An integrated concurrency and core-ISA architectural envelope definition, and test oracle, for IBM POWER multiprocessors

Kathryn E. Gray[1] Gabriel Kerneis[1+] Dominic Mulligan[1]
Christopher Pulte[1] Susmit Sarkar[2] Peter Sewell[1]

[1] University of Cambridge [1+]During work [2] University of St Andrews

What is an architecture spec?

What is an architecture spec?

Typically prose

What is an architecture spec?

Typically prose

What is an architecture spec?

Typically prose

Sometimes pseudocode

What is an architecture spec?

Typically prose

Sometimes pseudocode

Branch ***I-form***

b	target_addr	(AA=0 LK=0)
ba	target_addr	(AA=1 LK=0)
bl	target_addr	(AA=0 LK=1)
bla	target_addr	(AA=1 LK=1)

18	LI	AA	LK
0	6	30	31

```
if AA then NIA ←iea EXTS(LI || 0b00)
else       NIA ←iea CIA + EXTS(LI || 0b00)
if LK then LR ←iea CIA + 4
```

target_addr specifies the branch target address.

If AA=0 then the branch target address is the sum of LI || 0b00 sign-extended and the address of this instruction, with the high-order 32 bits of the branch target address set to 0 in 32-bit mode.

If AA=1 then the branch target address is the value LI || 0b00 sign-extended, with the high-order 32 bits of the branch target address set to 0 in 32-bit mode.

If LK=1 then the effective address of the instruction following the Branch instruction is placed into the Link Register.

Special Registers Altered:

LR (if LK=1)

Branch Conditional ***B-form***

bc	BO,BI,target_addr	(AA=0 LK=0)
bca	BO,BI,target_addr	(AA=1 LK=0)
bcl	BO,BI,target_addr	(AA=0 LK=1)
bcla	BO,BI,target_addr	(AA=1 LK=1)

16	BO	BI	BD	AA	LK
0	6	11	16	30	31

```
if (64-bit mode)
  then M ← 0
  else M ← 32
if ¬BO2 then CTR ← CTR - 1
ctr_ok ← BO2 | ((CTR[M:63] ≠ 0) ⊕ BO3)
cond_ok ← BO0 | (CR[BI+32] ≡ BO1)
if ctr_ok & cond_ok then
  if AA then NIA ←iea EXTS(BD || 0b00)
  else       NIA ←iea CIA + EXTS(BD || 0b00)
if LK then LR ←iea CIA + 4
```

BI+32 specifies the Condition Register bit to be tested. The BO field is used to resolve the branch as described in Figure 42. *target_addr* specifies the branch target address.

If AA=0 then the branch target address is the sum of BD || 0b00 sign-extended and the address of this instruction, with the high-order 32 bits of the branch target address set to 0 in 32-bit mode.

If AA=1 then the branch target address is the value BD || 0b00 sign-extended, with the high-order 32 bits of the branch target address set to 0 in 32-bit mode.

If LK=1 then the effective address of the instruction following the *Branch* instruction is placed into the Link Register.

Special Registers Altered:

CTR (if BO_2=0)
LR (if LK=1)

Extended Mnemonics:

Examples of extended mnemonics for *Branch Conditional*:

Extended:		**Equivalent to:**	
blt	target	bc	12,0,target
bne	cr2,target	bc	4,10,target
bdnz	target	bc	16,0,target

But ...

But ...

- *Not* executable test oracles
  - You can’t test h/w or s/w against prose

But ...

- *Not* executable test oracles
  - You can't test h/w or s/w against prose
- *Not* a clear guide to concurrent behaviour
  - Especially for weakly consistent IBM Power and ARM

But ...

- *Not* executable test oracles
  - You can't test h/w or s/w against prose
- *Not* a clear guide to concurrent behaviour
  - Especially for weakly consistent IBM Power and ARM
- A mass of instruction set detail

Specification as Artefact

We (show how to) make
architecture specs that are *real* technical artefacts

Specification as Artefact

We (show how to) make
architecture specs that are *real* technical artefacts

- Executable as test oracle

Specification as Artefact

We (show how to) make
architecture specs that are *real* technical artefacts

- Executable as test oracle
- Mathematically precise

Specification as Artefact

We (show how to) make
architecture specs that are *real* technical artefacts

- Executable as test oracle
- Mathematically precise
- Related to vendor pseudocode and intuition

Specification as Artefact

We (show how to) make
architecture specs that are *real* technical artefacts

- Executable as test oracle
- Mathematically precise
- Related to vendor pseudocode and intuition
- Clarify interface between ISA and concurrency

Specification as Artefact

We (show how to) make
architecture specs that are *real* technical artefacts

Specifically IBM POWER
- all non-FP non-vector "user" ISA (153 instructions)
- and concurrency model

Specification as Artefact

We (show how to) make
architecture specs that are *real* technical artefacts

Specifically IBM POWER
all non-FP non-vector "user" ISA (153 instructions)
and concurrency model

Applicable to ARM as well
See Modelling the ARMv8 Architecture, Operationally
Concurrency and ISA, POPL16

Not just an emulator

Not just an emulator

Emulator

PPCMEM2

Not just an emulator

Emulator

Written in C etc

A language with many faults

Intermingling of emulation detail
& semantics

PPCMEM2

Not just an emulator

Emulator

Written in C etc
A language with many faults

Intermingling of emulation detail
& semantics

PPCMEM2

Written in Lem & Sail
Languages for logic,
mathematics, and ISAs

Only spec detail
Emulation separated

Not just an emulator

Emulator

PPCMEM2

Running concurrent code:
Consider a lock

Not just an emulator

Emulator

T1 Lock
T1 Set critical section
T1 Unlock
T2 Lock
…

repeat

PPCMEM2

Running concurrent code:
Consider a lock

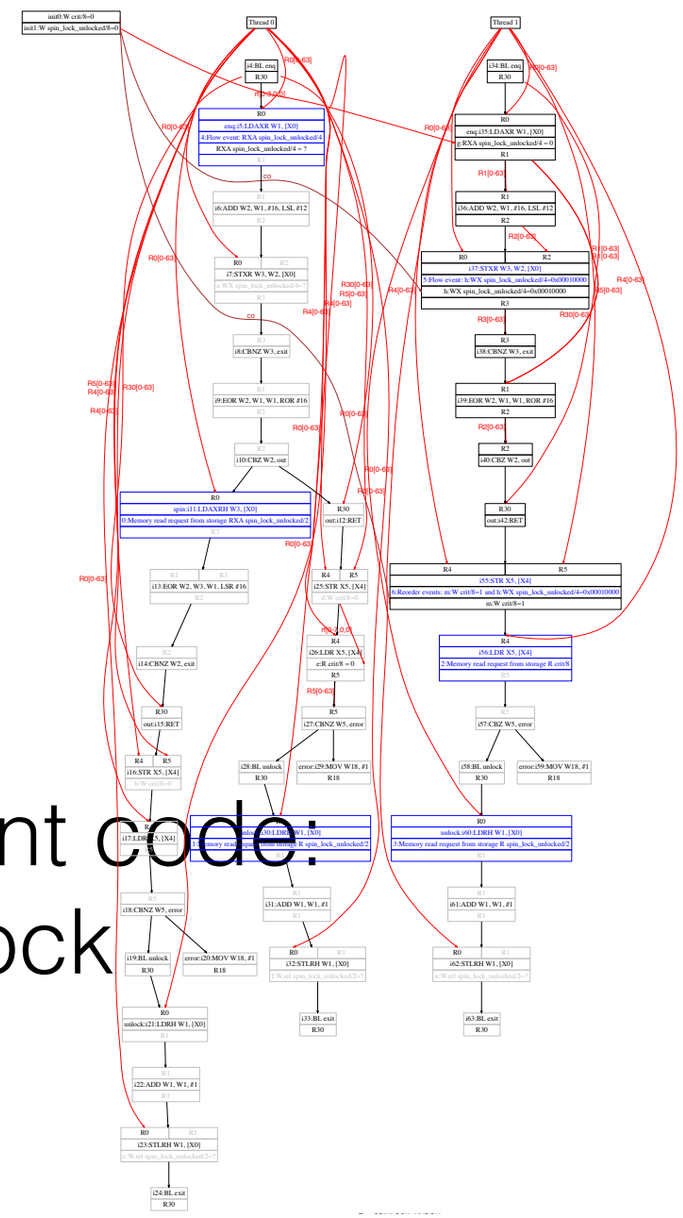
Not just an emulator

Emulator

T1 Lock
T1 Set critical section
T1 Unlock
T2 Lock
…

repeat

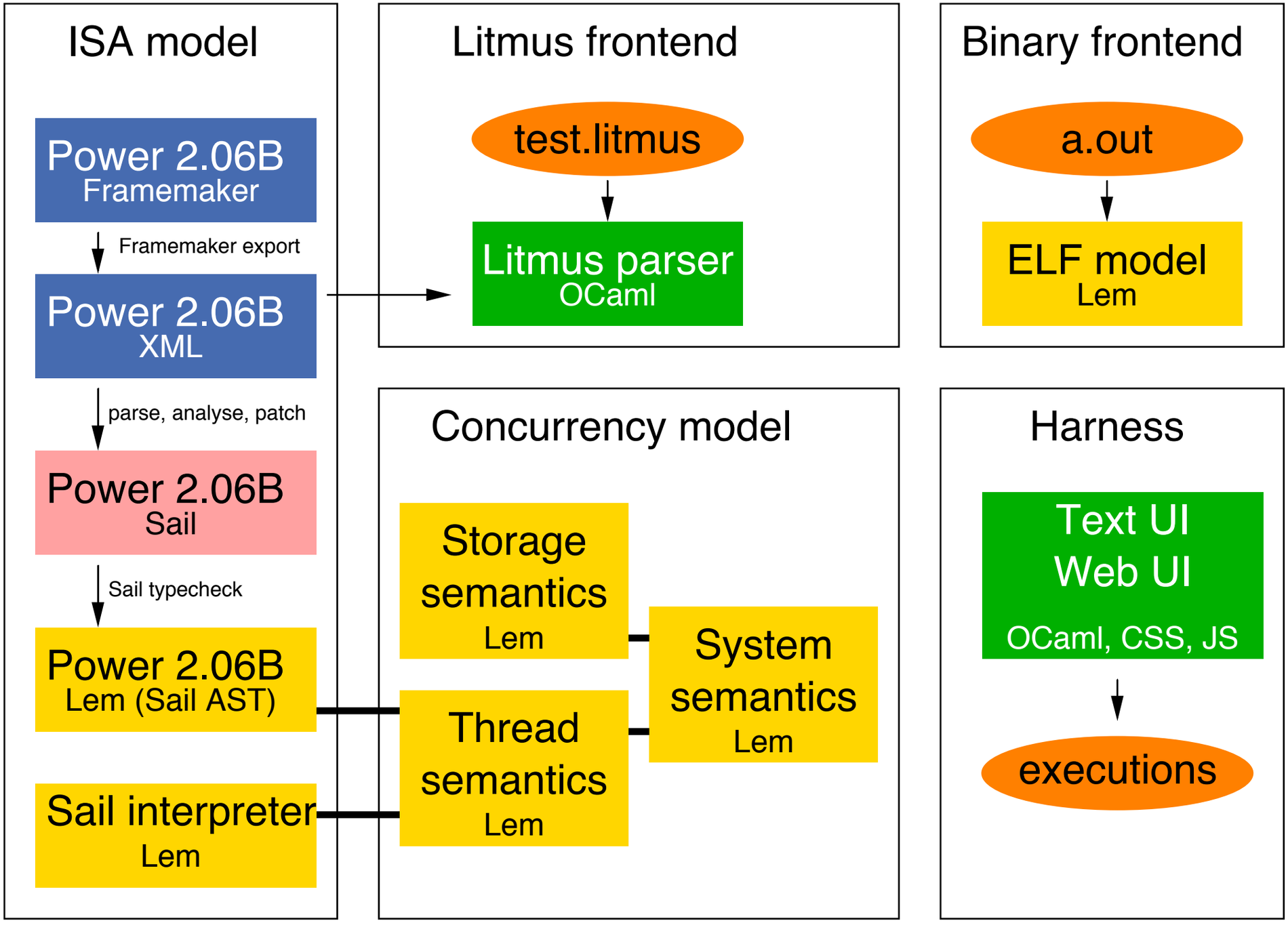
PPCMEM2



Running concurrent code:
Consider a lock

Beneficiaries

- Compiler writers
- Concurrency primitive implementors
- Security developers
- Hardware developers


ISA model
Power 2.06B
Framemaker
Framemaker export
Power 2.06B
XML
parse, analyse, patch
Power 2.06B
Sail
Sail typecheck
Power 2.06B
Lem (Sail AST)
Sail interpreter
Lem
Litmus frontend
test.litmus
Litmus parser
OCaml
Binary frontend
a.out
ELF model
Lem
Concurrency model
Storage
semantics
Lem
System
semantics
Lem
Thread
semantics
Lem
Harness
Text UI
Web UI
OCaml, CSS, JS
executions

Sample Instruction

Store Word with Update ***D-form***

stwu RS,D(RA)

37	RS	RA	D
0	6	11	16 31

```
EA ← (RA) + EXTS(D)
MEM(EA, 4) ← (RS)32:63
RA ← EA
```

Let the effective address (EA) be the sum (RA)+ D. $(RS)_{32:63}$ are stored into the word in storage addressed by EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

Special Registers Altered:

None

```
union ast member (bit[5], bit[5], bit[16]) Stwu

function clause decode (0b100101 :
                        (bit[5]) RS :
                        (bit[5]) RA :
                        (bit[16]) D as instr) =
  Stwu (RS,RA,D)

function clause execute (Stwu (RS, RA, D)) = {
 (bit[64]) EA := 0;
 EA := GPR[RA] + EXTS(D);
 GPR[RA] := EA;
 MEMw(EA,4) := (GPR[RS])[32 .. 63]
}
```

Sample Instruction

Store Word with Update ***D-form***

stwu RS,D(RA)

37	RS	RA	D
0	6	11	16 31

```
union
function clause
Stwu (RS,RA,D)

function clause
(bit[64])
EA
GPR[RA]
MEMw(EA,4)
}
```

```
EA ← (RA) + EXTS(D)
MEM(EA, 4) ← (RS)32:63
RA ← EA
```

Let the effective address (EA) be the sum (RA)+ D. $(RS)_{32:63}$ are stored into the word in storage addressed by EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

Special Registers Altered:

None

Sample Instruction

Store Word with Update ***D-form***

stwu RS,D(RA)

37	RS	RA	D
0	6	11	16 31

```
EA ← (RA) + EXTS(D)
MEM(EA, 4) ← (RS)32:63
RA ← EA
```

Let the effective address (EA) be the sum (RA)+ D. $(RS)_{32:63}$ are stored into the word in storage addressed by EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

Special Registers Altered:

None

```
union ast member (bit[5], bit[5], bit[16]) Stwu

function clause decode (0b100101 :
                        (bit[5]) RS :
                        (bit[5]) RA :
                        (bit[16]) D as instr) =
  Stwu (RS,RA,D)

function clause execute (Stwu (RS, RA, D)) = {
 (bit[64]) EA := 0;
 EA := GPR[RA] + EXTS(D);
 GPR[RA] := EA;
 MEMw(EA,4) := (GPR[RS])[32 .. 63]
}
```

Sample Instruction

Store Word with Update *D-form*

stwu RS,D(RA)

37	RS	RA	D
0	6	11	16 31

EA ← (RA) + EXTS(D)
MEM(EA, 4) ← $(RS)_{32:63}$
RA ← EA

Let the effective address (EA) be the sum (RA)+D. $(RS)_{32:63}$ are stored into the word in storage addressed by EA.

EA is placed into register RA.

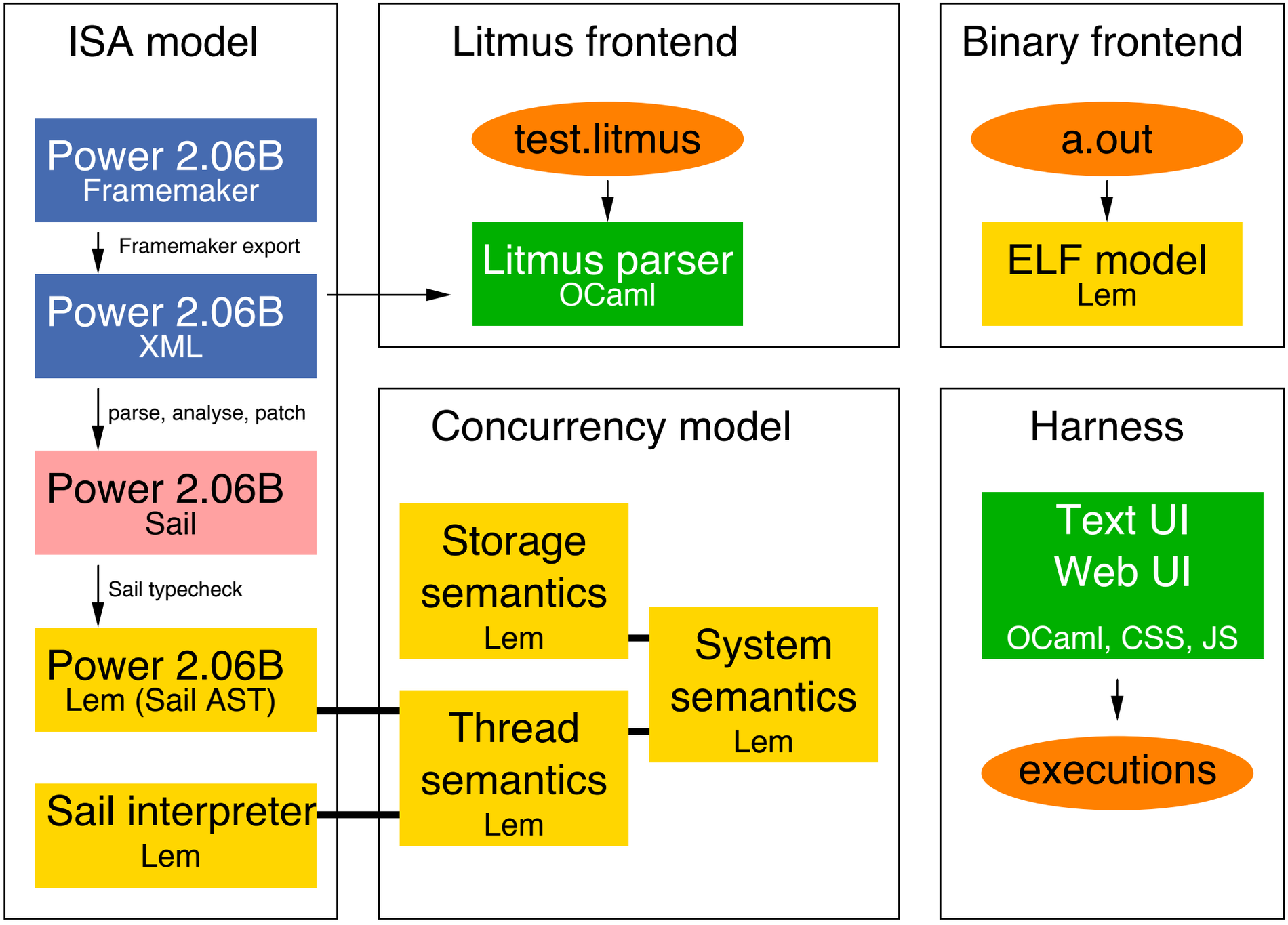
If RA=0, the instruction form is invalid.

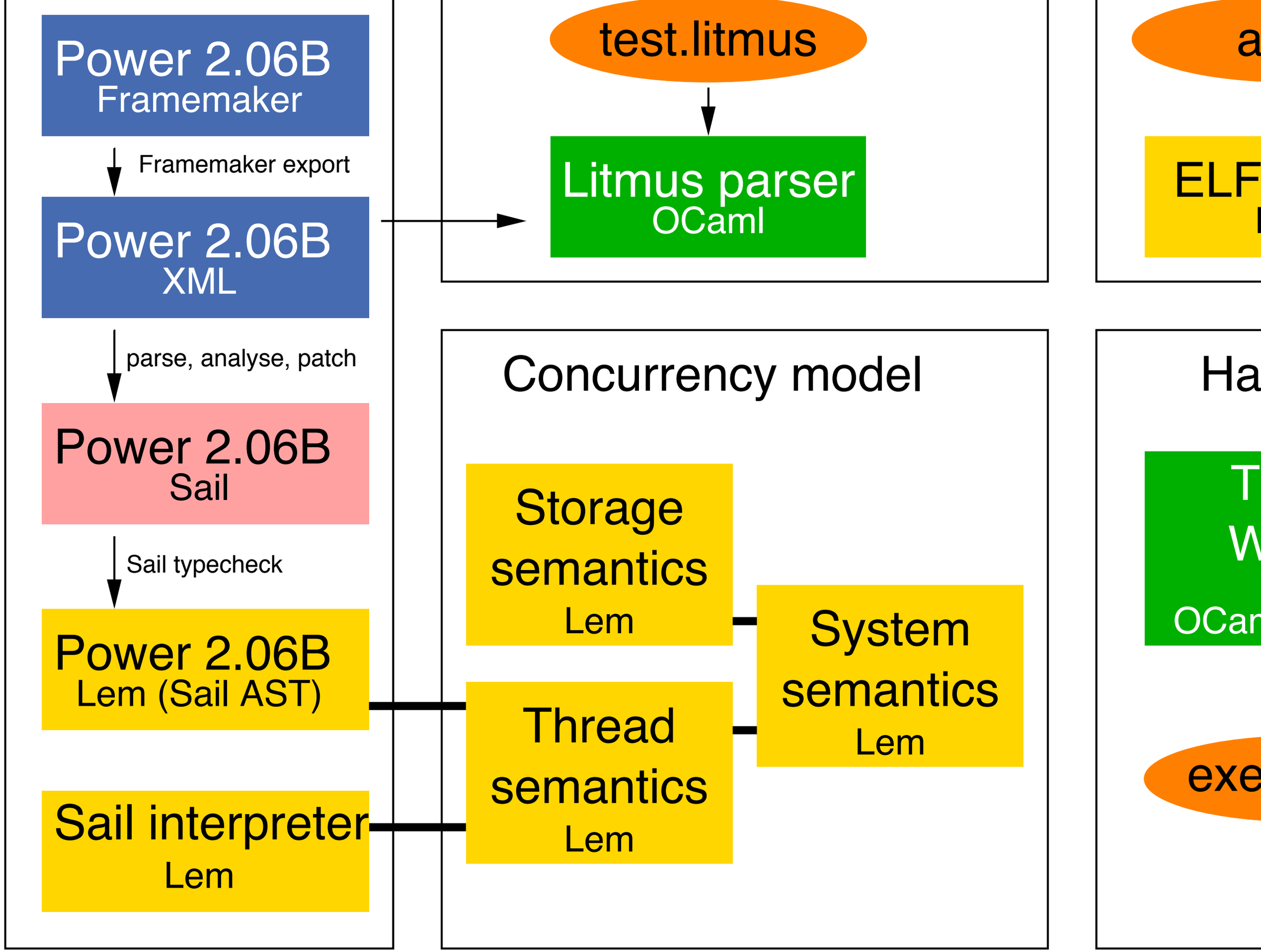
Special Registers Altered:
None

```
union ast member (bit[5], bit[5], bit[16]) Stwu

function clause decode (0b100101 :
                        (bit[5]) RS :
                        (bit[5]) RA :
                        (bit[16]) D as instr) =
  Stwu (RS,RA,D)

function clause execute (Stwu (RS, RA, D)) = {
  (bit[64]) EA := 0;
  EA := GPR[RA] + EXTS(D);
  GPR[RA] := EA;
  MEMw(EA,4) := (GPR[RS])[32 .. 63]
}
```

Sail:
for specifying concurrent ISAs

- C-like/ISA Pseudo-code like imperative language with
  - Built-in understanding of registers and memory
  - Type inference, including vector-size checking
- Formal interpreter
  - Executable for sequential or concurrent exploration
  - Analyses instructions for register/memory footprint


ISA model
Power 2.06B
Framemaker
Framemaker export
Power 2.06B
XML
parse, analyse, patch
Power 2.06B
Sail
Sail typecheck
Power 2.06B
Lem (Sail AST)
Sail interpreter
Lem
Litmus frontend
test.litmus
Litmus parser
OCaml
Binary frontend
a.out
ELF model
Lem
Concurrency model
Storage
semantics
Lem
System
semantics
Lem
Thread
semantics
Lem
Harness
Text UI
Web UI
OCaml, CSS, JS
executions


Power 2.06B
Framemaker
Framemaker export
Power 2.06B
XML
parse, analyse, patch
Power 2.06B
Sail
Sail typecheck
Power 2.06B
Lem (Sail AST)
Sail interpreter
Lem
test.litmus
Litmus parser
OCaml
Concurrency model
Storage semantics
Lem
Thread semantics
Lem
System semantics
Lem
ELF
Ha
T
W
OCam
exe

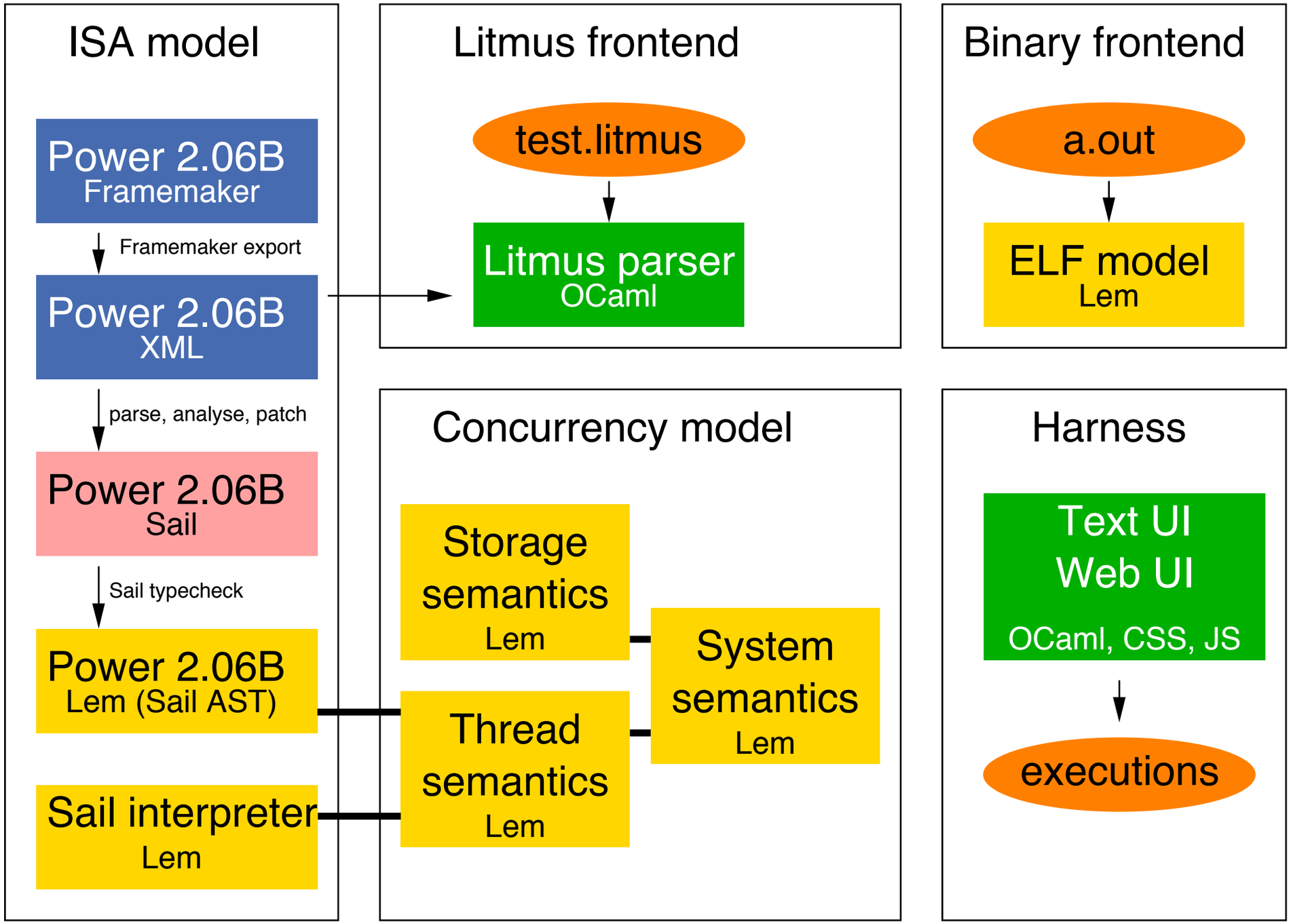
ISA + Concurrency Challenges

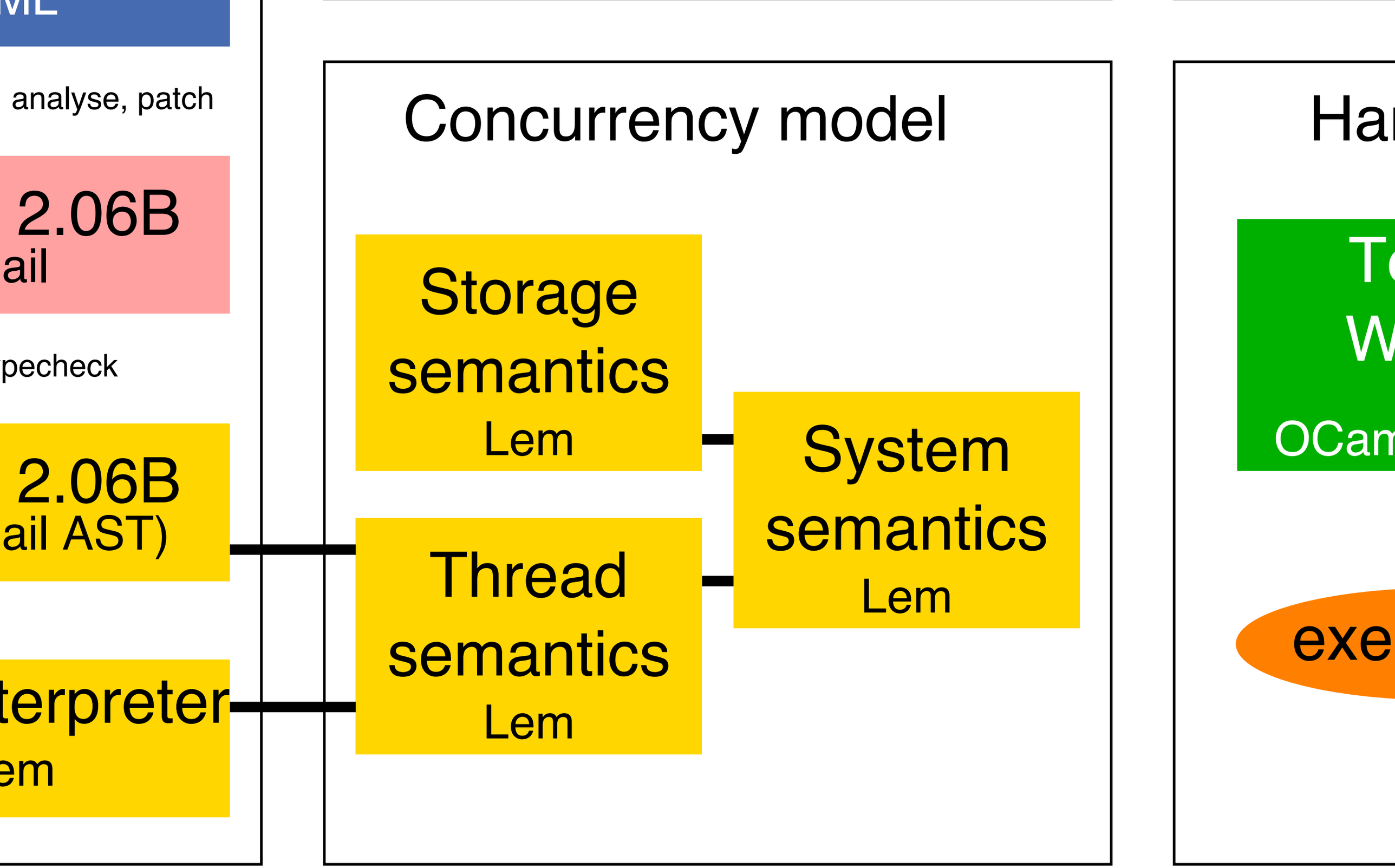
- No single program point
- No per-thread register state
- Register shadowing effects
- and more

No Per-thread register state

MP+sync+rs POWER

Thread 0		Thread 1	
stw r7,0(r1)	# x=1	lwz r5,0(r2)	# r5=y
sync	# sync	mr r6,r5	# r6=r5
stw r8,0(r2)	# y=1	lwz r5,0(r1)	# r5=x
Initial state: 0:r1=x, 0:r2=y, 0:r7=1, 0:r8=1, 1:r1=x, 1:r2=y, x=0			
Allowed: 1:r6=1, 1:r5=0			


ISA model
Power 2.06B
Framemaker
Framemaker export
Power 2.06B
XML
parse, analyse, patch
Power 2.06B
Sail
Sail typecheck
Power 2.06B
Lem (Sail AST)
Sail interpreter
Lem
Litmus frontend
test.litmus
Litmus parser
OCaml
Binary frontend
a.out
ELF model
Lem
Concurrency model
Storage
semantics
Lem
System
semantics
Lem
Thread
semantics
Lem
Harness
Text UI
Web UI
OCaml, CSS, JS
executions


ML
analyse, patch
2.06B
ail
pecheck
2.06B
ail AST)
terpreter
em
Concurrency model
Storage
semantics
Lem
System
semantics
Lem
Thread
semantics
Lem
Ha
To
W
OCam
exe

ML

analyse, patch

2.06B
ail

pecheck

2.06B
ail AST)

terpreter
em

Concurrency model

Storage semantics
Lem

System semantics
Lem

Thread semantics
Lem

Maintains tree of in-flight instructions

Ha

T
W
OCam

exe

ML

analyse, patch

2.06B
ail

pecheck

2.06B
ail AST)

terpreter
em

Concurrency mo

Abstraction of cache hierarchies and protocols, store buffers, etc

Storage semantics
Lem

System semantics
Lem

Thread semantics
Lem

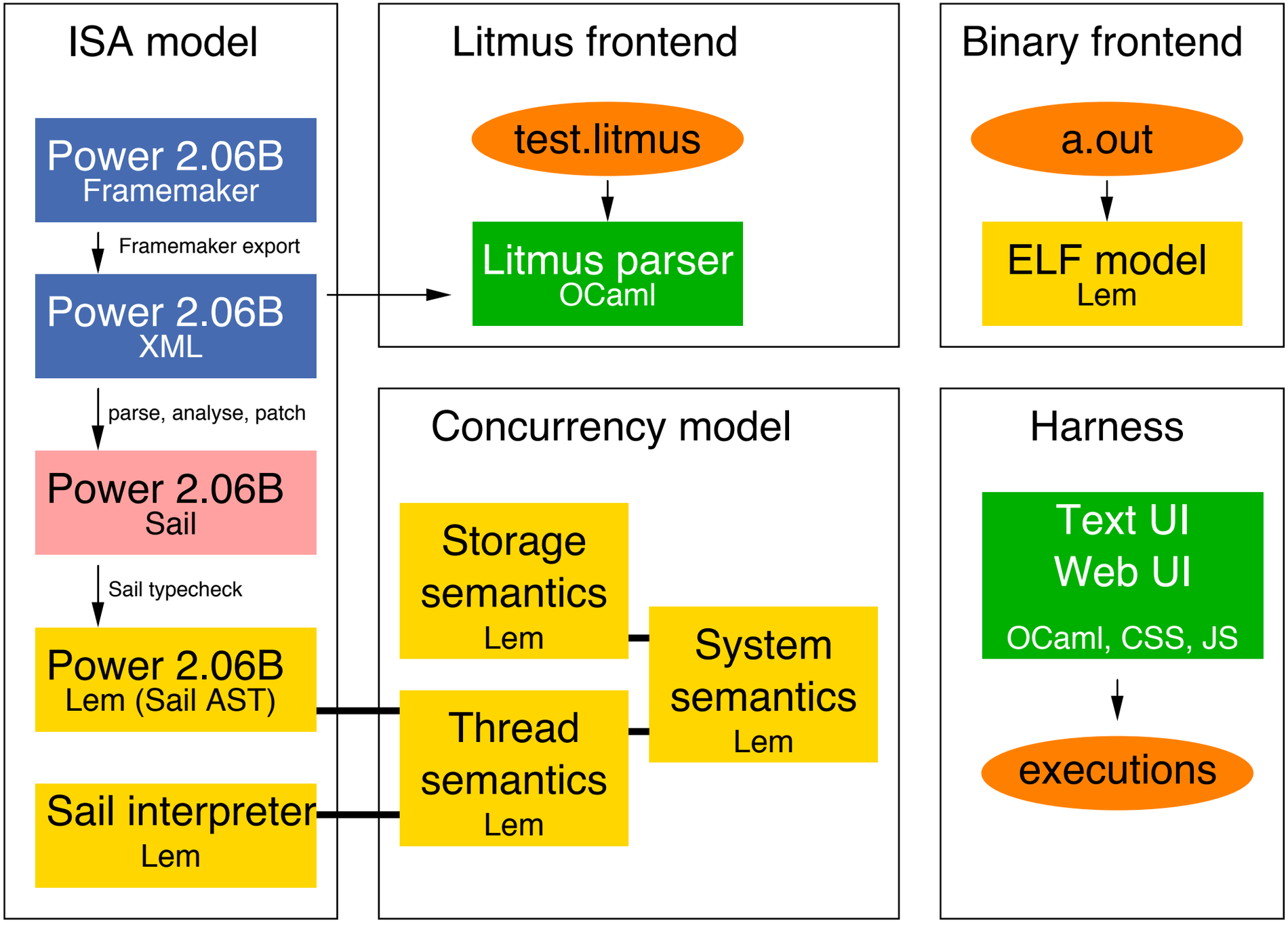
Maintains tree of in-flight instructions

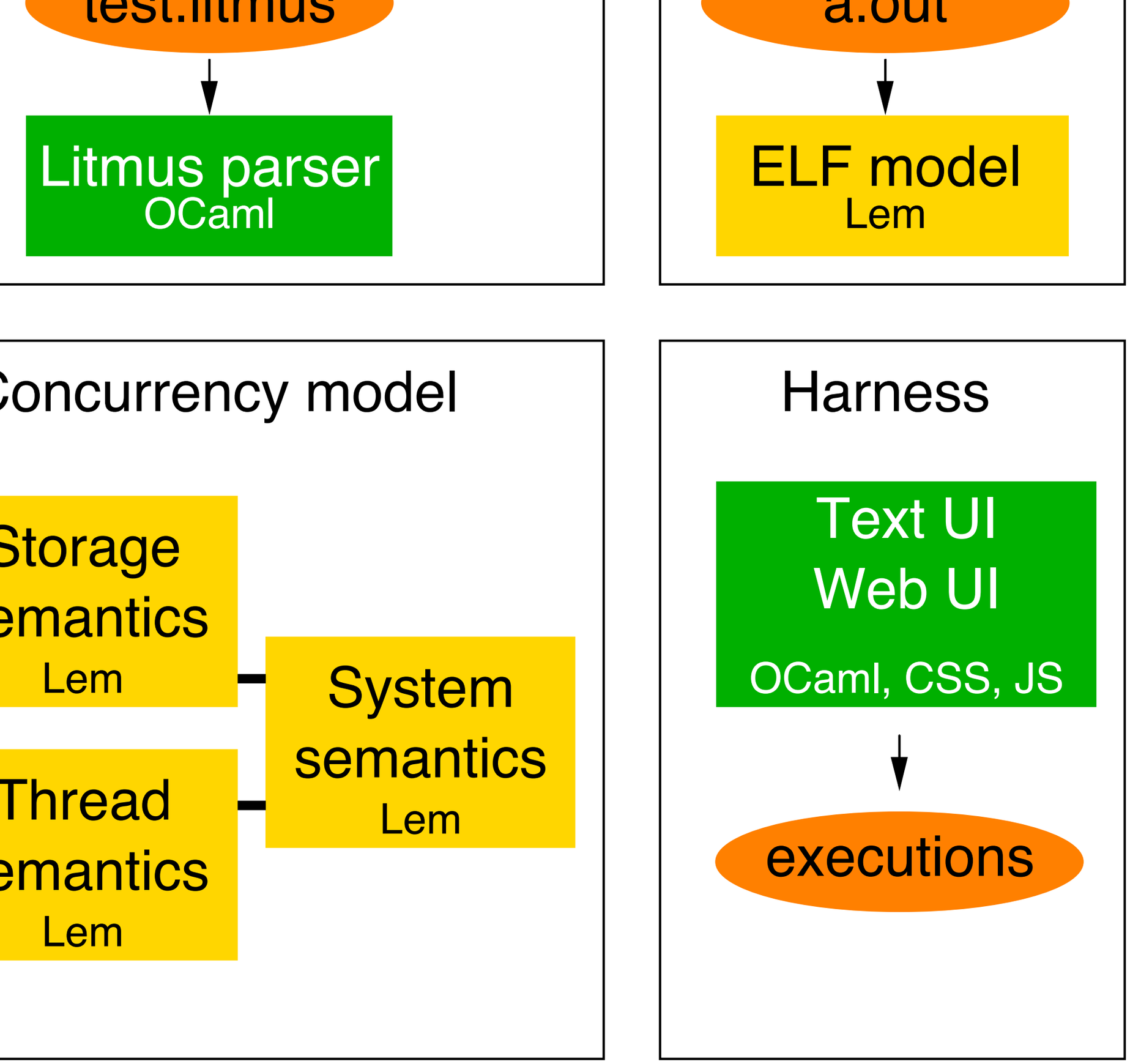
Har

T
W

OCam

exe


ISA model
Power 2.06B
Framemaker
Framemaker export
Power 2.06B
XML
parse, analyse, patch
Power 2.06B
Sail
Sail typecheck
Power 2.06B
Lem (Sail AST)
Sail interpreter
Lem
Litmus frontend
test.litmus
Litmus parser
OCaml
Binary frontend
a.out
ELF model
Lem
Concurrency model
Storage
semantics
Lem
System
semantics
Lem
Thread
semantics
Lem
Harness
Text UI
Web UI
OCaml, CSS, JS
executions


test.litmus
Litmus parser
OCaml
a.out
ELF model
Lem
Concurrency model
Storage semantics
Lem
Thread semantics
Lem
System semantics
Lem
Harness
Text UI
Web UI
OCaml, CSS, JS
executions

POWER Memory Model

www.cl.cam.ac.uk/users/sf502/pldi11/index.html

Google News Humor Films Food UK Comp Teaching misc

POWER Memory Model

```
PPC MP+sync+ctrl
"SyncdWW Rfe DpCtrldR Fre"
Cycle=Rfe DpCtrldR Fre SyncdWW
{
0:r2=x; 0:r4=y;
1:r2=y; 1:r4=x;
}
 P0            | P1            ;
 li r1,1       | lwz r1,0(r2)  ;
 stw r1,0(r2)  | cmpw r1,r1    ;
 sync          | beq  LC00     ;
 li r3,1       | LC00:         ;
 stw r3,0(r4)  | lwz r3,0(r4)  ;
exists
(1:r1=1 /\ 1:r3=0)
```

Select POWER Test | Select Options | Run Interactive | Help

System State

```
Storage subsystem state:
  writes seen   = {W x/8=0, W y/8=0, W x/4=1}
  coherence     = {W x/8=0 -> W x/4=1}
  writes_past_coherence_point = {W x/8=0, W y/8=0}
events propagated to:
    Thread 0: [W y/8=0 [0-7], W x/8=0 [0-7], W x/4=1 [0-3], Sync ]
    Thread 1: [W y/8=0 [0-7], W x/8=0 [0-7]]
  2 Propagate write to thread: W x/4=1 [0-3] to Thread 1
  unacknowledged Sync requests = {Sync }

Thread 0 state:
  unacknowledged Syncs = {Sync }
  old instructions
  ioid: 4    loc: 0x0000000000050000   addi RT=1 RA=0 SI=1
  ioid: 5    loc: 0x0000000000050004   stw RS=1 RA=2 D=0                    mem writes: [W x/4=1]
  new instructions
  ------------------------
  ioid: 6    loc: 0x0000000000050008   sync L=0
  ioid: 7    loc: 0x000000000005000c   addi RT=3 RA=0 SI=1
  ioid: 8    loc: 0x0000000000050010   stw RS=3 RA=4 D=0                    opcode:    instruction kind: write
    regs_in: {GPR4, GPR3[32..63]}   regs_out: {}   ioids_feeding_address: {}   nias: {succ}
    mem writes_read_from: []   committed_mem_writes: []   committed_barriers: []
    reg_reads: []
    reg_writes: []   committed: false   finished: false   micro_op_state: MOS_plain
    |    b := (GPR[4]);
    |    EA := (b + EXTS (64,D));
    |    MEMw(EA,4) := ((GPR[to_num (RS)])[32 .. 63])
    Env: D=0b0...0 [0..15], EA=0b0...0 [0..63], RA=0b00100 [0..4], RS=0b00011 [0..4], b=0b0...0 [0..63]
  0 (0:8) Register read: GPR4 = y from initialstate of 0-63:y

Thread 1 state:
  unacknowledged Syncs = {}
  Initial fetch address: 0x0000000000051000   (from notional predecessor ioid 2)
  1 (1:2) Fetch from address 0x0000000000051000 lwz RT=1 RA=2 D=0


Enabled transitions:
  0: (0:8) Register read: GPR4 = y from initialstate of 0-63:y
  1: (1:2) Fetch from address 0x0000000000051000 lwz RT=1 RA=2 D=0
  2: Propagate write to thread: W x/4=1 [0-3] to Thread 1
```

Validation

Sequential single instruction

- 6983 tests of fixed-point user-mode instruction

Concurrent litmus tests

- 2175 tests run exhaustively
  - including those from prior concurrency models

Conclusions

Combined ISA and concurrency model for IBM POWER

Developed w.r.t existing h/w
& in consultation with architects

Usable as reference model for future h/w & s/w

Usable for verification

Relevant for ARM and future models

http://www.cl.cam.ac.uk/~pes20/ppcmem2/