The α-Ram family - bit level models for parallelism and concurrency.

Alex Berka

Isynchronise Ltd.

BCTCS April 8th 2020

Problems in Parallelism and Concurrency.

1. “Models and languages for parallel computation.” D B. Skillicorn and D Talia, ACM Computing Surveys 30, no. 2 (1998): 123–169.

“Data Centers in the Wild: A Large Performance Study.” R Birke, L.Y Chen, E Smirni. IBM Research Report RZ 3820 (2012).

“The Future of the Data Center: Heterogeneous Computing.” Charles Rowe. (2016)
https://www.dataversity.net/future-data-center-heterogeneous-computing/#.

2. “What are the Fundamental Structures of Concurrency? We still don’t know.” S. Abramsky. ENTCS Volume 162, 2006, Pgs 37-41

“Beautiful Concurrency.” Simon Peyton-Jones. Chapter in “Beautiful Code” edited by A. Oram and G. Wilson, (O'Reilly, 2007).

“The problem with threads.” E.A. Lee IEEE Computer 39, no. 5 (2006): 33-42.

α-Ram family and Synchronic Computation.

(References and resources on www.isynchronise.com.)

Benefits of α-Ram machine models:

- One machine called the Synchronic A-Ram is put forward as a fine grained theoretical model for parallelism.
- 4 primitive instructions are sufficient to express any finite parallel or concurrent process.
- Languages have simple semantics and a neutral machine platform - easier to reason about languages.
- Concurrency protocols and other constructs can be given lightweight implementations, without introducing complexity from the particulars of higher level models.
- Good for detecting errors and misconceptions - better for language design.
- For hardware design physical constraints can be introduced in a least restrictive order - reduce bias towards existing architectural types.
- Unknown opportunities for developing innovative programming environments, architectures, and theoretical frameworks.

Some Background to Synchronic Computation:

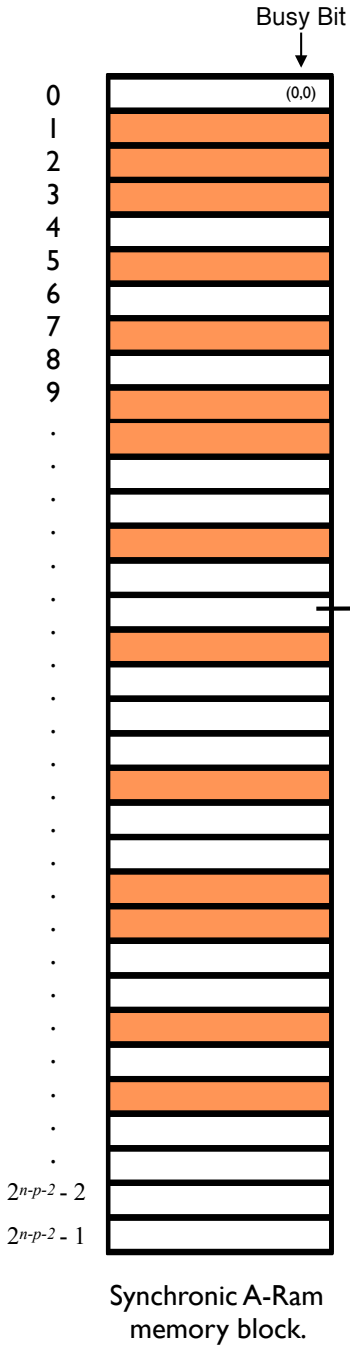
Questions the outlook that parallel/concurrent software can be adequately researched and developed in languages abstracted from hardware.

- Complete abstraction from hardware overkill - important opportunities for statically avoiding hazards/deadlocks lost.
- An abstract machine environment can maintain abstraction whilst avoiding side effects.

Deterministic parallelism can subsume non-deterministic concurrency for most (if not all) purposes.

- Planet can already be synchronised to near nanosecond precision.
- For larger machines introduce asynchrony only towards end of compilation - program semantics can be preserved.
- Oracles, and clocked agents hiding their internal workings from other clocked agents, can model non-determinism.

The A-Ram, with example of the Synchronic A-Ram.



Synchronic A-Ram memory block.

An A-ram is a tuple $\langle p, q, \sigma, \mu, \eta \rangle$.

1. $p \geq 3$ is an integer and is called the offset. $n = 2^p$ is the number of bits in a register.
2. $q \geq 0$ is called the opcode field, 2^{n-q} is the total number of bits in the memory block, and 2^{n-p-q} is the number of registers in the memory block.
3. $\sigma : N_{2^{(n-p-q)}} \times N_n \rightarrow \{0,1\}$. Let $\Sigma = \{ \sigma \mid \sigma : N_{2^{n-p-q}} \times N_n \rightarrow \{0,1\} \}$
4. μ is the marking, a subset $\mu \subseteq \left(N_{2^{(n-p-q)}} - \{0\} \right)$. Let $\mathrm{M} = P\left(N_{2^{n-p-q}} - \{0\} \right)$.
5. $\eta : \Sigma \times \mathrm{M} \rightarrow \Sigma \times \mathrm{M}$ is a total state transformation function.

A run of the A-Ram commences with an application of η to $\langle \sigma, \mu \rangle$. Thereafter $\langle \sigma, \mu \rangle = \eta(\sigma, \mu)$, either indefinitely, or until η identifies a tuple that halts the machine either successfully or in error.

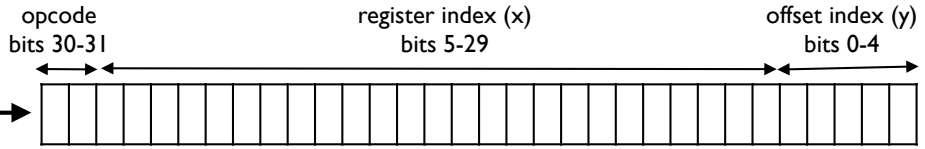


Table 1. Instruction format for 32 bit Synchronic A-Ram (p = 5, q = 2)

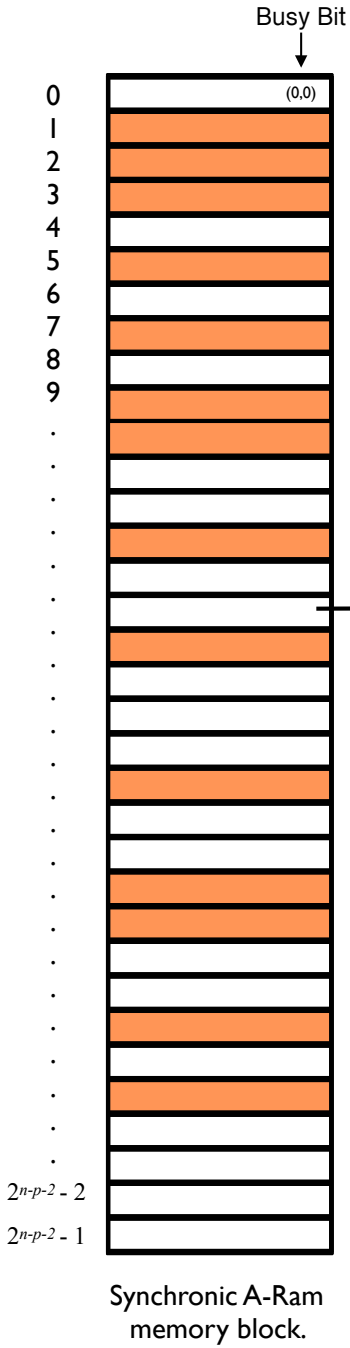
opcode	Instruction (assembly language)	action
0 0	**wrt0 x y**	Write 0 into (x,y).
0 1	**wrt1 x y**	Write 1 into (x,y).
1 0	**cond x y**	Examine contents of (x,y). If zero, mark instruction in next register for execution in next cycle. Else, mark instruction in next but one register.
1 1	**jump x y**	Mark all instructions from instruction x, up to an including instruction x+y for execution in the next cycle.

Table 2a. Instruction set for Synchronic A-Ram; a clocked, Concurrent Read, Exclusive Write Machine.

Basic Programming Element	**Synchronic A-Ram Instructions needed.**
Sequencing	Jump
Selection	Cond
Iteration	Jump and Cond
Parallel Fork	Jump

Table 2b. Basic elements for parallelism.

The A-Ram, with example of the Synchronic A-Ram.



Synchronic A-Ram memory block.

A Synchronic A-ram is a tuple $\langle p,2,\sigma,\mu,\eta \rangle$.

1. $p \geq 3$ is an integer and is called the offset. $n = 2^p$.
2. $\sigma : N_{2(n-p-2)} \times N_n \rightarrow \{0,1\}$. Let $\Sigma = \{ \sigma \mid \sigma : N_{2^{n-p-2}} \times N_n \rightarrow \{0,1\} \}$
3. μ is the marking, a subset $\mu \subseteq \left(N_{2(n-p-2)} - \{0\} \right)$. Let $\mathrm{M} = P\left(N_{2^{n-p-2}} - \{0\} \right)$.
4. $\eta : \Sigma \times \mathrm{M} \rightarrow \Sigma \times \mathrm{M}$ is a total state transformation function.

A run of the A-Ram commences with an application of η to $\langle \sigma,\mu \rangle$. Thereafter $\langle \sigma,\mu \rangle = \eta(\sigma,\mu)$, either indefinitely, or until η identifies a tuple that halts the machine either successfully or in error.

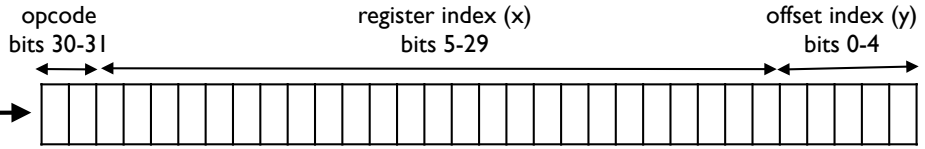


Table 1. Instruction format for 32 bit Synchronic A-Ram (p = 5, q = 2)

opcode	Instruction (assembly language)	action
0 0	**wrt0 x y**	Write 0 into (x,y).
0 1	**wrt1 x y**	Write 1 into (x,y).
1 0	**cond x y**	Examine contents of (x,y). If zero, mark instruction in next register for execution in next cycle. Else, mark instruction in next but one register.
1 1	**jump x y**	Mark all instructions from instruction x, up to an including instruction x+y for execution in the next cycle.

Table 2a. Instruction set for Synchronic A-Ram; a clocked, Concurrent Read, Exclusive Write Machine.

Basic Programming Element	**Synchronic A-Ram Instructions needed.**
Sequencing	Jump
Selection	Cond
Iteration	Jump and Cond
Parallel Fork	Jump

Table 2b. Basic elements for parallelism.

Simple Program Example.

Reg	cycle 10	Comment
0	0	reserved for machine status bits
1	wrt1 0 0	indicate machine is busy
2	jump 3 1	jump to initiate both jump gates
3	jump 5 0	1st AND gate includes carry thread
4	jump 10 0	2nd AND gate
5	cond 24 0	test input bit 0
6	jump 18 0	bit 0 is zero, so jump to exit sequence
7	cond 24 1	test input bit 1
8	jump 19 0	bit 1 is zero, so jump to exit sequence
9	jump 15 0	bit 0 and 1 are zero, so test result of 2nd AND gate
10	cond 24 2	test input bit 2
11	wrt0 24 4	bit 2 is zero, so write 0 into result bit of 2nd AND gate
12	cond 24 3	test input bit 3
13	wrt0 24 4	bit 3 is zero, so write 0 into result bit of 2nd AND gate
14	wrt1 24 4	bit 3 is 1, so write 1 into result bit of 2nd AND gate
15	cond 24 4	test result of 2nd AND gate
16	jump 20 1	2nd AND gate was zero, so write final result of zero
17	jump 21 1	2nd AND gate was one, so write final result of one
18	jump 19 0	longer exit sequence
19	jump 20 1	short exit sequence
20	wrt0 24 5	write final result of zero
21	jump 23 0	jump to halt instruction
22	wrt1 24 5	write final result of 1
23	wrt0 0 0	instruct machine to successfully halt
24	0 0 1 1 1 1	24 stores input bits 0-3, 2nd AND gate result in bit 4, and final result in bit 5

Fig 1: 4-input AND gate with two 2-input AND gates running in parallel

Environment and Languages for 32-bit Synchronic A-Ram.

Spatiale 1.0 environment and current status.

The project's current software tool was released in 2010.
Warning: Does not compile with recent versions of gcc - use gcc 3.3 or near version (MacOS Panther has it).

Spatiale -- (https://sourceforge.net/projects/spatiale/files/)

A unix console application written in C.
Simulator for a 32-bit Synchronic A-Ram.
Back end generates 32-bit Synchronic A-Ram machine code.

Front end languages:

Earth

User friendly assembly language close to the level of machine code.
Powerful enough to implement complex sequential digital circuits, with massive circuit parallelism.
Implements arithmetic-logic functional units and can concisely generate 000s lines of machine code.

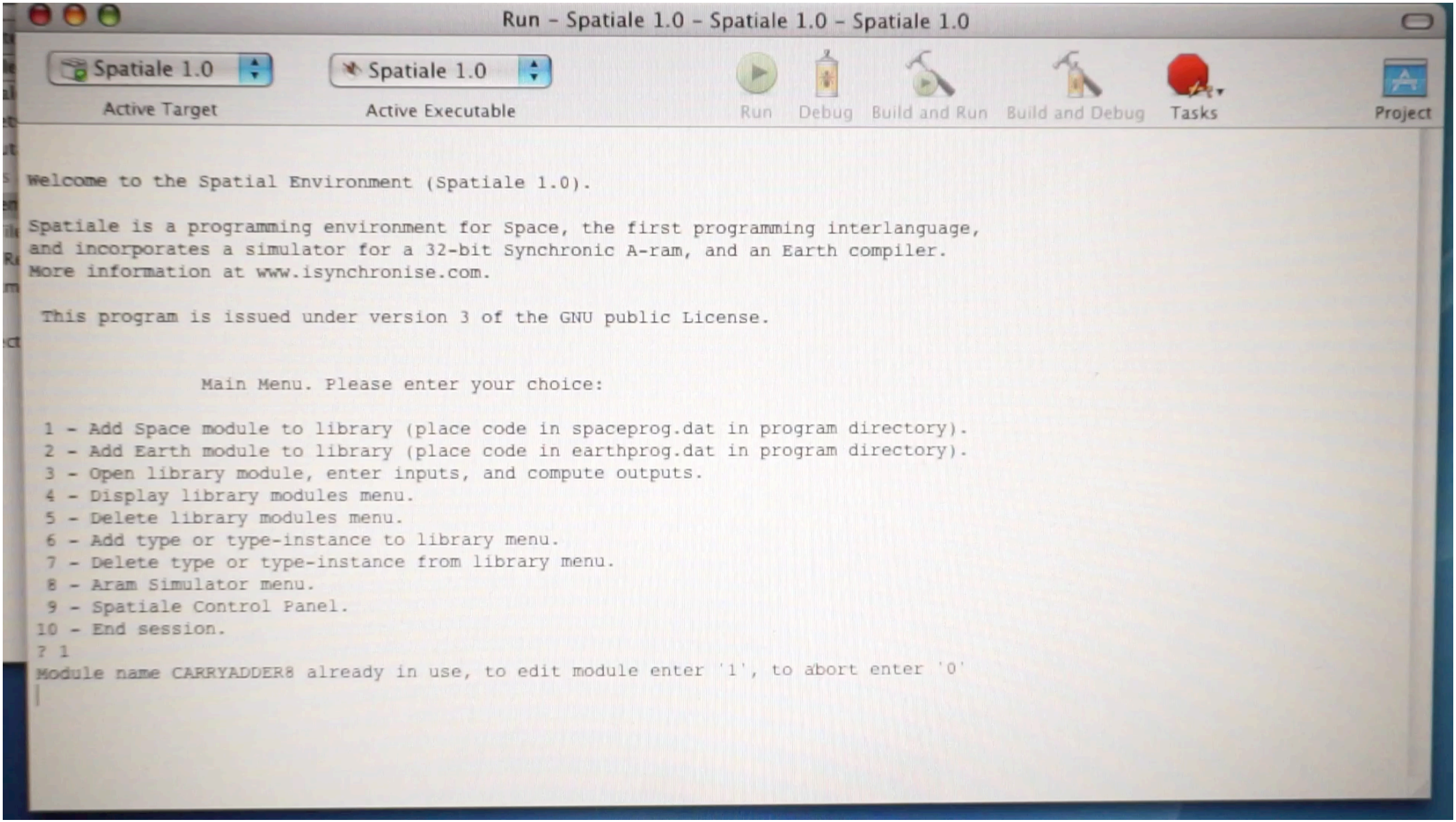
Space

Higher level language system with spatial semantics - uses Earth modules.
Future version will be fully general purpose for massive parallelism and concurrency.
Code has a circuit style with functionality comparable to C.

Some implemented low level modules.

Module for 32-bit Synchronic A-RAM	Lines of Earth code (w/o line breaks)	Lines of Compiled Machine Code (# registers)	Running time in A-Ram cycles
32 bit incrementer	21	262	4-67
Sequential 32-bit OR gate	12	100	3-66
Parallel 32-bit OR gate	28	140	23-30
Parallel 32-bit pairwise negate	18	136	4
Parallel 32-bit AND gate	n/a	97	14-18
8-bit serial adder	138	138	170-184
8-bit look-ahead carry adder	Space Program	1221	110
32-bit serial adder	138	138	674-736
32-bit barrel shift	129	2190	8
32-bit test for equality	43	424	16
32-bit shift left	24	139	7
prog source copy (R-RAM)	40	1845	7-9
prog target copy (W-RAM)	51	3548	7-9
prog copy register	71	5429	7-9
Jump Tree for starting 1024 threads	n/a	1057	3
Jump Tree for starting 65,536 threads	n/a	67,651	5

Table 3. Modules with register counts and cycle times.


Run – Spatiale 1.0 – Spatiale 1.0 – Spatiale 1.0
Spatiale 1.0
Active Target
Spatiale 1.0
Active Executable
Run
Debug
Build and Run
Build and Debug
Tasks
Project
Welcome to the Spatial Environment (Spatiale 1.0).
Spatiale is a programming environment for Space, the first programming interlanguage,
and incorporates a simulator for a 32-bit Synchronic A-ram, and an Earth compiler.
More information at www.isynchronise.com.
This program is issued under version 3 of the GNU public License.
Main Menu. Please enter your choice:
1 - Add Space module to library (place code in spaceprog.dat in program directory).
2 - Add Earth module to library (place code in earthprog.dat in program directory).
3 - Open library module, enter inputs, and compute outputs.
4 - Display library modules menu.
5 - Delete library modules menu.
6 - Add type or type-instance to library menu.
7 - Delete type or type-instance from library menu.
8 - Aram Simulator menu.
9 - Spatiale Control Panel.
10 - End session.
? 1
Module name CARRYADDER8 already in use, to edit module enter '1', to abort enter '0'

Sequential B-Ram - an α-Ram with countably infinite memory cells.

$p \gtrsim 3$ is an integer and is called the offset. $n = 2^p$ is the number of elements in a register, each element stores a bit. In this example $p = 4$, the register bit width $n = 16$. Each register has a cursor pointing to some memory block σ_i. Initially each register points to the same memory block in which it resides. The memory block has 512 registers.

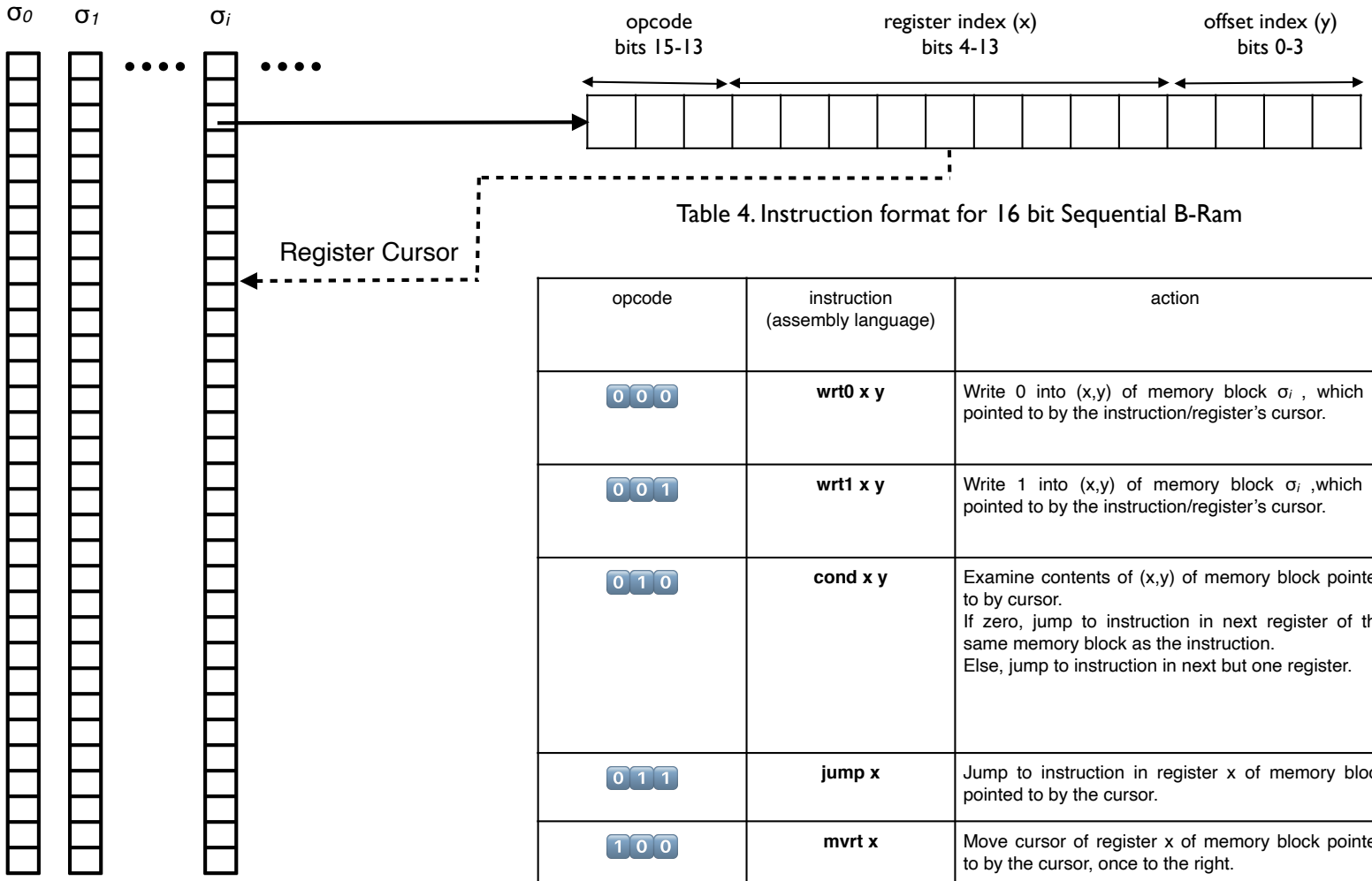


Table 4. Instruction format for 16 bit Sequential B-Ram

opcode	instruction (assembly language)	action
0 0 0	**wrt0 x y**	Write 0 into (x,y) of memory block σ_i , which is pointed to by the instruction/register's cursor.
0 0 1	**wrt1 x y**	Write 1 into (x,y) of memory block σ_i ,which is pointed to by the instruction/register's cursor.
0 1 0	**cond x y**	Examine contents of (x,y) of memory block pointed to by cursor. If zero, jump to instruction in next register of the same memory block as the instruction. Else, jump to instruction in next but one register.
0 1 1	**jump x**	Jump to instruction in register x of memory block pointed to by the cursor.
1 0 0	**mvrt x**	Move cursor of register x of memory block pointed to by the cursor, once to the right.
1 0 1	**mvlt x**	Move cursor of register x of memory block pointed to by cursor, once to the left..

Table 5. Instruction set for Sequential B-Ram

Sequential B-Ram program for incrementing an indefinitely large integer.

- Integer is stored as a succession of bits in the (0,0) cells of the succession of memory blocks after σ_0 with least significant bit in $\sigma_1(0,0)$.
- The (0,1) bit of a block will be set to indicate if the block contains the final, most significant bit, and reset otherwise. The most significant bit of the integer is always set.
- To run the program a second time, all the cursors of the registers marked * would have to be rewound to point to σ_0.

Tuple combination	Comment	Action
$\sigma(i,0,0)=0,\sigma(i,0,1)=0$	Less significant zero bit	Set bit, and halt machine.
$\sigma(i,0,0)=1,\sigma(i,0,1)=0$	Less significant one bit	Reset bit, and repeat loop.
$\sigma(i,0,0)=1,\sigma(i,0,1)=1$	Most significant bit (always one)	Reset bit, and set new final bit in next block, and halt machine.

Table 6. Pseudocode for *i* th iteration of main loop

Line	Instruction	Comment
0	....	// cell (0,0) of register 0 in σ_0 indicates business of Sequential B-Ram
1	wrt1 0 0	// Signal the Sequential B-Ram is busy, next, begin loop
2	mvrt 10	// move cursors of instructions marked * to next block
3	mvrt 13	
4	mvrt 16	
5	mvrt 18	
6	mvrt 20	
7	mvrt 21	
8	mvrt 24	
9	mvrt 25	
10	cond 0 1	// test if bit in current memory block is final *
11	jump 13	// bit is non-final, and and jump to test bit
12	jump 20	// bit is final, which is always reset
13	cond 0 0	// test non-final bit *
14	jump 16	// jump to set bit,and halt
15	jump 18	// or jump to reset bit, and repeat loop
16	wrt1 0 0	// set bit of non final bit *
17	wrt0 0 0	// halt
18	wrt0 0 0	// wrt0 into (0,0) of (next) memory block *
19	jump 2	
20	wrt0 0 1	// rewrite final bit status of current block *
21	wrt0 0 0	// reset bit *
22	mvrt 23	// move onto next block to write final bit
23	mvrt 24	// move onto next block to indicate final bit
24	wrt1 0 0	// *
25	wrt1 0 1	// *
26	wrt0 0 0	// halt

Fig 2: Sequential B-Ram stored program in memory block zero for incrementing indefinitely large integer.

Synchronic B-Ram - a parallel α-Ram with countably infinite memory cells.

$p \geq 3$ is an integer and is called the offset. $n = 2^p$ is the number of elements in a register, each element stores a bit. In this example $p = 4$, the register bit width $n = 16$. Each register has a cursor pointing to some memory block σ_i. Initially each register points to the same memory block in which it resides. The memory block has 512 registers.

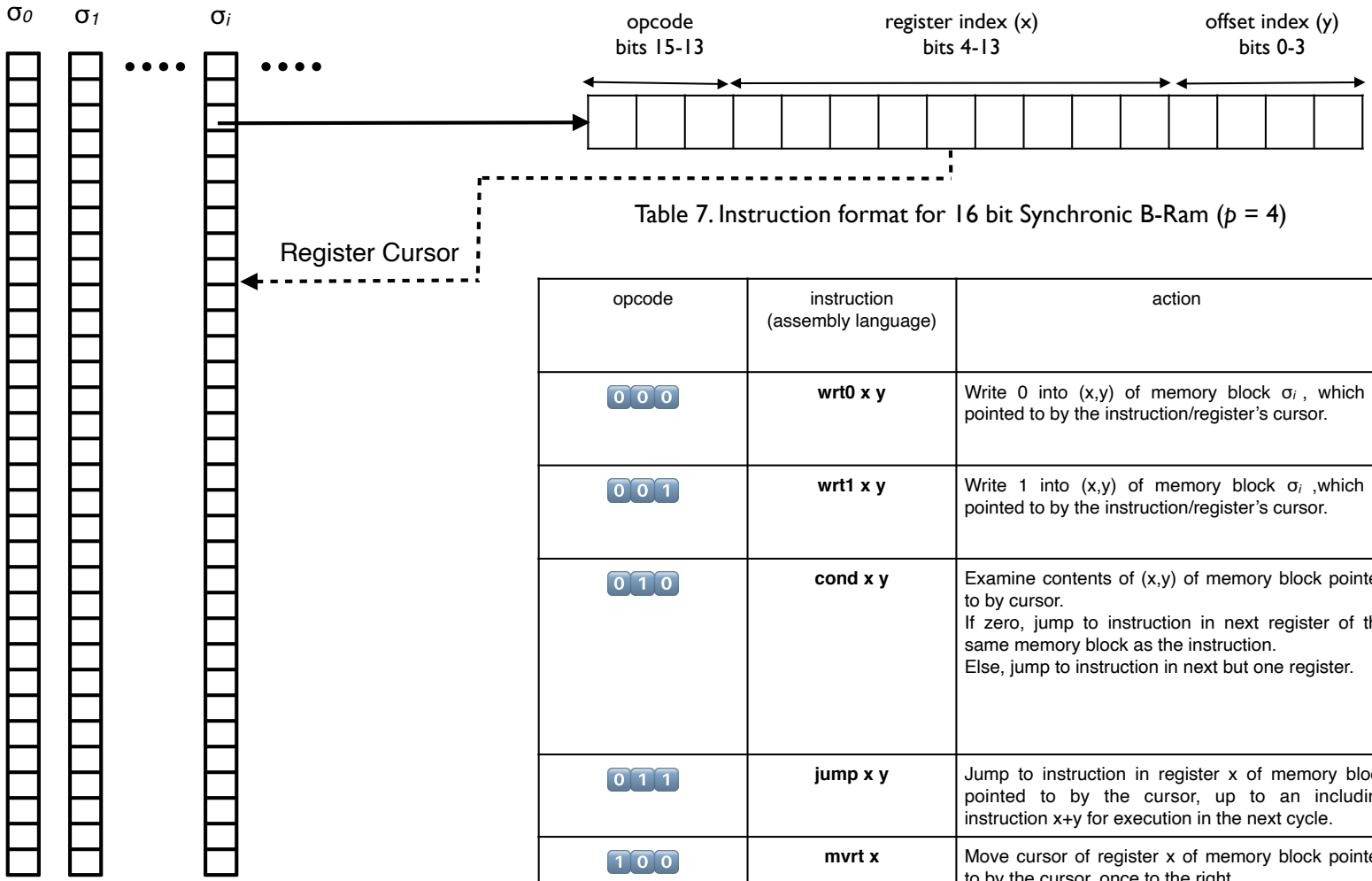


Table 7. Instruction format for 16 bit Synchronic B-Ram ($p = 4$)

opcode	instruction (assembly language)	action
0 0 0	**wrt0 x y**	Write 0 into (x,y) of memory block σ_i , which is pointed to by the instruction/register's cursor.
0 0 1	**wrt1 x y**	Write 1 into (x,y) of memory block σ_i ,which is pointed to by the instruction/register's cursor.
0 1 0	**cond x y**	Examine contents of (x,y) of memory block pointed to by cursor. If zero, jump to instruction in next register of the same memory block as the instruction. Else, jump to instruction in next but one register.
0 1 1	**jump x y**	Jump to instruction in register x of memory block pointed to by the cursor, up to an including instruction x+y for execution in the next cycle.
1 0 0	**mvrt x**	Move cursor of register x of memory block pointed to by the cursor, once to the right.
1 0 1	**mvlt x**	Move cursor of register x of memory block pointed to by cursor, once to the left.

Table 8. Instruction set for Synchronic B-Ram.

Related Work.

• Multi-Tape Turing Machine. Hopcroft,JE Ullman,JD “Introduction to Automata Theory, Languages, and Computation.” Pg 161. (Addison- Wesley 1979)

Parallelization of *n* instances of a 1-tape Turing machine with *m* states, results in an *n*-tape Turing Machine with m^n states. Berka A. V., “Interlanguages and Synchronic Models of Computation”, Pg 234. http://arxiv.org/pdf/1005.5183 published 2010.

• Petri Nets. Petri, Carl A. (1962). Kommunikation mit Automaten (Ph. D. thesis). University of Bonn.

Implementation of an *n*-input logic gate in one transition requires 2^n places.

• High Level Synthesis of sequential programs into RTL descriptions of Sequential Logic Circuits. I. Page, "Constructing Hardware-Software Systems from a Single Description," J. VLSI Signal Processing, Dec. 1996, pp. 87-107.

Source programs can generate huge RTL descriptions.
Bit level, but digital circuits are not fundamentally primitive structures.
Semantics of selection in C programs is not fundamental - based on constructs multiplexers/ demultiplexers.

- Proposal for another Invariance Thesis

(Sequential) Invariance and Parallel Computation Theses: Peter van Emde Boas. “Machine models and simulations.” In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, volume A: Algorithms and Complexity, pages 1–66. North-Holland, Amsterdam, 1990.

Parallel Invariance Thesis:

Reasonable parallel machines can simulate each other with constant factor overheads in space and time.

Conclusion.

α-Rams present:

- Ultra fine grained machine models.
- A simple semantics for languages - just 4 primitive instructions can express any concurrent module.
- A neutral machine platform for language and protocol design, without introducing biases from higher level models.
- Tractable simulations of complex what if scenarios.
- A basis for investigating processes running on a basic device, rather than in formalism abstracted from hardware.
- A member called the Synchronic A-Ram, which is put forward as a finite machine model for general purpose parallelism and concurrency.
- For hardware design physical constraints can be introduced in a least restrictive order, thereby reducing bias towards existing architectural types.
- B-Rams have infinite memories, and are Turing Computable.
- Machine concepts underlying Synchronic Computation.

Download slides and papers from isynchronise.com/resources.

α-Rams & Synchronic Computation (SC) project.

SC bootstraps the development of parallel languages and machine architectures based on a novel theoretical approach.

• Develop a non-graph based, textual language system incorporating an abstract machine environment (interlanguage), for concurrent expression and concise description of many-to-many relationships and DAGs.

• Use interlanguage features to provide insight for developing formal, bit-level machine models of parallel computation (α-Ram family).

• Use interlanguages and α-Rams to develop a general purpose environment (Space) for parallelism/concurrency.

• Use Space and α-Rams to develop specialised and more general purpose parallel architectures (Synchronic Engines).

• Using only enough informal logic and set theory to define α-Rams, attempt to recast mathematical structures, including logic and the axioms of set theory, as parallelised computational structures. Both constructive and non-constructive approaches could be pursued. (In contrast with Homotopy Type Theory, which is tied to the sequential λ-calculus, conventional tree syntax based formalisms, and abstracted from machines.)

• Further practical and theoretical objectives.

Table of Related Work 1

Warning: May contain value judgements.

Standard Formal Models	Lowest Level of Abstraction	Simple Semantics	Suitable for study of computability	Suitable for study of time and space complexity	Parallelism available	Tractable for computationally simulating Parallelism	Efficient for computationally simulating Random Access of Memory	Suitable for simulating processes and testing constructs in parallel/ concurrent language design
Turing Machine	bit level	yes	yes	yes	yes (Multi-Tape)	no	No	No
λ-calculus	below bit level	yes	yes	no	no	N/A	No	N/A
Register Machines (RASP etc..)	register	less so	less so	less so	no	N/A	Yes	N/A
RAM/Von Neumann Model	register	less so	less so	less so	no	N/A	Yes	N/A
α-RAM	bit level	informally yes formally perhaps less so	Don't know	Don't know	yes	yes	Yes	Yes

Table 10. Standard Models (& α-RAM).

Table of Related Work 2

Warning: May contain omissions and value judgements.

Parallel Model	Lowest Level of Abstraction	Involves notion of a machine	Associated Machine Architecture/ Implementation	General Purpose for Parallelism	Inherently deadlock free	Computationally efficient for Machine Model to execute Parallelism/ Concurrency	Deterministic mode of execution
λ-graph parallel rewriting	bit level	No	Von Neumann Network (VNN)	Yes	With Transactional Memory - Yes	No because of overheads incurred by graph manipulations, and recursion-to-iteration transformations.	Dependent on implementation
Dataflow/ Streaming Model/ Cellular Automata.	word/register	Yes	VNN//DSP/Coarse Grained Array (CGA)	No	Yes for Synchronous Reactive Languages	Yes	Dependent on implementation
Petri Nets	bit level	Yes	VNN/DSP/CGA	No	No	bit level - No higher -Yes	No
CSP	bit level	No	Various e.g. Edinburgh Concurrency Workbench	Yes	No	Yes	No
Occam	bit level	Yes	VNN/FPGA	Yes	No	Low processor utilization	No
Multi-threading	bit level	VNN generally assumed	VNN	Good for sequential task parallelism and divide and conquer, less so for everything else	No	Low processor utilization	No
Bulk Synchronous Model	register level	Yes	VNN	Single Program Multiple Data - no	Yes	low processor utilization	Yes
High Level Synthesis of C	bit level	Von Neumann Core attached to ASIC/FPGA	Von Neumann Core attached to ASIC/ FPGA	Normally only inner loops of sequential programs are dealt with.	Logic verified at the C level.	Yes for accelerated inner loops.	No
OpenACC	Can operate at sub-register level	Assumes Heterogeneous Machine	Heterogeneous (VNN/GPU/FPGA)	Yes	With Lock Free Programming - Yes	Low utilization of Von Neumann cores, GPUs and FPGAs more efficient	Yes
α-Ram/Space	bit level	Yes	Heterogeneous (Synchronic Engine in development/ GPU/FPGA)	Yes	Yes	Yes	Yes

Table 11. Parallel Models and α-Ram/Space

Formal definition of the Synchronic A-Ram

A Synchronic A-Ram is a tuple $\langle p,\sigma,\mu,\eta \rangle$. (We elide *q*, which is set to 2.)

1. Offset. $p \geq 3$ is an integer and is called the offset. Let $n = 2^p$. n is the number of elements in a register, where each element stores a member of the set $\{0,1\}$. (p is the minimum number of elements required to encode the position of an element in a register, using the alphabet of $\{0,1\}$.)
2. Memory. The memory block is a function $\sigma : N_{2^{(n-p-2)}} \text{ x } N_n \rightarrow \{0,1\}$ which takes a register index *x*, an element index *y*, and delivers the contents of the *y*th element of the yth register. Let $\Sigma = \{ \sigma \mid \sigma : N_{2^{n-p-2}} \text{ x } N_n \rightarrow \{0,1\} \}$ be the set of all possible memory blocks.
3. Marking. The initial marking $\mu \subseteq \left(N_{2^{(n-p-2)}} - \{0\} \right)$ represents a subset of the registers in the memory block, whose activation initiates the machine run. The zeroth register cannot be marked and is reserved for machine status bits. Let $\mathbf{M} = P\left(N_{2^{n-p-2}} - \{0\} \right)$ be the set of all markings.
4. State transformation function. The state transformation function is a total function $\eta : \Sigma \text{ x } \mathbf{M} \rightarrow \Sigma \text{ x } \mathbf{M}$. A run of the A-Ram commences with an application of η to $\langle \sigma,\mu \rangle$, which constitutes the first machine cycle. Thereafter, a succession of cycles proceeds, where the input to η is the output from the previous cycle, either indefinitely, or until an application of η classifies a tuple as terminating, requiring no further applications of η. The user may determine when a run terminates when the busy bit (0,0) is reset, and the success or type of failure of the run, by examining the other bits of register 0.

Preliminaries to formal definition of the state transformation function η for Synchronic A-Ram.

1. Let $i \in N_{2^{n-p-2}}$ be a variable pointing to a register.
   i. $\sigma_y(i)$ is the integer represented by the bits 0-4 in the offset cell in binary form, where the bit 0 is the least significant bit.
   ii. $\sigma_x(i)$ is the integer represented by the bits 5-29 in binary form, with the least significant bit in bit 5.
   iii. $\sigma_z(i)$ is the integer represented by the bits 0-31 in binary form, with the least significant bit in bit 0.
2. We now define a write function $\omega : \Sigma \text{ x } N_{2^{n-p-2}} \text{ x } N_n \text{ x } \{0,1\} \rightarrow \Sigma$, which modifies the content of one cell of one register in σ ; $\omega(\sigma,i,j,b) = \{\langle i,j,b \rangle\} \cup \{ \langle x,y,z \rangle \in \sigma \mid (x \neq i) \vee (y \neq j) \}$.
3. The effect of two consecutive writes $\langle \omega(\sigma,a,b,c);\ \omega(\sigma,f,g,h) \rangle =_{DF} \omega(\omega(\sigma,a,b,c),f,g,h)$.
4. The function $\eta : \Sigma \text{ x } \mathbf{M} \rightarrow \Sigma \text{ x } \mathbf{M}$ can now be described for the Synchronic A-Ram. For brevity the cases are evaluated in order from top to bottom, but could reformulated to run in parallel. The value of the function is followed on the right by it's associated condition, expressed as a conventional logical formalism.

Error Scheme for Synchronic A-Ram

Types of Errors.

1.*Marking Fail*. A legal marking cannot be a multiset. (In Petri Net parlance, each cycle in a run should have a safe marking.) If a register is activated more than once in the same cycle, then the marking is emptied and (0,1) is set.

2. *Write Fail.* The Synchronic A-Ram performs simultaneous writes, but is not a concurrent write machine in the conventional sense. If at least two instructions write to the same location (x,y) in the same cycle, then the marking is emptied and (0,2) is set.

3. *Halt Fail*. The final marking should always only activate the special halt instruction. If **wrt0 0 0** is not the only instruction in a marking, then the marking is emptied and error bit (0,3) is set. This error helps to ensure the module halt is meaningful, and as intended.

4.*Live Fail*. A program should not unexpectedly generate an empty marking. If the marking becomes empty (without the halt instruction having been activated in the previous cycle), then the marking is emptied and (0,4) is set, and the machine halts in failure. This error helps to ensure the module halt is meaningful, and as intended.

5.*Cond Fail*. If a cond instruction in the last but one register in the memory block is activated for execution, then the positive consequent of the cond is undefined. The machine ignores the instruction and writes a 1 to (0,5).

6.*Consequent Fail.* A cond instruction may also be misused if any two of the triplet of a cond instruction, it's negative consequent instruction, and positive consequent instruction are in a marking. If this occurs the marking is emptied and (0,6) is set.

7.*Active Fail*. An instruction should not have its contents modified in the same cycle in which it is active. If the marking contains a write to a bit within a marked register, then the marking is emptied and (0,7) is set.

8.*Jump Fail*. A jump instruction should not attempt to mark a register index which is illegal or does not exist. If the destination cell points to the zero register, or if the destination cell plus the offset exceeds the total number of registers in the memory block (), then the marking is emptied and (0,8) is set.

9.*Error Fail.* Programs may not write to the designated error bits. The marking that includes a wrt instruction to any bit in register zero, other than the zeroth bit, is illegal, and (0,9) is set.

Notes:

A Synchronic A-Ram machine cycle may be viewed as having a read phase, followed by a write phase. Therefore instructions **cond a b**, and a **wrtx a b** can occur in the same marking, where the cond instruction reads (a,b), before it is overwritten by the wrt instruction. The errors are evaluated sequentially within a cycle, but could be reformulated to be processed in parallel.

The proposed error detection scheme imposes a substantial computational cost for every machine cycle, not only for a simulation, but also for hardware based on the concept. Errors may however, be bypassed with a suitably designed language, compiler and programming methodology, thereby eliminating the need for much of the scheme's implementation. (No proof as yet.) The main benefit of the error scheme has been to assist in the debugging of Spatiale 1.0, and thus far, it can be reported that a wide range of compiled (massively parallel) programs, producing outputs as expected, without activating the error detection mechanisms that have been implemented.

$$\eta(\sigma,\mu)=$$

$\langle\, \omega(\sigma,0,0,0), \varnothing\rangle,$ if $\exists! i \in \mu\ (\sigma_z(i)=0)$	(0)	Successful halt
$\langle\, \langle\, \omega(\sigma,0,1,1);\ \omega(\sigma,0,0,0)\ \rangle, \varnothing\rangle,$ if μ is a multi set.	(1)	Marking Fail
$\langle\, \langle\, \omega(\sigma,0,2,1);\ \omega(\sigma,0,0,0)\ \rangle, \varnothing\rangle,$ if $\exists i,j \in \mu \begin{pmatrix} (i \neq j) \wedge (\sigma(i,n\text{-}1)=0) \wedge (\sigma(j,n\text{-}1)=0) \\ \wedge (\sigma_x(i)=\sigma_x(j)) \wedge (\sigma_y(i)=\sigma_y(j)) \end{pmatrix}$	(2)	Write Fail
$\langle\, \langle\, \omega(\sigma,0,3,1);\ \omega(\sigma,0,0,0)\ \rangle, \varnothing\rangle,$ if $\exists i,j \in \mu\ ((i \neq j) \wedge (\sigma_z(i)=0))$	(3)	Halt Fail
$\langle\, \langle\, \omega(\sigma,0,4,1);\ \omega(\sigma,0,0,0)\ \rangle, \varnothing\rangle,$ if $(\mu=\varnothing) \wedge (\sigma(0,0)=1)$	(4)	Live Fail
$\langle\, \langle\, \omega(\sigma,0,5,1);\ \omega(\sigma,0,0,0)\ \rangle, \varnothing\rangle,$ if $\exists i \in \mu \begin{pmatrix} (\sigma(i,n\text{-}1)=1) \wedge (\sigma(i,n\text{-}2)=0) \\ \wedge\ (i = 2^{n-p-2}-2) \end{pmatrix}$	(5)	Cond Fail
$\langle\, \langle\, \omega(\sigma,0,6,1);\ \omega(\sigma,0,0,0)\ \rangle, \varnothing\rangle,$ if $(\exists k \in 2^{n-p-2})(\exists i,j \in \mu) \begin{pmatrix} ((\sigma(k,n\text{-}1)=1) \wedge (\sigma(k,n\text{-}2)=0)) \\ \wedge \begin{pmatrix} ((i=k+1) \wedge (j=k+2)) \ \vee \\ \left((i=k) \wedge \begin{pmatrix} (j=k+1) \vee \\ (j=k+2) \end{pmatrix}\right) \end{pmatrix} \end{pmatrix}$	(6)	Consequent Fail
$\langle\, \langle\, \omega(\sigma,0,7,1);\ \omega(\sigma,0,0,0)\ \rangle, \varnothing\rangle$ if $\exists i,j \in \mu\ ((i \neq j) \wedge (\sigma(j,n\text{-}1)=0) \wedge (\sigma_x(j)=i))$	(7)	Active Fail
$\langle\, \langle\, \omega(\sigma,0,8,1);\ \omega(\sigma,0,0,0)\ \rangle, \varnothing\rangle$ if $\exists i \in \mu \begin{pmatrix} (\sigma(i,n\text{-}1)=1) \wedge (\sigma(i,n\text{-}2)=1) \\ \wedge\ (((\sigma_x(i)=0) \vee (\sigma_x(i)+\sigma_y(i) \geq 2^{n-p-2}))) \end{pmatrix}$	(8)	Jump Fail
$\langle\, \langle\, \omega(\sigma,0,9,1);\ \omega(\sigma,0,0,0)\ \rangle, \varnothing\rangle$ if $\exists i \in \mu((\sigma_x(i)=0) \wedge (\sigma_y(i) \neq 0))$	(9)	Error Fail
$\langle \eta_1(\sigma,\mu), \eta_2(\sigma,\mu) \rangle$ otherwise	(10)	Legal (non-halting) cycle

Fig 3. State Transformation Function for Synchronic A-Ram $\langle p,\sigma,\mu,\eta \rangle$, recall $n=2^p$.

The memory function $\eta_1 : \Sigma \mathrm{x} \mathrm{M} \rightarrow \Sigma$ is defined:

$W =_{\mathrm{DF}} \left\{ \langle i,r,s \rangle \in N_{2^{n-p-2}} \mathrm{x}\, N_{2^{n-p-2}} \mathrm{x}\, N_n \;\middle|\; \exists i \exists r \exists s \left(\left(i \in \mu \right) \wedge \left(\sigma(i,n-1) = 0 \right) \wedge \left(r = \sigma_x(i) \right) \wedge \left(s = \sigma_y(i) \right) \right) \right\}$ - write instructions' register indexes with their bit co-ordinates of write locations

$W' =_{\mathrm{DF}} \left\{ \langle r,s \rangle \in N_{2^{n-p-2}} \mathrm{x}\, N_n \;\middle|\; \exists t \exists i \exists r \exists s \left(\left(t \in W \right) \wedge \left(t = \langle i,r,s \rangle \right) \right) \right\}$ - bit co-ordinates of new write locations

$$\eta_1(\sigma,\mu) = \left(\bigcup_{\langle i,r,s \rangle \in W} \langle r,s,\sigma(i,n-2) \rangle \right)$$ - new writes to σ

$$\cup \left(\bigcup_{\langle r,s \rangle \in \left(\left(N_{2^{n-p-2}} \mathrm{x}\, N_n \right) - W' \right)} \langle r,s,\sigma(r,s) \rangle \right)$$ - rest of σ from previous cycle

N.B. η_1 will generate a function and not a relation, because a write fail is impossible (see previous slide).

The marking function $\eta_2 : \Sigma \mathrm{x} \mathrm{M} \rightarrow \mathrm{M}$ is defined: Let $i, j \in N,\ 0 < j \leq 2^p - 1$. Let $[i,0] \equiv i$ and $[i,j] \equiv i, i+1, \ldots i+j$. then

$$\eta_2(\sigma,\mu) = \left(\bigcup_{\left\{ i \mid (i \in \mu) \wedge (\sigma(i,n-1)=1) \wedge (\sigma(i,n-2)=0) \wedge \left(\sigma\left(\sigma_{x(i)},\sigma_{y(i)}\right)=0 \right) \right\}} i+1 \right)$$ - cond false consequents

$$\cup \left(\bigcup_{\left\{ i \mid (i \in \mu) \wedge (\sigma(i,n-1)=1) \wedge (\sigma(i,n-2)=0) \wedge \left(\sigma\left(\sigma_{x(i)},\sigma_{y(i)}\right)=1 \right) \right\}} i+2 \right)$$ - cond true consequents

$$\cup \left(\bigcup_{\left\{ i \mid (i \in \mu) \wedge (\sigma(i,n-1)=1) \wedge (\sigma(i,n-2)=1) \right\}} \left[\sigma_x(i), \sigma_y(i) \right] \right)$$ - jumps with offsets

N.B. η_2 may generate a multi-set marking, which will produce a marking fail in the next cycle.

Fig 4. Memory and marking functions for legal non-halting cycle in Synchronic A-Ram.