp.cs.uni-sb.de/sbpram`

SB-PRAM at Assembler Programming Level

p=4096 RISC-style processors (32bit)

Physically Shared Memory 2 GByte

Private Address Subspaces embedded into shared memory, private addresses (bit 31 set) relative to $BASE$ pointer

Unit Memory Access Time

Common Clock Signal synchronous on assembler level

Concurrent Write PRIORITY CRCW

Multiprefix Operations perform in 2 clock cycles

Processor Features

32 registers (32 bit)

Register R_0 denotes either the status register (for store/push/pop instructions) or the constant 0. R_1 is the program counter, R_2 the system stack pointer.

Each other register may be used as a stack pointer.

$BASE$ register automatically added to private addresses

A global COUNTER counts clock cycles; lsb of COUNTER is the MODULO bit of the status register, used to separate read/write access to same memory location.

Single precision IEEE 754 Floatingpoint arithmetic.

For more details see [6].

Processor Instruction Set

Notation: I_{19} denotes a 19-bit constant. S may be either a register R or a 13 bit constant I_{13}. $M(A)$ denotes a (shared) memory access at address A. For the delayed instructions (`ldg`, `popg`, `popng`, `popig`, `mpadd`, `mpmax`, `mpand`, `mpor`) the target register will be written at the end of the *following* clock cycle, for all other instructions at the end of the *same* clock cycle. For more details, see [8].

COMPUTE Instructions, INTEGER

`add` R_x, S, R_d	$R_d \leftarrow R_x + S$
`adc` R_x, S, R_d	$R_d \leftarrow R_x + S$+carry
`sub` R_x, S, R_d	$R_d \leftarrow S - R_x$
`sbc` R_x, S, R_d	$R_d \leftarrow S - R_x$−carry
`mul` R_x, S, R_d	$R_d \leftarrow R_x \cdot S$, bits 31...0
`mlhs` R_x, S, R_d	$R_d \leftarrow R_x \cdot S$, bits 63...32, signed
`mlhu` R_x, S, R_d	$R_d \leftarrow R_x \cdot S$, bits 63...32, unsigned
`rm` S, R_d	$R_d \leftarrow \lfloor \log(S) \rfloor$ (unsigned S) (-1 if $S = 0$)
`gethi` I_{19}, R_d	bits 13:32 of $R_d \leftarrow I_{19}$&0xfff80000
`getlo` I_{19}, R_d	bits 0:18 of $R_d \leftarrow I_{19}$, sign expanded
`and` R_x, S, R_d	$R_d \leftarrow R_x \& S$
`or` R_x, S, R_d	$R_d \leftarrow R_x \mid S$
`nand` R_x, S, R_d	$R_d \leftarrow \sim (R_x \& S)$
`xor` R_x, S, R_d	$R_d \leftarrow R_x \hat{} S$
`lsl` R_x, S, R_d	$R_d \leftarrow R_x \ll (S\%32)$, 0 expanded
`lsr` R_x, S, R_d	$R_d \leftarrow R_x \gg (S\%32)$, 0 expanded
`asl` R_x, S, R_d	$R_d \leftarrow R_x \ll (S\%32)$, compute overflow
`asr` R_x, S, R_d	$R_d \leftarrow R_x \ll (S\%32)$, sign expanded
`rol` R_x, S, R_d	$R_d \leftarrow R_x$ rotated left by $(S\%32)$
`ror` R_x, S, R_d	$R_d \leftarrow R_x$ rotated right by $(S\%32)$
`rocl` R_x, S, R_d	as `rol`, including carry bit
`rocr` R_x, S, R_d	as `ror`, including carry bit

COMPUTE Instructions, FLOATINGPOINT

`fadd` R_x, R_y, R_d	$R_d \leftarrow R_x + R_y$
`fada` R_x, R_y, R_d	$R_d \leftarrow \lvert R_x + R_y \rvert$
`fsub` R_x, R_y, R_d	$R_d \leftarrow R_y - R_x$
`fsba` R_x, R_y, R_d	$R_d \leftarrow \lvert R_y - R_x \rvert$

`fmul` R_x, R_y, R_d	$R_d \leftarrow R_y \cdot R_x$
`fmla` R_x, R_y, R_d	$R_d \leftarrow \lvert R_y \cdot R_x \rvert$
`ftoi` R_x, R_d	$R_d \leftarrow (\texttt{int}) R_x$
`itof` S, R_d	$R_d \leftarrow (\texttt{float}) S$

LOAD / STORE Instructions

`ldg` R_x, S, R_d	$R_d \leftarrow M(R_x + S)$
`popg` R_x, R_d	$R_d \leftarrow M(R_x - 1);\ R_x - -$
`popng` R_x, I_{13}, R_d	$R_d \leftarrow M(R_x + I_{13});\ R_x += I_{13}$
`popig` R_x, I_{13}, R_d	$R_d \leftarrow M(R_x + I_{13});\ R_x += I_{13} + 1$
`stg` R_s, R_x, I_{13}	$M(R_x + I_{13}) \leftarrow R_s$
`stgc` R_x	$M(R_x) \leftarrow 0$
`pshg` R_s, R_x	$M(R_x + I_{13}) \leftarrow R_s;\ R_x ++$
`pshg` R_s, R_x, I_{13}	$M(R_x + I_{13}) \leftarrow R_s;\ R_x += I_{13}$
`pshig` R_s, R_x, I_{13}	$M(R_x + I_{13}) \leftarrow R_s;\ R_x += I_{13} + 1$
`pshgc` R_x	$M(R_x) \leftarrow 0;\ R_x ++$

MULTIPREFIX Instructions (integer only)

R_x^q denotes register R_x of processor q. Q denotes the set of all processors that perform the same multiprefix operation as I do, to the same memory location in the same clock cycle. $ denotes my physical processor ID. & denotes bitwise AND, | denotes bitwise OR.

`mpadd` R_x, S, R_d	$M(R_x + S) \leftarrow M(R_x + S)_{old} + \sum_{q \in Q} R_d^q;$ $R_d \leftarrow M(R_x + S)_{old} + \sum_{q \in Q, q<\$} R_d^q$
`mpmax` R_x, S, R_d	$M(R_x + S) \leftarrow \max\{M(R_x + S)_{old}, \max_{q \in Q} R_d^q\};$ $R_d \leftarrow \max\{M(R_x + S)_{old}, \max_{q \in Q, q<\$} R_d^q\}$
`mpand` R_x, S, R_d	$M(R_x + S) \leftarrow M(R_x + S)_{old}\ \&\ \&_{q \in Q} R_d^q;$ $R_d \leftarrow M(R_x + S)_{old}\ \&\ \&_{q \in Q, q<\$} R_d^q$
`mpor` R_x, S, R_d	$M(R_x + S) \leftarrow M(R_x + S)_{old}\ \mid\ \mid_{q \in Q} R_d^q;$ $R_d \leftarrow M(R_x + S)_{old}\ \mid\ \mid_{q \in Q, q<\$} R_d^q$
`syncadd` R_s, R_x, I_{13}	$M(R_x + I_{13}) \leftarrow M(R_x + I_{13})_{old} + \sum_{q \in Q} R_s^q;$
`syncmax` R_s, R_x, I_{13}	$M(R_x + I_{13}) \leftarrow \max\{M(R_x + I_{13})_{old} \max_{q \in Q} R_s^q\};$
`syncand` R_s, R_x, I_{13}	$M(R_x + I_{13}) \leftarrow M(R_x + I_{13})_{old}\ \&\ \&_{q \in Q} R_s^q;$
`syncor` R_s, R_x, I_{13}	$M(R_x + I_{13}) \leftarrow M(R_x + I_{13})_{old}\ \mid\ \mid_{q \in Q} R_d^q;$

JUMP Instructions

cc denotes an integer condition code, fcc a floatingpoint condition code, based on status register flags.

`b`cc I_{19}	if (cc) $PC \leftarrow PC + I_{19}$
`fb`fcc I_{19}	if (fcc) $PC \leftarrow PC + I_{19}$
`j`cc I_{19}	if (cc) $PC \leftarrow I_{19}$
R_x, S	if (cc) $PC \leftarrow R_x + S$
`fj`fcc I_{19}	if (fcc) $PC \leftarrow I_{19}$
R_x, S	if (fcc) $PC \leftarrow R_x + S$

`bsrg` R_s, I_{19}	$M(R_s + +) \leftarrow + + PC;\ PC \leftarrow PC + I_{19}$
`jsrg` R_s, R_x, I_{13}	$M(R_s + +) \leftarrow + + PC;\ PC \leftarrow R_x + I_{13}$
`jexg` R_s, I_{19}	$M(R_s + +) \leftarrow PC;\ PC \leftarrow I_{19}$
`sysc`	call system routine R_5, par. in $R_7, ..., R_{10}$

Condition Codes

Integer Condition Codes		Floatingpoint Condition Codes	
`ra`	ALWAYS	`mi`	N (NEGATIVE)
`mi`	N (NEGATIVE)	`eq`	Z (ZERO)
`eq`	Z (ZERO)	`inf`	INFINITY
`vs`	V (OVERFLOW)	`nan`	NaN
`cs`	C (CARRY)	`iex`	IEX
`ls`	$C + Z$	`iop`	IOP (ILLEGAL OPER.)
`le`	$Z + N * \overline{V} + \bar{N} * V$	`uf`	UNDERFLOW
`lt`	$N * \overline{V} + \bar{N} * V$	`of`	OVERFLOW
`np`	NEVER	`pl`	$\overline{N}$ (/NEGATIVE)
`pl`	$\overline{N}$ (/NEGATIVE)	`ne`	$\bar{Z}$ (/ZERO)
`ne`	$\overline{Z}$ (/ZERO)	`nin`	/INFINITY
`vc`	$\overline{V}$ (/OVERFLOW)	`gle`	/NaN
`cc`	$\overline{C}$ (/CARRY)	`nie`	/IEX
`hi`	$C * Z$	`nio`	/IOP
`gt`	$N * V * \overline{Z} + \bar{N} * \bar{V} * \bar{Z}$	`nuf`	/UNDERFLOW
`ge`	$N * V + \overline{N} * \bar{V}$	`nof`	/OVERFLOW
`ms`	MODULO	`ogt`	$\overline{NaN + Z + N}$
`fs`	FULL	`oge`	$Z + \overline{Nan + N}$
`ns`	NLAST	`olt`	$N * \overline{(NAN + Z)}$
`ss`	SHADOW	`ole`	$Z + (N * \overline{NaN})$
`lz`	LOAD-ZERO	`ogl`	$\overline{NaN + Z}$
`lm`	LOAD-NEGATIVE	`fxs`	FLOATEX
`ds`	DATAEX	`ass`	ACS (accum. UF+OF+INF)
`ks`	CONTEX	`ais`	ACI (accum. IEX)
`mc`	/MODULO	`ule`	$NaN + Z + N$
`fc`	/FULL	`ult`	$NaN + N * \overline{Z}$
`nc`	/NLAST	`uge`	$NaN + Z + \overline{N}$
`sc`	/SHADOW	`ugt`	$NaN + \overline{N + Z}$
`ln`	/LOAD-ZERO	`ueq`	$NaN + Z$
`lp`	/LOAD-NEGATIVE	`fxc`	/FLOATEX
`dc`	/DATAEX	`asc`	/ACS
`kc`	/CONTEX	`aic`	/ACI

PRAM Assembler Shorthands

The SB-PRAM assembler `prass` generates COFF files from assembler sources. See [4] for a detailed description of syntax and options.

shorthand	for SB-PRAM instruction
`gethi S,Rd`	`gethi S>>13,Rd`
`geth S,Rd`	`gethi S,Rd`
`mov S,Rd`	`add R0,S,Rd`
`nop`	`gethi 0,R0`
`ret`	`popg R2,R1`
`ldgn Rx,S,Rd`	`ldg Rx,S,Rd`+ opt. `nop`

The assembler inserts a `nop` after `ldgn` automatically if `Rd` is used in the next instruction. Similar for `mpaddn`, `mpmaxn`, `mpandn`, `mporn`, `popgn`, `popign`, `popngn`

PRAMOS / Simulator Syscalls

Nr.	type	name	parameters (C declaration)
0	int	open	`char *file, int mode, flags`
1	int	read	`int fd, void *buf, int num`
2	int	write	`int fd, void *buf, int num`
3	int	close	`int fd`
4	int	lseek	`int fd, int offst, int orign`
5	int	sys_std_open	`int fd` open stdin/-out/-err
6	int	sys_abort	`int pc, int reason`
7	int	sys_getnr	get my physical processor ID
8	int	sys_exit	call OS program exit routine
9	int	sys_getct	get the global counter
12	int	sys_getbas	get BASE register
13	void	sys_putbas	`int base` write BASE register

For further syscalls see [3]

PRAM Simulator Environment File

`pramsim` needs the file `.pramsimrc` for setup of the simulator itself, located in your Fork95 source directory (see these in your `example` directory). Options:
`--datastart` determines offset for memory layout.
`--lazymalloc` for space economy during simulation.

PRAM Loader Environment File

The loader has been integrated into the simulator / OS. File `.ldrc` in your Fork95 source directory contains machine resource organization: number `procs` of available processors, maximum size `textlen` of text segment, maximum size of shared (`sheap`) and private (`pstack` and `pheap`) memory subspaces in the physical shared memory. These values are needed when loading and starting your Fork95 program (either by simulator or real machine).

Important PRAM Simulator Commands

Start the simulator by `pramsim` *myprog* for your compiled program *myprog* (COFF format). Commands:

`init` p q Simulate $p \times q$ processors. $p \leq 128$, $q \leq 32$.
`init` initialize program execution with same processors
`g` start / continue simulation
`r` q show all registers of processor q
`m` a show memory from address a in HEX
`k` a show memory from address a in HEX/ASCII
`help` show list of all available commands

Simulation can be stopped by pressing `Ctrl-C`. — An X-Windows based GUI for `pramsim` is under construction.

Languages and Compilers for SB-PRAM

Fork95 [7], contact the author for more details.
C [5], compiler based on gcc, with **p4** macro package

Help

Concerning simulator and assembler, mail questions to Stefan Franziskus, `stefran@obelix.cs.uni-sb.de`.

Concerning the operating system, ask Dr. Thomas Grün, `tgr@cs.uni-sb.de`.

Concerning the general hardware concept, ask Dr. Jörg Keller, `jkeller@cs.uni-sb.de`.

To get an account on the SB-PRAM (once it is running), contact Dr. Arno Formella, `formella@cs.uni-sb.de`.

Online Software and Documentation

A preliminary distribution of the SB-PRAM system software tools including assembler, linker, loader and simulator can be obtained via the Fork95 homepage `http://www.informatik.uni-trier.de/~kessler/fork95.html`, or by ftp from `ftp.informatik.uni-trier.de` in directory `/pub/users/Kessler`.

References

[1] F. Abolhassan, J. Keller, W.J. Paul. On the Cost–Effectiveness of PRAMs. Proc. 3rd IEEE Symp. on Par. and Distr. Processing, IEEE CS press, 1991.

[2] F. Abolhassan, R. Drefenstedt, J. Keller, W.J. Paul, D. Scheerer. On the Physical Design of PRAMs. Computer Journal **36**(8) 756–762, 1993.

[3] A. Crauser. Using SB-PRAM system calls in assembler language. Internal paper (1995), see `fork/pram/doc/sys1.ps`

[4] S. Franziskus. Der PRAM-Assembler `prass`. Internal paper (1995), see `fork/pram/doc/prass.ps`

[5] T. Grün, T. Rauber, J. Röhrig. The Programming Environment of the SB-PRAM. Proc. 7th IASTED/ISMM Int. Conf. on Par. and Distr. Computing and Systems, Wash. DC, 1995.

[6] J. Keller, W. J. Paul, D. Scheerer. Realization of PRAMs: Processor Design. Proc. WDAG'94, 8th Int. Workshop on Distributed Algorithms, Springer LNCS vol. 857, 17–27, 1994

[7] C. W. Kessler, H. Seidl. Integrating synchronous and asynchronous paradigms: The Fork95 programming language. Proc. MPPM-95 Int. IEEE Conf. on Massively Parallel Programming Models, Berlin, 1995.

[8] D. Scheerer. Der SB-PRAM Prozessor. PhD dissertation, Universität Saarbrücken, 1995.