### The Semantics of x86 Multiprocessor Machine Code

### The HOL Specification

Susmit Sarkar Peter Sewell Francesco Zappa Nardelli Scott Owens Tom Ridge Thomas Braibant Magnus O. Myreen Jade Alglave

 $^1 \rm University$  of Cambridge  $^2 \rm INRIA$  October 30, 2008



i

BRIEF CONTENTS ii

### **Brief Contents**

I $x86$ _coretypes	5
II x86_types	7
III x86_ast	10
IV x86_axiomatic_model	12
V x86_niceness_statement	18
VI x86_seq_monad	20
VII x86_event_monad	24
VIII x86_opsem	29
$IX x86\_decoder$	36
$ m X  m x86_{-}$	42
XI x86_program	44
XII x86_sequential_axiomatic_model	47
$ ext{XIII}  ext{ x86\_drf}$	50
XIV x86_hb_machine	53
XV x86_lts_ops	58
XVI x86_hb_machine_thms	60
Index	63

### **Full Contents**

Int	${f troduction}$		1
RE	ADME-spec-pul	blic	1
Ι	x86_coret	ypes	5
	_	$type\_abbrev\_Ximm$	6
	_	Xreg	6
	_	Xeflags	6
	_	Xea	6
	_	type_abbrev_proc	6
	_	$type\_abbrev\_program\_order\_index$	6
	_	iiid	6
II	x86_type		7
		$type\_abbrev\_address$	8
	_	$type\_abbrev\_value$	8
	_	$type\_abbrev\_eiid$	8
	_	$type\_abbrev\_reln$	8
	_	reg	8
	_	dirn	8
	_	location	8
	_	barrier	8
	_	action	8
	_	event	8
	_	event_structure	8
	_	$type\_abbrev\_state\_constraint$	8
	_	type_abbrev_view_orders	8
	_	execution_witness	8
	_	type_abbrev_eiid_state	8
	_	next_eiid	8
	_	$initial\_eiid\_state$	9
	_	$machine\_visible\_label$	9
	_	$lts\_monad\_visible\_label$	9
	_	$label \dots \dots$	9
	_	$type\_abbrev\_machine\_label$	9
	_	$type\_abbrev\_lts\_monad\_label$	9

FULL CONTENTS	i	iv

	_	LTS	9
III	x86_ast	10	0
	_	Xrm	
	_	$rm\_is\_memory\_access$	_
	_	Xdest_src	
	_	Ximm_rm	
	_	Xbinop_name	
	_	$Xmonop\_name$	
	_	Xcond	
	_	Xinstruction	
	_	$Xpre\_g1$	
		$Xpre\_g2$	
		Xinst	
	_	Atust	1
IV	x86 axio	omatic_model 12	2
	_		3
	_	$value\_of$	
	_	proc	
	_	$mem\_load$	
	_	mem_store	
	_	$mem\_barrier$	
	_	iiids	
	_	writes	
	_	reads	
	_	locked	
	_	po	
		$po\_strict$	
	_	po_iico	
		$well\_formed\_event\_structure$	
	_	$sub\_event\_structure$	_
	_		_
	_	F. Comp. T. C.	_
	_		
	_	····	
	_	<i>3</i> · · · · · · · · ·	5
	_	and the same of th	5
	_	write_serialization_candidates	
	_	lock_serialization_candidates	
	_	$reads\_from\_map\_candidates\_old$	
	_	reads_from_map_candidates	
	_	happens_before	
	_	check_causality	
	_	check_rfmap_written	
	_	$check\_rfmap\_initial$	
	_	$state\_updates$	
	_	check_final	
	_	$state\_updates\_mem$	6

FULL CONTENTS	$\mathbf{v}$
---------------	--------------

	_	check_final_mem
	_	check_atomicity
	_	<i>valid_execution</i>
	_	restrict_execution_witness
$\mathbf{V}$	v86 nico	ness_statement 18
V	xoo_ince	
	_	nice_execution
	_	niceness_thm
<b>37T</b>	9 <i>C</i>	mand 20
VI	$x86\_seq$	
	_	type_abbrev_x86_state
	_	XREAD_REG
	_	<i>XREAD_EIP</i>
	_	$XREAD\_EFLAG$
	_	$XREAD\_MEM$
	_	$XWRITE\_REG$
	_	<i>XWRITE_EIP</i>
	_	XWRITE_EFLAG
	_	<i>XWRITE_MEM</i>
	_	<i>XREAD_MEM_BYTES</i>
	_	$type\_abbrev\_M$
	_	$constT\_seq$
	_	$addT\_seq$
	_	$lockT\_seq$
	_	$failure T\_seq$
	_	$seqT\_seq$
	_	$parT\_seq$
	_	$parT\_unit\_seq$
		1
	_	
	_	3 1
	_	$write\_eflag\_seq$
	_	read_eflag_seq
	_	write_eip_seq
	_	$read\_eip\_seq$
	_	$write\_mem\_seq$
	_	read_mem_seq
	_	$read\_m32\_seq$
	_	$write\_m32\_seq$
	_	constT
	_	addT
	_	lockT
	_	failure T
	_	seqT
	_	parT
	_	$parT\_unit$
	_	write_reg
	_	read_reg

FULL CONTENTS	vi

	_	$write\_eip$	23
	_	$read\_eip$	23
	_	write_eflag	23
	_	$read\_eflag$	23
	_	write_m32	23
	_	$read_{-}m32$	23
	_	$option\_apply$	23
		operon_uppeg	20
VII	x86 ev	ent_monad	<b>24</b>
	_	$type\_abbrev\_M$	25
	_	$event\_structure\_empty$	25
	_	$event\_structure\_lock$	25
		event_structure_union	25
		event_structure_bigunion	$\frac{25}{25}$
			$\frac{25}{25}$
	_	event_structure_seq_union	$\frac{25}{25}$
	_	$mapT_{-}ev$	
	_	$choiceT_{-}ev$	25
	_	$constT\_ev$	25
	_	$discardT_{-}ev$	25
	_	$addT_{-}ev$	25
	_	$lockT_{-}ev$	26
	_	$failure T_{-}ev$	26
	_	$seqT_{-}ev$	26
	_	$parT_{-}ev$	26
	_	$parT\_unit\_ev$	26
	_	$write\_location\_ev$	26
	_	$read\_location\_ev$	26
	_	write_reg_ev	27
	_	read_reg_ev	27
	_	$write\_eip\_ev$	27
	_	$read\_eip\_ev$	27
	_	$write\_eflag\_ev$	27
	_	$read\_eflag\_ev$	27
	_	aligned32	27
	_	$write\_m32\_ev$	27
	_		
	_	$read_m32\_ev$	27
	_	constT	28
	_	addT	28
	_	lockT	28
	_	failureT	28
	_	seqT	28
	_	parT	28
	_	$parT\_unit$	28
	_	write_reg	28
	_	$read\_reg$	28
	_	write_eip	28
	_	$read\_eip$	28
	_	write_eflag	28

FULL CONTENTS	vii

		read_eflag
	_	
	_	write_m32
	_	$read_m32 \dots \dots 28$
VIII	$x86_{-0}$	-
	_	$ea\_Xr$
	_	$ea\_Xi$
	_	ea_Xrm_base
	_	ea_Xrm_index
	_	ea_Xrm
	_	ea_Xdest
	_	ea_Xsrc
	_	$ea\_Ximm\_rm$
	_	read_ea
	_	
	_	$read\_dest\_ea$
	_	write_ea
	_	jump_to_ea
	_	call_dest_from_ea 31
	_	$get\_ea\_address$
	_	$bump_eip$
	_	$byte\_parity$
	_	write_PF
	_	write_ZF
	_	write_SF
	_	write_logical_eflags
	_	$write\_arith\_eflags\_except\_CF$
	_	write_arith_eflags
		* 3
	_	, ,
	_	$add\_with\_carry\_out$
	_	$sub\_with\_borrow\_out$
	_	write_arith_result
	_	$write\_arith\_result\_no\_CF$
	_	$write\_arith\_result\_no\_write \dots 32$
	_	$write\_logical\_result$
	_	$write\_logical\_result\_no\_write$
	_	$write\_result\_erase\_eflags$
	_	write_binop
	_	write_monop
	_	read_cond
	_	$x86\_exec\_pop$
	_	x86_exec_push
		x86_exec_popad
	_	1 1
	_	<i>x86_exec_pushad</i>
	_	<i>x86_exec_pop_eip</i>
	_	$x86\_exec\_push\_eip \dots 33$
	_	$x86\_exec\_call \dots $
	_	$x86\_exec\_ret$

TX x86_exec   34	FUL	L CONTEN	TTS	viii
- STR_SPACE_AUX 37 - bytebits 37 - check_opcode 37 - read_SIB . 37 - read_ModRM 37 - is_hex_add 37 - process_hex_add 37 - x86_match_step 37 - x86_moop 38 - x86_moop 38 - x86_moop 38 - b2reg 38 - decode_Xr32 38 - decode_SIB 38 - decode_Xr32 38 - decode_Xrm32 38 - decode_Xrm32 38 - decode_Xronst 39 - decode_Xconst 39 - decode_Xconst 39 - decode_Ximm_rm 39 - x86_select_op 39 - x86_select_op 39 - x86_select_op 39 - x86_select_op 39 - x86_decode_nrefixes 40 - dest_src_args_ok 41 - x86_decode_tytes 41 - rm_args_ok 41 - rx86_drecode_instruction 41  X x86 iiid_dummy 45 - x86_program 44 - DOMAIN 45 - type_abbrev_program_word8 45		- -		
- STR_SPACE_AUX 37 - bytebits 37 - check_opcode 37 - read_SIB . 37 - read_ModRM 37 - is_bex_add 37 - process_hex_add 37 - x86_match_step 37 - x86_monop 38 - x86_monop 38 - b2reg 38 - decode_Xr32 38 - decode_SIB 38 - decode_Xrm32 38 - decode_Xrm32 38 - decode_Xrm32 38 - decode_Xconst 39 - decode_Xconst 39 - decode_Xconst 39 - decode_Ximm_rm 39 - x86_select_op 39 - x86_select_op 39 - x86_select_op 39 - x86_select_op 39 - x86_decode_nrefires 40 - dest_accesses_memory 40 - x86_decode_prefires 41 - dest_src_args_ok 41 - rm_args_ok 41 - rm_args_ok 41 - rm_args_ok 41 - rm_args_ok 41 - x86_decode_instruction 41  X x86 iiid_dummy 45 - x86_program 44 - DOMAIN 45 - type_abbrev_program_word8 45	IX	$x86_{-}dec$	$\operatorname{coder}$	36
- bytebits		_		
- check_opcode		_		37
- read_ModRM		_		37
-       is_hex_add       37         -       process_hex_add       37         -       x86_match_step       37         -       x86_binop       38         -       x86_monop       38         -       b2reg       38         -       decode_Xr32       38         -       decode_SIB       38         -       decode_XIB       38         -       decode_Xconst       39         -       decode_Xconst       39         -       decode_Xconst_or_zero       39         -       decode_Xdest_src       39         -       decode_Ximm_rm       39         -       decode_Ximm_rm       39         -       x86_select_op       40         -       x86_select_op       40         -       x86_decode_se       41         -       x86_decode_se <td></td> <td>_</td> <td>read_SIB</td> <td>37</td>		_	read_SIB	37
- process_hex_add 37 - x86_match_step 37 - x86_monop 38 - x86_monop 38 - x86_monop 38 - b2reg 38 - decode_Xr32 38 - decode_XI32 38 - decode_XI32 38 - decode_Xrm32 38 - decode_Xconst 39 - decode_Xconst 39 - decode_Xconst_or_zero 39 - decode_Xconst_or_zero 39 - x86_select_op 39 - x86_select_op 39 - x86_select_op 39 - x86_syntax 39 - x86_syntax 39 - x86_decode_prefixes 40 - x86_decode_prefixes 40 - x86_decode_prefixes 40 - x86_decode_bytes 41 - rm_args_ok 41 - dest_arcesses_memory 41 - dest_arcesses_ok 41 - mm_args_ok 41 - imm_rm_args_ok 41 - imm_rm_args_ok 41 - x86_derode_instruction 41  X x86 iiid_dummy 43 - x86_execute_some 43 - X86_program 44 - DOMAIN 45 - type_abbrev_program_word8 45		_	$read\_ModRM$	37
- x86_binop 38 - x86_monop 38 - x86_monop 38 - b2reg 38 - decode_Xr32 38 - decode_Xr32 38 - decode_Xrm32 38 - decode_Xrm32 38 - decode_Xrm32 38 - decode_Xconst 39 - decode_Xconst 39 - decode_Xconst 39 - decode_Xconst 39 - decode_Ximm_rm 39 - x86_select_op 39 - x86_syntax 39 - x86_syntax 39 - x86_decode_aux 40 - x86_decode_prefixes 40 - x86_decode_prefixes 40 - x86_decode_bytes 41 - x86_decode 41 - x86_decode 41 - x86_decode 41 - x86_decode 5ytes 41 - imm_rm_arys_ok 41 - x86_derol_formed_instruction 41  X x86 iiid_dummy 43 - x86_well_formed_instruction 41  X x86_program 44 - DOMAIN 45 - type_abbrev_program_word8 45		_	<i>is_hex_add</i>	37
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	process_hex_add	37
-       x86_monop       38         -       b2reg       38         -       decode_Xr32       38         -       decode_SIB       38         -       decode_Xconst       39         -       decode_Xconst       39         -       decode_Xconst_or_zero       39         -       decode_Ximm_rm       39         -       x86_select_op       39         -       x86_select_op       39         -       x86_syntax       39         -       x86_decode_aux       40         -       x86_decode_aux       40         -       x86_decode_prefixes       40         -       x86_decode_brefixes       40         -       x86_decode_bytes       41         -       rm_arys_ok       41         -       rm_arys_ok       41         -       rm_arys_ok       41         -       x86_well_formed_instruction       41 <t< td=""><td></td><td>_</td><td>x86_match_step</td><td>37</td></t<>		_	x86_match_step	37
-       b2reg       38         -       decode_XR32       38         -       decode_SIB       38         -       decode_Xconst       39         -       decode_Xconst       39         -       decode_Xconst_or_zero       39         -       decode_Ximm_rm       39         -       x86_select_op       39         -       x86_select_op       39         -       x86_sec_decode       40         -       x86_decode_aux       40         -       x86_decode_prefixes       40         -       x86_decode_betises       41         -       x86_decode       41         -       x86_decode       41         -       x86_decode       41         -       <		_	x86_binop	38
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	x86_monop	38
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	b2reg	38
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$decode\_Xr32 \dots \dots$	38
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$decode\_SIB$	38
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$decode\_Xrm32$	38
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$decode\_Xconst$	39
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$decode\_Xdest\_src$	39
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$decode\_Xconst\_or\_zero$	39
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$decode\_Ximm\_rm$	39
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$x86\_select\_op \dots \dots$	39
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$X\_SOME$	39
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$x86\_syntax$	39
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$x86\_decode\_aux$	40
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$x86\_decode\_prefixes$	40
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$dest\_accesses\_memory$	40
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	x86_lock_ok	40
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	<i>x86_decode</i>	41
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$x86\_decode\_bytes \dots \dots$	41
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$rm\_args\_ok$	41
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$dest\_src\_args\_ok$	41
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$imm\_rm\_args\_ok$	41
$egin{array}{cccccccccccccccccccccccccccccccccccc$		_		41
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		_	$x86\_well\_formed\_instruction$	41
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	$\mathbf{v}$	Q.C		49
-       x86_execute_some       43         -       X86_NEXT       43         XI       x86_program       44         -       DOMAIN       45         -       type_abbrev_program_word8       45	Λ	X0U_	iiid damma	
-       X86_NEXT       43         XI       x86_program       44         -       DOMAIN       45         -       type_abbrev_program_word8       45		_	· · · · · · · · · · · · · · · · · · ·	
XI x86_program		_		
- DOMAIN		_	A80_NEA1	43
- DOMAIN	ΧI	x86_pro	ogram	44
$- type\_abbrev\_program\_word8 \dots \dots$		-		45
		_		_
		_		

FULL CONTENTS	ix

	_	read_mem_bytes	45
	_	$decode\_program\_fun \dots \dots$	45
	_	$decode\_program\_rel$	45
	_	$type\_abbrev\_run\_skeleton$	45
	_	$run\_skeleton\_wf$	45
	_	$x86\_event\_execute$	45
	_	$x86\_execute\_with\_eip\_check \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots$	45
	_	$event\_structures\_of\_run\_skeleton \dots \dots \dots \dots \dots \dots \dots \dots$	45
	_	$x86\_semantics$	46
	_	$lts\_po\_of\_event\_structure$	46
XII	v86 sc	equential_axiomatic_model	47
<b>711</b>		sequential_execution	48
	_	$so\_to\_vo$	48
	_		48
	_	$so\_to\_write\_serialization$	48
	_		
	_	so_to_rfmap	48
	_	so_to_exec_witness	48
	_	valid_sequential_execution	48
	_	competes	48
	_	race_free	48
	_	restrictE	48
	_	prefixes	49
	_	sequential_race_free	49
XIII	x86_c	drf	<b>50</b>
	_	wfes	51
	_	$maximal\_es$	51
	_	deleteE	51
	_	$extend\_so$	51
	_	$locked\_ready$	51
	_	$ind\_hyp1$	51
	_	sequential_order_thm	51
	_	pre	52
	_	$ind\_concl$	52
	_	$ind\_concl2$	52
	_	$ind\_assum$	52
	_	$data\_race\_free\_thm$	52
XIV	O.C 1.	ah maghina	<b>53</b>
ΛIV	XOU_I	nb_machine	
	_	clause_name	54
	_	funupd	54
	_	funupd2	54
	_	linear_order_extend	54
	_	type_abbrev_message	54
	_	$type\_abbrev\_machine\_state$	54

FULL CONTENTS	x

	_	initial_machine_state	54
	_	$execution\_witness\_of\_machine\_state$	54
	_	$event\_set\_of\_machine\_state$	54
	_	$next\_eiid\_es$	54
	_	tlang_typing	54
	_	final_state	57
XV	x86_lts	ops	<b>58</b>
	_	lts_state	59
	_	eip_tracked_lts	59
	_	$eip\_tracked\_lts\_initial$	59
	_	$lts\_parallel$	59
	_	$traces\_of\_lts$	59
	_	$completed\_traces\_of\_lts$	59
	-	$states\_of\_trace$	59
XVI	x86_h	b_machine_thms	<b>6</b> 0
	_	$machine\_lts$	61
	_	$machine\_execution\_of\_event\_structure \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $	
	_	$final\_states$	61
	_	$hb\_equivalence\_thm1 \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots$	61
	_	partial_view_orders_well_formed	61
	_	$partial\_valid\_execution$	61
	_	$hb\_equivalence\_thm2$	62
	_	$hb\_machine\_progress\_thm$	62
	_	nv_mwninv_progress_uint	02
Index			63

#### Introduction

This document is an automatically typeset version of the HOL definitions described in the paper:

The Semantics of x86 Multiprocessor Machine Code. Susmit Sarkar, Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Tom Ridge, Thomas Braibant, Magnus Myreen, Jade Alglave. In Proc. POPL 2009.

That paper, and the full HOL definitions, are available from http://www.cl.cam.ac.uk/users/pes20/weakmemory. The README from the HOL tarball is reproduced below.

#### README-spec-public

Here are the HOL sources for the x86 multiprocessor semantics.

```
To Build
```

It is known to work with (at least) HOL revision 6031, in the recent PolyML port. To build it, first, install PolyML 5.2 from

http://sourceforge.net/project/showfiles.php?group\_id=148318

with something like the following:

sudo make install

```
wget http://downloads.sourceforge.net/polyml/polyml.5.2.tar.gz?modtime=1214245187&big_mirror=0
tar -zxvf polyml.5.2.tar.gz
cd polyml.5.2
./configure --prefix=/usr
make
```

(you may need a different prefix, and/or to ensure that your PATH contains the install directory/bin and your LD\_LIBRARY\_PATH contains the install directory/lib)

Then install HOL into a directory HOL-poly with something like:

```
svn co https://hol.svn.sf.net/svnroot/hol/HOL HOL-poly
cd HOL-poly
poly < tools-poly/smart-configure.sml
bin/build -expk -symlink</pre>
```

(ignore the mllex error at the end), and ensure that HOL-poly/bin is in your PATH. Add -r6031 after "co" above to get that particular version of HOL.

Finally, just type "make" in this directory (this takes around 15min on an Intel Core2 2.4Ghz machine).

2

```
Contents
 =======
An automatically typeset version of the definitions
 _____
alldoc.ps
alldoc.pdf
         This includes the HOL definitions in the files below, except those
         marked [*]. The HOL proof scripts are not typeset.
The definition of the semantics
x86_coretypesScript.sml
         core type definitions, shared by the sequential and event-based semantics % \left( \frac{1}{2}\right) =\frac{1}{2}\left( \frac{1}{2}\right) +\frac{1}{2}\left( \frac{1}{2}\right) +\frac
x86_typesScript.sml
         types for the event-based semantics: event, event_structure, execution_witness, etc.
x86_astScript.sml
         abstract syntax (AST) for x86 instructions
x86_axiomatic_modelScript.sml
         the x86-cc axiomatic memory model
x86_niceness_statementScript.sml
         the definition of "nice" x86-cc executions
x86_seq_monadScript.sml
         the sequential state state monad type constructor and combinators
x86_event_monadScript.sml
         the event monad type constructor and combinators
x86_opsemScript.sml
         the operational semantics for x86 AST instructions, above one of the monads
x86_decoderScript.sml
         decoding from machine code bytes to x86 AST assembly instructions
x86_Script.sml
         the top level for the sequential semantics
```

x86\_programScript.sml

the top level for the event semantics

The proof that the semantics builds well-formed event structures x86\_event\_opsem\_wfScript.sml [\*] x86\_program\_event\_structure\_wfScript.sml [\*] The proof that the axiomatic model is equivalent to one restricted to nice executions \_\_\_\_\_\_ tactic.sml [\*] x86\_niceness\_proofScript.sml [\*] The data-race-freedom development \_\_\_\_\_ x86\_sequential\_axiomatic\_modelScript.sml the definition of sequential execution used in the data-race-freedom development x86\_axiomatic\_model\_thmsScript.sml [\*] auxiliary theorems about the axiomatic model, used in the data-race-freedom development x86\_drfScript.sml the statements and proof scripts for the main data-race-freedom theorems An abstract machine that corresponds to the axiomatic memory model \_\_\_\_\_\_ x86\_hb\_machineScript.sml the hb machine definition x86\_lts\_opsScript.sml auxiliary definitions for operations over labelled transition systems x86\_hb\_machine\_thmsScript.sml statements of theorems about the hb machine, and hand proofs thereof Other auxiliary files [\*] bit\_listScript.sml [\*] decoderScript.sml [\*] opmonScript.sml utilScript.sml [\*] utilLib.sig [\*]

utilLib.sml	[*]
HolDoc.sml	[*]
HolDoc.sig	[*]

README-spec-public this file

LICENSE-spec-public the BSD-style license

# $\begin{array}{c} {\rm Part~I} \\ {\rm x86\_coretypes} \end{array}$

iiid 6

```
type_abbrev Ximm : word32

Xreg = EAX | EBX | ECX | EDX | ESP | EBP | ESI | EDI

Xeflags = X_CF | X_PF | X_AF | X_ZF | X_SF | X_OF

Xea =
    XEA_I of Ximm(* constant *)
    | XEA_R of Xreg(* register name *)
    | XEA_M of word32(* memory address *)

type_abbrev proc : num

type_abbrev program_order_index : num
```

 $iiid = \{\!\!\{ \text{proc}: \text{proc};$ 

program\_order\_index : num]

 $\begin{array}{c} {\rm Part~II} \\ {\rm x86\_types} \end{array}$ 

```
type_abbrev address : Ximm
type_abbrev value: Ximm
type_abbrev eiid : num
type\_abbrev reln : 'a\#'a \rightarrow bool
\mathsf{reg} = Reg32 \ \mathbf{of} \ \mathsf{Xreg} \mid Reg1 \ \mathbf{of} \ \mathsf{Xeflags} \mid RegEIP
\mathsf{dirn} = R \mid W
location = Location_{REG} of proc reg
            LOCATION_MEM of address
barrier = Leence | Seence | Meence
action = Access of dirn location value
event = \langle | eiid : eiid |
           iiid:iiid;
           action : action
\mathsf{event\_structure} = \langle\!\!\langle \ \mathit{procs} : \mathsf{proc} \ \mathsf{set}; \\
                       events: event set;
                       intra\_causality: {\tt event\ reln};
                       atomicity : event set set
type\_abbrev state_constraint : location \rightarrow value option
type\_abbrev view_orders : proc \rightarrow event reln
execution\_witness =
⟨ initial_state : state_constraint;
  vo: view\_orders;
  write\_serialization: event reln;
  lock_serialization : event reln;
  rfmap : event reln
type_abbrev eiid_state : eiid set
```

LTS

```
(\text{next\_eiid}: \texttt{eiid\_state} \rightarrow (\texttt{eiid\#eiid\_state}) \\ \texttt{set}) \\ eiids = \{(\texttt{eiid}, eiids \ \cup \{\texttt{eiid}\}) \ | \ \\ \texttt{eiid} \ | \ \\ \neg(\texttt{eiid} \ \in \ eiids)\}
(initial\_eiid\_state : eiid\_state) = \{\}
machine\_visible\_label =
\{ mvl\_event : event; 
   mvl\_iico: event set;
   mvl\_first\_of\_instruction : bool;
   mvl\_last\_of\_instruction: bool;
   mvl\_locked : bool
lts\_monad\_visible\_label =
\{ lmvl\_iiid : iiid; 
   lmvl\_action: action}
\mathsf{label} = \mathsf{Vis} \ \mathbf{of} \ 'a \mid \mathsf{TAU}
\mathbf{type\_abbrev} \ \mathsf{machine\_label} : \mathsf{machine\_visible\_label} \ \mathsf{label}
\mathbf{type\_abbrev}\ \mathsf{lts\_monad\_label}: \mathsf{lts\_monad\_visible\_label}\ \mathsf{label}
\label{eq:loss_loss} \begin{split} \mathsf{LTS} = & \{ \textit{states}: 'state \; \mathsf{set}; \\ \textit{initial}: 'state; \\ & . \end{split}
             final: ('state\#'value) set;
             trans: ('state\#('visible label)\#'state)set)
```

Part III

 $x86\_ast$ 

Xinst 11

```
Xrm = XR \text{ of } Xreg(* register *)
| XM of (word2#Xreg) option Xreg option Ximm(* mem[2^{scale}] * index + base + displacement] *)
(\text{rm\_is\_memory\_access}(\text{XM } i \ b \ d) = \mathbf{T}) \land
(rm\_is\_memory\_access(XR r) = F)
Xdest_src = XRM_I of Xrm Ximm(* mnemonic r/m32, imm32 or mnemonic r/m32, imm8 (sign-extended) *)
         XRM_R of Xrm Xreg(* mnemonic r/m32, r32 *)
        | XR_RM of Xreg Xrm(* mnemonic r32, r/m32 *)
Ximm_rm = X_{LRM} \text{ of } Xrm(*r/m32 *)
     XI of Ximm(* imm32 or imm8 (sign-extended) *)
Xbinop\_name = XADD \mid XAND \mid XCMP \mid XOR \mid XSHL \mid XSHR \mid XSAR \mid XSUB \mid XTEST \mid XXOR
\mathsf{Xmonop\_name} = \mathrm{XDEC} \mid \mathrm{XINC} \mid \mathrm{XNOT} \mid \mathrm{XNEG}
\mathsf{Xcond} = X\_ALWAYS \mid X\_E \mid X\_NE
Xinstruction = XBINOP of Xbinop\_name Xdest\_src
             | XMONOP of Xmonop_name Xrm
              XCMPXCHG of Xrm Xreg
              XXADD of Xrm Xreg
              XXCHG of Xrm Xreg
              \rm XLEA \ of \ Xdest\_src
             XPOP of Xrm
              X_{PUSH} of X_{imm_rm}
              XCALL of Ximm_rm
              XRET of Ximm
              XMOV of Xcond Xdest_src
              XJUMP of Xcond Ximm
              XLOOP of Xcond Ximm(* Here Xcond over approximates possibilities *)
              XPUSHAD
             | XPOPAD
Xpre\_g1 = XLOCK \mid XG1\_NONE
Xpre\_g2 = XBRANCH\_TAKEN \mid XBRANCH\_NOT\_TAKEN \mid XG2\_NONE
Xinst = XPREFIX of Xpre\_g1 Xpre\_g2 Xinstruction
```

## $\begin{array}{c} {\rm Part~IV} \\ {\rm x86\_axiomatic\_model} \end{array}$

 $po\_strict$  13

```
loc e =
case e.action of
   Access d l v \rightarrow \text{Some } l
 \parallel \_ \rightarrow \text{None}
value_of e =
case e.action of
   Access d\ l\ v \to {
m Some}\ v
 \parallel \_ \rightarrow None
\operatorname{proc} e = e.iiid.\operatorname{proc}
mem_load e =
case e.action of
   ACCESS R(LOCATION_MEM a)v \to \mathbf{T}
\parallel \_ \rightarrow \mathbf{F}
mem_store e =
case e.action of
   Access W(Location_mem a)v \rightarrow \mathbf{T}
\parallel _ 
ightarrow {f F}
mem_barrier e =
case e.action of
(* Barrier bar -> T \parallel *)
_{-}\rightarrow \mathbf{F}
iiids E = \{e.iiid \mid e \in E.events\}
writes E = \{e \mid e \in E.events \land \exists l \ v.e.action = Access W \ l \ v\}
reads E = \{e \mid e \in E.events \land \exists l \ v.e.action = Access R \ l \ v\}
locked E e = (e \in \mathbf{bigunion} \ E.atomicity)
po E =
\{(e_1, e_2) \mid (e_1.iiid.\mathsf{proc} = e_2.iiid.\mathsf{proc}) \land 
                e_1.iiid.\mathsf{program\_order\_index} \leq e_2.iiid.\mathsf{program\_order\_index} \wedge
                e_1 \in E.events \land e_2 \in E.events
po_strict E =
\{(e_1, e_2) \mid (e_1.iiid.\mathsf{proc} = e_2.iiid.\mathsf{proc}) \land
                e_1.iiid.program_order_index < e_2.iiid.program_order_index \land
                e_1 \in E.events \land e_2 \in E.events
```

viewed\_events E p =

 $\{e \mid e \in E.events \land ((proc \ e = p) \lor mem\_store \ e)\}$ 

po\_iico  $E = \text{po\_strict } E \cup E.intra\_causality$ well\_formed\_event\_structure E = $(\forall iiid. \mathbf{finite} \{ \mathsf{eiid} \mid \exists e \in (E.events). (e.iiid = iiid) \land (e.\mathsf{eiid} = \mathsf{eiid}) \}) \land$ (finite E.procs)  $\land$  $(\forall e \in (E.events).proc \ e \in E.procs) \land$  $(\forall e_1 \ e_2 \in (E.events).(e_1.eiid = e_2.eiid) \land (e_1.iiid = e_2.iiid) \implies (e_1 = e_2)) \land$  $(DOM\ E.intra\_causality) \subseteq E.events \land$ (range  $E.intra\_causality$ )  $\subseteq E.events \land$  $acyclic(E.intra\_causality) \land$  $(\forall (e_1, e_2) \in (E.intra\_causality).(e_1.iiid = e_2.iiid)) \land$  $(\forall (e_1, e_2) \in (E.intra\_causality). \neg (mem\_store e_1)) \land$  $(\forall (e_1 \in \text{writes } E)e_2.$  $\neg(e_1=e_2) \land$  $(e_2 \in \text{writes } E \vee e_2 \in \text{reads } E) \wedge$  $(e_1.iiid = e_2.iiid) \wedge$  $(loc e_1 = loc e_2) \land$  $(\exists p \ r. loc \ e_1 = Some (Location_reg \ p \ r))$  $(e_1, e_2) \in E.intra\_causality^+ \lor$  $(e_2, e_1) \in E.intra\_causality^+) \land$  $(\forall es \in (E.atomicity). \exists e \in es. mem\_load \ e) \land$  $PER\ E.events\ E.atomicity\ \land$  $(\forall es \in (E.atomicity). \forall e_1 \ e_2 \in es.(e_1.iiid = e_2.iiid)) \land$  $(\forall es \in (E.atomicity). \forall e_1 \in es. \forall e_2 \in (E.events). (e_1.iiid = e_2.iiid) \implies e_2 \in es) \land (e_1.iiid = e_2.iiid) \implies e_3 \in es) \land (e_1.iiid = e_3.iiid) \implies e_3 \in es) \land (e_1.iiid = e_3.iiid) \implies e_3 \in es) \land (e_3.iiid = e_3.iiid) \implies (e_3.iiid = e_3.iiid) \implies (e_3.iid = e_3.iid = e_3.ii$  $(\forall e \in (E.events). \forall p \ r. (loc \ e = Some (Location\_reg \ p \ r)) \implies (p = proc \ e))$ sub\_event\_structure E'E = $(E'.procs = E.procs) \land$  $E'.events \subseteq E.events \land$  $(E'.atomicity = PER\_RESTRICT\ E.atomicity\ E'.events) \land$  $(E'.intra\_causality = E.intra\_causality|_{E'\ events})$ preserved\_program\_order E = $\{(e_1, e_2) \mid (e_1, e_2) \in (\text{po\_strict } E) \land$  $((\exists p \ r.(loc \ e_1 = loc \ e_2) \land$ (loc  $e_1 = \text{Some (Location\_reg } p \ r))) \lor$  $(\text{mem\_load } e_1 \land \text{mem\_load } e_2) \lor$  $(\text{mem\_store } e_1 \land \text{mem\_store } e_2) \lor$ (mem\_load  $e_1 \land \text{mem\_store } e_2$ )  $\lor$ (mem\_store  $e_1 \land \text{mem\_load} \ e_2 \land (\text{loc} \ e_1 = \text{loc} \ e_2)) \lor$  $((\text{mem\_load } e_1 \vee \text{mem\_store } e_1) \wedge \text{locked } E e_2) \vee$ (locked  $E e_1 \land (\text{mem\_load } e_2 \lor \text{mem\_store } e_2)))$ }

check\_causality 15

```
view\_orders\_well\_formed E vo =
(\forall p \in (E.procs). linear\_order(vo p)(viewed\_events E p) \land
   \forall e \in (\text{viewed\_events } E \ p). \mathbf{finite} \{ e' \mid (e', e) \in (vo \ p) \}) \land
(\forall p. \neg (p \in E.procs) \implies (vo p = \{\}))
get_l_stores E l =
\{e \mid e \in E.events \land mem\_store \ e \land (loc \ e = Some \ l)\}
write_serialization_candidates_old E =
let per\_location\_store\_sets = \{es \mid \exists l.es = \text{get\_Lstores} \ E \ l\} in
let per\_location\_store\_set\_linearisations = \{strict\_linearisations \ es \mid es \in per\_location\_store\_sets\} in
\mathbf{let} \ \ choices = all\_choices \ per\_location\_store\_set\_linearisations \ \ \mathbf{in}
{bigunion lin \mid lin \in choices}
write\_serialization\_candidates \ E \ cand =
(\forall (e_1, e_2) \in cand.
   \exists l.e_1 \in (\text{get\_Lstores } E \ l) \land e_2 \in (\text{get\_Lstores } E \ l)) \land e_2 \in (\text{get\_Lstores } E \ l)
(\forall l. \text{ strict\_linear\_order}( cand |_{(\text{get\_l\_stores } E \ l)})
                               (\text{get\_l\_stores } E \ l))
lock_serialization_candidates E=
   let lin_{-}ec = strict_{-}linearisations E.atomicity in
      \{\{(e_1, e_2) \mid \exists (es_1, es_2) \in lin.e_1 \in es_1 \land e_2 \in es_2\}
       | lin \in lin\_ec \}
reads_from_map_candidates_old E =
{\bf let} \ \ reads\_and\_their\_possible\_writes =
      \{(er, ews) \mid er \in E.events \land
                            \exists l \ v.(er.action = Access R \ l \ v) \land
                                     (ews = \{ew \mid ew \in E.events \land (ew.action = Access W \ l \ v)\})\} in
\{\mathit{rfmap} \mid (\mathbf{range} \ \mathit{rfmap}) \subseteq (\mathit{DOM} \ \mathit{reads\_and\_their\_possible\_writes}) \ \land \\
               \forall (ew, er) \in rfmap. \exists ews. (er, ews) \in reads\_and\_their\_possible\_writes \land ew \in ews \}
reads_from_map_candidates E rfmap =
   \forall (ew, er) \in rfmap.er \in E.events \land ew \in E.events \land
         \exists l \ v.(er.action = Access R \ l \ v) \land
               (ew.action = Access W l v)
happens_before E X =
   E.intra\_causality \cup
   (preserved\_program\_order E) \cup
   X.write\_serialization \cup
   X.lock\_serialization \cup
   X.rfmap
```

check\_final\_mem 16

```
check\_causality E vo happens before =
  \forall p \in (E.procs). \operatorname{acyclic}((\operatorname{strict}(vo\ p)) \cup happensbefore)
check_rfmap_written E vo rfmap =
\forall p \in (E.procs).
  \forall (\mathit{ew}, \mathit{er}) \in (\mathit{rfmap}|_{(\mathit{viewed\_events}\ E\ p)}).
      \forall ew' \in (\text{writes } E).
         \neg(ew = ew') \land (ew, ew') \in (vo\ p) \land (ew', er) \in (vo\ p)
          \implies \neg (\text{loc } ew = \text{loc } ew')
check_rfmap_initial E \ vo \ rfmap \ initial_state =
\forall p \in (E.procs).
  \forall er \in (((reads E) \setminus (range rfmap)))
            \cap viewed_events E p).
      \exists l \ v.(er.action = Access R \ l \ v) \land
               (initial\_state \ l = Some \ v) \land
               \forall ew' \in \text{writes } E.
                    (ew', er) \in (vo\ p) \implies \neg(loc\ ew' = loc\ er)
state_updates E vo write\_serialization l =
case l of
  Location_mem a \rightarrow
      \{value\_of\ ew\ |\ ew\ \in\ maximal\_elements(get\_Lstores\ E\ l)write\_serialization\}
\parallel Location_reg p r \rightarrow
      \{value\_of\ ew\ |\ ew\ \in\ maximal\_elements(get\_l\_stores\ E\ l)(vo\ p)\}
{\it check\_final}\ E\ vo\ initial\_state\ final\_state\_opt\ write\_serialization =
if finite E.events then
\exists final_state.(final_state_opt = Some final_state) \land
\forall l.if (state\_updates \ E \ vo \ write\_serialization \ l) = \{\} \ then
         final_state l = initial\_state l
      else
         (\text{final\_state } l) \in (\text{state\_updates } E \text{ vo write\_serialization } l)
else
final\_state\_opt = None
state\_updates\_mem\ E\ write\_serialization\ a =
\{\text{value\_of } ew \mid ew \in maximal\_elements(get\_l\_stores } E(\text{LOCATION\_MEM } a)\} write\_serialization\}
(check_final_mem E initial_state write_serialization None =
\neg(finite E.events)) \land
(check\_final\_mem\ E\ initial\_state\ write\_serialization(Some\ final\_state) =
finite E.events \land
\forall a.if (state\_updates\_mem \ E \ write\_serialization \ a) = \{\} \ then
         final_state a = initial\_state(Location\_mem a)
      else
         (\text{final\_state } a) \in (\text{state\_updates\_mem } E \text{ } write\_serialization } a))
```

restrict\_execution\_witness 17

```
check_atomicity E vo =
  \forall p \in (E.procs). \forall es \in (E.atomicity).
     \forall e_1 \ e_2 \in es.(e_1, e_2) \in (vo \ p) \implies
       \forall e.(e_1, e) \in (vo \ p) \land (e, e_2) \in (vo \ p) \implies e \in es
valid_execution E X =
   view_orders_well_formed E X.vo \land
  X.write\_serialization \in write\_serialization\_candidates E \land
  X.lock\_serialization \in lock\_serialization\_candidates E \land
  X.rfmap \in \text{reads\_from\_map\_candidates } E \land
  check_causality E X.vo(\text{happens\_before } E X) \land
   check_rfmap_written E X.vo X.rfmap \land
   check_rfmap_initial E X.vo X.rfmap X.initial\_state \land
   check_atomicity E X.vo
restrict_execution_witness X E =
(*final\_state\_opt := ... *)
  \begin{split} &vo := &(\lambda p. \ (X.vo \ p)|_{E.events}); \\ &write\_serialization := & X.write\_serialization|_{E.events} \,; \end{split}
  lock\_serialization := X.lock\_serialization|_{E.events};
  rfmap := RRESTRICTX.rfmap \ E.events
```

## ${\bf Part~V}$ ${\bf x86\_niceness\_statement}$

 $niceness\_thm$  19

```
\begin{split} &\text{nice\_execution} \ E \ X = \forall p \in (E.procs). \\ &(\text{po\_strict} \ E)|_{(\text{viewed\_events} \ E \ p \ \backslash \text{mem\_store})} \subseteq (X.vo \ p) \\ &\text{niceness\_thm} = \\ &\forall E \ X. (\text{well\_formed\_event\_structure} \ E \ \land \\ & \text{valid\_execution} \ E \ X) \implies \\ &\exists X'. \text{valid\_execution} \ E \ X' \land \text{nice\_execution} \ E \ X' \land \\ & (X' \ \{ vo := (\lambda p.\{\}) \}) = X \ \{ vo := (\lambda p.\{\}) \} ) \end{split}
```

# $\begin{array}{c} {\rm Part\ VI} \\ {\rm \bf x86\_seq\_monad} \end{array}$

 $parT\_seq$  21

```
type\_abbrev \times 86\_state : (Xreg \rightarrow word32) \#(* - general-purpose 32-bit registers *)
(word32)\#(* - eip *)
(Xeflags \rightarrow bool option) \#(* - eflags *)
(word32 → word8 option)(* - unsegmented memory *)
XREAD_REG i((r, eip, f, m) : x86\_state) = r i
XREAD\_EIP((r, eip, f, m) : x86\_state) = eip
XREAD_EFLAG i((r, eip, f, m) : x86\_state) = f i
XREAD_MEM i((r, eip, f, m) : x86\_state) = m i
XWRITE_REG i \ x((r, eip, f, m) : x86\_state) = ((i = +x)r, eip, f, m) : x86\_state
XWRITE_EIP x((r, eip, f, m) : x86\_state) = (r, x, f, m) : x86\_state
XWRITE_EFLAG i \ x((r, eip, f, m) : x86\_state) = (r, eip, (i = +x)f, m) : x86\_state
XWRITE_MEM i \ x((r, eip, f, m) : x86\_state) = (r, eip, f, (i = +x)m) : x86\_state
XREAD\_MEM\_BYTES \ n \ a \ s =
if n = 0 then [] else XREAD_MEM a \in XREAD\_MEM\_BYTES(n - 1)(a + 1w)s
type\_abbrev M : x86\_state \rightarrow ('a\#x86\_state) option
(\text{constT\_seq} : 'a \rightarrow 'a \text{ M})x = \lambda y.\text{Some } (x, y)
(addT\_seq : 'a \rightarrow 'b M \rightarrow ('a\#'b)M)x s =
\lambda y.\mathbf{case} \ s \ y \ \mathbf{of} \ \mathrm{None} \to \mathrm{None} \parallel \mathrm{Some} \ (z,t) \to \mathrm{Some} \ ((x,z),t)
(lockT\_seq : 'a M \rightarrow 'a M)s = s
(failureT\_seq : 'a M) = \lambda y.None
(\text{seqT\_seq} : 'a \ \mathsf{M} \rightarrow ('a \rightarrow 'b \ \mathsf{M}) \rightarrow 'b \ \mathsf{M})s \ f =
\lambda y.\mathbf{case} \ s \ y \ \mathbf{of} \ \mathrm{None} \to \mathrm{None} \parallel \mathrm{Some} \ (z,t) \to f \ z \ t
(parT\_seq : 'a M \rightarrow 'b M \rightarrow ('a\#'b)M)s t =
\lambda y.\mathbf{case} \ s \ y \ \mathbf{of} \ \mathrm{None} \to \mathrm{None} \parallel \mathrm{Some} \ (a,z) \to
   case t \ z \ \text{of} \ \text{None} \to \text{None} \parallel \text{Some} \ (b, x) \to \text{Some} \ ((a, b), x)
```

lockT

```
(parT\_unit\_seq : unit M \rightarrow unit M \rightarrow unit M)s t =
\lambda y.\mathbf{case} \ s \ y \ \mathbf{of} \ \mathrm{None} \to \mathrm{None} \parallel \mathrm{Some} \ (a,z) \to
  case t \ z of None \rightarrow None \parallel Some (b, x) \rightarrow Some ((), x)
(write_reg_seq ii \ r \ x) : unit M =
\lambda s.\text{Some}((), \text{XWRITE\_REG} \ r \ x \ s)
(read\_reg\_seq ii r) : Ximm M =
\lambda s.Some (XREAD_REG r s, s)
(write_eflag_seq ii f x): unit M =
(\lambda s.Some ((), XWRITE\_EFLAG f x s))
(read\_eflag\_seq ii f) : bool M =
(\lambda s.\mathbf{case} XREAD_EFLAG f s of None \rightarrow None \parallel Some b \rightarrow Some (b,s))
(write_eip_seq ii x) : unit M =
\lambda s.\text{Some}((), \text{XWRITE\_EIP} \ x \ s)
(read\_eip\_seq\ ii): Ximm\ M =
\lambda s. \text{Some (XREAD\_EIP } s, s)
(write_mem_seq ii \ a \ x): unit M =
(\lambda s.\mathbf{case} \ \mathrm{XREAD\_MEM} \ a \ s \ \mathbf{of} \ \mathrm{None} \to \mathrm{None} \parallel \mathrm{Some} \ y \to \mathrm{Some} \ ((), \mathrm{XWRITE\_MEM} \ a(\mathrm{Some} \ x)s))
(read\_mem\_seq ii a) : word8 M =
(\lambda s.\mathbf{case} XREAD_MEM a \ s \ \mathbf{of} None \rightarrow None \parallel Some x \rightarrow Some (x, s))
(read_m32\_seq ii a) : Ximm M =
\operatorname{seqT\_seq}(\operatorname{parT\_seq}(\operatorname{read\_mem\_seq}\ ii(a+0w))(\operatorname{parT\_seq}(\operatorname{read\_mem\_seq}\ ii(a+1w))
         (parT\_seq(read\_mem\_seq\ ii(a+2w))(read\_mem\_seq\ ii(a+3w))))
  (\lambda(x0, x1, x2, x3). constT_seq(bytes2word[x0; x1; x2; x3]))
(write_m32\_seq ii \ a \ w) : unit M =
(let bs = word2bytes 4 w in
   parT_unit_seq(write_mem_seq\ ii(a+0w)(EL\ 0\ bs))(parT_unit_seq(write_mem_seq\ ii(a+1w)(EL\ 1\ bs))
(parT\_unit\_seq(write\_mem\_seq\ ii(a+2w)(EL\ 2\ bs))(write\_mem\_seq\ ii(a+3w)(EL\ 3\ bs)))))
(\text{constT}: 'a \rightarrow 'a \text{ M}) = \text{constT\_seq}
(addT : 'a \rightarrow 'b M \rightarrow ('a\#'b)M) = addT\_seq
```

option\_apply 23

```
(\mathrm{lock}\mathrm{T}:\mathsf{unit}\;\mathsf{M}\to\mathsf{unit}\;\mathsf{M})=\mathrm{lock}\mathrm{T}\_\mathrm{seq}
```

 $(failureT : unit M) = failureT\_seq$ 

$$(\operatorname{seq}T : 'a \ \mathsf{M} \to (('a \to 'b \ \mathsf{M}) \to 'b \ \mathsf{M})) = \operatorname{seq}T \operatorname{\_seq}$$

$$(parT : 'a M \rightarrow 'b M \rightarrow ('a\#'b)M) = parT\_seq$$

 $(parT\_unit : unit M \rightarrow unit M \rightarrow unit M) = parT\_unit\_seq$ 

 $(\text{write\_reg}: \mathit{iiid} \rightarrow \mathsf{Xreg} \rightarrow \mathsf{Ximm} \rightarrow \mathsf{unit}\;\mathsf{M}) = \mathsf{write\_reg\_seq}$ 

 $(\operatorname{read\_reg}:\mathit{iiid} \to \mathsf{Xreg} \to \mathsf{Ximm}\;\mathsf{M}) = \operatorname{read\_reg\_seq}$ 

 $(write\_eip : iiid \rightarrow Ximm \rightarrow unit M) = write\_eip\_seq$ 

 $(read\_eip : iiid \rightarrow Ximm M) = read\_eip\_seq$ 

 $(write\_eflag : iiid \rightarrow Xeflags \rightarrow bool option \rightarrow unit M) = write\_eflag\_seq$ 

 $(read\_eflag : iiid \rightarrow Xeflags \rightarrow bool M) = read\_eflag\_seq$ 

 $(write\_m32 : iiid \rightarrow Ximm \rightarrow Ximm \rightarrow unit M) = write\_m32\_seq$ 

 $(read\_m32 : iiid \rightarrow Ximm \rightarrow Ximm M) = read\_m32\_seq$ 

option\_apply  $x f = \mathbf{if} \ x = \text{None then None else } f(\mathbf{the} \ x)$ 

# $\begin{array}{c} {\rm Part\ VII} \\ {\rm \bf x86\_event\_monad} \end{array}$

 $addT_{-}ev$  25

```
type\_abbrev M : eiid\_state \rightarrow ((eiid\_state \#'a\#event\_structure)set)
event\_structure\_empty = \langle procs := \{ \}; events := \{ \}; intra\_causality := \{ \}; atomicity := \{ \} \rangle
event\_structure\_lock\ es = \{\ procs := es.procs; events := es.events; intra\_causality := es.intra\_causality; atomicity := \mathbf{if}\}
event_structure_union es_1 \ es_2 =
            \{procs := es_1.procs \cup es_2.procs;\}
               events := es_1.events \cup es_2.events;
               intra\_causality := es_1.intra\_causality \cup es_2.intra\_causality;
               atomicity := es_1.atomicity \cup es_2.atomicity
event_structure_bigunion(ess : event_structure set) =
              \langle\!\!\langle procs := \mathbf{bigunion} \{ es.procs \mid es \in ess \}; 
               events := \mathbf{bigunion} \{es.events \mid es \in ess\};
               intra\_causality := \mathbf{bigunion} \{es.intra\_causality \mid es \in ess\};
               atomicity := \mathbf{bigunion} \{ es. atomicity \mid es \in ess \} \}
event_structure_seq_union es_1 \ es_2 =
            \{procs := es_1.procs \cup es_2.procs;\}
               events := es_1.events \cup es_2.events;
               intra\_causality := es_1.intra\_causality
                                           \cup es<sub>2</sub>.intra_causality
                                            \cup \{(e_1, e_2)\}
                                                     |e_1| \in (maximal\_elements\ es_1.events\ es_1.intra\_causality)
                                                        \land e_2 \in (minimal\_elements\ es_2.events\ es_2.intra\_causality)\};
               atomicity := es_1.atomicity \cup es_2.atomicity
(\text{mapT-ev}: ('a \rightarrow 'b) \rightarrow 'a \ \mathsf{M} \rightarrow 'b \ \mathsf{M})f \ s =
\lambda eiid\_next : eiid_state.
   \mathbf{let} \ \ t = s \ eiid\_next \ \ \mathbf{in}
      \{(eiid\_next', f \ x, es)\}
                             |(eiid\_next', x, es) \in t\}
(\text{choiceT_ev} : 'a \ \mathsf{M} \to 'a \ \mathsf{M} \to 'a \ \mathsf{M}) s \ s' =
\lambda eiid\_next: eiid_state.s eiid\_next \cup s' eiid\_next
(\text{constT_ev}: 'a \rightarrow 'a \text{ M})x = \lambda eiid\_next.\{(eiid\_next, x, \text{event\_structure\_empty})\}
(\operatorname{discardT_ev}: 'a \ \mathsf{M} \to \mathsf{unit} \ \mathsf{M})s =
\lambda eiid\_next.let (t:(eiid\_state\#'a\#event\_structure)set) = s \ eiid\_next \ in
   image(\lambda(eiid\_next', v, es).(eiid\_next', (), es))t
```

 $read\_location\_ev$  26

```
(addT_ev : 'a \rightarrow 'b M \rightarrow ('a\#'b)M)x s =
\lambda eiid\_next.let (t:(eiid\_state\#'b\#event\_structure)set) = s \ eiid\_next in
   image(\lambda(eiid\_next', v, es).(eiid\_next', (x, v), es))t
(lockT_ev : 'a M \rightarrow 'a M)s =
\lambda eiid\_next.let (t:(eiid\_state\#'a\#event\_structure)set) = s \ eiid\_next \ in
   image(\lambda(eiid\_next', v, es).(eiid\_next', v, event\_structure\_lock\ es))t
(failureT_ev : 'a M) = \lambda eiid_next. \{ \}
(\text{seqT\_ev} : 'a \ \mathsf{M} \rightarrow ('a \rightarrow 'b \ \mathsf{M}) \rightarrow 'b \ \mathsf{M})s \ f =
\lambda eiid\_next : eiid_state.
let t = s \ eiid\_next in
   bigunion{let t' = f \ x \ eiid\_next' in
                            \{(eiid\_next'', x', event\_structure\_seq\_union \ es \ es')
                             |(eiid\_next'', x', es') \in t'\}
                   |(eiid\_next', x, es) \in t\}
(parT_ev : 'a M \rightarrow 'b M \rightarrow ('a\#'b)M)s s' =
\lambda eiid\_next : eiid_state.
let t = s \ eiid\_next in
   bigunion{let t' = s' \ eiid\_next' in
                             \{(eiid\_next'', (x, x'), event\_structure\_union \ es \ es') \ | \ (eiid\_next'', x', es') \in \ t'\} 
                   |(eiid\_next', x, es) \in t\}
(parT\_unit\_ev : unit M \rightarrow unit M \rightarrow unit M)s s' =
\lambda eiid\_next : eiid_state.
let t = s \ eiid\_next in
   bigunion{let t' = s' \ eiid\_next' in
                            \{(\mathit{eiid\_next''}, (), \mathit{event\_structure\_union}\ \mathit{es}\ \mathit{es'})
                             |(eiid\_next'', (), es') \in t'\}
                   |(eiid\_next', (), es) \in t\}
(write_location_ev ii l x): unit M =
\lambda eiid\_next.\{(eiid\_next',
                  (),
                  \{procs := \{ii.proc\};
                     events := \{ \langle eiid := eiid'; \rangle \}
                                     iiid := ii;
                                     action := Access W | l | x \rangle;
                     intra\_causality := \{\};
                     atomicity := \{\}\}) | (eiid', eiid\_next') \in next\_eiid \ eiid\_next\}
```

 $read\_m32\_ev$  27

```
(read\_location\_ev \ ii \ l) : value M =
\lambda eiid\_next.\{(eiid\_next',
               \{procs := \{ii.proc\};
                 events := \{ \langle eiid := eiid'; \rangle \}
                              iiid := ii;
                              action := Access R l x \};
                 intra\_causality := \{\};
                 atomicity := \{\}\}
          |x \in UNIV \land (eiid', eiid\_next') \in next\_eiid \ eiid\_next\}
(write_reg_ev ii r x): unit M =
write_location_ev ii(Location_{REG} ii.proc(Reg32 r))x
(read\_reg\_ev \ ii \ r) : value M =
read_location_ev ii(Location_{REG} ii.proc(Reg32 r))
(write_eip_ev ii x): unit M =
write_location_ev ii(Location_reg ii.proc RegEIP)x
(read\_eip\_ev ii) : value M =
read_location_ev ii(Location_reg ii.proc RegEIP)
(write\_eflag\_ev \ ii \ f \ bo) : unit M =
case bo of
Some b \rightarrow
(write_location_ev ii(Location\_reg\ ii.proc(Reg1\ f))(if\ b\ then\ 1w\ else\ 0w))
\parallel None \rightarrow
  choiceT\_ev
     (write_location_ev ii(Location_{REG} ii.proc(Reg1 f))0w)
     (write_location_ev ii(Location_{REG} ii.proc(Reg1 f))1w)
(\text{read\_eflag\_ev } ii f) : \text{bool M} =
mapT_ev(\lambda x.(x = 0w))
(read\_location\_ev ii(Location\_reg ii.proc(Reg1 f)))
aligned 32 a = ((a \&\& 3w) = 0w)
(write_m32_ev ii \ a \ x) : unit M =
if aligned 32 \ a then
write_location_ev ii(Location\_mem a)x
else
failure T\_ev
```

 $read_m32$  28

```
(read_m32_ev ii a) : Ximm M =
if aligned 32 \ a then
read_location_ev ii(Location_mem a)
else
failure T\_ev
(\text{constT}: 'a \rightarrow 'a \text{ M}) = \text{constT\_ev}
(addT : 'a \rightarrow 'b M \rightarrow ('a\#'b)M) = addT_ev
(\mathrm{lock} T : \mathsf{unit}\ \mathsf{M} \to \mathsf{unit}\ \mathsf{M}) = \mathrm{lock} T_- \mathrm{ev}
(failureT : unit M) = failureT_ev
(\operatorname{seqT}: 'a \ \mathsf{M} \to (('a \to 'b \ \mathsf{M}) \to 'b \ \mathsf{M})) = \operatorname{seqT\_ev}
(parT : 'a M \rightarrow 'b M \rightarrow ('a\#'b)M) = parT_ev
(parT\_unit : unit M \rightarrow unit M \rightarrow unit M) = parT\_unit\_ev
(write\_reg : iiid \rightarrow Xreg \rightarrow Ximm \rightarrow unit M) = write\_reg\_ev
(\text{read\_reg} : iiid \rightarrow \mathsf{Xreg} \rightarrow \mathsf{Ximm} \ \mathsf{M}) = \text{read\_reg\_ev}
(write\_eip : iiid \rightarrow Ximm \rightarrow unit M) = write\_eip\_ev
(read\_eip : iiid \rightarrow Ximm M) = read\_eip\_ev
(write\_eflag : iiid \rightarrow Xeflags \rightarrow bool option \rightarrow unit M) = write\_eflag\_ev
(read\_eflag : iiid \rightarrow Xeflags \rightarrow bool M) = read\_eflag\_ev
(write\_m32 : iiid \rightarrow Ximm \rightarrow Ximm \rightarrow unit M) = write\_m32\_ev
```

 $(read_m32 : iiid \rightarrow Ximm \rightarrow Ximm M) = read_m32_ev$ 

## Part VIII x86\_opsem

 $jump\_to\_ea$  30

```
ea_Xr(r : Xreg) = constT(XEA_R r)
ea_Xi(i : Ximm) = constT(XEA_I i)
(ea_Xrm_base ii None = constT 0w) \land
(ea_Xrm_base ii(Some r) = read_reg ii r)
(ea\_Xrm\_index \ ii \ None = constT(0w : Ximm)) \land
(ea\_Xrm\_index \ ii(SomE \ (s: word2, r)) =
\operatorname{seqT}(\operatorname{read\_reg}\ ii\ r)(\lambda idx.\operatorname{constT}(\mathbf{n2w}(2**\mathbf{w2n}\ s)*idx)))
(ea_Xrm\ ii(Xr\ r) = ea_Xr\ r) \land
(ea_Xrm\ ii(XM\ i\ b\ d) =
seqT
  (parT(ea_Xrm_index ii i)(ea_Xrm_base ii b))
     (\lambda(idx, b). constT(XEA_M(idx + b + d))))
(ea_Xdest\ ii(XRM_I\ rm\ i) = ea_Xrm\ ii\ rm) \land
(ea_Xdest ii(X_{RM_R} rm r) = ea_Xrm ii rm) \land
(ea\_Xdest \ ii(XR\_RM \ r \ rm) = ea\_Xr \ r)
(ea\_Xsrc\ ii(XRM\_I\ rm\ i) = ea\_Xi\ i) \land
(ea\_Xsrc\ ii(XRM\_R\ rm\ r) = ea\_Xr\ r) \land
(ea\_Xsrc\ ii(XR\_RM\ r\ rm) = ea\_Xrm\ ii\ rm)
(ea\_Ximm\_rm \ ii(XI\_RM \ rm) = ea\_Xrm \ ii \ rm) \land
(ea_Ximm_rm ii(XI i) = ea_Xi i)
(read_ea\ ii(Xea_I\ i) = constT\ i) \land
(read\_ea\ ii(Xea\_r) = read\_reg\ ii\ r) \land
(read\_ea\ ii(Xea\_m\ a) = read\_m32\ ii\ a)
read_src_ea ii ds = \text{seqT}(\text{ea\_Xsrc} ii ds)(\lambda ea. \text{addT} ea(\text{read\_ea} ii ea))
read_dest_ea ii \ ds = \operatorname{seqT}(\operatorname{ea\_Xdest} \ ii \ ds)(\lambda ea. \operatorname{addT} \ ea(\operatorname{read\_ea} \ ii \ ea))
(write_ea ii(XEA_I i)x = failureT) \land (* one cannot store into a constant *)
(write_ea ii(XEA_R r)x = write_reg ii r x) \land
(write_ea ii(XEA_M a)x = write_m32 ii a x)
(jump_to_ea ii \ eip(XEA_I \ i) = write_eip \ ii(eip + i)) \land
(jump_to_ea ii \ eip(XEA_R \ r) = seqT(read\_reg \ ii \ r)(write_eip \ ii)) \land
(jump\_to\_ea \ ii \ eip(Xea\_m \ a) = seqT(read\_m32 \ ii \ a)(write\_eip \ ii))
```

erase\_eflags 31

```
(call_dest_from_ea\ ii\ eip(Xea_i) = constT(eip + i)) \land
(call_dest_from_ea ii \ eip(XEA_R \ r) = read_reg \ ii \ r) \land
(call\_dest\_from\_ea \ ii \ eip(Xea\_m \ a) = read\_m32 \ ii \ a)
(\text{get\_ea\_address}(XEA_I i) = 0w) \land
(\text{get\_ea\_address}(XEA_R r) = 0w) \land
(get_ea_address(Xea_m a) = a)
bump_{eip} ii len rest =
parT_unit rest(seqT(read\_eip\ ii)(\lambda x. write\_eip\ ii(x + len)))
byte_parity = EVEN \ o \ \mathbf{length} \ o \ \mathbf{filter} \ \mathbf{I} \ o \ n2bits \ 8 \ o \ \mathbf{w2n}
write_PF ii \ w = \text{write\_eflag} \ ii \ X_PF(Some (byte\_parity \ w))
write_ZF ii\ w = \text{write\_eflag}\ ii\ X_ZF(\text{Some}\ (w = 0w))
write_SF ii \ w = write\_eflag \ ii \ X\_SF(Some (word\_msb \ w))
write_logical_eflags ii \ w =
parT_unit(write_PF ii w)
(parT_unit(write_ZF ii w)
(parT_unit(write\_SF ii w))
(parT_unit(write_eflag ii X_OF(Some F))
(parT\_unit(write\_eflag\ ii\ X\_CF(Some\ F))
               (write_eflag ii X_AF None)))))
write_arith_eflags_except_CF ii w =
parT_unit(write_PF ii w)
(parT_unit(write_ZF ii w)
(parT_unit(write_SF ii w)
(parT_unit(write_eflag ii X_OF None)
               (write_eflag ii X_AF NONE))))
write_arith_eflags ii(w, c) =
parT_unit(write_eflag ii X_CF(Some c))(write_arith_eflags_except_CF ii w)
erase_eflags ii =
parT_unit(write_eflag ii X_PF None)
(parT_unit(write_eflag ii X_ZF None)
(parT_unit(write_eflag ii X_SF None)
(parT_unit(write_eflag ii X_OF None)
(parT_unit(write_eflag ii X_CF None)
            (write_eflag ii X_AF None)))))
```

 $x86\_exec\_pop$  32

```
add_with_carry_out(x: Ximm)y = (x + y, 2 * *32 \le \mathbf{w2n} \ x + \mathbf{w2n} \ y)
sub\_with\_borrow\_out(x : Ximm)y = (x - y, x < +y)
write_arith_result ii(w, c)ea = parT_unit(write_arith_eflags ii(w, c))(write_ea ii ea w)
write_arith_result_no_CF ii w ea =
(parT_unit(write_arith_eflags_except_CF ii w)(write_ea ii ea w))
write_arith_result_no_write ii(w, c) = (write_arith_eflags ii(w, c))
write_logical_result ii\ w\ ea = (parT_unit(write_logical_eflags\ ii\ w)(write_logical_eflags\ ii\ ea\ w))
write_logical_result_no_write ii \ w = (write_logical_eflags \ ii \ w)
write_result_erase_eflags ii\ w\ ea = (parT_unit(erase_eflags\ ii)(write_ea\ ii\ ea\ w))
(write_binop ii XADD x y ea = (write_arith_result ii(add_with_carry_out x y)ea)) \land
(write_binop ii XSUB x y ea = (write_arith_result ii(sub_with_borrow_out x y)ea)) \land
(write_binop ii XCMP x y ea = (write_arith_result_no_write ii(sub_with_borrow_out <math>x y))) \land
(write_binop ii XTEST x y ea = (write_logical_result_no_write ii(x&&y))) <math>\land
(write_binop ii Xand x y ea = (write_logical_result ii(x \& \& y) ea)) \land
(write_binop ii \text{ XXOR } x \text{ } y \text{ } ea = (\text{write\_logical\_result } ii(x??y)ea)) \land
(write_binop ii XOR x y ea = (write_logical_result ii(x!!y)ea)) \land
(write_binop ii XSHL x y ea = (write_result_erase_eflags ii(x \ll w2n y)ea)) \land
(write_binop ii XSHR x y ea = (write_result_erase_eflags ii(<math>x >>> \mathbf{w2n} \ y)ea)) \land
(write_binop ii XSAR x \ y \ ea = (write_result_erase_eflags \ ii(x \gg w2n \ y)ea))
(write_monop ii XNOT x ea = write_ea ii ea(\neg x)) \land
(write_monop ii XDEC x ea = write_arith_result_no_CF ii(x - 1w)ea) \land
(write_monop ii XINC x ea = write_arith_result_no_CF ii(x + 1w)ea) \land
(write_monop ii Xneg x ea =
parT_unit(write_arith_result_no_CF\ ii(0w - x)ea)
             (write_eflag ii X_CF None))
(read\_cond\ ii\ X\_ALWAYS = constT\ T) \land
(read\_cond\ ii\ X\_E = read\_eflag\ ii\ X\_ZF) \land
(read\_cond\ ii\ X\_NE = seqT(read\_eflag\ ii\ X\_ZF)(\lambda b.\ constT(\neg b)))
x86_exec_pop ii \ rm =
\operatorname{seqT}(\operatorname{seqT}(\operatorname{read\_reg}\ ii\ \operatorname{ESP})(\lambda esp.\operatorname{addT}\ esp(\operatorname{write\_reg}\ ii\ \operatorname{ESP}(esp+4w))))
     (\lambda(old\_esp, x). seqT(parT(ea\_Xrm\ ii\ rm)(read\_m32\ ii\ old\_esp))
                                (\lambda(ea, w). write\_ea \ ii \ ea \ w))
```

 $x86\_exec\_call$  33

```
x86_exec_push ii imm_rm =
(seqT
   (parT(seqT(ea\_Ximm\_rm\ ii\ imm\_rm)(\lambda ea.\ read\_ea\ ii\ ea))
           (\text{seqT}(\text{read\_reg } ii \text{ ESP})(\lambda w. \text{constT}(w-4w))))
   (\lambda(w, esp). parT\_unit(write\_m32 \ ii \ esp \ w)(write\_reg \ ii \ ESP \ esp)))
x86_exec_popad ii =
\operatorname{seqT}(\operatorname{read\_reg}\ ii\ \operatorname{ESP})(\lambda \operatorname{original\_esp}).
parT_unit(write_reg\ ii\ ESP(original_esp + 32w))(
parT_unit(seqT(read_m32\ ii(original_esp))(\lambda x. write_reg\ ii\ EDI\ x))(
parT_unit(seqT(read_m32\ ii(original_esp + 4w))(\lambda x. write_reg\ ii\ ESI\ x))(
parT_unit(seqT(read_m32\ ii(original_esp + 8w))(\lambda x. write_reg\ ii\ EBP\ x))(
(* The next 4 bytes of stack (containing saved value of ESP) are skipped*)
parT_unit(seqT(read_m32\ ii(original_esp + 16w))(\lambda x. write_reg\ ii\ EBX\ x))(
parT_unit(seqT(read_m32\ ii(original_esp + 20w))(\lambda x. write_reg\ ii\ EDX\ x))(
parT_unit(seqT(read_m32\ ii(original_esp + 24w))(\lambda x. write_reg\ ii\ ECX\ x))(
              (\text{seqT}(\text{read\_m32} \ ii(\textit{original\_esp} + 28w))(\lambda x. \text{write\_reg} \ ii \text{ EAX } x))))))))))
x86_exec_pushad ii =
\operatorname{seqT}(\operatorname{read\_reg}\ ii\ \operatorname{ESP})(\lambda \operatorname{original\_esp}.
parT_unit(write_reg\ ii\ ESP(original_esp-32w))(
parT_unit(seqT(read\_reg\ ii\ EAX)(\lambda w.\ write\_m32\ ii(original\_esp-4w)w))(
parT_unit(seqT(read\_reg\ ii\ ECX)(\lambda w.\ write\_m32\ ii(original\_esp-8w)w))(
parT_unit(seqT(read_reg\ ii\ EDX)(\lambda w. write_m32\ ii(original_esp-12w)w))(
parT_unit(seqT(read_reg\ ii\ EBX)(\lambda w.write_m32\ ii(original_esp-16w)w))(
parT_unit(write_m32\ ii(original_esp - 20w)original_esp)
parT_unit(seqT(read_reg ii EBP)(\lambda w. write_m32 ii(original_esp - 24w)w))(
parT_unit(seqT(read_reg_ii ESI)(\lambda w. write_m32 ii(original_esp - 28w)w))(
              (\text{seqT}(\text{read\_reg } ii \text{ EDI})(\lambda w. \text{write\_m} 32 \ ii(\textit{original\_esp} - 32w)w)))))))))))))
x86_\text{exec\_pop\_eip} ii =
\operatorname{seqT}(\operatorname{seqT}(\operatorname{read\_reg}\ ii\ \operatorname{ESP})(\lambda esp.\operatorname{addT}\ esp(\operatorname{write\_reg}\ ii\ \operatorname{ESP}(esp+4w))))
     (\lambda(old\_esp, x). seqT(read\_m32 \ ii \ old\_esp)
                                   (\lambda w. \text{write\_eip } ii \ w))
x86_exec_push_eip ii =
(seqT
   (parT(read\_eip\ ii)
           (\text{seqT}(\text{read\_reg } ii \text{ ESP})(\lambda w. \text{constT}(w-4w))))
   (\lambda(w, esp). parT\_unit(write\_m32 \ ii \ esp \ w)(write\_reg \ ii \ ESP \ esp)))
x86_exec_call ii imm_rm len =
seqT(parT(read_reg ii ESP)
              (parT(read\_eip\ ii)
                       (ea\_Ximm\_rm\ ii\ imm\_rm))(\lambda(old\_esp,(old\_eip,ea)).
let bumped\_eip = old\_eip + len in
```

 $x86\_exec$  34

```
seqT(call\_dest\_from\_ea\ ii\ bumped\_eip\ ea)(\lambda destw.
     (parT_unit(write_m32 ii old_esp bumped_eip)
          (parT_unit(write_reg\ ii\ ESP(old_esp-4w))
                  (write_eip ii destw)))))
x86_exec_ret ii imm =
seqT(read\_reg\ ii\ ESP)(\lambda old\_esp.
seqT(addT \ old\_esp(read\_m32 \ ii \ old\_esp))(\lambda(old\_esp, retw).
     parT_unit(write_reg\ ii\ ESP(old_esp + imm + 4w))
                  (write\_eip \ ii \ retw)))
(x86\_exec\ ii(XBINOP\ binop\_name\ ds)len = bump\_eip\ ii\ len
(seqT
  (parT(read\_src\_ea ii ds)(read\_dest\_ea ii ds))
       (\lambda((ea\_src, val\_src), (ea\_dest, val\_dest)).
          write_binop ii binop_name val_dest val_src ea_dest))) \lambda
(x86\_exec\ ii(Xmonop\_name\ rm)len = bump\_eip\ ii\ len
(seqT
  (\text{seqT}(\text{ea\_Xrm } ii \ rm)(\lambda ea. \text{addT } ea(\text{read\_ea } ii \ ea)))
       (\lambda(ea\_dest, val\_dest).
          write_monop ii \ monop\_name \ val\_dest \ ea\_dest))) \land
(x86\_exec\ ii(Xxadd\ rm\ r)len = bump\_eip\ ii\ len
(seqT
  (parT(seqT(ea\_Xrm\ ii\ rm)(\lambda ea.addT\ ea(read\_ea\ ii\ ea)))
          (parT(constT(XEA_R r))(read\_reg ii r)))
  (\lambda((ea\_dest, val\_dest), (ea\_src, val\_src)).
        seqT(write_ea ii ea_src val_dest)
             (\lambda x. \text{ write\_binop } ii \text{ XADD } val\_src \ val\_dest \ ea\_dest)))) \land
(x86\_exec\ ii(Xxchg\ rm\ r)len =
(if rm_is_memory_access rm then lockT else I)
(bump_eip ii len
(if rm = (XR r) then constT() else
  (seqT
     (parT(seqT(ea\_Xrm\ ii\ rm)(\lambda ea.addT\ ea(read\_ea\ ii\ ea)))
             (parT(constT(XEA_R r))(read\_reg ii r)))
     (\lambda((ea\_dest, val\_dest), (ea\_src, val\_src)).
          (parT_unit(write_ea ii ea_src val_dest)
                       (write_ea ii \ ea\_dest \ val\_src)))))) \wedge
(x86\_exec\ ii(XCMPXCHG\ rm\ r)len = bump\_eip\ ii\ len
(seqT
  (parT(seqT(ea\_Xrm\ ii\ rm)(\lambda ea. addT\ ea(read\_ea\ ii\ ea)))
          (read\_reg\ ii\ EAX))
  (\lambda((dest\_ea, dest\_val), acc).
        parT_unit(write_binop ii XCMP acc dest_val(XEA_R EAX))
                     (if acc = dest\_val then
                        seqT(read\_reg\ ii\ r)(\lambda src\_val.\ write\_ea\ ii\ dest\_ea\ src\_val)
                     else
                        write_reg ii \text{ EAX } dest\_val)))) \wedge
```

 $x86\_execute$  35

```
(x86\_exec\ ii(XPOP\ rm)len = bump\_eip\ ii\ len(x86\_exec\_pop\ ii\ rm)) \land
(x86\_exec\ ii(XPUSH\ imm\_rm)len = bump\_eip\ ii\ len(x86\_exec\_push\ ii\ imm\_rm)) \land
(x86\_exec\ ii(XCALL\ imm\_rm)len = x86\_exec\_call\ ii\ imm\_rm\ len) \land
(x86\_exec\ ii(Xret\ imm)len = x86\_exec\_ret\ ii\ imm) \land
(x86\_exec \ ii(XLEA \ ds)len = bump\_eip \ ii \ len
(seqT
  ((parT(ea\_Xsrc\ ii\ ds)(ea\_Xdest\ ii\ ds)))
       (\lambda(ea\_src, ea\_dest).
             write_ea ii\ ea\_dest(get\_ea\_address\ ea\_src)))) \land
(x86\_exec\ ii(Xmov\ c\ ds)len = bump\_eip\ ii\ len
(seqT
  ((parT(read\_src\_ea ii ds))
               (parT(read\_cond\ ii\ c)(ea\_Xdest\ ii\ ds))))
        (\lambda((ea\_src, val\_src), (g, ea\_dest)).
             if g then write_ea ii ea_dest val_src else constT()))) \land
(x86\_exec\ ii(XJUMP\ c\ imm)len = (
seqT
  (parT(read\_eip\ ii)(read\_cond\ ii\ c))
     (\lambda(eip, g). \text{ write\_eip } ii(\mathbf{if} \ g \ \mathbf{then} \ eip + len + imm \ \mathbf{else} \ eip + len)))) \land
(x86\_exec\ ii(XLOOP\ c\ imm)len =
seqT(parT(read\_eip\ ii))
        parT(read\_cond ii c)
             (\text{seqT}(\text{read\_reg } ii \text{ ECX})(\lambda ecx. \text{constT}(ecx - 1w)))))
     (\lambda(eip, guard, new\_ecx).
          parT_unit(write_reg ii ECX new_ecx)
                       (write_eip ii
                          (\mathbf{if} \neg (new\_ecx = 0w) \land guard
                           then eip + len + imm else eip + len)))) \land
(x86\_exec\ ii(Xpushad)len = bump\_eip\ ii\ len(
x86_exec_pushad ii)) \land
(x86\_exec ii(XPOPAD)len = bump\_eip ii len(
x86_exec_popad ii)
(x86_execute ii(XPREFIX XG1_NONE q2 i)len = x86_exec ii i len) \land
(x86_execute ii(XPREFIX\ XLOCK\ g2\ i)len = lockT(x86_exec\ ii\ i\ len))
```

## $\begin{array}{c} {\rm Part~IX} \\ {\rm x86\_decoder} \end{array}$

 $x86\_match\_step$  37

```
(STR\_SPACE\_AUX \ n"" = "") \land
(STR\_SPACE\_AUX \ n(STRING \ c \ s) =
if n = 0 then STRING\# "(STRING\ c(STR\_SPACE\_AUX\ 1\ s))
                        else STRING c(STR\_SPACE\_AUX(n-1)s)
bytebits = hex2bits \ o \ STR\_SPACE\_AUX \ 2
check_opcode s =
let y = (n2bits \ 3 \ o \ char2num \ o \ hd \ o \ TL \ o \ explode)s in
assert(\lambda g.g\text{``Reg/Opcode''} = y)
read\_SIB =
assign\_drop"Base" 3 \gg assign\_drop" Index" 3 \gg assign\_drop" Scale" 2 \gg assign\_drop Scal
option_try[
assert(\lambda g.(g\text{``Mod''} = [\mathbf{F}; \mathbf{F}]) \land (g\text{``Base''} = [\mathbf{T}; \mathbf{F}; \mathbf{T}])) \gg assign\_drop\text{``disp32"}32;
assert(\lambda g.(g\text{``Mod''} = [\mathbf{F}; \mathbf{F}]) \land \neg (g\text{``Base''} = [\mathbf{T}; \mathbf{F}; \mathbf{T}]));
assert(\lambda g.(g"Mod" = [T; F])) \gg assign\_drop"disp8"8;
assert(\lambda g.(g"Mod" = [\mathbf{F}; \mathbf{T}])) \gg assign\_drop"disp32"32]
read\_ModRM =
assign\_drop"R/M"3 \gg assign\_drop"Reg/Opcode"3 \gg assign\_drop"Mod"2 \gg
option_try[
assert(\lambda g.(g"Mod" = [T; T]));
assert(\lambda g.(g"Mod" = [\mathbf{F}; \mathbf{F}]) \land \neg (g"R/M" = [\mathbf{F}; \mathbf{F}; \mathbf{T}]) \land \neg (g"R/M" = [\mathbf{T}; \mathbf{F}; \mathbf{T}]));
assert(\lambda g.(g\text{``Mod''} = [\mathbf{F}; \mathbf{F}]) \land (g\text{``R/M''} = [\mathbf{T}; \mathbf{F}; \mathbf{T}])) \gg assign\_drop\text{``disp32"}32;
assert(\lambda g.(g"Mod" = [\mathbf{T}; \mathbf{F}]) \land \neg (g"R/M" = [\mathbf{F}; \mathbf{F}; \mathbf{T}])) \gg assign\_drop"disp8"8;
assert(\lambda g.(g\text{``Mod''} = [\mathbf{F}; \mathbf{T}]) \land \neg (g\text{``R/M''} = [\mathbf{F}; \mathbf{F}; \mathbf{T}])) \gg assign\_drop\text{``disp32''}32;
assert(\lambda g. \neg (g^{"}Mod" = [T; T]) \land (g^{"}R/M" = [F; F; T])) \gg read\_SIB]
is_hex_add x = ((implode \ o \ DROP \ 2 \ o \ explode)x = "+rd")
process\_hex\_add name =
let n = (hex2num \ o \ implode \ o \ TAKE \ 2 \ o \ explode)name \ in
option\_try[drop\_eq(n2bits\ 8(n+0)) \gg assign"reg"(n2bits\ 3\ 0);
                                  drop\_eq(n2bits\ 8(n+1)) \gg assign\text{"reg"}(n2bits\ 3\ 1);
                                  drop\_eq(n2bits\ 8(n+2)) \gg assign\text{"reg"}(n2bits\ 3\ 2);
                                  drop\_eq(n2bits\ 8(n+3)) \gg assign\text{"reg"}(n2bits\ 3\ 3);
                                  drop\_eq(n2bits\ 8(n+4)) \gg assign\text{"reg"}(n2bits\ 3\ 4);
                                  drop\_eq(n2bits\ 8(n+5)) \gg assign\text{"reg"}(n2bits\ 3\ 5);
                                  drop\_eq(n2bits\ 8(n+6)) \gg assign\text{"reg"}(n2bits\ 3\ 6);
                                  drop\_eq(n2bits\ 8(n+7)) \gg assign"reg"(n2bits\ 3\ 7)
```

 $decode\_Xrm32$  38

```
x86_match_step name =
if is\_hex\ name \land (STRLEN\ name = 2) then (* opcode e.g. FE, 83 and 04 *)
drop\_eq(n2bits\ 8(hex2num\ name))
else if is_hex_add name then (* opcode + rd, e.g. F8+rd *)
process_hex_add name
else if name = "1" then (* constant 1 *)
assign\_drop\ name\ 0
else if mem name ["ib"; "cb"; "rel8"; "imm8"] then (* 8-bit immediate or address *)
assign\_drop\ name\ 8
else if mem name ["iw"; "cw"; "imm16"] then (* 16-bit immediate or address *)
assign_drop name 16
else if mem name ["id"; "cd"; "rel32"; "imm32"] then (* 32-bit immediate or address *)
assign\_drop\ name\ 32
else if name = "/r" then (* normal reg + reg/mem *)
read_ModRM
else if mem name ["/0"; "/1"; "/2"; "/3"; "/4"; "/5"; "/6"; "/7"] then (* opcode extension in ModRM *)
read\_ModRM \gg check\_opcode name
else
option\_fail
x86_binop =
[("ADD", XADD); ("AND", XAND); ("CMP", XCMP); ("OR", XOR); ("SAR", XSAR); ("SHR", XSHR); ("SHL", XSHL); ("SU
x86_monop =
[("DEC", XDEC); ("INC", XINC); ("NOT", XNOT); ("NEG", XNEG)]
b2reg\ q\ name = (EL(bits2num(q\ name))[EAX; ECX; EDX; EBX; ESP; EBP; ESI; EDI]): Xreg
decode\_Xr32 \ g \ name =
if name = \text{``EAX''} then EAX else
if g"reg" = [] then b2reg g"Reg/Opcode" else b2reg g"reg"
decode\_SIB g =
let scaled\_index = (if \ g"Index" = [F; F; T] \ then None else Some (b2w g"Scale", b2reg g"Index")) in
if g"Base" = [T; F; T] then (* the special case indicated by "[*]" *)
  if g"Mod" = [F; F] then XM scaled\_index None(b2w g"disp32") else
  if g"Mod" = [T; F] then XM scaled\_index(SOME EBP)(b2w g"disp8") else
  (* g "Mod" = [F;T] *)XM scaled\_index(Some EBP)(b2w g"disp32")
else (* normal cases, just need to read off the correct displacement *)
  let disp = (if (q^m Mod^n = [T; F]) then sw2sw((b2w q^m disp8^n)) : word8) else b2w q^m disp32^n) in
  let disp = (if (g"Mod" = [F; F]) then 0w else disp) in
    XM scaled_index(Some (b2reg g"Base"))disp
decode\_Xrm32 \ g \ name =
if name = \text{``EAX''} then XR EAX else
if (q^{\text{``Mod''}} = [\mathbf{F}; \mathbf{F}]) \land (q^{\text{``R}}/M^{\text{''}} = [\mathbf{T}; \mathbf{F}; \mathbf{T}]) then XM NONE NONE (b2w \ q^{\text{``disp}}32^{\text{''}}) else
```

 $x86\_syntax$  39

```
if \neg (g \text{``Mod''} = [T; T]) \land (g \text{``R/M''} = [F; F; T]) then decode_SIB g else
if (g^{"}Mod" = [F; F]) then XM NONE(SOME (b2reg g^{"}R/M"))0w else
if (g^{"}Mod" = [T; F]) then XM NONE(SOME (b2reg g^{"}R/M"))(sw2sw : word8 \rightarrow word32(b2w g^{"}disp8")) else
if (g\text{``Mod"} = [\mathbf{F}; \mathbf{T}]) then XM NONE(SOME (b2reg g\text{``R/M"}))(b2w\ g\text{``disp32"}) else
if (g^{"}Mod" = [T; T]) then XR(b2reg g^{"}R/M") else XR(b2reg g^{"}reg")
decode\_Xconst\ name\ g =
if name = \text{``imm8''} then sw2sw : word8 \rightarrow word32(b2w g\text{``ib''}) else
if name = \text{``rel8''} then sw2sw : word8 \rightarrow word32(b2w g\text{``cb''}) else
if name = \text{``imm}16\text{''} then sw2sw : \text{word}16 \rightarrow \text{word}32(b2w g\text{``iw''}) else
if name = \text{"imm}32\text{"} then b2w g\text{"id"} else
if name = \text{"rel32"} then b2w g\text{"cd"} else
if name = "1" then 1w else 0w
{\tt decode\_Xdest\_src}\ g\ dest\ src =
if src = \text{``r}32\text{''} then XRM_R(decode_Xrm32 g dest)(decode_Xr32 g src) else
if src = \text{"r/m}32\text{" then XR\_RM}(\text{decode\_Xr}32 \ g \ dest)(\text{decode\_Xrm}32 \ g \ src) else
if src = \text{``m''} then XR_RM(decode_Xr32 g \ dest)(decode_Xrm32 g \ src) else
                           XRM_I(decode\_Xrm32 \ g \ dest)(decode\_Xconst \ src \ g)
decode\_Xconst\_or\_zero ts g =
if length ts < 2 then 0w else decode_Xconst(EL \ 1 \ ts)g
decode_Ximm_rm ts g =
if mem (EL 1 ts) ["r/m32"; "r32"]
 then XI_RM(decode_Xrm32 \ g(EL 1 \ ts))
 else XI(decode\_Xconst(EL 1 ts)g)
x86\_select\_op\ name\ list = snd(hd(filter(\lambda x. fst\ x = name)list))
X_SOME f = \text{Some } o(\lambda(g, w).(f g, w))
x86_syntax ts =
if length ts = 0 then option\_fail else
if mem (hd ts) (map fst x86_binop) then X_SOME(\lambda g.^x86_syntax_binop) else
if mem (hd ts) (map fst x86_monop) then X_SOME(\lambda g.^x86_syntax_monop) else
if hd ts = \text{"POP"} then X_SOME(\lambda g.\text{XPOP}(\text{decode\_Xrm32 } g(EL 1 \ ts))) else
if hd ts = \text{"PUSH"} then X_SOME(\lambda g.XPUSH(decode_Ximm_rm ts g)) else
if hd ts = \text{"PUSHAD"} then X_SOME(\lambda g.XPUSHAD) else
if hd ts = "POPAD" then X_SOME(\lambda g.XPOPAD) else
if hd ts = \text{``CMPXCHG''} then X_SOME(\lambda g.XCMPXCHG(decode_Xrm32 g(EL\ 1\ ts))(decode_Xr32 g(EL\ 2\ ts))) else
\textbf{if} \ \ \textbf{hd} \ \ ts = \text{``XCHG''} \ \textbf{then} \ \ \ \text{X\_SOME}(\lambda g. \text{XXCHG}(\text{decode\_Xrm32} \ \ g(EL\ 1\ ts))(\text{decode\_Xr32} \ \ g(EL\ 2\ ts))) \ \textbf{else}
if hd ts = \text{"XADD"} then X_SOME(\lambda g.XXADD(decode_Xrm32 g(EL\ 1\ ts))(decode_Xr32 g(EL\ 2\ ts))) else
if hd ts = \text{"JMP"} then X_SOME(\lambda g.XJUMP X_ALWAYS(decode_Xconst_or_zero ts g)) else
\mathbf{if} \ \mathbf{hd} \ ts = \text{``JE''} \ \mathbf{then} \ \ \mathbf{X\_SOME}(\lambda g. \mathbf{X} \texttt{JUMP} \ \mathbf{X\_E}(\text{decode\_Xconst\_or\_zero} \ ts \ g)) \ \mathbf{else}
```

 $x86\_lock\_ok$  40

```
if hd ts = "JNE" then X_SOME(\lambda g.XJUMP X_NE(decode_Xconst_or_zero ts g)) else
if hd ts = \text{``LEA''} then X_SOME(\lambda g.XLEA(decode_Xdest_src g(EL\ 1\ ts)(EL\ 2\ ts))) else
if hd ts = \text{``LOOP''} then X_SOME(\lambda q.XLOOP X_ALWAYS(decode_Xconst_or_zero ts q)) else
if hd ts = \text{``LOOPE''} then X_SOME(\lambda g.XLOOP X_E(decode_Xconst_or_zero ts g)) else
if hd ts = \text{``LOOPNE''} then X_SOME(\lambda g.\text{XLOOP X_NE}(\text{decode\_Xconst\_or\_zero } ts g)) else
if hd ts = \text{``MOV''} then X_SOME(\lambda g.XMOV X_ALWAYS(decode_Xdest_src g(EL\ 1\ ts)(EL\ 2\ ts))) else
if hd ts = \text{``CMOVE''} then X_SOME(\lambda g. XMOV X_E(\text{decode\_Xdest\_src } g(EL\ 1\ ts)(EL\ 2\ ts)))) else
if hd ts = \text{``CMOVNE''} then X_SOME(\lambda g.XMOV X_NE(decode_Xdest_src g(EL\ 1\ ts)(EL\ 2\ ts))) else
if hd ts = \text{``CALL''} then X_SOME(\lambda g.XCALL(decode_Ximm_rm ts g)) else
if hd ts = \text{``RET''} then X_SOME(\lambda q.XRET(decode_Xconst_or_zero ts \ q)) else option_fail
x86\_decode\_aux = (match\_list x86\_match\_step tokenise(x86\_syntax o tokenise)o
                     map(\lambda s.let \ x = STR\_SPLIT[\#"]"]s \ in (EL 0 \ x, EL 1 \ x)))^x86\_syntax\_list
x86\_decode\_prefixes w =
if length w < 16 then (XG1_NONE, XG2_NONE, w) else
let bt1 = (TAKE \ 8 \ w = n2bits \ 8(hex2num"2E")) in
let bnt1 = (TAKE \ 8 \ w = n2bits \ 8(hex2num"3E")) in
let l1 = (TAKE \ 8 \ w = n2bits \ 8(hex2num"F0")) in
let bt2 = (TAKE\ 8(DROP\ 8\ w) = n2bits\ 8(hex2num"2E")) \land l1 in
let bnt2 = (TAKE\ 8(DROP\ 8\ w) = n2bits\ 8(hex2num"3E")) \land l1 in
let l2 = (TAKE\ 8(DROP\ 8\ w) = n2bits\ 8(hex2num"F0")) \land (bt1 \lor bnt1) in
let g1 = \mathbf{if} \ l1 \lor l2 then XLOCK else XG1_NONE in
let g2 = \mathbf{if} \ bt1 \lor bt2 then XBRANCH_TAKEN else XG2_NONE in
let g2 = \mathbf{if} \ bnt1 \lor bnt2 then XBRANCH_NOT_TAKEN else g2 in
let n = \mathbf{if} \ bt1 \lor bnt1 \lor l1 \ \mathbf{then} \ (\mathbf{if} \ bt2 \lor bnt2 \lor l2 \ \mathbf{then} \ 16 \ \mathbf{else} \ 8) \ \mathbf{else} \ 0 \ \mathbf{in}
  (g1, g2, DROP \ n \ w)
(\text{dest\_accesses\_memory}(\text{XRM\_I } rm \ i) = \text{rm\_is\_memory\_access } rm) \land
(\text{dest\_accesses\_memory}(\text{XRM\_R} \ rm \ r) = \text{rm\_is\_memory\_access} \ rm) \land
(dest\_accesses\_memory(XR\_RM \ r \ rm) = \mathbf{F})
(x86\_lock\_ok(XBINOP\ binop\_name\ ds) =
mem binop_name [XADD; XAND; XOR; XSUB; XXOR] ∧
dest\_accesses\_memory ds) \land
(x86_lock_ok(XMONOP monop_name rm) =
mem monop\_name [XDEC; XINC; XNEG; XNOT] \land
rm_is_memory_access rm) \wedge
(x86\_lock\_ok(XXADD \ rm \ r) = rm\_is\_memory\_access \ rm) \land
(x86\_lock\_ok(Xxchg rm r) = rm\_is\_memory\_access rm) \land
(x86\_lock\_ok(XCMPXCHG \ rm \ r) = rm\_is\_memory\_access \ rm) \land
(x86\_lock\_ok(XPOP rm) = \mathbf{F}) \land
(x86\_lock\_ok(XLEA \ ds) = \mathbf{F}) \land
(x86\_lock\_ok(XPUSH\ imm\_rm) = \mathbf{F}) \land
(x86\_lock\_ok(XCALL\ imm\_rm) = \mathbf{F}) \land
(x86\_lock\_ok(XRET\ imm) = \mathbf{F}) \land
(x86\_lock\_ok(Xmov \ c \ ds) = \mathbf{F}) \land
```

```
(x86\_lock\_ok(XJUMP \ c \ imm) = \mathbf{F}) \land
(x86\_lock\_ok(XLOOP \ c \ imm) = \mathbf{F}) \land
(x86\_lock\_ok(XPUSHAD) = \mathbf{F}) \land
(x86\_lock\_ok(XPOPAD) = F)
x86_decode w =
let (g1, g2, w) = x86_decode_prefixes w in
let result = x86\_decode\_aux w in
if result = None then None else
  let (i, w) = the result in
     if \neg (g1 = \text{XLOCK}) \lor \text{x86\_lock\_ok} \ i \ \text{then} \ \text{Some} \ (\text{Xprefix} \ g1 \ g2 \ i, w) \ \text{else} \ \text{None}
x86\_decode\_bytes \ b = x86\_decode(FOLDR[]@(map \ w2bits \ b))
(\text{rm\_args\_ok}(X_R r) = \mathbf{T}) \land
(rm\_args\_ok(XM None None t3) = T) \land
(rm\_args\_ok(XM None(Some br)t3) = T) \land
(rm\_args\_ok(XM(SOME (s, ir))NONE t3) = \neg(ir = ESP)) \land
(rm\_args\_ok(XM(SOME (s, ir))(SOME br)t3) = \neg(ir = ESP))
(\text{dest\_src\_args\_ok}(X_{RM\_I} \ rm \ i) = \text{rm\_args\_ok} \ rm) \land
(\text{dest\_src\_args\_ok}(X_{RM\_R} \ rm \ r) = \text{rm\_args\_ok} \ rm) \land
(\text{dest\_src\_args\_ok}(X_{R\_RM} \ r \ rm) = \text{rm\_args\_ok} \ rm)
(imm_rm_args_ok(X_{I_RM} rm) = rm_args_ok rm) \land
(imm_rm_args_ok(XI i) = T)
(x86\_args\_ok(Xbinop\_name\ ds) = dest\_src\_args\_ok\ ds) \land
(x86\_args\_ok(Xmonop\_name\ rm) = rm\_args\_ok\ rm) \land
(x86\_args\_ok(XXADD \ rm \ r) = rm\_args\_ok \ rm) \land
(x86\_args\_ok(Xxchg rm r) = rm\_args\_ok rm) \land
(x86\_args\_ok(Xcmpxchg rm r) = rm\_args\_ok rm) \land
(x86\_args\_ok(XPOP rm) = rm\_args\_ok rm) \land
(x86\_args\_ok(XPUSH\ imm\_rm) = imm\_rm\_args\_ok\ imm\_rm) \land
(x86\_args\_ok(XCALL\ imm\_rm) = imm\_rm\_args\_ok\ imm\_rm) \land
(x86\_args\_ok(Xret imm) = w2n imm < 2 * *16) \land
(x86\_args\_ok(XMOV \ c \ ds) = dest\_src\_args\_ok \ ds \land mem \ c \ [X\_NE; X\_E; X\_ALWAYS]) \land (* partial list *)
(x86\_args\_ok(XJUMP \ c \ imm) = \mathbf{F}) \land
(x86\_args\_ok(XLOOP \ c \ imm) = mem \ c \ [X_NE; X_E; X_ALWAYS]) \land
(x86\_args\_ok(XPUSHAD) = T) \land
(x86\_args\_ok(XPOPAD) = T)
(x86_well_formed_instruction(XPREFIX XLOCK g2 i) = x86_lock_ok i \land x86_args_ok i) \land
(x86\_well\_formed\_instruction(XPREFIX XG1\_NONE q2 i) = x86\_args\_ok i)
```

Part X

 $x86_{-}$ 

 $X86\_NEXT$  43

```
 \begin{tabular}{ll} \be
```

# $\begin{array}{c} {\rm Part~XI} \\ {\rm x86\_program} \end{array}$

```
DOMAIN f = \{x \mid \neg(f \mid x = \text{None})\}\
type_abbrev program_word8 : (address → word8 option)
type\_abbrev program_Xinst : (address \rightarrow (Xinst#num) option)
read_mem_bytes n \ a \ m =
if n = 0 then [] else m a \in \text{read\_mem\_bytes}(n-1)(a+1w)m
(\text{decode\_program\_fun: program\_word8} \rightarrow \text{address} \rightarrow (\text{Xinst}\#\text{num}) \text{ option}) prog\_word8 \ a =
let xs = map the(read\_mem\_bytes 20 \ a \ prog\_word8) in (* read next 20 bytes *)
let decode = x86\_decode\_bytes xs in (* attempt to decode *)
case decode of
None \rightarrow None(* if decoding fails, then fail *)
 \parallel Some (i, w) \rightarrow (* \text{ otherwise extract content } *)
   let n = 20 - (length \ w \ div \ 8) in (* calc length of instruction *)
   if every(\lambda x. \neg (x = \text{None})) (read_mem_bytes n \ a \ prog\_word8) (* check the memory is there *)
    then Some (i, n)
    else None
(decode\_program\_rel : program\_word8 \rightarrow program\_Xinst \rightarrow bool)
   prog\_word8 \ prog\_Xinst =
\forall a.\mathbf{case} \ prog\_Xinst \ a \ \mathbf{of}
         None \to \mathbf{T}
    \parallel \text{Some } (inst, n) \rightarrow \text{decode\_program\_fun } prog\_word8 \ a = \text{Some } (inst, n)
\mathbf{type\_abbrev} run_skeleton : (\mathsf{proc} \to (\mathsf{program\_order\_index} \to \mathsf{address} \ \mathsf{option}))
(run\_skeleton\_wf : address set \rightarrow run\_skeleton \rightarrow bool) addrs rs =
(\forall p \ i \ i'.((\neg((rs \ p \ i') = \text{None})) \land (i < i')) \implies (\neg((rs \ p \ i) = \text{None}))) \land
(\forall p \ i \ a.(rs \ p \ i = SOME \ a) \implies a \in addrs) \land
finite\{p \mid \exists i \ a.rs \ p \ i = \text{Some} \ a\}
x86_event_execute = x86_event_opsem $x86_execute
x86_execute_with_eip_check ii inst len eip =
let s = (x86\_event\_execute \ ii \ inst \ len){} in
   \{E
      \exists eiid\_next \ x.
         (eiid\_next, x, E) \in s \land
        \forall v \ ev.((ev.action = (Access R(Location_{REG} \ ii.proc RegEIP)v)) \land
                        ev \in E.events) \implies (v = eip)
   }
```

```
(event\_structures\_of\_run\_skeleton : program\_Xinst \rightarrow run\_skeleton \rightarrow event\_structure set)
                                               prog\_Xinst \ rs =
let Ess = \{x86\_\text{execute\_with\_eip\_check} \{ \text{proc} := p; \text{program\_order\_index} := i \} inst(\mathbf{n2w} \ n) eip \}
                           (rs \ p \ i = Some \ eip) \land (Some \ (inst, n) = prog\_Xinst \ eip) \}
 {event_structure_bigunion Es \mid Es \in all\_choices Ess}
 (x86\_semantics : program\_word8 \rightarrow state\_constraint \rightarrow
 (run_skeleton#program_Xinst#((event_structure#(execution_witness set))set))set)
 prog\_word8\ initial\_state =
let x1 = \{(rs, prog\_Xinst) \mid rs, prog\_Xinst \mid run\_skeleton\_wf(DOMAIN prog\_Xinst)rs \land \}
                                                                                                                                                decode_program_rel prog_word8 prog_Xinst} in
let x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst, Es) \in x1 \land x3 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst
                                                                                                      (Es = event\_structures\_of\_run\_skeleton prog\_Xinst rs)} in
let x\beta = \{(rs, prog\_Xinst, \{(E, Xs) \mid E \in Es \land As\}\}\}
                                                                                                                   (Xs = \{X \mid \text{valid\_execution } E \mid X \land X)
                                                                                                                                                            (X.initial\_state = initial\_state)\})\})
                      |(rs, prog\_Xinst, Es) \in x2\} in
 x3
 (lts_po_of_event_structure (* : event_structure -> (event_set,unit,machine_visible_label) LTS *))E =
 \{ states := POW \ E.events; \}
      initial := \{\};
     final := if finite E.events then {(E.events, ())} else {};
      trans := \{(s, (Vis \langle mvl\_event := e;
                                                            mvl\_iico := \{e' \mid e' \in s \land (e', e) \in E.intra\_causality\};
                                                            mvl\_first\_of\_instruction := \neg(\exists (e' \in s).e'.iiid = e.iiid);
                                                            mvl\_last\_of\_instruction := \neg(\exists (e' \in (E.events \setminus s)).e'.iiid = e.iiid);
                                                            mvl\_locked := locked E e
                                                       \rangle), s \cup \{e\}) | e \in E.events \land \neg (e \in s) \land \neg (e \in s)
                                                                                                                                              e \in minimal\_elements(E.events \setminus s)(po\_iico E)\}
```

## $\begin{array}{c} {\rm Part~XII} \\ \\ {\rm x86\_sequential\_axiomatic\_model} \end{array}$

restrictE 48

```
sequential_execution E so =
       linear_order so E.events \land
       (\forall (es \in (E.atomicity))(e_1 \in es)(e_2 \in es)e.
             (e_1, e) \in so \land (e, e_2) \in so \implies e \in es)
so_to_vo E \ so = \lambda p.\mathbf{if} \ p \in E.procs \ \mathbf{then} \ so|_{(viewed\_events \ E \ p)} \ \mathbf{else} \ \{\}
so\_to\_write\_serialization so =
\{(e_1, e_2) \mid (e_1, e_2) \in (\text{strict } so) \land \text{mem\_store } e_1 \land \text{mem\_store } e_2 \land (\text{loc } e_1 = \text{loc } e_2)\}
so\_to\_lock\_serialization E so =
\{(e_1, e_2) \mid (e_1, e_2) \in (\text{strict } so) \land \}
                                  \exists es_1 \ es_2 \in (E.atomicity). \neg (es_1 = es_2) \land e_1 \in es_1 \land e_2 \in es_2
so\_to\_rfmap E so =
\{(ew, er) \mid (ew, er) \in so \land ew \in (writes E) \land er \in (reads E) \land (loc er = loc ew) \land ev \in (reads E) \land (loc er = loc ew) \land ev \in (reads E) \land (loc er = loc ew) \land ev \in (reads E) \land (loc er = loc ew) \land ev \in (reads E) \land (loc er = loc ew) \land ev \in (reads E) \land (loc er = loc ew) \land (loc er = loc ev) \land (loc er = loc ev
                                  (\forall ew' \in (\text{writes } E). \neg (ew = ew') \land (ew, ew') \in so \land (ew', er) \in so
                                         \neg(\log ew = \log ew'))
so\_to\_exec\_witness\ E\ initial\_state\ so=
\{ initial\_state := initial\_state; \}
     vo := so\_to\_vo E so;
     write\_serialization := so\_to\_write\_serialization so;
     lock\_serialization := so\_to\_lock\_serialization E so;
     rfmap := so\_to\_rfmap \ E \ so\}
valid_sequential_execution E initial_state so =
       sequential_execution E so \land
        valid_execution E(so\_to\_exec\_witness\ E\ initial\_state\ so)
competes E X =
\{(e_1, e_2) \mid \neg(e_1 = e_2) \land (\text{loc } e_1 = \text{loc } e_2) \land \}
((e_1 \in \text{writes } E \land \text{mem\_store } e_1 \land e_2 \in \text{reads } E) \lor
       (e_2 \in \text{writes } E \land \text{mem\_store } e_2 \land e_1 \in \text{reads } E))
  \ ((happens_before E(X)^+ \cup ((happens_before E(X)^+)^{-1})
race_free E X = \forall e_1 \ e_2 \in (E.events).
       \neg((e_1, e_2) \in \text{ competes } E X)
restrictE E es =
\{procs := E.procs;
     events := E.events \cap es;
     intra\_causality := E.intra\_causality|_{es};
      atomicity := PER\_RESTRICT \ E. atomicity \ es
```

sequential\_race\_free 49

```
\begin{array}{l} \text{prefixes} \ E\ X = \{E' \mid \text{sub\_event\_structure} \ E'\ E \land \forall e_1\ e_2. \\ e_2 \in E'.events \land (e_1,e_2) \in (\text{happens\_before} \ E\ X) \Longrightarrow \\ e_1 \in E'.events \} \end{array} \begin{array}{l} \text{sequential\_race\_free} \ E\ X = \forall (E' \in (\text{prefixes} \ E\ X)) so. \\ \text{valid\_sequential\_execution} \ E'\ X.initial\_state\ so \Longrightarrow \\ \forall e_1\ e_2. \neg ((e_1,e_2) \in \text{competes} \ E' \\ \text{(so\_to\_exec\_witness} \ E'\ X.initial\_state\ so))} \end{array}
```

Part XIII

 $x86\_drf$ 

sequential\_order\_thm 51

```
wfes E =
(\forall iiid. \mathbf{finite} \{ \mathsf{eiid} \mid \exists e \in (E.events). (e.iiid = iiid) \land (e.\mathsf{eiid} = \mathsf{eiid}) \}) \land
(finite E.procs) \land
(\forall e \in (E.events).\mathsf{proc}\ e\ \in\ E.procs) \ \land
(\forall e_1 \ e_2 \in (E.events).(e_1.eiid = e_2.eiid) \land (e_1.iiid = e_2.iiid) \implies (e_1 = e_2)) \land
(DOM\ E.intra\_causality) \subseteq E.events \land
(range E.intra\_causality) \subseteq E.events \land
acyclic(E.intra\_causality) \land
(\forall (e_1, e_2) \in (E.intra\_causality).(e_1.iiid = e_2.iiid)) \land
(\forall (e_1 \in \text{writes } E)e_2.
\neg(e_1=e_2) \land
(e_2 \in \text{writes } E \lor e_2 \in \text{reads } E) \land
(e_1.iiid = e_2.iiid) \land
(loc e_1 = loc e_2) \land
(\exists p \ r. loc \ e_1 = Some (Location_reg \ p \ r))
(e_1, e_2) \in E.intra\_causality^+ \lor
(e_2, e_1) \in E.intra\_causality^+) \land
PER\ E.events\ E.atomicity\ \land
(\forall es \in (E.atomicity). \forall e_1 \ e_2 \in es.(e_1.iiid = e_2.iiid)) \land
(\forall es \in (E.atomicity). \forall e_1 \in es. \forall e_2 \in (E.events). (e_1.iiid = e_2.iiid) \implies e_2 \in es) \land
(\forall e \in (E.events). \forall p \ r.(loc \ e = Some (Location_reg \ p \ r)) \implies (p = proc \ e))
maximal_es E X = maximal\_elements E.events(happens\_before E X)
delete E \ E \ e =
\{procs := E.procs;
  events := E.events \ DELETE \ e;
  intra\_causality := E.intra\_causality|_{(E.events\ DELETE\ e)};
   atomicity := PER\_RESTRICT\ E. atomicity(E. events\ DELETE\ e)
extend_so E so e =
so \cup \{(e_1, e) \mid e_1 \mid e_1 \in E.events\}
locked_ready E X es =
\forall e \ e'.e \in es \land (e,e') \in \text{happens\_before } E \ X \implies e' \in es
ind_hyp1 E X so =
\forall es \ e.
es \; \in \; E.atomicity \land
(\exists e'.e' \in es \land locked\_ready \ E \ X \ es) \land
e \in maximal\_elements\ E.events\ so
e \in es
```

 $data\_race\_free\_thm$  52

```
sequential\_order\_thm =
\forall E \ X. \text{ wfes } E \land \mathbf{finite} \ E.events \land
race_free E X \wedge \text{valid\_execution } E X
 \implies \exists so.
valid_sequential_execution E X.initial\_state so \land
ind_hyp1 E X so \land
(happens_before E(so\_to\_exec\_witness\ E\ X.initial\_state\ so))
   \subseteq (\text{strict } so) \land
(X.write\_serialization = so\_to\_write\_serialization so) \land
(X.lock\_serialization = so\_to\_lock\_serialization E so) \land
(X.rfmap = so\_to\_rfmap E so)
pre E \ X \ es = \text{restrictE} \ E(\text{biginter}\{E''.events \mid E'' \mid es \subseteq E''.events \land E'' \in \text{prefixes} \ E \ X\})
ind_concl E' X' so =
valid_sequential_execution E' X'.initial\_state so \land
(happens_before E'(so\_to\_exec\_witness\ E'\ X'.initial\_state\ so)) \subseteq (strict\ so) \land
(X'.write\_serialization = so\_to\_write\_serialization so) \land
(X'.lock\_serialization = so\_to\_lock\_serialization E'so) \land
(X'.rfmap = so\_to\_rfmap E' so)
ind_concl2 E' X' so = \text{ind\_concl } E' X' so \land \text{ind\_hyp1 } E' X' so
ind_assum E E' X' =
card E'.events < \mathbf{card} \ E.events \land
wfes E' \wedge
finite E'. events \wedge
sequential_race_free E' X' \wedge
valid_execution E' X'
data\_race\_free\_thm =
\forall E \ X. \ \text{well\_formed\_event\_structure} \ E \land \mathbf{finite} \ E.events \land
sequential_race_free E X \wedge \text{valid\_execution } E X
race_free E X \wedge \exists so.
valid_sequential_execution E X.initial\_state so \land
ind_hyp1 E X so \land
(happens_before E(so\_to\_exec\_witness\ E\ X.initial\_state\ so))
   \subseteq (\text{strict } so) \land
(X.write\_serialization = so\_to\_write\_serialization so) \land
(X.lock\_serialization = so\_to\_lock\_serialization E so) \land
(X.rfmap = so\_to\_rfmap E so)
```

## $\begin{array}{c} {\rm Part~XIV} \\ {\rm \bf x86\_hb\_machine} \end{array}$

 $tlang\_typing$  54

```
clause_name x = \mathbf{T}
f \oplus (x \mapsto y) = \lambda x'.if x' = x then y else f(x')
funupd2 f \ w \ x \ y = f \oplus (w \mapsto ((f \ w) \oplus (x \mapsto y)))
linear_order_extend r y = r \cup \{(x, y) \mid (x, x) \in r\}
type_abbrev message : (event#address#value#(event set))
type_abbrev machine_state :
      (*E*)event_structure#
      (*M*)(proc \rightarrow address \rightarrow (value\#(event option)) option)#
      (*F^*)(\mathsf{proc} \to \mathsf{proc} \to \mathsf{message} \; \mathsf{list}) \#
      (*G*)(address \rightarrow event list)#
(* TODO: now we're using X, instead of having an erasure property, maybe we should lose the explicit G? *)
      (*Rg*)(proc \rightarrow reg \rightarrow (value\#(event option)) option)#
      (*X*)execution_witness
(initial\_machine\_state : state\_constraint \rightarrow machine\_state) initial state =
      (\{ events := \{ \}; intra\_causality := \{ \}; atomicity := \{ \} \},
         (\lambda p.\lambda a.\mathbf{case}\ initial state(\text{Location\_mem }a)\ \mathbf{of}\ \text{None} \to \text{None}\ \|\ \text{Some}\ v \to \text{Some}\ (v, \text{None})),
         (\lambda p_1.\lambda p_2.NIL),
         (\lambda a.NIL),
         (\lambda p.\lambda r.\mathbf{case}\ initial state(\mathtt{Location\_Reg}\ p\ r)\ \mathbf{of}\ \mathtt{None}\to\mathtt{None}\ \|\ \mathtt{Some}\ v\to\mathtt{Some}\ (v,\mathtt{None})),
         \langle | initial\_state := initialstate; \rangle
           vo := \lambda p.\{\};
           write\_serialization := \{\};
           lock\_serialization := \{\};
           rfmap := \{\}\}
execution_witness_of_machine_state(E, M, F, G, Rg, X) = X
event_set_of_machine_state(E, M, F, G, Rq, X) = E.events
next_eiid_es E = \text{next\_eiid} \{ e.\text{eiid} \mid e \in E.events \}
(* enqueuing sub-writes *)
(\forall P \ M \ \mathbf{F} \ G \ Rg \ e \ F' \ a \ v \ p \ E \ X(X' : \mathsf{execution\_witness}) iico \ mvl.
(clause_name "enqueue-mem-write" \land
(e = mvl.mvl\_event) \land
```

tlang\_typing 55

```
(iico = mvl.mvl_{iico}) \land
(e.action = Access W(Location\_mem a)v) \land
(p = e.iiid.proc) \land
(\forall q. q \in P \implies ((F' \ p \ q) = ([(e, a, (v, iico))] + +(\mathbf{F} \ p \ q)))) \land
(\forall p' \ q.p' \in P \implies q \in P \implies \neg(p = p') \implies (F' \ p' \ q = \mathbf{F} \ p' \ q)) \land 
(X' = X)
machine\_trans\ P(E, M, \mathbf{F}, G, Rg, X)(Vis\ mvl)(E, M, F', G, Rg, X')
) \land
(* Read own memory *)
(\forall P \ M \ \mathbf{F} \ G \ Rg \ e \ a \ v \ p \ E \ E' \ X \ X' \ eo \ iico \ mvl.
(clause_name "mem-read" \wedge
(e = mvl.mvl\_event) \land
(iico = mvl.mvl_{iico}) \land
(e.action = Access R(Location_mem a)v) \land
(p = e.iiid.proc) \land
(M \ p \ a = \text{Some} \ (v, eo)) \land
(\neg \exists e' \ v' \ iico'. \mathbf{mem} \ (e', a, v', iico') \ (\mathbf{F} \ p \ p)) \land 
(E' = \{ events := E.events \cup \{e\} \};
         intra\_causality := E.intra\_causality \cup \{(e', e) \mid e' \in iico\};
         atomicity := E.atomicity \rangle \land
(X' = X \ (vo := X.vo \oplus (p \mapsto (linear\_order\_extend(X.vo p)e));
             rfmap := \mathbf{case} \ eo \ \mathbf{of} \ \mathrm{None} \to X.rfmap
                               \| \text{ Some } ew \to X.rfmap \cup \{(ew, e)\} \} \rangle \wedge
(* TODO: would it be cleaner to pull eo from the most recent write in our X.vo p, instead of recording eo in M?*)
(\forall e'.(e',e) \in (\text{happens\_before } E' X')^+ \implies e' \in \text{viewed\_events } E' p \implies (e',e') \in X.vo p)
) \implies
machine\_trans\ P(E, M, \mathbf{F}, G, Rg, X)(Vis\ mvl)(E', M, \mathbf{F}, G, Rg, X')
(* delivering sub-writes, case (1), where eiid has already been seen by *)
(* Ga (and no subwrite of a Ga-older eiid is pending for this processor) *)
(\forall P \ M \ \mathbf{F} \ G \ Rg \ M' \ F' \ p \ q \ e \ a \ v \ G_0 \ G_1 \ E \ X \ X' \ iico.
((clause\_name "deliver-mem-write-1") \land
(e.action = Access W(Location\_mem a)v) \land
(\mathbf{F} = \text{funupd2 } F' \ p \ q((F' \ p \ q) + + [(e, a, (v, iico))])) \land
(G \ a = (G_0 + + [e] + + G_1)) \land
(\forall p' \ e' \ v'.p' \in P \implies \mathbf{mem} \ (e', a, v') \ (\mathbf{F} \ p' \ q) \implies \neg (\mathbf{mem} \ e' \ G_1)) \land 
(M' = \text{funupd2} \ M \ q \ a(\text{SOME} \ (v, \text{SOME} \ e))) \land
(X' = X \ (vo := X.vo \oplus (q \mapsto (linear\_order\_extend(X.vo q)e)))) \land
(\forall e'.(e',e) \in (\text{happens\_before } E X')^+ \implies e' \in \text{viewed\_events } E q \implies (e',e') \in X.vo q)
machine\_trans\ P(E, M, \mathbf{F}, G, Rg, X) TAU(E, M', F', G, Rg, X')
) \
```

 $tlang\_typing$  56

```
(\forall P \ M \ \mathbf{F} \ G \ Rg \ M' \ F' \ G' \ p \ q \ a \ e \ v \ E \ E' \ X \ X' \ iico.
((clause_name "deliver-mem-write-2") ∧
(e.action = Access W(Location_mem a)v) \land
(\mathbf{F} = (\text{funupd2 } F' \ p \ q((F' \ p \ q) + + ([(e, a, (v, iico))])))) \land 
(\neg(\mathbf{mem}\ e\ (G\ a))) \land
(G' = G \oplus (a \mapsto ([e] + + G \ a))) \land 
(\forall p' \ e' \ v'.p' \in P \implies \mathbf{mem} \ (e', a, v') \ (\mathbf{F} \ p' \ q) \implies \neg(\mathbf{mem} \ e' \ (G \ a))) \land 
(M' = \text{funupd2 } M \ q \ a(\text{Some} \ (v, \text{Some} \ e))) \land
(E' = \{ events := E.events \cup \{e\} \};
         intra\_causality := E.intra\_causality \cup \{(e', e) \mid e' \in iico\};
         atomicity := E.atomicity \rangle \land
(X' = X \ (vo := X.vo \oplus (q \mapsto (linear\_order\_extend(X.vo \ q)e));
             write\_serialization := X.write\_serialization \cup \{(e', e) \mid \mathbf{mem} \ e' \ (G \ a)\}\}) \land
(\forall e'.(e',e) \in (\text{happens\_before } E X')^+ \implies e' \in \text{viewed\_events } E q \implies (e',e') \in (X.vo q))
machine\_trans\ P(E, M, \mathbf{F}, G, Rg, X) TAU(E', M', F', G', Rg, X')
(\forall P \ M \ \mathbf{F} \ G \ Rg \ e \ r \ v \ p \ E \ E' \ X \ X' \ eo \ iico \ mvl.
(clause_name "reg-read" \wedge
(e = mvl.mvl\_event) \land
(iico = mvl.mvl_{iico}) \land
(e.action = Access R(Location_{REG} p r)v) \land
(Rg \ p \ r = Some \ (v, eo)) \land
(E' = \{ events := E.events \cup \{e\}; \}
         intra\_causality := E.intra\_causality \cup \{(e', e) \mid e' \in iico\};
         atomicity := E.atomicity \rangle \rangle \wedge
(X' = X \ (vo := X.vo \oplus (p \mapsto (linear\_order\_extend(X.vo p)e));
             rfmap := \mathbf{case} \ eo \ \mathbf{of} \ \mathrm{None} \to X.rfmap
                              \| \text{ Some } ew \to X.rfmap \cup \{(ew, e)\} \rangle ) \land
(\forall e'.(e',e) \in (\text{happens\_before } E' X')^+ \implies e' \in \text{viewed\_events } E' p \implies (e',e') \in X.vo p)
machine\_trans\ P(E, M, \mathbf{F}, G, Rg, X)(Vis\ mvl)(E', M, \mathbf{F}, G, Rg, X')
(\forall P \ M \ \mathbf{F} \ G \ Rq \ Rq' \ e \ r \ v \ p \ E \ E' \ X \ X' \ iico \ mvl.
(clause_name "reg-write" ∧
(e = mvl.mvl\_event) \land
(iico = mvl.mvl_{iico}) \land
(e.action = Access W(Location_{REG} p r)v) \land
(Rg' = \text{funupd2} \ Rg \ p \ r(\text{Some} \ (v, \text{Some} \ e))) \land
(E' = \{ events := E.events \cup \{e\};
         intra\_causality := E.intra\_causality \cup \{(e', e) \mid e' \in iico\};
         atomicity := E.atomicity \rangle \land
```

final\_state 57

```
 \begin{array}{l} (X' = X \ (vo := X.vo \oplus (p \mapsto (\operatorname{linear\_order\_extend}(X.vo \ p)e)))) \land \\ (\forall e'.(e',e) \in (\operatorname{happens\_before} \ E' \ X')^+ \implies e' \in \operatorname{viewed\_events} \ E' \ p \implies (e',e') \in X.vo \ p) ) \Longrightarrow \\ machine\_trans \ P(E,M,\mathbf{F},G,Rg,X)(\operatorname{Vis} \ mvl)(E',M,\mathbf{F},G,Rg',X') \\ ) \end{array}
```

 $\text{final\_state } p \ s = \exists n. (p \ n, p(n+1)) = (\text{Some } s, \text{None})$ 

 $\begin{array}{c} {\rm Part~XV} \\ {\rm x86\_lts\_ops} \end{array}$ 

 $states\_of\_trace$  59

 $lts\_state = LEAF \ of \ value \mid LEFT \ of \ lts\_state \mid RIGHT \ of \ lts\_state \mid PAIR \ of \ lts\_state \ lts\_state$ 

```
(eip\_tracked\_lts : ('s, 'a, lts\_monad\_visible\_label)LTS \rightarrow address \rightarrow program\_order\_index \rightarrow (address\#program\_order\_index)
  \{ states := \{ (eip, po, s) \mid eip \in UNIV \land (po = po\_initial) \land s \in lts.states \} ; \}
              initial := (eip\_initial, po\_initial, lts.initial);
            \mathit{final} := \{ ((\mathit{eip}, \mathit{po\_initial}, s), x) \mid \mathit{eip} \in \mathit{UNIV} \land (s, x) \in \mathit{lts.final} \};
              trans := \{((eip, po\_initial, s), l, (eip', po\_initial, s')) \mid (s, l, s') \in lts.trans \land l
              (\forall lmvl\ p\ v.((l = Vis\ lmvl) \land (lmvl.lmvl_{action} = Access\ R(Location\_reg\ p\ RegeIP)v)) \implies ((eip = v) \land (eip' = v))
              (\forall lmvl\ p\ v.((l = Vis\ lmvl) \land (lmvl.lmvl_{action} = Access\ W(Location\_reg\ p\ RegEIP)v)) \implies (eip' = eip)) \land (eip' = eip))
              ((\neg(\exists lmvl\ p\ v\ d.(l = Vis\ lmvl) \land (lmvl.lmvl_{action} = Access\ d(Location\_reg\ p\ RegEIP)v))) \implies (eip' = eip))(i)
             }]
 (eip\_tracked\_lts\_initial: address \rightarrow (address\#program\_order\_index\#lts\_state, unit, lts\_monad\_visible\_label)LTS) eip=tracked\_lts\_initial: address \rightarrow (address\#program\_order\_index\#lts\_state, unit, lts\_monad\_visible\_label)LTS) eip=tracked\_lts\_initial: address \rightarrow (address\#program\_order\_index\#lts\_state, unit, lts\_monad\_visible\_label)LTS) eip=tracked\_lts\_initial: address \rightarrow (address\#program\_order\_index\#lts\_state, unit, lts\_monad\_visible\_label)
let s = (eip, 0, \text{Leaf } 0w) in
 \{ states := \{ s \};
             initial := s;
            final := \{(s, ())\};
             trans := \{\}\}
 (lts\_parallel: ('s1, 'a1, 'vl)LTS \rightarrow ('s2, 'a2, 'vl)LTS \rightarrow (('s1\#'s2), ('a1\#'a2), 'vl)LTS) lts1 \ lts2 = ('s1, 'a1, 'vl)LTS) lts1 \ lts2 = ('s1, 'vl)LTS) \ lts1 \ lts2 =
  \{ states := \{ (s1, s2) \mid s1 \in lts1.states \land s2 \in lts2.states \} ; \}
              initial := (lts1.initial, lts2.initial);
            final := \{((s1, s2), (x1, x2)) \mid (s1, x1) \in lts1.final \land (s2, x2) \in lts2.final\};
             trans := \{((s1, s2), \text{TAU}, (s1', s2)) \mid (s1, \text{TAU}, s1') \in lts1.trans \land s2 \in lts2.states\} \cup trans := \{((s1, s2), \text{TAU}, (s1', s2)) \mid (s1, \text{TAU}, s1') \in lts1.trans \land s2 \in lts2.states\} \cup trans := \{((s1, s2), \text{TAU}, (s1', s2)) \mid (s1, \text{TAU}, s1') \in lts1.trans \land s2 \in lts2.states\} \cup trans := \{((s1, s2), \text{TAU}, (s1', s2)) \mid (s1, \text{TAU}, s1') \in lts1.trans \land s2 \in lts2.states\} \cup trans := \{((s1, s2), \text{TAU}, (s1', s2)) \mid (s1, \text{TAU}, s1') \in lts1.trans \land s2 \in lts2.states\} \cup trans := \{((s1, s2), \text{TAU}, (s1', s2)) \mid (s1, \text{TAU}, s1') \in lts1.trans \land s2 \in lts2.states\} \cup trans := \{((s1, s2), \text{TAU}, (s1', s2)) \mid (s1, \text{TAU}, s1') \in lts1.trans \land s2 \in lts2.states\} \cup trans := \{(s1, s2), \text{TAU}, (s1', s2), \text{TAU}, (s
                                             \{((s1, s2), \text{TAU}, (s1, s2')) \mid (s2, \text{TAU}, s2') \in lts2.trans \land s1 \in lts1.states\} \cup \{((s1, s2), \text{TAU}, (s1, s2')) \mid (s2, \text{TAU}, s2') \in lts2.trans \land s1 \in lts1.states\} \cup \{((s1, s2), \text{TAU}, (s1, s2')) \mid (s2, \text{TAU}, s2') \in lts2.trans \land s1 \in lts1.states\} \cup \{((s1, s2), \text{TAU}, (s1, s2')) \mid (s2, \text{TAU}, s2') \in lts2.trans \land s1 \in lts1.states\}
                                             \{((s1, s2), l, (s1', s2')) \mid (s1, l, s1') \in lts1.trans \land (s2, l, s2') \in lts2.trans \land \neg (l = TAU)\}
\rangle
 (\text{traces\_of\_lts}: ('s, 'v, 'vl) LTS \rightarrow ('s\#(\text{num} \rightarrow (('vl \text{ label})\#'s) \text{ option})) set) lts =
 \{(x,t) \mid (x = lts.initial) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \implies (lts.initial, l',s') \in lts.trans) \land (\forall l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (lts.initial, l',s') \in lts.trans) \land (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 = \text{Some} \ (l',s')) \mapsto (\exists l' \ s'.(t \ 0 =
                                                                                                                                                                (\forall i \ l' \ s'.(t(i+1) = \text{Some} \ (l', s')) \implies \exists l \ s.(t \ i = \text{Some} \ (l, s)) \land (s, l', s') \in lts.trans)\}
 \{(x,t) \mid (x = lts.initial) \land
                                                                                (\forall l' \ s'.(t \ 0 = \text{Some} \ (l', s')) \implies (lts.initial, l', s') \in lts.trans) \land
                                                                                ((t \ 0 = \text{None}) \implies \neg(\exists l' \ s'.(lts.initial, l', s') \in lts.trans)) \land
                                                                                (\forall i \ l' \ s'.(t(i+1) = \text{SOME} \ (l',s')) \implies \exists l \ s.(t \ i = \text{SOME} \ (l,s)) \land (s,l',s') \in lts.trans) \land
                                                                                (\forall i \ l \ s.(t \ i = \text{Some} \ (l, s)) \implies (t(i+1) = \text{None}) \implies \neg(\exists l' \ s'.(s, l', s') \in lts.trans))\}
 (states\_of\_trace(trs : ('s\#(num \rightarrow ((('vl label)\#'s) option)))set) : 's set) =
 \{s \mid \exists n \ l \ tr.tr \in trs \land ((\mathbf{snd} \ tr)n = \mathrm{Some} \ (l,s))\}
 \{\mathbf{fst} \ tr \mid tr \in trs\}
```

#### $\begin{array}{c} {\rm Part~XVI} \\ {\rm x86\_hb\_machine\_thms} \end{array}$

```
(\text{machine\_lts}: \text{proc set} \rightarrow \text{state\_constraint} \rightarrow (\text{machine\_state}, \text{unit}, \text{machine\_visible\_label}) \text{LTS}) ps \ initial\_state =
\{ states := (UNIV : machine\_state set); \}
  initial := initial_machine_state initial_state;
  final := \{\};
  trans := \{(s1, l, s2) \mid machine\_trans \ ps \ s1 \ l \ s2\}\}
(machine\_execution\_of\_event\_structure\ E\ initial\_state) =
let lts\_prog = lts\_po\_of\_event\_structure E in
let lts\_machine = machine\_lts(E.procs)initial\_state in
let lts = lts_parallel lts_prog lts_machine in
completed_traces_of_lts lts
final\_states init\_state E trs =
\{st \mid \exists path \ lbl.((\{\}, init\_state), path) \in trs \land \}
                       final_state path(lbl, (E.events, st))}
hb_{equivalence\_thm1} =
\forall E X.
well_formed_event_structure E \wedge
finite E.events \land
valid_execution E X \wedge
nice_execution E X
 \Longrightarrow
\exists M \mathbf{F} G Rq.
(E, M, \mathbf{F}, G, Rq, X) \in
final\_states(initial\_machine\_state\ X.initial\_state)
                    (machine\_execution\_of\_event\_structure\ E\ X.initial\_state)
partial_view_orders_well_formed E \ vo =
(\forall p \in (E.procs).
  (\exists es \ es'.
     (viewed_events E p = es \cup es') \wedge
     (\forall e \in es'. \text{mem\_store } e) \land
      linear\_order(vo\ p)es) \land
  \forall e \in (\text{viewed\_events } E \ p). \mathbf{finite} \{ e' \mid (e', e) \in (vo \ p) \}) \land
(\forall p. \neg (p \in E.procs) \implies (vo p = \{\}))
partial_valid_execution E X =
   partial_view_orders_well_formed E X.vo \land
   X.write\_serialization \in write\_serialization\_candidates E \land
   X.lock\_serialization \in lock\_serialization\_candidates E \land
   X.rfmap \in reads\_from\_map\_candidates E \land
   check_causality E X.vo(\text{happens\_before } E X) \land
   check_rfmap_written E X.vo X.rfmap \land
   check_rfmap_initial E X.vo X.rfmap X.initial\_state \land
   check_atomicity E X.vo
```

```
hb\_equivalence\_thm2 =
\forall E \ M \ \mathbf{F} \ G \ Rg \ X.
finite E.events \land
well_formed_event_structure E \wedge
(E, M, \mathbf{F}, G, Rg, X) \in
final\_states(initial\_machine\_state\ X.initial\_state)
              (machine\_execution\_of\_event\_structure\ E\ X.initial\_state)
partial_valid_execution E X
hb_machine_progress_thm =
\forall E \ mst \ es \ path \ lbl \ init.
((\{\}, (initial\_machine\_state init)), path) \in
traces_of_lts(lts_parallel(lts_po_of_event_structure E)(machine_lts E.procs\ init)) \land
final\_state\ path(lbl,(es,mst))
\exists mst'.
(mst, Tau, mst') \in (machine\_lts \ E.procs \ init).trans \lor
(\exists es' l.
(es, l, es') \in (lts\_po\_of\_event\_structure\ E).trans \land
(mst, l, mst') \in (machine\_lts \ E.procs \ init).trans) \lor
((\forall p \ q.((\mathbf{fst}(\mathbf{snd}(\mathbf{snd} \ mst)))p \ q = [])) \land
(\forall es' \ l. \neg ((es, l, es') \in (lts\_po\_of\_event\_structure \ E).trans)))
```

#### Index

action, 8	$dest\_src\_args\_ok, 41$
$add\_with\_carry\_out, 32$	dirn, 8
addT, 22, 28	$discardT_{-}ev,25$
$addT_{-}ev,25$	DOMAIN, 45
$addT\_seq,\ 21$	V1 1 20
aligned 32, 27	$ea\_Xdest$ , 30
10 00	$ea_Xi$ , 30
b2reg, 38	$ea\_Ximm\_rm, 30$
barrier, 8	$ea\_Xr$ , 30
bump_eip, 31	$ea\_Xrm$ , 30
byte_parity, 31	ea_Xrm_base, 30
by tebits, 37	$ea\_Xrm\_index, 30$
	$ea\_Xsrc, 30$
call_dest_from_ea, 31	$eip\_tracked\_lts, 59$
$check\_atomicity, 17$	$eip\_tracked\_lts\_initial, 59$
check_causality, 15	$erase\_eflags,\ 31$
check_final, 16	event,8
check_final_mem, 16	$event\_set\_of\_machine\_state,\ 54$
check_opcode, 37	$event\_structure,~8$
check_rfmap_initial, 16	$event\_structure\_bigunion,\ 25$
check_rfmap_written, 16	$event\_structure\_empty,\ 25$
$choice T_{-}ev, 25$	$event\_structure\_lock,\ 25$
clause_name, 54	$event\_structure\_seq\_union, \ 25$
competes, 48	$event\_structure\_union, \ 25$
$completed\_traces\_of\_lts, 59$	$event\_structures\_of\_run\_skeleton,\ 45$
constT, 22, 28	$execution\_witness,~8$
$constT\_ev, 25$	$execution\_witness\_of\_machine\_state,\ 54$
$constT\_seq,\ 21$	$extend\_so,\ 51$
$data\_race\_free\_thm,  52$	failure T, 23, 28
$decode\_program\_fun, 45$	$failure T_{-}ev, 26$
$decode\_program\_rel, 45$	$failure T\_seq, \ 21$
$decode\_SIB,\ 38$	$final\_state, 57$
$decode\_Xconst, 39$	$final\_states, 61$
$decode\_Xconst\_or\_zero, 39$	funupd, 54
$decode\_Xdest\_src, 39$	funupd2, 54
$decode\_Ximm\_rm, 39$	fanapaz, or
$decode\_Xr32, \frac{38}{38}$	$get\_ea\_address, 31$
$decode\_Xrm32, 38$	$get\_l\_stores, 15$
deleteE, $51$	J
$dest\_accesses\_memory, 40$	$happens\_before, 15$
2000_2000000_111011101 g, 10	* * *

INDEX 64

$hb\_equivalence\_thm1, 61$	$parT_{-}ev, 26$
hb_equivalence_thm2, 62	$parT\_seq$ , 21
hb_machine_progress_thm, 62	$parT\_unit, 23, 28$
	$parT\_unit\_ev, 26$
iiid, 6	$parT\_unit\_seq, 22$
$iiid\_dummy, 43$	$partial\_valid\_execution, 61$
iiids, 13	partial_view_orders_well_formed, 61
$imm\_rm\_args\_ok, 41$	po, 13
$ind\_assum,\ 52$	$po\_iico, 14$
$ind\_concl, 52$	po_strict, 13
$ind\_concl2, 52$	pre, 52
$ind\_hyp1$ , 51	prefixes, 49
$initial\_eiid\_state, 9$	preserved_program_order, 14
$initial\_machine\_state, 54$	proc, 13
$is\_hex\_add, 37$	process_hex_add, 37
$jump\_to\_ea, \frac{30}{}$	race_free, 48
1-1-1 0	$read\_cond, 32$
label, 9	$read\_dest\_ea, 30$
linear_order_extend, 54	$read\_ea, 30$
loc, 13	$read\_eflag, 23, 28$
location, 8	$read\_eflag\_ev, 27$
lock_serialization_candidates, 15	$read\_eflag\_seq, 22$
locked, 13	$read\_eip, 23, 28$
locked_ready, 51	$read\_eip\_ev, 27$
lockT, 22, 28	$read\_eip\_seq, \frac{22}{}$
$lockT_{-}ev$ , 26	$read\_location\_ev, 26$
$lockT\_seq, 21$	$read_{-}m32, 23, 28$
LTS, 9	$read\_m32\_ev, \frac{27}{}$
lts_monad_visible_label, 9	$read\_m32\_seq, 22$
lts_parallel, 59	$read\_mem\_bytes, 45$
lts_po_of_event_structure, 46	$read\_mem\_seq,\ 22$
lts_state, 59	$read\_ModRM, 37$
$machine\_execution\_of\_event\_structure, \ 61$	$read\_reg, 23, 28$
machine_lts, 61	$read\_reg\_ev, 27$
machine_visible_label, 9	$read\_reg\_seq, 22$
$mapT_{-}ev, 25$	$read\_SIB, 37$
$maximal\_es, 51$	$read\_src\_ea, 30$
mem_barrier, 13	reads, 13
mem_load, 13	$reads\_from\_map\_candidates,\ 15$
mem_store, 13	$reads\_from\_map\_candidates\_old,\ 15$
,	reg, 8
$next\_eiid$ , 8	$restrict\_execution\_witness, 17$
$next\_eiid\_es, 54$	restrictE, 48
$nice\_execution, 19$	$rm\_args\_ok, 41$
$niceness\_thm,19$	$rm\_is\_memory\_access, 11$
ontion annly 22	$run\_skeleton\_wf$ , 45
$option\_apply, 23$	seqT, 23, 28
parT, 23, 28	30q1, 20, 20

INDEX 65

$seqT_{-}ev, 26$	$write\_arith\_result\_no\_CF, 32$
$seqT\_seq, 21$	$write\_arith\_result\_no\_write, 32$
$sequential\_execution, 48$	$write\_binop, \frac{32}{}$
$sequential\_order\_thm, 51$	$write\_ea, 30$
sequential_race_free, 49	$write\_eflag, 23, 28$
$so\_to\_exec\_witness, 48$	$write\_eflag\_ev, 27$
$so\_to\_lock\_serialization, 48$	$write\_eflag\_seq, 22$
$so\_to\_rfmap, 48$	$write\_eip,\ 23,\ 28$
$so\_to\_vo, 48$	$write\_eip\_ev, 27$
$so\_to\_write\_serialization, 48$	$write\_eip\_seq, \frac{22}{}$
state_updates, 16	$write\_location\_ev, 26$
state_updates_mem, 16	$write\_logical\_eflags, 31$
states_of_trace, 59	$write\_logical\_result, 32$
$STR\_SPACE\_AUX, 37$	$write\_logical\_result\_no\_write, 32$
sub_event_structure, 14	write_m32, 23, 28
$sub\_with\_borrow\_out, 32$	$write\_m32\_ev, 27$
	$write\_m32\_seq, 22$
$tlang\_typing, 54$	$write\_mem\_seq, 22$
$traces\_of\_lts, 59$	$write\_monop, \frac{3}{32}$
$type\_abbrev\_address,\ 8$	$write\_PF, 31$
$type\_abbrev\_eiid, \ 8$	$write\_reg, 23, 28$
$type\_abbrev\_eiid\_state,\ 8$	$write\_reg\_ev, 27$
$type\_abbrev\_lts\_monad\_label, \ 9$	$write\_reg\_seq, 22$
$type\_abbrev\_M$ , 21, 25	$write\_result\_erase\_eflags, 32$
$type\_abbrev\_machine\_label, \ 9$	$write\_serialization\_candidates,\ 15$
$type\_abbrev\_machine\_state, 54$	$write\_serialization\_candidates\_old, 15$
$type\_abbrev\_message, 54$	$write\_SF, 31$
$type\_abbrev\_proc, 6$	$write\_ZF, 31$
$type\_abbrev\_program\_order\_index, \ 6$	writes, 13
$type\_abbrev\_program\_word8,\ 45$	
$type\_abbrev\_program\_Xinst,\ 45$	$x86\_args\_ok, 41$
$type\_abbrev\_reln, 8$	$x86\_binop, 38$
$type\_abbrev\_run\_skeleton, 45$	$x86\_decode, 41$
$type\_abbrev\_state\_constraint, \ 8$	$x86\_decode\_aux, 40$
$type\_abbrev\_value,\ 8$	$x86\_decode\_bytes, 41$
$type\_abbrev\_view\_orders,\ 8$	$x86\_decode\_prefixes, 40$
$type\_abbrev\_x86\_state,\ 21$	$x86\_event\_execute, 45$
$type\_abbrev\_Ximm, 6$	$x86\_exec, 34$
	$x86\_exec\_call, 33$
valid_execution, 17	$x86\_exec\_pop, 32$
valid_sequential_execution, 48	$x86\_exec\_pop\_eip, 33$
$value\_of$ , 13	$x86\_exec\_popad, 33$
$view\_orders\_well\_formed, 14$	$x86\_exec\_push, 33$
$viewed\_events, 14$	$x86\_exec\_push\_eip, 33$
11 6 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	$x86\_exec\_pushad, 33$
well_formed_event_structure, 14	$x86\_exec\_ret, 34$
wfes, 51	$x86\_execute, 35$
write_arith_eflags, 31	$x86\_execute\_some, 43$
write_arith_eflags_except_CF, 31	$x86\_execute\_with\_eip\_check,\ 45$
write_arith_result, 32	

INDEX 66

```
x86_lock_ok, 40
x86\_match\_step, 37
x86\_monop, 38
X86\_NEXT, 43
x86\_select\_op, \frac{39}{}
x86\_semantics, 46
x86\_syntax, 39
x86\_well\_formed\_instruction,\ 41
X\_SOME, 39
Xbinop\_name, 11
Xcond, 11
Xdest\_src, 11
Xea, 6
Xeflags, 6
Ximm\_rm, 11
Xinst, 11
Xinstruction, 11
Xmonop\_name, 11
Xpre_g1, 11
Xpre_g2, 11
XREAD\_EFLAG, 21
XREAD_EIP, 21
XREAD\_MEM, 21
XREAD\_MEM\_BYTES, \frac{21}{}
XREAD\_REG, 21
Xreg, 6
Xrm, 11
XWRITE\_EFLAG, 21
XWRITE_EIP, 21
XWRITE_MEM, 21
XWRITE\_REG, 21
```