The Semantics of x86 Multiprocessor Machine Code

The HOL Specification

Susmit Sarkar[1] Peter Sewell[1] Francesco Zappa Nardelli[2]
Scott Owens[1] Tom Ridge[1] Thomas Braibant[2]
Magnus O. Myreen[1] Jade Alglave[2]

[1]University of Cambridge [2]INRIA

October 30, 2008

Brief Contents

Full Contents

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:

  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
  sudo make install

(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

(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).
```

```
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

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
```

Part I

x86_coretypes

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 =⟨[proc : proc;
program_order_index : num]⟩

Part II

x86_types

type_abbrev address : Ximm

type_abbrev value : Ximm

type_abbrev eiid : num

type_abbrev reln : $'a\#'a \rightarrow$ bool

reg = $\textsc{Reg32}$ **of** Xreg | $\textsc{Reg1}$ **of** Xeflags | $\textsc{RegEIP}$

dirn = R | W

location = $\textsc{Location\_reg}$ **of** proc reg
| $\textsc{Location\_mem}$ **of** address

barrier = $\textsc{Lfence}$ | $\textsc{Sfence}$ | $\textsc{Mfence}$

action = $\textsc{Access}$ **of** dirn location value

event =⟨| eiid : eiid;
iiid : *iiid*;
action : *action*|⟩

event_structure =⟨| *procs* : proc set;
events : event set;
intra_causality : event reln;
atomicity : event set set|⟩

type_abbrev state_constraint : location → value option

type_abbrev view_orders : proc → 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

$(\text{next\_eiid} : \textsf{eiid\_state} \rightarrow (\textsf{eiid}\#\textsf{eiid\_state})\textsf{set})\, eiids = \{(\textsf{eiid}, eiids \cup \{\textsf{eiid}\}) \mid \textsf{eiid} \mid \neg(\textsf{eiid} \in eiids)\}$

$(\text{initial\_eiid\_state} : \textsf{eiid\_state}) = \{\}$

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*⦄

label = VIS **of** *'a* | TAU

type_abbrev machine_label : machine_visible_label label

type_abbrev lts_monad_label : lts_monad_visible_label label

LTS =⦃ *states* : *'state* set;
initial : *'state*;
final : (*'state*#*'value*)set;
trans : (*'state*#(*'visible* label)#*'state*)set⦄

Part III

x86_ast

Xrm = XR **of** Xreg(* register *)
| XM **of** (word2#Xreg) option Xreg option Ximm(* mem[2^{scale} * index + base + displacement] *)

(rm_is_memory_access(XM i b d) = **T**) $\wedge$
(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 = XI_RM **of** Xrm(* r/m32 *)
| XI **of** Ximm(* imm32 or imm8 (sign-extended) *)

Xbinop_name = XADD | XAND | XCMP | XOR | XSHL | XSHR | XSAR | XSUB | XTEST | XXOR

Xmonop_name = XDEC | XINC | XNOT | XNEG

Xcond = X_ALWAYS | X_E | 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
| XLEA **of** Xdest_src
| XPOP **of** Xrm
| XPUSH **of** Ximm_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 | XG1_NONE

Xpre_g2 = XBRANCH_TAKEN | XBRANCH_NOT_TAKEN | XG2_NONE

Xinst = XPREFIX **of** Xpre_g1 Xpre_g2 Xinstruction

Part IV

x86_axiomatic_model

loc $e =$
case $e.action$ **of**
  $\textsc{Access}\ d\ l\ v \rightarrow \textsc{Some}\ \ l$
$\|$ _ $\rightarrow$ $\textsc{None}$

value_of $e =$
case $e.action$ **of**
  $\textsc{Access}\ d\ l\ v \rightarrow \textsc{Some}\ \ v$
$\|$ _ $\rightarrow$ $\textsc{None}$

proc $e = e.iiid.\mathsf{proc}$

mem_load $e =$
case $e.action$ **of**
  $\textsc{Access}\ \mathrm{R}(\textsc{Location\_mem}\ a)v \rightarrow \mathbf{T}$
$\|$ _ $\rightarrow \mathbf{F}$

mem_store $e =$
case $e.action$ **of**
  $\textsc{Access}\ \mathrm{W}(\textsc{Location\_mem}\ a)v \rightarrow \mathbf{T}$
$\|$ _ $\rightarrow \mathbf{F}$

mem_barrier $e =$
case $e.action$ **of**
(* Barrier bar -> T || *)
_ $\rightarrow \mathbf{F}$

iiids $E = \{e.iiid \mid e\ \in\ E.events\}$

writes $E = \{e \mid e\ \in\ E.events \wedge \exists l\ v.e.action = \textsc{Access}\ \mathrm{W}\ l\ v\}$

reads $E = \{e \mid e\ \in\ E.events \wedge \exists l\ v.e.action = \textsc{Access}\ \mathrm{R}\ l\ v\}$

locked $E\ e = (e\ \in\ \textbf{bigunion}\ E.atomicity)$

po $E =$
$\{(e_1, e_2) \mid (e_1.iiid.\mathsf{proc} = e_2.iiid.\mathsf{proc}) \wedge$
  $e_1.iiid.\mathsf{program\_order\_index} \leq e_2.iiid.\mathsf{program\_order\_index} \wedge$
  $e_1\ \in\ E.events \wedge e_2\ \in\ E.events\}$

po_strict $E =$
$\{(e_1, e_2) \mid (e_1.iiid.\mathsf{proc} = e_2.iiid.\mathsf{proc}) \wedge$
  $e_1.iiid.\mathsf{program\_order\_index} < e_2.iiid.\mathsf{program\_order\_index} \wedge$
  $e_1\ \in\ E.events \wedge e_2\ \in\ E.events\}$

$\text{po\_iico } E = \text{po\_strict } E \cup E.\mathit{intra\_causality}$

$\text{well\_formed\_event\_structure } E =$
$(\forall \mathit{iiid}.\mathbf{finite}\{\mathsf{eiid} \mid \exists e \in (E.\mathit{events}).(e.\mathit{iiid} = \mathit{iiid}) \wedge (e.\mathsf{eiid} = \mathsf{eiid})\}) \wedge$
$(\mathbf{finite}\ E.\mathit{procs}) \wedge$
$(\forall e \in (E.\mathit{events}).\mathsf{proc}\ e \in E.\mathit{procs}) \wedge$
$(\forall e_1\ e_2 \in (E.\mathit{events}).(e_1.\mathsf{eiid} = e_2.\mathsf{eiid}) \wedge (e_1.\mathit{iiid} = e_2.\mathit{iiid}) \implies (e_1 = e_2)) \wedge$
$(\mathit{DOM}\ E.\mathit{intra\_causality}) \subseteq E.\mathit{events} \wedge$
$(\mathbf{range}\ E.\mathit{intra\_causality}) \subseteq E.\mathit{events} \wedge$
$\text{acyclic}(E.\mathit{intra\_causality}) \wedge$
$(\forall (e_1, e_2) \in (E.\mathit{intra\_causality}).(e_1.\mathit{iiid} = e_2.\mathit{iiid})) \wedge$
$(\forall (e_1, e_2) \in (E.\mathit{intra\_causality}).\neg(\text{mem\_store } e_1)) \wedge$
$(\forall (e_1 \in \text{writes } E) e_2.$
$\neg(e_1 = e_2) \wedge$
$(e_2 \in \text{writes } E \vee e_2 \in \text{reads } E) \wedge$
$(e_1.\mathit{iiid} = e_2.\mathit{iiid}) \wedge$
$(\text{loc } e_1 = \text{loc } e_2) \wedge$
$(\exists p\ r.\, \text{loc } e_1 = \textsc{Some}\ (\textsc{Location\_reg}\ p\ r))$
$\implies$
$(e_1, e_2) \in E.\mathit{intra\_causality}^+ \vee$
$(e_2, e_1) \in E.\mathit{intra\_causality}^+) \wedge$
$(\forall \mathit{es} \in (E.\mathit{atomicity}).\exists e \in \mathit{es}.\, \text{mem\_load } e) \wedge$
$\mathit{PER}\ E.\mathit{events}\ E.\mathit{atomicity} \wedge$
$(\forall \mathit{es} \in (E.\mathit{atomicity}).\forall e_1\ e_2 \in \mathit{es}.(e_1.\mathit{iiid} = e_2.\mathit{iiid})) \wedge$
$(\forall \mathit{es} \in (E.\mathit{atomicity}).\forall e_1 \in \mathit{es}.\forall e_2 \in (E.\mathit{events}).(e_1.\mathit{iiid} = e_2.\mathit{iiid}) \implies e_2 \in \mathit{es}) \wedge$
$(\forall e \in (E.\mathit{events}).\forall p\ r.(\text{loc } e = \textsc{Some}\ (\textsc{Location\_reg}\ p\ r)) \implies (p = \mathsf{proc}\ e))$

$\text{sub\_event\_structure } E'\ E =$
$(E'.\mathit{procs} = E.\mathit{procs}) \wedge$
$E'.\mathit{events} \subseteq E.\mathit{events} \wedge$
$(E'.\mathit{atomicity} = \mathit{PER\_RESTRICT}\ E.\mathit{atomicity}\ E'.\mathit{events}) \wedge$
$(E'.\mathit{intra\_causality} = E.\mathit{intra\_causality}|_{E'.\mathit{events}})$

$\text{preserved\_program\_order } E =$
$\{(e_1, e_2) \mid (e_1, e_2) \in (\text{po\_strict } E) \wedge$
$((\exists p\ r.(\text{loc } e_1 = \text{loc } e_2) \wedge$
$(\text{loc } e_1 = \textsc{Some}\ (\textsc{Location\_reg}\ p\ r))) \vee$
$(\text{mem\_load } e_1 \wedge \text{mem\_load } e_2) \vee$
$(\text{mem\_store } e_1 \wedge \text{mem\_store } e_2) \vee$
$(\text{mem\_load } e_1 \wedge \text{mem\_store } e_2) \vee$
$(\text{mem\_store } e_1 \wedge \text{mem\_load } e_2 \wedge (\text{loc } e_1 = \text{loc } e_2)) \vee$
$((\text{mem\_load } e_1 \vee \text{mem\_store } e_1) \wedge \text{locked } E\ e_2) \vee$
$(\text{locked } E\ e_1 \wedge (\text{mem\_load } e_2 \vee \text{mem\_store } e_2)))\}$

$\text{viewed\_events } E\ p =$
$\{e \mid e \in E.\mathit{events} \wedge ((\mathsf{proc}\ e = p) \vee \text{mem\_store } e)\}$

view_orders_well_formed $E\ vo =$
$(\forall p \in (E.procs).\ \text{linear\_order}(vo\ p)(\text{viewed\_events}\ E\ p) \wedge$
$\forall e \in (\text{viewed\_events}\ E\ p).\mathbf{finite}\{e' \mid (e', e) \in (vo\ p)\}) \wedge$
$(\forall p.\neg(p \in E.procs) \implies (vo\ p = \{\}))$

get_l_stores $E\ l =$
$\{e \mid e \in E.events \wedge \text{mem\_store}\ e \wedge (\text{loc}\ e = \text{Some}\ l)\}$

write_serialization_candidates_old $E =$
let $per\_location\_store\_sets = \{es \mid \exists l.es = \text{get\_l\_stores}\ E\ l\}$ **in**
let $per\_location\_store\_set\_linearisations = \{\text{strict\_linearisations}\ es \mid es \in per\_location\_store\_sets\}$ **in**
let $choices = all\_choices\ per\_location\_store\_set\_linearisations$ **in**
$\{\mathbf{bigunion}\ lin \mid lin \in choices\}$

write_serialization_candidates $E\ cand =$
$(\forall (e_1, e_2) \in cand.$
$\exists l.e_1 \in (\text{get\_l\_stores}\ E\ l) \wedge e_2 \in (\text{get\_l\_stores}\ E\ l)) \wedge$
$(\forall l.\ \text{strict\_linear\_order}(\ cand|_{(\text{get\_l\_stores}\ E\ l)})$
$(\text{get\_l\_stores}\ E\ l))$

lock_serialization_candidates $E =$
let $lin\_ec = \text{strict\_linearisations}\ E.atomicity$ **in**
$\{\{(e_1, e_2) \mid \exists (es_1, es_2) \in lin.e_1 \in es_1 \wedge e_2 \in es_2\}$
$\mid lin \in lin\_ec\}$

reads_from_map_candidates_old $E =$
let $reads\_and\_their\_possible\_writes =$
$\{(er, ews) \mid er \in E.events \wedge$
$\exists l\ v.(er.action = \text{Access R}\ l\ v) \wedge$
$(ews = \{ew \mid ew \in E.events \wedge (ew.action = \text{Access W}\ l\ v)\})\}$ **in**
$\{rfmap \mid (\mathbf{range}\ rfmap) \subseteq (DOM\ reads\_and\_their\_possible\_writes) \wedge$
$\forall (ew, er) \in rfmap.\exists ews.(er, ews) \in reads\_and\_their\_possible\_writes \wedge ew \in ews\}$

reads_from_map_candidates $E\ rfmap =$
$\forall (ew, er) \in rfmap.er \in E.events \wedge ew \in E.events \wedge$
$\exists l\ v.(er.action = \text{Access R}\ l\ v) \wedge$
$(ew.action = \text{Access W}\ l\ v)$

happens_before $E\ X =$
$E.intra\_causality\ \cup$
$(\text{preserved\_program\_order}\ E) \cup$
$X.write\_serialization\ \cup$
$X.lock\_serialization\ \cup$
$X.rfmap$

$$\text{check\_causality } E\ vo\ happensbefore =$$
$$\quad \forall p \in (E.procs).\ \text{acyclic}((\text{strict}(vo\ p)) \cup\ happensbefore)$$

$$\begin{array}{l}
\text{check_rfmap_written } E\ vo\ rfmap = \\
\forall p \in (E.procs). \\
\quad \forall (ew, er) \in (\,rfmap|_{(\text{viewed_events } E\ p)}). \\
\qquad \forall ew' \in (\text{writes } E). \\
\qquad\quad \neg(ew = ew') \wedge (ew, ew') \in (vo\ p) \wedge (ew', er) \in (vo\ p) \\
\qquad\quad\ \implies \neg(\text{loc } ew = \text{loc } ew')
\end{array}$$

$$\begin{array}{l}
\text{check_rfmap_initial } E\ vo\ rfmap\ initial_state = \\
\forall p \in (E.procs). \\
\quad \forall er \in (((\text{reads } E) \setminus (\textbf{range}\ rfmap)) \\
\qquad\qquad \cap\ \text{viewed_events } E\ p). \\
\qquad \exists l\ v.(er.action = \textsc{Access}\ \mathrm{R}\ l\ v) \wedge \\
\qquad\qquad (initial_state\ l = \textsc{Some}\ \ v) \wedge \\
\qquad\qquad \forall ew' \in \text{writes } E. \\
\qquad\qquad\quad (ew', er) \in (vo\ p) \implies \neg(\text{loc } ew' = \text{loc } er)
\end{array}$$

$$\begin{array}{l}
\text{state_updates } E\ vo\ write_serialization\ l = \\
\textbf{case}\ \ l\ \ \textbf{of} \\
\quad \textsc{Location_mem}\ a \rightarrow \\
\qquad \{\text{value_of } ew \mid ew\ \in\ maximal_elements(\text{get_l_stores } E\ l)write_serialization\} \\
\ \|\ \textsc{Location_reg}\ p\ r \rightarrow \\
\qquad \{\text{value_of } ew \mid ew\ \in\ maximal_elements(\text{get_l_stores } E\ l)(vo\ p)\}
\end{array}$$

$$\begin{array}{l}
\text{check_final } E\ vo\ initial_state\ final_state_opt\ write_serialization = \\
\textbf{if finite}\ E.events\ \ \textbf{then} \\
\exists\,\text{final_state}\,.(final_state_opt = \textsc{Some}\ \ \text{final_state}) \wedge \\
\forall l.\textbf{if}\ (\text{state_updates } E\ vo\ write_serialization\ l) = \{\}\ \textbf{then} \\
\qquad\quad \text{final_state } l = initial_state\ l \\
\qquad \textbf{else} \\
\qquad\quad (\text{final_state } l) \in (\text{state_updates } E\ vo\ write_serialization\ l) \\
\ \textbf{else} \\
final_state_opt = \textsc{None}
\end{array}$$

$$\begin{array}{l}
\text{state_updates_mem } E\ write_serialization\ a = \\
\{\text{value_of } ew \mid ew\ \in\ maximal_elements(\text{get_l_stores } E(\textsc{Location_mem}\ a))write_serialization\}
\end{array}$$

$$\begin{array}{l}
(\text{check_final_mem } E\ initial_state\ write_serialization\ \textsc{None} = \\
\neg(\textbf{finite}\ E.events)) \wedge \\
(\text{check_final_mem } E\ initial_state\ write_serialization(\textsc{Some}\ \ \text{final_state}) = \\
\textbf{finite}\ E.events \wedge \\
\forall a.\textbf{if}\ (\text{state_updates_mem } E\ write_serialization\ a) = \{\}\ \textbf{then} \\
\qquad\quad \text{final_state } a = initial_state(\textsc{Location_mem}\ a) \\
\qquad \textbf{else} \\
\qquad\quad (\text{final_state } a) \in (\text{state_updates_mem } E\ write_serialization\ a))
\end{array}$$

check_atomicity E vo =
$\quad \forall p \in (E.procs).\forall es \in (E.atomicity).$
$\quad\quad \forall e_1\ e_2 \in es.(e_1, e_2) \in (vo\ p) \implies$
$\quad\quad\quad \forall e.(e_1, e) \in (vo\ p) \wedge (e, e_2) \in (vo\ p) \implies e \in es$

valid_execution E X =
$\quad$ view_orders_well_formed E $X.vo$ $\wedge$
$\quad$ $X.write\_serialization \in$ write_serialization_candidates E $\wedge$
$\quad$ $X.lock\_serialization \in$ lock_serialization_candidates E $\wedge$
$\quad$ $X.rfmap \in$ reads_from_map_candidates E $\wedge$
$\quad$ check_causality E $X.vo$(happens_before E X) $\wedge$
$\quad$ check_rfmap_written E $X.vo$ $X.rfmap$ $\wedge$
$\quad$ check_rfmap_initial E $X.vo$ $X.rfmap$ $X.initial\_state$ $\wedge$
$\quad$ check_atomicity E $X.vo$

restrict_execution_witness X E =
$\langle\!|$ $initial\_state := X.initial\_state;$
$\quad$ (*final_state_opt := ... *)
$\quad vo := (\lambda p.\ (X.vo\ p)|_{E.events});$
$\quad write\_serialization := X.write\_serialization|_{E.events};$
$\quad lock\_serialization := X.lock\_serialization|_{E.events};$
$\quad rfmap := RRESTRICT\, X.rfmap\ E.events |\!\rangle$

Part V

x86_niceness_statement

$$\text{nice\_execution}\ E\ X = \forall p \in (E.procs).$$
$$(\text{po\_strict}\ E)|_{(\text{viewed\_events}\ E\ p\ \setminus\ \text{mem\_store})} \subseteq (X.vo\ p)$$

$$\begin{aligned}
&\text{niceness_thm} = \\
&\forall E\ X.(\text{well_formed_event_structure}\ E \wedge \\
&\qquad \text{valid_execution}\ E\ X) \implies \\
&\qquad \exists X'.\,\text{valid_execution}\ E\ X' \wedge \text{nice_execution}\ E\ X' \wedge \\
&\qquad\quad (X' \llparenthesis\ vo := (\lambda p.\{\}) \rrparenthesis = X \llparenthesis\ vo := (\lambda p.\{\}) \rrparenthesis)
\end{aligned}$$

Part VI

x86_seq_monad

type_abbrev $\mathsf{x86\_state} : (\mathsf{Xreg} \to \mathsf{word32})\#$(* - general-purpose 32-bit registers *)
$(\mathsf{word32})\#$(* - eip *)
$(\mathsf{Xeflags} \to \mathsf{bool\ \ option})\#$(* - eflags *)
$(\mathsf{word32} \to \mathsf{word8\ \ option})$(* - unsegmented memory *)

$\text{XREAD\_REG}\ \ i((r, eip, f, m) : \mathsf{x86\_state}) = r\ i$

$\text{XREAD\_EIP}((r, eip, f, m) : \mathsf{x86\_state}) = eip$

$\text{XREAD\_EFLAG}\ \ i((r, eip, f, m) : \mathsf{x86\_state}) = f\ i$

$\text{XREAD\_MEM}\ \ i((r, eip, f, m) : \mathsf{x86\_state}) = m\ i$

$\text{XWRITE\_REG}\ \ i\ x((r, eip, f, m) : \mathsf{x86\_state}) = ((i = +x)r, eip, f, m) : \mathsf{x86\_state}$

$\text{XWRITE\_EIP}\ \ x((r, eip, f, m) : \mathsf{x86\_state}) = (r, x, f, m) : \mathsf{x86\_state}$

$\text{XWRITE\_EFLAG}\ \ i\ x((r, eip, f, m) : \mathsf{x86\_state}) = (r, eip, (i = +x)f, m) : \mathsf{x86\_state}$

$\text{XWRITE\_MEM}\ \ i\ x((r, eip, f, m) : \mathsf{x86\_state}) = (r, eip, f, (i = +x)m) : \mathsf{x86\_state}$

$\text{XREAD\_MEM\_BYTES}\ \ n\ a\ s =$
if $n = 0$ **then** $[\,]$ **else** $\text{XREAD\_MEM}\ \ a\ s \in \text{XREAD\_MEM\_BYTES}(n-1)(a + 1w)s$

type_abbrev $\mathsf{M} : \mathsf{x86\_state} \to ('a\#\mathsf{x86\_state})\ \mathsf{option}$

$(\text{constT\_seq} : 'a \to 'a\ \mathsf{M})x = \lambda y.\textsc{Some}\ (x, y)$

$(\text{addT\_seq} : 'a \to 'b\ \mathsf{M} \to ('a\#'b)\mathsf{M})x\ s =$
$\lambda y.$**case** $s\ y$ **of** $\textsc{None} \to \textsc{None} \parallel \textsc{Some}\ (z, t) \to \textsc{Some}\ ((x, z), t)$

$(\text{lockT\_seq} : 'a\ \mathsf{M} \to 'a\ \mathsf{M})s = s$

$(\text{failureT\_seq} : 'a\ \mathsf{M}) = \lambda y.\textsc{None}$

$(\text{seqT\_seq} : 'a\ \mathsf{M} \to ('a \to 'b\ \mathsf{M}) \to 'b\ \mathsf{M})s\ f =$
$\lambda y.$**case** $s\ y$ **of** $\textsc{None} \to \textsc{None} \parallel \textsc{Some}\ (z, t) \to f\ z\ t$

$(\text{parT\_seq} : 'a\ \mathsf{M} \to 'b\ \mathsf{M} \to ('a\#'b)\mathsf{M})s\ t =$
$\lambda y.$**case** $s\ y$ **of** $\textsc{None} \to \textsc{None} \parallel \textsc{Some}\ (a, z) \to$
  case $t\ z$ **of** $\textsc{None} \to \textsc{None} \parallel \textsc{Some}\ (b, x) \to \textsc{Some}\ ((a, b), x)$

$(\text{parT\_unit\_seq} : \mathsf{unit\ M} \to \mathsf{unit\ M} \to \mathsf{unit\ M})s\ t =$
$\lambda y.\mathbf{case}\ \ s\ y\ \ \mathbf{of}\ \ \textsc{None} \to \textsc{None} \parallel \textsc{Some}\ (a, z) \to$
$\quad \mathbf{case}\ \ t\ z\ \ \mathbf{of}\ \ \textsc{None} \to \textsc{None} \parallel \textsc{Some}\ (b, x) \to \textsc{Some}\ ((), x)$

$(\text{write\_reg\_seq}\ \ ii\ r\ x) : \mathsf{unit\ M} =$
$\lambda s.\textsc{Some}\ ((), \text{XWRITE\_REG}\ \ r\ x\ s)$

$(\text{read\_reg\_seq}\ \ ii\ r) : \mathsf{Ximm\ M} =$
$\lambda s.\textsc{Some}\ (\text{XREAD\_REG}\ \ r\ s, s)$

$(\text{write\_eflag\_seq}\ \ ii\ f\ x) : \mathsf{unit\ M} =$
$(\lambda s.\textsc{Some}\ ((), \text{XWRITE\_EFLAG}\ \ f\ x\ s))$

$(\text{read\_eflag\_seq}\ \ ii\ f) : \mathsf{bool\ M} =$
$(\lambda s.\mathbf{case}\ \ \text{XREAD\_EFLAG}\ \ f\ s\ \ \mathbf{of}\ \ \textsc{None} \to \textsc{None} \parallel \textsc{Some}\ \ b \to \textsc{Some}\ (b, s))$

$(\text{write\_eip\_seq}\ \ ii\ x) : \mathsf{unit\ M} =$
$\lambda s.\textsc{Some}\ ((), \text{XWRITE\_EIP}\ \ x\ s)$

$(\text{read\_eip\_seq}\ \ ii) : \mathsf{Ximm\ M} =$
$\lambda s.\textsc{Some}\ (\text{XREAD\_EIP}\ \ s, s)$

$(\text{write\_mem\_seq}\ \ ii\ a\ x) : \mathsf{unit\ M} =$
$(\lambda s.\mathbf{case}\ \ \text{XREAD\_MEM}\ \ a\ s\ \ \mathbf{of}\ \ \textsc{None} \to \textsc{None} \parallel \textsc{Some}\ \ y \to \textsc{Some}\ ((), \text{XWRITE\_MEM}\ \ a(\textsc{Some}\ \ x)s))$

$(\text{read\_mem\_seq}\ \ ii\ a) : \mathsf{word8\ M} =$
$(\lambda s.\mathbf{case}\ \ \text{XREAD\_MEM}\ \ a\ s\ \ \mathbf{of}\ \ \textsc{None} \to \textsc{None} \parallel \textsc{Some}\ \ x \to \textsc{Some}\ (x, s))$

$(\text{read\_m32\_seq}\ \ ii\ a) : \mathsf{Ximm\ M} =$
$\text{seqT\_seq}(\text{parT\_seq}(\text{read\_mem\_seq}\ \ ii(a + 0w))(\text{parT\_seq}(\text{read\_mem\_seq}\ \ ii(a + 1w))$
$\qquad (\text{parT\_seq}(\text{read\_mem\_seq}\ \ ii(a + 2w))(\text{read\_mem\_seq}\ \ ii(a + 3w)))))$
$\quad (\lambda(\mathit{x0}, \mathit{x1}, \mathit{x2}, \mathit{x3}).\ \text{constT\_seq}(\mathit{bytes2word}[\mathit{x0}; \mathit{x1}; \mathit{x2}; \mathit{x3}]))$

$(\text{write\_m32\_seq}\ \ ii\ a\ w) : \mathsf{unit\ M} =$
$(\mathbf{let}\ \ bs = \mathit{word2bytes}\ 4\ w\ \ \mathbf{in}$
$\quad \text{parT\_unit\_seq}(\text{write\_mem\_seq}\ \ ii(a + 0w)(\mathit{EL}\ 0\ bs))(\text{parT\_unit\_seq}(\text{write\_mem\_seq}\ \ ii(a + 1w)(\mathit{EL}\ 1\ bs))$
$(\text{parT\_unit\_seq}(\text{write\_mem\_seq}\ \ ii(a + 2w)(\mathit{EL}\ 2\ bs))(\text{write\_mem\_seq}\ \ ii(a + 3w)(\mathit{EL}\ 3\ bs)))))$

$(\text{constT} : {'a} \to {'a}\ \mathsf{M}) = \text{constT\_seq}$

$(\text{addT} : {'a} \to {'b}\ \mathsf{M} \to ({'a}\#{'b})\mathsf{M}) = \text{addT\_seq}$

$(\text{lockT} : \mathsf{unit\ M} \rightarrow \mathsf{unit\ M}) = \text{lockT\_seq}$

$(\text{failureT} : \mathsf{unit\ M}) = \text{failureT\_seq}$

$(\text{seqT} : {}'a\ \mathsf{M} \rightarrow (({}'a \rightarrow {}'b\ \mathsf{M}) \rightarrow {}'b\ \mathsf{M})) = \text{seqT\_seq}$

$(\text{parT} : {}'a\ \mathsf{M} \rightarrow {}'b\ \mathsf{M} \rightarrow ({}'a\#{}'b)\mathsf{M}) = \text{parT\_seq}$

$(\text{parT\_unit} : \mathsf{unit\ M} \rightarrow \mathsf{unit\ M} \rightarrow \mathsf{unit\ M}) = \text{parT\_unit\_seq}$

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

$(\text{read\_reg} : iiid \rightarrow \mathsf{Xreg} \rightarrow \mathsf{Ximm\ M}) = \text{read\_reg\_seq}$

$(\text{write\_eip} : iiid \rightarrow \mathsf{Ximm} \rightarrow \mathsf{unit\ M}) = \text{write\_eip\_seq}$

$(\text{read\_eip} : iiid \rightarrow \mathsf{Ximm\ M}) = \text{read\_eip\_seq}$

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

$(\text{read\_eflag} : iiid \rightarrow \mathsf{Xeflags} \rightarrow \mathsf{bool\ M}) = \text{read\_eflag\_seq}$

$(\text{write\_m32} : iiid \rightarrow \mathsf{Ximm} \rightarrow \mathsf{Ximm} \rightarrow \mathsf{unit\ M}) = \text{write\_m32\_seq}$

$(\text{read\_m32} : iiid \rightarrow \mathsf{Ximm} \rightarrow \mathsf{Ximm\ M}) = \text{read\_m32\_seq}$

$\text{option\_apply}\ \ x\ f = \mathbf{if}\ \ x = \text{N}\textsc{one}\ \ \mathbf{then}\ \ \text{N}\textsc{one}\ \ \mathbf{else}\ \ f(\mathbf{the}\ \ x)$

Part VII

x86_event_monad

$\textbf{type\_abbrev}\ \mathsf{M} : \mathsf{eiid\_state} \rightarrow ((\mathsf{eiid\_state}\#'a\#\mathsf{event\_structure})\mathsf{set})$

$\text{event\_structure\_empty} = \langle\!| \mathit{procs} := \{\}; \mathit{events} := \{\}; \mathit{intra\_causality} := \{\}; \mathit{atomicity} := \{\} |\!\rangle$

$\text{event\_structure\_lock}\ \mathit{es} = \langle\!| \mathit{procs} := \mathit{es.procs}; \mathit{events} := \mathit{es.events}; \mathit{intra\_causality} := \mathit{es.intra\_causality}; \mathit{atomicity} := \textbf{if}$

$$
\begin{aligned}
&\text{event\_structure\_union}\ \mathit{es}_1\ \mathit{es}_2 = \\
&\qquad \langle\!| \mathit{procs} := \mathit{es}_1.\mathit{procs}\ \cup\ \mathit{es}_2.\mathit{procs}; \\
&\qquad\ \ \mathit{events} := \mathit{es}_1.\mathit{events}\ \cup\ \mathit{es}_2.\mathit{events}; \\
&\qquad\ \ \mathit{intra\_causality} := \mathit{es}_1.\mathit{intra\_causality}\ \cup\ \mathit{es}_2.\mathit{intra\_causality}; \\
&\qquad\ \ \mathit{atomicity} := \mathit{es}_1.\mathit{atomicity}\ \cup\ \mathit{es}_2.\mathit{atomicity} |\!\rangle
\end{aligned}
$$

$$
\begin{aligned}
&\text{event\_structure\_bigunion}(\mathit{ess} : \mathsf{event\_structure\ set}) = \\
&\qquad \langle\!| \mathit{procs} := \textbf{bigunion}\{\mathit{es.procs} \mid \mathit{es}\ \in\ \mathit{ess}\}; \\
&\qquad\ \ \mathit{events} := \textbf{bigunion}\{\mathit{es.events} \mid \mathit{es}\ \in\ \mathit{ess}\}; \\
&\qquad\ \ \mathit{intra\_causality} := \textbf{bigunion}\{\mathit{es.intra\_causality} \mid \mathit{es}\ \in\ \mathit{ess}\}; \\
&\qquad\ \ \mathit{atomicity} := \textbf{bigunion}\{\mathit{es.atomicity} \mid \mathit{es}\ \in\ \mathit{ess}\} |\!\rangle
\end{aligned}
$$

$$
\begin{aligned}
&\text{event\_structure\_seq\_union}\ \mathit{es}_1\ \mathit{es}_2 = \\
&\qquad \langle\!| \mathit{procs} := \mathit{es}_1.\mathit{procs}\ \cup\ \mathit{es}_2.\mathit{procs}; \\
&\qquad\ \ \mathit{events} := \mathit{es}_1.\mathit{events}\ \cup\ \mathit{es}_2.\mathit{events}; \\
&\qquad\ \ \mathit{intra\_causality} := \mathit{es}_1.\mathit{intra\_causality} \\
&\qquad\qquad\qquad\qquad \cup\ \mathit{es}_2.\mathit{intra\_causality} \\
&\qquad\qquad\qquad\qquad \cup \{(e_1, e_2) \\
&\qquad\qquad\qquad\qquad\qquad \mid e_1\ \in (\mathit{maximal\_elements}\ \mathit{es}_1.\mathit{events}\ \mathit{es}_1.\mathit{intra\_causality}) \\
&\qquad\qquad\qquad\qquad\qquad\quad \wedge\ e_2\ \in (\mathit{minimal\_elements}\ \mathit{es}_2.\mathit{events}\ \mathit{es}_2.\mathit{intra\_causality})\}; \\
&\qquad\ \ \mathit{atomicity} := \mathit{es}_1.\mathit{atomicity}\ \cup\ \mathit{es}_2.\mathit{atomicity} |\!\rangle
\end{aligned}
$$

$$
\begin{aligned}
&(\text{mapT\_ev} : ('a \rightarrow 'b) \rightarrow 'a\ \mathsf{M} \rightarrow 'b\ \mathsf{M}) f\ s = \\
&\lambda \mathit{eiid\_next} : \mathsf{eiid\_state}. \\
&\quad \textbf{let}\ \ t = s\ \mathit{eiid\_next}\ \ \textbf{in} \\
&\qquad \{(\mathit{eiid\_next}', f\ x, \mathit{es}) \\
&\qquad\qquad\qquad \mid (\mathit{eiid\_next}', x, \mathit{es}) \in\ t\}
\end{aligned}
$$

$$
\begin{aligned}
&(\text{choiceT\_ev} : 'a\ \mathsf{M} \rightarrow 'a\ \mathsf{M} \rightarrow 'a\ \mathsf{M}) s\ s' = \\
&\lambda \mathit{eiid\_next} : \mathsf{eiid\_state}. s\ \mathit{eiid\_next}\ \cup\ s'\ \mathit{eiid\_next}
\end{aligned}
$$

$(\text{constT\_ev} : 'a \rightarrow 'a\ \mathsf{M}) x = \lambda \mathit{eiid\_next}.\{(\mathit{eiid\_next}, x, \text{event\_structure\_empty})\}$

$$
\begin{aligned}
&(\text{discardT\_ev} : 'a\ \mathsf{M} \rightarrow \mathsf{unit}\ \mathsf{M}) s = \\
&\lambda \mathit{eiid\_next}.\textbf{let}\ (t : (\mathsf{eiid\_state}\#'a\#\mathsf{event\_structure})\mathsf{set}) = s\ \mathit{eiid\_next}\ \ \textbf{in} \\
&\quad \textbf{image}(\lambda(\mathit{eiid\_next}', v, \mathit{es}).(\mathit{eiid\_next}', (), \mathit{es}))t
\end{aligned}
$$

$(\text{addT\_ev} : {}'a \to {}'b\ \mathsf{M} \to ({}'a\#{}'b)\mathsf{M})x\ s =$
$\lambda \mathit{eiid\_next}.\mathbf{let}\ (t : (\mathsf{eiid\_state}\#{}'b\#\mathsf{event\_structure})\mathsf{set}) = s\ \mathit{eiid\_next}\ \ \mathbf{in}$
$\quad \mathbf{image}(\lambda(\mathit{eiid\_next}', v, \mathit{es}).(\mathit{eiid\_next}', (x, v), \mathit{es}))t$

$(\text{lockT\_ev} : {}'a\ \mathsf{M} \to {}'a\ \mathsf{M})s =$
$\lambda \mathit{eiid\_next}.\mathbf{let}\ (t : (\mathsf{eiid\_state}\#{}'a\#\mathsf{event\_structure})\mathsf{set}) = s\ \mathit{eiid\_next}\ \ \mathbf{in}$
$\quad \mathbf{image}(\lambda(\mathit{eiid\_next}', v, \mathit{es}).(\mathit{eiid\_next}', v, \text{event\_structure\_lock}\ \ \mathit{es}))t$

$(\text{failureT\_ev} : {}'a\ \mathsf{M}) = \lambda \mathit{eiid\_next}.\{\}$

$(\text{seqT\_ev} : {}'a\ \mathsf{M} \to ({}'a \to {}'b\ \mathsf{M}) \to {}'b\ \mathsf{M})s\ f =$
$\lambda \mathit{eiid\_next} : \mathsf{eiid\_state}.$
$\mathbf{let}\ \ t = s\ \mathit{eiid\_next}\ \ \mathbf{in}$
$\quad \mathbf{bigunion}\{\mathbf{let}\ \ t' = f\ x\ \mathit{eiid\_next}'\ \ \mathbf{in}$
$\qquad\qquad \{(\mathit{eiid\_next}'', x', \text{event\_structure\_seq\_union}\ \ \mathit{es}\ \mathit{es}')$
$\qquad\qquad \mid (\mathit{eiid\_next}'', x', \mathit{es}') \in\ t'\}$
$\qquad \mid (\mathit{eiid\_next}', x, \mathit{es}) \in\ t\}$

$(\text{parT\_ev} : {}'a\ \mathsf{M} \to {}'b\ \mathsf{M} \to ({}'a\#{}'b)\mathsf{M})s\ s' =$
$\lambda \mathit{eiid\_next} : \mathsf{eiid\_state}.$
$\mathbf{let}\ \ t = s\ \mathit{eiid\_next}\ \ \mathbf{in}$
$\quad \mathbf{bigunion}\{\mathbf{let}\ \ t' = s'\ \mathit{eiid\_next}'\ \ \mathbf{in}$
$\qquad\qquad \{(\mathit{eiid\_next}'', (x, x'), \text{event\_structure\_union}\ \ \mathit{es}\ \mathit{es}')$
$\qquad\qquad \mid (\mathit{eiid\_next}'', x', \mathit{es}') \in\ t'\}$
$\qquad \mid (\mathit{eiid\_next}', x, \mathit{es}) \in\ t\}$

$(\text{parT\_unit\_ev} : \mathsf{unit}\ \mathsf{M} \to \mathsf{unit}\ \mathsf{M} \to \mathsf{unit}\ \mathsf{M})s\ s' =$
$\lambda \mathit{eiid\_next} : \mathsf{eiid\_state}.$
$\mathbf{let}\ \ t = s\ \mathit{eiid\_next}\ \ \mathbf{in}$
$\quad \mathbf{bigunion}\{\mathbf{let}\ \ t' = s'\ \mathit{eiid\_next}'\ \ \mathbf{in}$
$\qquad\qquad \{(\mathit{eiid\_next}'', (), \text{event\_structure\_union}\ \ \mathit{es}\ \mathit{es}')$
$\qquad\qquad \mid (\mathit{eiid\_next}'', (), \mathit{es}') \in\ t'\}$
$\qquad \mid (\mathit{eiid\_next}', (), \mathit{es}) \in\ t\}$

$(\text{write\_location\_ev}\ \ \mathit{ii}\ l\ x) : \mathsf{unit}\ \mathsf{M} =$
$\lambda \mathit{eiid\_next}.\{(\mathit{eiid\_next}',$
$\qquad (),$
$\qquad \langle\!| \mathit{procs} := \{\mathit{ii}.\mathsf{proc}\};$
$\qquad\quad \mathit{events} := \{\langle\!| \mathsf{eiid} := \mathit{eiid}';$
$\qquad\qquad\qquad \mathit{iiid} := \mathit{ii};$
$\qquad\qquad\qquad \mathit{action} := \text{Access W}\ l\ x |\!\rangle\};$
$\qquad\quad \mathit{intra\_causality} := \{\};$
$\qquad\quad \mathit{atomicity} := \{\}|\!\rangle) \mid (\mathit{eiid}', \mathit{eiid\_next}') \in\ \text{next\_eiid}\ \ \mathit{eiid\_next}\}$

$(\text{read\_location\_ev}\ \ ii\ l) : \mathsf{value\ M} =$
$\lambda eiid\_next.\{(eiid\_next',$
$\quad x,$
$\quad ⦇\ procs := \{ii.\mathsf{proc}\};$
$\quad\quad events := \{⦇\ \mathsf{eiid} := eiid';$
$\quad\quad\quad iiid := ii;$
$\quad\quad\quad action := \textsc{Access}\ \mathrm{R}\ l\ x⦈\};$
$\quad\quad intra\_causality := \{\};$
$\quad\quad atomicity := \{\}⦈)$
$\quad \mid x\ \in\ UNIV \wedge (eiid', eiid\_next') \in\ \text{next\_eiid}\ \ eiid\_next\}$

$(\text{write\_reg\_ev}\ \ ii\ r\ x) : \mathsf{unit\ M} =$
$\text{write\_location\_ev}\ \ ii(\textsc{Location\_reg}\ ii.\mathsf{proc}(\textsc{Reg32}\ r))x$

$(\text{read\_reg\_ev}\ \ ii\ r) : \mathsf{value\ M} =$
$\text{read\_location\_ev}\ \ ii(\textsc{Location\_reg}\ ii.\mathsf{proc}(\textsc{Reg32}\ r))$

$(\text{write\_eip\_ev}\ \ ii\ x) : \mathsf{unit\ M} =$
$\text{write\_location\_ev}\ \ ii(\textsc{Location\_reg}\ ii.\mathsf{proc}\ \textsc{RegEIP})x$

$(\text{read\_eip\_ev}\ \ ii) : \mathsf{value\ M} =$
$\text{read\_location\_ev}\ \ ii(\textsc{Location\_reg}\ ii.\mathsf{proc}\ \textsc{RegEIP})$

$(\text{write\_eflag\_ev}\ \ ii\ f\ bo) : \mathsf{unit\ M} =$
case bo **of**
$\textsc{Some}\ \ b \rightarrow$
$(\text{write\_location\_ev}\ \ ii(\textsc{Location\_reg}\ ii.\mathsf{proc}(\textsc{Reg1}\ f))(\mathbf{if}\ \ b\ \ \mathbf{then}\ \ 1w\ \ \mathbf{else}\ \ 0w))$
$\parallel \textsc{None} \rightarrow$
$\quad \text{choiceT\_ev}$
$\quad\quad (\text{write\_location\_ev}\ \ ii(\textsc{Location\_reg}\ ii.\mathsf{proc}(\textsc{Reg1}\ f))0w)$
$\quad\quad (\text{write\_location\_ev}\ \ ii(\textsc{Location\_reg}\ ii.\mathsf{proc}(\textsc{Reg1}\ f))1w)$

$(\text{read\_eflag\_ev}\ \ ii\ f) : \mathsf{bool\ M} =$
$\text{mapT\_ev}(\lambda x.(x = 0w))$
$(\text{read\_location\_ev}\ \ ii(\textsc{Location\_reg}\ ii.\mathsf{proc}(\textsc{Reg1}\ f)))$

$\text{aligned32}\ \ a = ((a \&\& 3w) = 0w)$

$(\text{write\_m32\_ev}\ \ ii\ a\ x) : \mathsf{unit\ M} =$
if $\text{aligned32}\ \ a$ **then**
$\text{write\_location\_ev}\ \ ii(\textsc{Location\_mem}\ a)x$
else
failureT_ev

$(\text{read\_m32\_ev}\ \ ii\ a) : \mathsf{Ximm\ M} =$
if aligned32 a **then**
read_location_ev $ii(\textsc{Location\_mem}\ a)$
else
failureT_ev

$(\text{constT} : {}'a \rightarrow {}'a\ \mathsf{M}) = \text{constT\_ev}$

$(\text{addT} : {}'a \rightarrow {}'b\ \mathsf{M} \rightarrow ({}'a\#{}'b)\mathsf{M}) = \text{addT\_ev}$

$(\text{lockT} : \mathsf{unit\ M} \rightarrow \mathsf{unit\ M}) = \text{lockT\_ev}$

$(\text{failureT} : \mathsf{unit\ M}) = \text{failureT\_ev}$

$(\text{seqT} : {}'a\ \mathsf{M} \rightarrow (({}'a \rightarrow {}'b\ \mathsf{M}) \rightarrow {}'b\ \mathsf{M})) = \text{seqT\_ev}$

$(\text{parT} : {}'a\ \mathsf{M} \rightarrow {}'b\ \mathsf{M} \rightarrow ({}'a\#{}'b)\mathsf{M}) = \text{parT\_ev}$

$(\text{parT\_unit} : \mathsf{unit\ M} \rightarrow \mathsf{unit\ M} \rightarrow \mathsf{unit\ M}) = \text{parT\_unit\_ev}$

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

$(\text{read\_reg} : iiid \rightarrow \mathsf{Xreg} \rightarrow \mathsf{Ximm\ M}) = \text{read\_reg\_ev}$

$(\text{write\_eip} : iiid \rightarrow \mathsf{Ximm} \rightarrow \mathsf{unit\ M}) = \text{write\_eip\_ev}$

$(\text{read\_eip} : iiid \rightarrow \mathsf{Ximm\ M}) = \text{read\_eip\_ev}$

$(\text{write\_eflag} : iiid \rightarrow \mathsf{Xeflags} \rightarrow \mathsf{bool\ \ option} \rightarrow \mathsf{unit\ M}) = \text{write\_eflag\_ev}$

$(\text{read\_eflag} : iiid \rightarrow \mathsf{Xeflags} \rightarrow \mathsf{bool\ M}) = \text{read\_eflag\_ev}$

$(\text{write\_m32} : iiid \rightarrow \mathsf{Ximm} \rightarrow \mathsf{Ximm} \rightarrow \mathsf{unit\ M}) = \text{write\_m32\_ev}$

$(\text{read\_m32} : iiid \rightarrow \mathsf{Ximm} \rightarrow \mathsf{Ximm\ M}) = \text{read\_m32\_ev}$

Part VIII

x86_opsem

$\text{ea\_Xr}(r : \mathsf{Xreg}) = \text{constT}(\text{X}\textsc{ea\_r}\ r)$

$\text{ea\_Xi}(i : \mathsf{Ximm}) = \text{constT}(\text{X}\textsc{ea\_i}\ i)$

$(\text{ea\_Xrm\_base}\ \ ii\ \textsc{None} = \text{constT}\ \ 0w) \wedge$
$(\text{ea\_Xrm\_base}\ \ ii(\textsc{Some}\ \ r) = \text{read\_reg}\ \ ii\ r)$

$(\text{ea\_Xrm\_index}\ \ ii\ \textsc{None} = \text{constT}(0w : \mathsf{Ximm})) \wedge$
$(\text{ea\_Xrm\_index}\ \ ii(\textsc{Some}\ (s : \mathsf{word2}, r)) =$
$\text{seqT}(\text{read\_reg}\ \ ii\ r)(\lambda idx.\,\text{constT}(\mathbf{n2w}(2 ** \mathbf{w2n}\ \ s) * idx)))$

$(\text{ea\_Xrm}\ \ ii(\text{X}\textsc{r}\ r) = \text{ea\_Xr}\ \ r) \wedge$
$(\text{ea\_Xrm}\ \ ii(\text{X}\textsc{m}\ i\ b\ d) =$
seqT
$\quad(\text{parT}(\text{ea\_Xrm\_index}\ \ ii\ i)(\text{ea\_Xrm\_base}\ \ ii\ b))$
$\qquad(\lambda(idx, b).\,\text{constT}(\text{X}\textsc{ea\_m}(idx + b + d))))$

$(\text{ea\_Xdest}\ \ ii(\text{X}\textsc{rm\_i}\ rm\ i) = \text{ea\_Xrm}\ \ ii\ rm) \wedge$
$(\text{ea\_Xdest}\ \ ii(\text{X}\textsc{rm\_r}\ rm\ r) = \text{ea\_Xrm}\ \ ii\ rm) \wedge$
$(\text{ea\_Xdest}\ \ ii(\text{X}\textsc{r\_rm}\ r\ rm) = \text{ea\_Xr}\ \ r)$

$(\text{ea\_Xsrc}\ \ ii(\text{X}\textsc{rm\_i}\ rm\ i) = \text{ea\_Xi}\ \ i) \wedge$
$(\text{ea\_Xsrc}\ \ ii(\text{X}\textsc{rm\_r}\ rm\ r) = \text{ea\_Xr}\ \ r) \wedge$
$(\text{ea\_Xsrc}\ \ ii(\text{X}\textsc{r\_rm}\ r\ rm) = \text{ea\_Xrm}\ \ ii\ rm)$

$(\text{ea\_Ximm\_rm}\ \ ii(\text{X}\textsc{i\_rm}\ rm) = \text{ea\_Xrm}\ \ ii\ rm) \wedge$
$(\text{ea\_Ximm\_rm}\ \ ii(\text{X}\textsc{i}\ i) = \text{ea\_Xi}\ \ i)$

$(\text{read\_ea}\ \ ii(\text{X}\textsc{ea\_i}\ i) = \text{constT}\ \ i) \wedge$
$(\text{read\_ea}\ \ ii(\text{X}\textsc{ea\_r}\ r) = \text{read\_reg}\ \ ii\ r) \wedge$
$(\text{read\_ea}\ \ ii(\text{X}\textsc{ea\_m}\ a) = \text{read\_m32}\ \ ii\ a)$

$\text{read\_src\_ea}\ \ ii\ ds = \text{seqT}(\text{ea\_Xsrc}\ \ ii\ ds)(\lambda ea.\,\text{addT}\ \ ea(\text{read\_ea}\ \ ii\ ea))$

$\text{read\_dest\_ea}\ \ ii\ ds = \text{seqT}(\text{ea\_Xdest}\ \ ii\ ds)(\lambda ea.\,\text{addT}\ \ ea(\text{read\_ea}\ \ ii\ ea))$

$(\text{write\_ea}\ \ ii(\text{X}\textsc{ea\_i}\ i)x = \text{failureT}) \wedge$ (* one cannot store into a constant *)
$(\text{write\_ea}\ \ ii(\text{X}\textsc{ea\_r}\ r)x = \text{write\_reg}\ \ ii\ r\ x) \wedge$
$(\text{write\_ea}\ \ ii(\text{X}\textsc{ea\_m}\ a)x = \text{write\_m32}\ \ ii\ a\ x)$

$(\text{jump\_to\_ea}\ \ ii\ eip(\text{X}\textsc{ea\_i}\ i) = \text{write\_eip}\ \ ii(eip + i)) \wedge$
$(\text{jump\_to\_ea}\ \ ii\ eip(\text{X}\textsc{ea\_r}\ r) = \text{seqT}(\text{read\_reg}\ \ ii\ r)(\text{write\_eip}\ \ ii)) \wedge$
$(\text{jump\_to\_ea}\ \ ii\ eip(\text{X}\textsc{ea\_m}\ a) = \text{seqT}(\text{read\_m32}\ \ ii\ a)(\text{write\_eip}\ \ ii))$

$(\text{call\_dest\_from\_ea}\ \ ii\ \ eip(\text{X}\textsc{ea\_i}\ i) = \text{constT}(eip + i)) \wedge$
$(\text{call\_dest\_from\_ea}\ \ ii\ \ eip(\text{X}\textsc{ea\_r}\ r) = \text{read\_reg}\ \ ii\ r) \wedge$
$(\text{call\_dest\_from\_ea}\ \ ii\ \ eip(\text{X}\textsc{ea\_m}\ a) = \text{read\_m32}\ \ ii\ a)$

$(\text{get\_ea\_address}(\text{X}\textsc{ea\_i}\ i) = 0w) \wedge$
$(\text{get\_ea\_address}(\text{X}\textsc{ea\_r}\ r) = 0w) \wedge$
$(\text{get\_ea\_address}(\text{X}\textsc{ea\_m}\ a) = a)$

$\text{bump\_eip}\ \ ii\ len\ rest =$
$\text{parT\_unit}\ \ rest(\text{seqT}(\text{read\_eip}\ \ ii)(\lambda x.\ \text{write\_eip}\ \ ii(x + len)))$

$\text{byte\_parity} = \mathit{EVEN}\ o\ \mathbf{length}\ \ o\ \mathbf{filter}\ \mathbf{I}\ o\ \mathit{n2bits}\ 8\ o\ \mathbf{w2n}$

$\text{write\_PF}\ \ ii\ w = \text{write\_eflag}\ \ ii\ \text{X\_PF}(\textsc{Some}\ (\text{byte\_parity}\ \ w))$

$\text{write\_ZF}\ \ ii\ w = \text{write\_eflag}\ \ ii\ \text{X\_ZF}(\textsc{Some}\ (w = 0w))$

$\text{write\_SF}\ \ ii\ w = \text{write\_eflag}\ \ ii\ \text{X\_SF}(\textsc{Some}\ (\mathit{word\_msb}\ w))$

$\text{write\_logical\_eflags}\ \ ii\ w =$
$\text{parT\_unit}(\text{write\_PF}\ \ ii\ w)$
$(\text{parT\_unit}(\text{write\_ZF}\ \ ii\ w)$
$(\text{parT\_unit}(\text{write\_SF}\ \ ii\ w)$
$(\text{parT\_unit}(\text{write\_eflag}\ \ ii\ \text{X\_OF}(\textsc{Some}\ \ \mathbf{F}))$
$(\text{parT\_unit}(\text{write\_eflag}\ \ ii\ \text{X\_CF}(\textsc{Some}\ \ \mathbf{F}))$
$\quad\quad(\text{write\_eflag}\ \ ii\ \text{X\_AF}\ \textsc{None})))))$

$\text{write\_arith\_eflags\_except\_CF}\ \ ii\ w =$
$\text{parT\_unit}(\text{write\_PF}\ \ ii\ w)$
$(\text{parT\_unit}(\text{write\_ZF}\ \ ii\ w)$
$(\text{parT\_unit}(\text{write\_SF}\ \ ii\ w)$
$(\text{parT\_unit}(\text{write\_eflag}\ \ ii\ \text{X\_OF}\ \textsc{None})$
$\quad\quad(\text{write\_eflag}\ \ ii\ \text{X\_AF}\ \textsc{None}))))$

$\text{write\_arith\_eflags}\ \ ii(w, c) =$
$\text{parT\_unit}(\text{write\_eflag}\ \ ii\ \text{X\_CF}(\textsc{Some}\ \ c))(\text{write\_arith\_eflags\_except\_CF}\ \ ii\ w)$

$\text{erase\_eflags}\ \ ii =$
$\text{parT\_unit}(\text{write\_eflag}\ \ ii\ \text{X\_PF}\ \textsc{None})$
$(\text{parT\_unit}(\text{write\_eflag}\ \ ii\ \text{X\_ZF}\ \textsc{None})$
$(\text{parT\_unit}(\text{write\_eflag}\ \ ii\ \text{X\_SF}\ \textsc{None})$
$(\text{parT\_unit}(\text{write\_eflag}\ \ ii\ \text{X\_OF}\ \textsc{None})$
$(\text{parT\_unit}(\text{write\_eflag}\ \ ii\ \text{X\_CF}\ \textsc{None})$
$\quad\quad(\text{write\_eflag}\ \ ii\ \text{X\_AF}\ \textsc{None})))))$

$\text{add\_with\_carry\_out}(x : \mathsf{Ximm})y = (x + y, 2 ** 32 \leq \mathbf{w2n}\ x + \mathbf{w2n}\ y)$

$\text{sub\_with\_borrow\_out}(x : \mathsf{Ximm})y = (x - y, x <+ y)$

$\text{write\_arith\_result}\ \ ii(w, c)ea = \text{parT\_unit}(\text{write\_arith\_eflags}\ \ ii(w, c))(\text{write\_ea}\ \ ii\ ea\ w)$

$\text{write\_arith\_result\_no\_CF}\ \ ii\ w\ ea =$
$(\text{parT\_unit}(\text{write\_arith\_eflags\_except\_CF}\ \ ii\ w)(\text{write\_ea}\ \ ii\ ea\ w))$

$\text{write\_arith\_result\_no\_write}\ \ ii(w, c) = (\text{write\_arith\_eflags}\ \ ii(w, c))$

$\text{write\_logical\_result}\ \ ii\ w\ ea = (\text{parT\_unit}(\text{write\_logical\_eflags}\ \ ii\ w)(\text{write\_ea}\ \ ii\ ea\ w))$

$\text{write\_logical\_result\_no\_write}\ \ ii\ w = (\text{write\_logical\_eflags}\ \ ii\ w)$

$\text{write\_result\_erase\_eflags}\ \ ii\ w\ ea = (\text{parT\_unit}(\text{erase\_eflags}\ \ ii)(\text{write\_ea}\ \ ii\ ea\ w))$

$(\text{write\_binop}\ \ ii\ \text{X}\textsc{add}\ x\ y\ ea = (\text{write\_arith\_result}\ \ ii(\text{add\_with\_carry\_out}\ \ x\ y)ea)) \land$
$(\text{write\_binop}\ \ ii\ \text{X}\textsc{sub}\ x\ y\ ea = (\text{write\_arith\_result}\ \ ii(\text{sub\_with\_borrow\_out}\ \ x\ y)ea)) \land$
$(\text{write\_binop}\ \ ii\ \text{X}\textsc{cmp}\ x\ y\ ea = (\text{write\_arith\_result\_no\_write}\ \ ii(\text{sub\_with\_borrow\_out}\ \ x\ y))) \land$
$(\text{write\_binop}\ \ ii\ \text{X}\textsc{test}\ x\ y\ ea = (\text{write\_logical\_result\_no\_write}\ \ ii(x \&\& y))) \land$
$(\text{write\_binop}\ \ ii\ \text{X}\textsc{and}\ x\ y\ ea = (\text{write\_logical\_result}\ \ ii(x \&\& y)ea)) \land$
$(\text{write\_binop}\ \ ii\ \text{X}\textsc{xor}\ x\ y\ ea = (\text{write\_logical\_result}\ \ ii(x ?? y)ea)) \land$
$(\text{write\_binop}\ \ ii\ \text{X}\textsc{or}\ x\ y\ ea = (\text{write\_logical\_result}\ \ ii(x !! y)ea)) \land$
$(\text{write\_binop}\ \ ii\ \text{X}\textsc{shl}\ x\ y\ ea = (\text{write\_result\_erase\_eflags}\ \ ii(x \ll \mathbf{w2n}\ \ y)ea)) \land$
$(\text{write\_binop}\ \ ii\ \text{X}\textsc{shr}\ x\ y\ ea = (\text{write\_result\_erase\_eflags}\ \ ii(x >>> \mathbf{w2n}\ \ y)ea)) \land$
$(\text{write\_binop}\ \ ii\ \text{X}\textsc{sar}\ x\ y\ ea = (\text{write\_result\_erase\_eflags}\ \ ii(x \gg \mathbf{w2n}\ \ y)ea))$

$(\text{write\_monop}\ \ ii\ \text{X}\textsc{not}\ x\ ea = \text{write\_ea}\ \ ii\ ea(\neg x)) \land$
$(\text{write\_monop}\ \ ii\ \text{X}\textsc{dec}\ x\ ea = \text{write\_arith\_result\_no\_CF}\ \ ii(x - 1w)ea) \land$
$(\text{write\_monop}\ \ ii\ \text{X}\textsc{inc}\ x\ ea = \text{write\_arith\_result\_no\_CF}\ \ ii(x + 1w)ea) \land$
$(\text{write\_monop}\ \ ii\ \text{X}\textsc{neg}\ x\ ea =$
$\text{parT\_unit}(\text{write\_arith\_result\_no\_CF}\ \ ii(0w - x)ea)$
$\qquad (\text{write\_eflag}\ \ ii\ \text{X\_CF}\ \textsc{None}))$

$(\text{read\_cond}\ \ ii\ \text{X\_ALWAYS} = \text{constT}\ \ \mathbf{T}) \land$
$(\text{read\_cond}\ \ ii\ \text{X\_E} = \text{read\_eflag}\ \ ii\ \text{X\_ZF}) \land$
$(\text{read\_cond}\ \ ii\ \text{X\_NE} = \text{seqT}(\text{read\_eflag}\ \ ii\ \text{X\_ZF})(\lambda b.\,\text{constT}(\neg b)))$

$\text{x86\_exec\_pop}\ \ ii\ rm =$
$\text{seqT}(\text{seqT}(\text{read\_reg}\ \ ii\ \text{ESP})(\lambda esp.\,\text{addT}\ \ esp(\text{write\_reg}\ \ ii\ \text{ESP}(esp + 4w))))$
$\quad (\lambda(old\_esp, x).\,\text{seqT}(\text{parT}(\text{ea\_Xrm}\ \ ii\ rm)(\text{read\_m32}\ \ ii\ old\_esp))$
$\qquad\qquad (\lambda(ea, w).\,\text{write\_ea}\ \ ii\ ea\ w))$

$\text{x86\_exec\_push}\ \ ii\ \ imm\_rm =$
$(\text{seqT}$
$\quad(\text{parT}(\text{seqT}(\text{ea\_Ximm\_rm}\ \ ii\ \ imm\_rm)(\lambda ea.\, \text{read\_ea}\ \ ii\ \ ea))$
$\qquad\quad(\text{seqT}(\text{read\_reg}\ \ ii\ \text{ESP})(\lambda w.\, \text{constT}(w - 4w))))$
$\quad(\lambda(w, esp).\, \text{parT\_unit}(\text{write\_m32}\ \ ii\ \ esp\ w)(\text{write\_reg}\ \ ii\ \text{ESP}\ esp)))$

$\text{x86\_exec\_popad}\ \ ii =$
$\text{seqT}(\text{read\_reg}\ \ ii\ \text{ESP})(\lambda original\_esp.$
$\text{parT\_unit}(\text{write\_reg}\ \ ii\ \text{ESP}(original\_esp + 32w))($
$\text{parT\_unit}(\text{seqT}(\text{read\_m32}\ \ ii(original\_esp))(\lambda x.\, \text{write\_reg}\ \ ii\ \text{EDI}\ x))($
$\text{parT\_unit}(\text{seqT}(\text{read\_m32}\ \ ii(original\_esp + 4w))(\lambda x.\, \text{write\_reg}\ \ ii\ \text{ESI}\ x))($
$\text{parT\_unit}(\text{seqT}(\text{read\_m32}\ \ ii(original\_esp + 8w))(\lambda x.\, \text{write\_reg}\ \ ii\ \text{EBP}\ x))($
(* The next 4 bytes of stack (containing saved value of ESP) are skipped*)
$\text{parT\_unit}(\text{seqT}(\text{read\_m32}\ \ ii(original\_esp + 16w))(\lambda x.\, \text{write\_reg}\ \ ii\ \text{EBX}\ x))($
$\text{parT\_unit}(\text{seqT}(\text{read\_m32}\ \ ii(original\_esp + 20w))(\lambda x.\, \text{write\_reg}\ \ ii\ \text{EDX}\ x))($
$\text{parT\_unit}(\text{seqT}(\text{read\_m32}\ \ ii(original\_esp + 24w))(\lambda x.\, \text{write\_reg}\ \ ii\ \text{ECX}\ x))($
$\qquad\quad(\text{seqT}(\text{read\_m32}\ \ ii(original\_esp + 28w))(\lambda x.\, \text{write\_reg}\ \ ii\ \text{EAX}\ x)))))))))$

$\text{x86\_exec\_pushad}\ \ ii =$
$\text{seqT}(\text{read\_reg}\ \ ii\ \text{ESP})(\lambda original\_esp.$
$\text{parT\_unit}(\text{write\_reg}\ \ ii\ \text{ESP}(original\_esp - 32w))($
$\text{parT\_unit}(\text{seqT}(\text{read\_reg}\ \ ii\ \text{EAX})(\lambda w.\, \text{write\_m32}\ \ ii(original\_esp - 4w)w))($
$\text{parT\_unit}(\text{seqT}(\text{read\_reg}\ \ ii\ \text{ECX})(\lambda w.\, \text{write\_m32}\ \ ii(original\_esp - 8w)w))($
$\text{parT\_unit}(\text{seqT}(\text{read\_reg}\ \ ii\ \text{EDX})(\lambda w.\, \text{write\_m32}\ \ ii(original\_esp - 12w)w))($
$\text{parT\_unit}(\text{seqT}(\text{read\_reg}\ \ ii\ \text{EBX})(\lambda w.\, \text{write\_m32}\ \ ii(original\_esp - 16w)w))($
$\text{parT\_unit}(\text{write\_m32}\ \ ii(original\_esp - 20w)original\_esp)($
$\text{parT\_unit}(\text{seqT}(\text{read\_reg}\ \ ii\ \text{EBP})(\lambda w.\, \text{write\_m32}\ \ ii(original\_esp - 24w)w))($
$\text{parT\_unit}(\text{seqT}(\text{read\_reg}\ \ ii\ \text{ESI})(\lambda w.\, \text{write\_m32}\ \ ii(original\_esp - 28w)w))($
$\qquad\quad(\text{seqT}(\text{read\_reg}\ \ ii\ \text{EDI})(\lambda w.\, \text{write\_m32}\ \ ii(original\_esp - 32w)w))))))))))$

$\text{x86\_exec\_pop\_eip}\ \ ii =$
$\text{seqT}(\text{seqT}(\text{read\_reg}\ \ ii\ \text{ESP})(\lambda esp.\, \text{addT}\ \ esp(\text{write\_reg}\ \ ii\ \text{ESP}(esp + 4w))))$
$\quad(\lambda(old\_esp, x).\, \text{seqT}(\text{read\_m32}\ \ ii\ \ old\_esp)$
$\qquad\qquad\qquad(\lambda w.\, \text{write\_eip}\ \ ii\ \ w))$

$\text{x86\_exec\_push\_eip}\ \ ii =$
$(\text{seqT}$
$\quad(\text{parT}(\text{read\_eip}\ \ ii)$
$\qquad\quad(\text{seqT}(\text{read\_reg}\ \ ii\ \text{ESP})(\lambda w.\, \text{constT}(w - 4w))))$
$\quad(\lambda(w, esp).\, \text{parT\_unit}(\text{write\_m32}\ \ ii\ \ esp\ w)(\text{write\_reg}\ \ ii\ \text{ESP}\ esp)))$

$\text{x86\_exec\_call}\ \ ii\ \ imm\_rm\ \ len =$
$\text{seqT}(\text{parT}(\text{read\_reg}\ \ ii\ \text{ESP})$
$\qquad\quad(\text{parT}(\text{read\_eip}\ \ ii)$
$\qquad\qquad(\text{ea\_Ximm\_rm}\ \ ii\ \ imm\_rm)))(\lambda(old\_esp, (old\_eip, ea)).$
$\textbf{let}\ \ bumped\_eip = old\_eip + len\ \ \textbf{in}$

```
seqT(call_dest_from_ea ii bumped_eip ea)(λdestw.
    (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)(λold_esp.
seqT(addT old_esp(read_m32 ii old_esp))(λ(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))
      (λ((ea_src, val_src), (ea_dest, val_dest)).
        write_binop ii binop_name val_dest val_src ea_dest))) ∧
(x86_exec ii(XMONOP monop_name rm)len = bump_eip ii len
(seqT
  (seqT(ea_Xrm ii rm)(λea. addT ea(read_ea ii ea)))
      (λ(ea_dest, val_dest).
        write_monop ii monop_name val_dest ea_dest))) ∧
(x86_exec ii(XXADD rm r)len = bump_eip ii len
(seqT
  (parT(seqT(ea_Xrm ii rm)(λea. addT ea(read_ea ii ea)))
        (parT(constT(XEA_R r))(read_reg ii r)))
  (λ((ea_dest, val_dest), (ea_src, val_src)).
      seqT(write_ea ii ea_src val_dest)
          (λx. write_binop ii XADD val_src val_dest ea_dest)))) ∧
(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)(λea. addT ea(read_ea ii ea)))
          (parT(constT(XEA_R r))(read_reg ii r)))
    (λ((ea_dest, val_dest), (ea_src, val_src)).
        (parT_unit(write_ea ii ea_src val_dest)
                  (write_ea ii ea_dest val_src))))))) ∧
(x86_exec ii(XCMPXCHG rm r)len = bump_eip ii len
(seqT
  (parT(seqT(ea_Xrm ii rm)(λea. addT ea(read_ea ii ea)))
        (read_reg ii EAX))
  (λ((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)(λsrc_val. write_ea ii dest_ea src_val)
                 else
                  write_reg ii EAX dest_val)))) ∧
```

$(\text{x86\_exec}\ \ ii(\textsc{Xpop}\ rm) len = \text{bump\_eip}\ \ ii\ len(\text{x86\_exec\_pop}\ \ ii\ rm)) \wedge$
$(\text{x86\_exec}\ \ ii(\textsc{Xpush}\ imm\_rm) len = \text{bump\_eip}\ \ ii\ len(\text{x86\_exec\_push}\ \ ii\ imm\_rm)) \wedge$
$(\text{x86\_exec}\ \ ii(\textsc{Xcall}\ imm\_rm) len = \text{x86\_exec\_call}\ \ ii\ imm\_rm\ len) \wedge$
$(\text{x86\_exec}\ \ ii(\textsc{Xret}\ imm) len = \text{x86\_exec\_ret}\ \ ii\ imm) \wedge$
$(\text{x86\_exec}\ \ ii(\textsc{Xlea}\ ds) len = \text{bump\_eip}\ \ ii\ len$
$(\text{seqT}$
$\quad((\text{parT}(\text{ea\_Xsrc}\ \ ii\ ds)(\text{ea\_Xdest}\ \ ii\ ds)))$
$\qquad(\lambda(ea\_src, ea\_dest).$
$\qquad\quad \text{write\_ea}\ \ ii\ ea\_dest(\text{get\_ea\_address}\ \ ea\_src)))) \wedge$
$(\text{x86\_exec}\ \ ii(\textsc{Xmov}\ c\ ds) len = \text{bump\_eip}\ \ ii\ len$
$(\text{seqT}$
$\quad((\text{parT}(\text{read\_src\_ea}\ \ ii\ ds)$
$\qquad\qquad(\text{parT}(\text{read\_cond}\ \ ii\ c)(\text{ea\_Xdest}\ \ ii\ ds))))$
$\qquad(\lambda((ea\_src, val\_src), (g, ea\_dest)).$
$\qquad\quad \textbf{if}\ \ g\ \ \textbf{then}\ \ \text{write\_ea}\ \ ii\ ea\_dest\ val\_src\ \ \textbf{else}\ \ \text{constT}()))) \wedge$
$(\text{x86\_exec}\ \ ii(\textsc{Xjump}\ c\ imm) len = ($
seqT
$\quad(\text{parT}(\text{read\_eip}\ \ ii)(\text{read\_cond}\ \ ii\ c))$
$\qquad(\lambda(eip, g).\, \text{write\_eip}\ \ ii(\textbf{if}\ \ g\ \ \textbf{then}\ \ eip + len + imm\ \ \textbf{else}\ \ eip + len)))) \wedge$
$(\text{x86\_exec}\ \ ii(\textsc{Xloop}\ c\ imm) len =$
$\text{seqT}(\text{parT}(\text{read\_eip}\ \ ii)($
$\qquad\text{parT}(\text{read\_cond}\ \ ii\ c)$
$\qquad\quad(\text{seqT}(\text{read\_reg}\ \ ii\ \text{ECX})(\lambda ecx.\, \text{constT}(ecx - 1w)))))$
$\quad(\lambda(eip, guard, new\_ecx).$
$\qquad\text{parT\_unit}(\text{write\_reg}\ \ ii\ \text{ECX}\ new\_ecx)$
$\qquad\qquad(\text{write\_eip}\ \ ii$
$\qquad\qquad\quad(\textbf{if}\ \neg(new\_ecx = 0w) \wedge guard$
$\qquad\qquad\quad\ \textbf{then}\ \ eip + len + imm\ \ \textbf{else}\ \ eip + len)))) \wedge$
$(\text{x86\_exec}\ \ ii(\textsc{Xpushad}) len = \text{bump\_eip}\ \ ii\ len($
$\text{x86\_exec\_pushad}\ \ ii)) \wedge$
$(\text{x86\_exec}\ \ ii(\textsc{Xpopad}) len = \text{bump\_eip}\ \ ii\ len($
$\text{x86\_exec\_popad}\ \ ii))$

$(\text{x86\_execute}\ \ ii(\textsc{Xprefix}\ \textsc{Xg1\_none}\ g2\ i) len = \text{x86\_exec}\ \ ii\ i\ len) \wedge$
$(\text{x86\_execute}\ \ ii(\textsc{Xprefix}\ \textsc{Xlock}\ g2\ i) len = \text{lockT}(\text{x86\_exec}\ \ ii\ i\ len))$

Part IX

x86_decoder

$(\text{STR\_SPACE\_AUX}\ \ n\text{``''} = \text{``''}) \wedge$
$(\text{STR\_SPACE\_AUX}\ \ n(\mathit{STRING}\ c\ s) =$
$\textbf{if}\ \ n = 0\ \ \textbf{then}\ \ \mathit{STRING\#}\text{`` ''}(\mathit{STRING}\ c(\text{STR\_SPACE\_AUX}\ 1\ s))$
$\qquad\qquad \textbf{else}\ \ \mathit{STRING}\ c(\text{STR\_SPACE\_AUX}(n-1)s))$

$\text{bytebits} = \mathit{hex2bits}\ o\ \text{STR\_SPACE\_AUX}\ 2$

$\text{check\_opcode}\ \ s =$
$\textbf{let}\ \ y = (\mathit{n2bits}\ 3\ o\ \mathit{char2num}\ o\ \textbf{hd}\ \ o\ \mathit{TL}\ o\ \textbf{explode})s\ \ \textbf{in}$
$\mathit{assert}(\lambda g.g\text{``Reg/Opcode''} = y)$

$\text{read\_SIB} =$
$\mathit{assign\_drop}\text{``Base''}3 \gg \mathit{assign\_drop}\text{``Index''}3 \gg \mathit{assign\_drop}\text{``Scale''}2 \gg$
$\mathit{option\_try}[$
$\mathit{assert}(\lambda g.(g\text{``Mod''} = [\mathbf{F};\mathbf{F}]) \wedge (g\text{``Base''} = [\mathbf{T};\mathbf{F};\mathbf{T}])) \gg \mathit{assign\_drop}\text{``disp32''}32;$
$\mathit{assert}(\lambda g.(g\text{``Mod''} = [\mathbf{F};\mathbf{F}]) \wedge \neg(g\text{``Base''} = [\mathbf{T};\mathbf{F};\mathbf{T}]));$
$\mathit{assert}(\lambda g.(g\text{``Mod''} = [\mathbf{T};\mathbf{F}])) \gg \mathit{assign\_drop}\text{``disp8''}8;$
$\mathit{assert}(\lambda g.(g\text{``Mod''} = [\mathbf{F};\mathbf{T}])) \gg \mathit{assign\_drop}\text{``disp32''}32]$

$\text{read\_ModRM} =$
$\mathit{assign\_drop}\text{``R/M''}3 \gg \mathit{assign\_drop}\text{``Reg/Opcode''}3 \gg \mathit{assign\_drop}\text{``Mod''}2 \gg$
$\mathit{option\_try}[$
$\mathit{assert}(\lambda g.(g\text{``Mod''} = [\mathbf{T};\mathbf{T}]));$
$\mathit{assert}(\lambda g.(g\text{``Mod''} = [\mathbf{F};\mathbf{F}]) \wedge \neg(g\text{``R/M''} = [\mathbf{F};\mathbf{F};\mathbf{T}]) \wedge \neg(g\text{``R/M''} = [\mathbf{T};\mathbf{F};\mathbf{T}]));$
$\mathit{assert}(\lambda g.(g\text{``Mod''} = [\mathbf{F};\mathbf{F}]) \wedge (g\text{``R/M''} = [\mathbf{T};\mathbf{F};\mathbf{T}])) \gg \mathit{assign\_drop}\text{``disp32''}32;$
$\mathit{assert}(\lambda g.(g\text{``Mod''} = [\mathbf{T};\mathbf{F}]) \wedge \neg(g\text{``R/M''} = [\mathbf{F};\mathbf{F};\mathbf{T}])) \gg \mathit{assign\_drop}\text{``disp8''}8;$
$\mathit{assert}(\lambda g.(g\text{``Mod''} = [\mathbf{F};\mathbf{T}]) \wedge \neg(g\text{``R/M''} = [\mathbf{F};\mathbf{F};\mathbf{T}])) \gg \mathit{assign\_drop}\text{``disp32''}32;$
$\mathit{assert}(\lambda g.\neg(g\text{``Mod''} = [\mathbf{T};\mathbf{T}]) \wedge (g\text{``R/M''} = [\mathbf{F};\mathbf{F};\mathbf{T}])) \gg \text{read\_SIB}]$

$\text{is\_hex\_add}\ \ x = ((\textbf{implode}\ \ o\ \mathit{DROP}\ 2\ o\ \ \textbf{explode})x = \text{``+rd''})$

$\text{process\_hex\_add}\ \ \mathit{name} =$
$\textbf{let}\ \ n = (\mathit{hex2num}\ o\ \textbf{implode}\ \ o\ \mathit{TAKE}\ 2\ o\ \textbf{explode})\mathit{name}\ \ \textbf{in}$
$\mathit{option\_try}[\mathit{drop\_eq}(\mathit{n2bits}\ 8(n+0)) \gg \mathit{assign}\text{``reg''}(\mathit{n2bits}\ 3\ 0);$
$\qquad\qquad \mathit{drop\_eq}(\mathit{n2bits}\ 8(n+1)) \gg \mathit{assign}\text{``reg''}(\mathit{n2bits}\ 3\ 1);$
$\qquad\qquad \mathit{drop\_eq}(\mathit{n2bits}\ 8(n+2)) \gg \mathit{assign}\text{``reg''}(\mathit{n2bits}\ 3\ 2);$
$\qquad\qquad \mathit{drop\_eq}(\mathit{n2bits}\ 8(n+3)) \gg \mathit{assign}\text{``reg''}(\mathit{n2bits}\ 3\ 3);$
$\qquad\qquad \mathit{drop\_eq}(\mathit{n2bits}\ 8(n+4)) \gg \mathit{assign}\text{``reg''}(\mathit{n2bits}\ 3\ 4);$
$\qquad\qquad \mathit{drop\_eq}(\mathit{n2bits}\ 8(n+5)) \gg \mathit{assign}\text{``reg''}(\mathit{n2bits}\ 3\ 5);$
$\qquad\qquad \mathit{drop\_eq}(\mathit{n2bits}\ 8(n+6)) \gg \mathit{assign}\text{``reg''}(\mathit{n2bits}\ 3\ 6);$
$\qquad\qquad \mathit{drop\_eq}(\mathit{n2bits}\ 8(n+7)) \gg \mathit{assign}\text{``reg''}(\mathit{n2bits}\ 3\ 7)]$

x86_match_step *name* =
if *is_hex name* $\wedge$ (*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 *g name* = (*EL*(*bits2num*(*g name*))[EAX; ECX; EDX; EBX; ESP; EBP; ESI; EDI]) : Xreg

decode_Xr32 *g name* =
if *name* = “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** (*g*“Mod” = [**T**; **F**]) **then** *sw2sw*((*b2w g*“disp8”) : word8) **else** *b2w g*“disp32”) **in**
  let *disp* = (**if** (*g*“Mod” = [**F**; **F**]) **then** 0*w* **else** *disp*) **in**
    XM *scaled_index*(SOME (b2reg *g*“Base”))*disp*

decode_Xrm32 *g name* =
if *name* = “EAX” **then** XR EAX **else**
if (*g*“Mod” = [**F**; **F**]) $\wedge$ (*g*“R/M” = [**T**; **F**; **T**]) **then** XM NONE NONE(*b2w g*“disp32”) **else**

if $\neg(g$"Mod" $= [\mathbf{T};\mathbf{T}]) \wedge (g$"R/M" $= [\mathbf{F};\mathbf{F};\mathbf{T}])$ **then** decode_SIB g **else**
if $(g$"Mod" $= [\mathbf{F};\mathbf{F}])$ **then** XM NONE(SOME (b2reg g"R/M"))$0w$ **else**
if $(g$"Mod" $= [\mathbf{T};\mathbf{F}])$ **then** XM NONE(SOME (b2reg g"R/M"))($sw2sw$: word8 $\rightarrow$ word32($b2w$ g"disp8")) **else**
if $(g$"Mod" $= [\mathbf{F};\mathbf{T}])$ **then** XM NONE(SOME (b2reg g"R/M"))($b2w$ g"disp32") **else**
if $(g$"Mod" $= [\mathbf{T};\mathbf{T}])$ **then** XR(b2reg g"R/M") **else** XR(b2reg g"reg")

decode_Xconst *name* g =
if *name* = "imm8" **then** $sw2sw$: word8 $\rightarrow$ word32($b2w$ g"ib") **else**
if *name* = "rel8" **then** $sw2sw$: word8 $\rightarrow$ word32($b2w$ g"cb") **else**
if *name* = "imm16" **then** $sw2sw$: word16 $\rightarrow$ word32($b2w$ g"iw") **else**
if *name* = "imm32" **then** $b2w$ g"id" **else**
if *name* = "rel32" **then** $b2w$ g"cd" **else**
if *name* = "1" **then** $1w$ **else** $0w$

decode_Xdest_src g *dest src* =
if *src* = "r32" **then** XRM_R(decode_Xrm32 g *dest*)(decode_Xr32 g *src*) **else**
if *src* = "r/m32" **then** XR_RM(decode_Xr32 g *dest*)(decode_Xrm32 g *src*) **else**
if *src* = "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 = 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* = "POP" **then** X_SOME($\lambda g.$XPOP(decode_Xrm32 g(*EL* 1 *ts*))) **else**
if **hd** *ts* = "PUSH" **then** X_SOME($\lambda g.$XPUSH(decode_Ximm_rm *ts* g)) **else**
if **hd** *ts* = "PUSHAD" **then** X_SOME($\lambda g.$XPUSHAD) **else**
if **hd** *ts* = "POPAD" **then** X_SOME($\lambda g.$XPOPAD) **else**
if **hd** *ts* = "CMPXCHG" **then** X_SOME($\lambda g.$XCMPXCHG(decode_Xrm32 g(*EL* 1 *ts*))(decode_Xr32 g(*EL* 2 *ts*))) **els**
if **hd** *ts* = "XCHG" **then** X_SOME($\lambda g.$XXCHG(decode_Xrm32 g(*EL* 1 *ts*))(decode_Xr32 g(*EL* 2 *ts*))) **else**
if **hd** *ts* = "XADD" **then** X_SOME($\lambda g.$XXADD(decode_Xrm32 g(*EL* 1 *ts*))(decode_Xr32 g(*EL* 2 *ts*))) **else**
if **hd** *ts* = "JMP" **then** X_SOME($\lambda g.$XJUMP X_ALWAYS(decode_Xconst_or_zero *ts* g)) **else**
if **hd** *ts* = "JE" **then** X_SOME($\lambda g.$XJUMP X_E(decode_Xconst_or_zero *ts* g)) **else**

if hd *ts* = "JNE" **then** X_SOME(λg.XJUMP X_NE(decode_Xconst_or_zero *ts g*)) **else**
if hd *ts* = "LEA" **then** X_SOME(λg.XLEA(decode_Xdest_src *g*(*EL* 1 *ts*)(*EL* 2 *ts*))) **else**
if hd *ts* = "LOOP" **then** X_SOME(λg.XLOOP X_ALWAYS(decode_Xconst_or_zero *ts g*)) **else**
if hd *ts* = "LOOPE" **then** X_SOME(λg.XLOOP X_E(decode_Xconst_or_zero *ts g*)) **else**
if hd *ts* = "LOOPNE" **then** X_SOME(λg.XLOOP X_NE(decode_Xconst_or_zero *ts g*)) **else**
if hd *ts* = "MOV" **then** X_SOME(λg.XMOV X_ALWAYS(decode_Xdest_src *g*(*EL* 1 *ts*)(*EL* 2 *ts*))) **else**
if hd *ts* = "CMOVE" **then** X_SOME(λg.XMOV X_E(decode_Xdest_src *g*(*EL* 1 *ts*)(*EL* 2 *ts*))) **else**
if hd *ts* = "CMOVNE" **then** X_SOME(λg.XMOV X_NE(decode_Xdest_src *g*(*EL* 1 *ts*)(*EL* 2 *ts*))) **else**
if hd *ts* = "CALL" **then** X_SOME(λg.XCALL(decode_Ximm_rm *ts g*)) **else**
if hd *ts* = "RET" **then** X_SOME(λg.XRET(decode_Xconst_or_zero *ts g*)) **else** *option_fail*

x86_decode_aux = (*match_list* x86_match_step *tokenise*(x86_syntax *o tokenise*)*o*
map(λs.**let** x = *STR_SPLIT*[#"|"]*s* **in** (*EL* 0 *x*, *EL* 1 *x*)))ˆ*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")) ∧ *l1* **in**
let *bnt2* = (*TAKE* 8(*DROP* 8 *w*) = *n2bits* 8(*hex2num*"3E")) ∧ *l1* **in**
let *l2* = (*TAKE* 8(*DROP* 8 *w*) = *n2bits* 8(*hex2num*"F0")) ∧ (*bt1* ∨ *bnt1*) **in**
let *g1* = **if** *l1* ∨ *l2* **then** XLOCK **else** XG1_NONE **in**
let *g2* = **if** *bt1* ∨ *bt2* **then** XBRANCH_TAKEN **else** XG2_NONE **in**
let *g2* = **if** *bnt1* ∨ *bnt2* **then** XBRANCH_NOT_TAKEN **else** *g2* **in**
let *n* = **if** *bt1* ∨ *bnt1* ∨ *l1* **then** (**if** *bt2* ∨ *bnt2* ∨ *l2* **then** 16 **else** 8) **else** 0 **in**
(*g1*, *g2*, *DROP n w*)

(dest_accesses_memory(XRM_I *rm i*) = rm_is_memory_access *rm*) ∧
(dest_accesses_memory(XRM_R *rm r*) = rm_is_memory_access *rm*) ∧
(dest_accesses_memory(XR_RM *r rm*) = **F**)

(x86_lock_ok(XBINOP *binop_name ds*) =
mem *binop_name* [XADD; XAND; XOR; XSUB; XXOR] ∧
dest_accesses_memory *ds*) ∧
(x86_lock_ok(XMONOP *monop_name rm*) =
mem *monop_name* [XDEC; XINC; XNEG; XNOT] ∧
rm_is_memory_access *rm*) ∧
(x86_lock_ok(XXADD *rm r*) = rm_is_memory_access *rm*) ∧
(x86_lock_ok(XXCHG *rm r*) = rm_is_memory_access *rm*) ∧
(x86_lock_ok(XCMPXCHG *rm r*) = rm_is_memory_access *rm*) ∧
(x86_lock_ok(XPOP *rm*) = **F**) ∧
(x86_lock_ok(XLEA *ds*) = **F**) ∧
(x86_lock_ok(XPUSH *imm_rm*) = **F**) ∧
(x86_lock_ok(XCALL *imm_rm*) = **F**) ∧
(x86_lock_ok(XRET *imm*) = **F**) ∧
(x86_lock_ok(XMOV *c ds*) = **F**) ∧

(x86_lock_ok(XJUMP c imm) = **F**) $\wedge$
(x86_lock_ok(XLOOP c imm) = **F**) $\wedge$
(x86_lock_ok(XPUSHAD) = **F**) $\wedge$
(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 = XLOCK) $\vee$ x86_lock_ok i **then** SOME (XPREFIX g1 g2 i, w) **else** NONE

x86_decode_bytes b = x86_decode(FOLDR[ ] @ (**map** w2bits b))

(rm_args_ok(XR r) = **T**) $\wedge$
(rm_args_ok(XM NONE NONE t3) = **T**) $\wedge$
(rm_args_ok(XM NONE(SOME br)t3) = **T**) $\wedge$
(rm_args_ok(XM(SOME (s, ir))NONE t3) = $\neg$(ir = ESP)) $\wedge$
(rm_args_ok(XM(SOME (s, ir))(SOME br)t3) = $\neg$(ir = ESP))

(dest_src_args_ok(XRM_I rm i) = rm_args_ok rm) $\wedge$
(dest_src_args_ok(XRM_R rm r) = rm_args_ok rm) $\wedge$
(dest_src_args_ok(XR_RM r rm) = rm_args_ok rm)

(imm_rm_args_ok(XI_RM rm) = rm_args_ok rm) $\wedge$
(imm_rm_args_ok(XI i) = **T**)

(x86_args_ok(XBINOP binop_name ds) = dest_src_args_ok ds) $\wedge$
(x86_args_ok(XMONOP monop_name rm) = rm_args_ok rm) $\wedge$
(x86_args_ok(XXADD rm r) = rm_args_ok rm) $\wedge$
(x86_args_ok(XXCHG rm r) = rm_args_ok rm) $\wedge$
(x86_args_ok(XCMPXCHG rm r) = rm_args_ok rm) $\wedge$
(x86_args_ok(XPOP rm) = rm_args_ok rm) $\wedge$
(x86_args_ok(XPUSH imm_rm) = imm_rm_args_ok imm_rm) $\wedge$
(x86_args_ok(XCALL imm_rm) = imm_rm_args_ok imm_rm) $\wedge$
(x86_args_ok(XRET imm) = **w2n** imm < 2 ** 16) $\wedge$
(x86_args_ok(XMOV c ds) = dest_src_args_ok ds $\wedge$ **mem** c [X_NE; X_E; X_ALWAYS]) $\wedge$ (* partial list *)
(x86_args_ok(XJUMP c imm) = **F**) $\wedge$
(x86_args_ok(XLOOP c imm) = **mem** c [X_NE; X_E; X_ALWAYS]) $\wedge$
(x86_args_ok(XPUSHAD) = **T**) $\wedge$
(x86_args_ok(XPOPAD) = **T**)

(x86_well_formed_instruction(XPREFIX XLOCK g2 i) = x86_lock_ok i $\wedge$ x86_args_ok i) $\wedge$
(x86_well_formed_instruction(XPREFIX XG1_NONE g2 i) = x86_args_ok i)

Part X

x86_

iiid_dummy $=\langle\!|$ proc $:= 0$; program_order_index $:= 0|\!\rangle$

x86_execute_some $i\ w\ s$ = option_apply(x86_execute iiid_dummy $i\ w\ s$)($\lambda t.$SOME (**snd** t))

X86_NEXT s =
let e = XREAD_EIP s **in** (* read eip *)
let xs = **map the**(XREAD_MEM_BYTES 20 $e\ s$) **in** (* read next 20 bytes *)
let m = x86_decode_bytes xs **in** (* attempt to decode *)
if m = NONE **then** NONE **else** (* if decoded fail, return NONE *)
  let (i, w) = **the** m **in** (* otherwise extract content *)
  let $n = 20 -$ (**length** w **div** 8) **in** (* calc length of instruction *)
    if **every**($\lambda x.\neg(x =$ NONE))(XREAD_MEM_BYTES $n\ e\ s$)(* check that the memory is there *)
     then x86_execute_some i(**n2w** n)s **else** NONE(* execute the instruction *)

Part XI

x86_program

$\text{DOMAIN } f = \{x \mid \neg(f\ x = \text{NONE})\}$

type_abbrev program_word8 : (address → word8 option)

type_abbrev program_Xinst : (address → (Xinst#num) option)

$\text{read\_mem\_bytes}\ n\ a\ m =$
$\textbf{if}\ n = 0\ \textbf{then}\ []\ \textbf{else}\ m\ a \in \text{read\_mem\_bytes}(n-1)(a+1w)m$

(decode_program_fun : program_word8 → address → (Xinst#num) 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 → NONE(* if decoding fails, then fail *)
 ‖ SOME (i, w) → (* 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 → program_Xinst → bool)
  prog_word8 prog_Xinst =
$\forall a.$**case** *prog_Xinst a* **of**
    NONE → **T**
  ‖ SOME (*inst*, n) → decode_program_fun *prog_word8 a* = SOME (*inst*, n)

type_abbrev run_skeleton : (proc → (program_order_index → address option))

(run_skeleton_wf : address set → run_skeleton → bool)*addrs rs* =
$(\forall p\ i\ i'.((\neg((rs\ p\ i') = \text{NONE})) \wedge (i < i')) \implies (\neg((rs\ p\ i) = \text{NONE}))) \wedge$
$(\forall p\ i\ a.(rs\ p\ i = \text{SOME}\ \ a) \implies a \in addrs) \wedge$
$\textbf{finite}\{p \mid \exists i\ a.rs\ p\ i = \text{SOME}\ \ a\}$

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**
  $\{E$
   $\mid \exists eiid\_next\ x.$
    $(eiid\_next, x, E) \in s \wedge$
    $\forall v\ ev.((ev.action = (\text{ACCESS R}(\text{LOCATION\_REG}\ ii.\mathsf{proc}\ \text{REGEIP})v)) \wedge$
        $ev \in E.events) \implies (v = eip)$
  $\}$

(event_structures_of_run_skeleton : program_Xinst → run_skeleton → event_structure set)
$\quad$ *prog_Xinst rs* =
let $Ess = \{$x86_execute_with_eip_check$\langle\!|$ proc $:= p$; program_order_index $:= i |\!\rangle\, inst(\mathbf{n2w}\ n)\, eip \mid$
$\quad (rs\ p\ i = \textsc{Some}\ \ eip) \wedge (\textsc{Some}\ (inst, n) = prog\_Xinst\ eip)\}$
in
$\{$event_structure_bigunion $Es \mid Es \in all\_choices\ Ess\}$

(x86_semantics : program_word8 → state_constraint →
(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 $prog\_Xinst)rs \wedge$
$\quad$ decode_program_rel $prog\_word8\ prog\_Xinst\}$ **in**

let $x2 = \{(rs, prog\_Xinst, Es) \mid (rs, prog\_Xinst) \in x1 \wedge$
$\quad (Es =$ event_structures_of_run_skeleton $prog\_Xinst\ rs)\}$ **in**

let $x3 = \{(rs, prog\_Xinst, \{(E, Xs) \mid E \in Es \wedge$
$\quad (Xs = \{X \mid$ valid_execution $E\ X \wedge$
$\quad\quad (X.initial\_state = initial\_state)\})\})$
$\quad \mid (rs, prog\_Xinst, Es) \in x2\}$ **in**
$x3$

(lts_po_of_event_structure (* : event_structure -> (event set,unit,machine_visible_label) LTS *))E =
$\langle\!|$ $states := POW\ E.events$;
$\quad initial := \{\}$;
$\quad final :=$ **if finite** $E.events$ **then** $\{(E.events, ())\}$ **else** $\{\}$;
$\quad trans := \{(s, (\textsc{Vis}\langle\!|\ mvl\_event := e$;
$\quad\quad mvl\_iico := \{e' \mid e' \in s \wedge (e', e) \in E.intra\_causality\}$;
$\quad\quad mvl\_first\_of\_instruction := \neg(\exists(e' \in s). e'.iiid = e.iiid)$;
$\quad\quad mvl\_last\_of\_instruction := \neg(\exists(e' \in (E.events \setminus s)). e'.iiid = e.iiid)$;
$\quad\quad mvl\_locked :=$ locked $E\ e$
$\quad |\!\rangle), s \cup \{e\}) \mid e \in E.events \wedge \neg(e \in s) \wedge$
$\quad\quad e \in minimal_elements(E.events \setminus s)($po_iico $E)\}|\!\rangle$

Part XII

x86_sequential_axiomatic_model

sequential_execution $E\ so =$
$\quad$ linear_order $so\ E.events \wedge$
$\quad (\forall (es \in (E.atomicity))(e_1 \in es)(e_2 \in es)e.$
$\qquad (e_1, e) \in so \wedge (e, e_2) \in so \implies e \in es)$

so_to_vo $E\ so = \lambda p.\textbf{if}\ p \in E.procs\ \textbf{then}\ so|_{(\text{viewed\_events}\ E\ p)}\ \textbf{else}\ \{\}$

so_to_write_serialization $so =$
$\{(e_1, e_2) \mid (e_1, e_2) \in (\text{strict}\ so) \wedge \text{mem\_store}\ e_1 \wedge \text{mem\_store}\ e_2 \wedge (\text{loc}\ e_1 = \text{loc}\ e_2)\}$

so_to_lock_serialization $E\ so =$
$\{(e_1, e_2) \mid (e_1, e_2) \in (\text{strict}\ so) \wedge$
$\qquad \exists es_1\ es_2 \in (E.atomicity).\neg(es_1 = es_2) \wedge e_1 \in es_1 \wedge e_2 \in es_2\}$

so_to_rfmap $E\ so =$
$\{(ew, er) \mid (ew, er) \in so \wedge ew \in (\text{writes}\ E) \wedge er \in (\text{reads}\ E) \wedge (\text{loc}\ er = \text{loc}\ ew) \wedge$
$\qquad (\forall ew' \in (\text{writes}\ E).\neg(ew = ew') \wedge (ew, ew') \in so \wedge (ew', er) \in so$
$\qquad\quad \implies$
$\qquad\quad \neg(\text{loc}\ ew = \text{loc}\ ew'))\}$

so_to_exec_witness $E\ initial\_state\ so =$
$\langle\!|\ initial\_state := initial\_state;$
$\quad vo := \text{so\_to\_vo}\ E\ so;$
$\quad write\_serialization := \text{so\_to\_write\_serialization}\ so;$
$\quad lock\_serialization := \text{so\_to\_lock\_serialization}\ E\ so;$
$\quad rfmap := \text{so\_to\_rfmap}\ E\ so |\!\rangle$

valid_sequential_execution $E\ initial\_state\ so =$
$\quad$ sequential_execution $E\ so \wedge$
$\quad$ valid_execution $E(\text{so\_to\_exec\_witness}\ E\ initial\_state\ so)$

competes $E\ X =$
$\{(e_1, e_2) \mid \neg(e_1 = e_2) \wedge (\text{loc}\ e_1 = \text{loc}\ e_2) \wedge$
$((e_1 \in \text{writes}\ E \wedge \text{mem\_store}\ e_1 \wedge e_2 \in \text{reads}\ E) \vee$
$\quad (e_2 \in \text{writes}\ E \wedge \text{mem\_store}\ e_2 \wedge e_1 \in \text{reads}\ E))\}$
$\setminus ((\text{happens\_before}\ E\ X)^+ \cup ((\text{happens\_before}\ E\ X)^+)^{-1})$

race_free $E\ X = \forall e_1\ e_2 \in (E.events).$
$\quad \neg((e_1, e_2) \in \text{competes}\ E\ X)$

restrictE $E\ es =$
$\langle\!|\ procs := E.procs;$
$\quad events := E.events \cap es;$
$\quad intra\_causality := E.intra\_causality|_{es};$
$\quad atomicity := PER\_RESTRICT\ E.atomicity\ es |\!\rangle$

$$\text{prefixes } E\ X = \{E' \mid \text{sub\_event\_structure } E'\ E \wedge \forall e_1\ e_2.\ e_2 \in E'.\mathit{events} \wedge (e_1, e_2) \in (\text{happens\_before } E\ X) \implies e_1 \in E'.\mathit{events}\}$$

$$\text{sequential\_race\_free } E\ X = \forall (E' \in (\text{prefixes } E\ X))\mathit{so}.\ \text{valid\_sequential\_execution } E'\ X.\mathit{initial\_state}\ \mathit{so} \implies \forall e_1\ e_2. \neg((e_1, e_2) \in \text{competes } E'\ (\text{so\_to\_exec\_witness } E'\ X.\mathit{initial\_state}\ \mathit{so}))$$

Part XIII

x86_drf

wfes $E =$
$(\forall iiid.\mathbf{finite}\{\mathsf{eiid} \mid \exists e \in (E.events).(e.iiid = iiid) \wedge (e.\mathsf{eiid} = \mathsf{eiid})\}) \wedge$
$(\mathbf{finite}\ E.procs) \wedge$
$(\forall e \in (E.events).\mathsf{proc}\ e \ \in\ E.procs) \wedge$
$(\forall e_1\ e_2 \in (E.events).(e_1.\mathsf{eiid} = e_2.\mathsf{eiid}) \wedge (e_1.iiid = e_2.iiid) \implies (e_1 = e_2)) \wedge$
$(DOM\ E.intra\_causality) \subseteq\ E.events \wedge$
$(\mathbf{range}\ \ E.intra\_causality) \subseteq\ E.events \wedge$
$\mathrm{acyclic}(E.intra\_causality) \wedge$
$(\forall (e_1, e_2) \in (E.intra\_causality).(e_1.iiid = e_2.iiid)) \wedge$
$(\forall (e_1 \in \mathrm{writes}\ \ E)e_2.$
$\neg(e_1 = e_2) \wedge$
$(e_2\ \in\ \mathrm{writes}\ \ E \vee e_2\ \in\ \mathrm{reads}\ \ E) \wedge$
$(e_1.iiid = e_2.iiid) \wedge$
$(\mathrm{loc}\ \ e_1 = \mathrm{loc}\ \ e_2) \wedge$
$(\exists p\ r.\,\mathrm{loc}\ \ e_1 = \textsc{Some}\ (\textsc{Location\_reg}\ p\ r))$
$\implies$
$(e_1, e_2) \in E.intra\_causality^+ \vee$
$(e_2, e_1) \in E.intra\_causality^+) \wedge$
$PER\ E.events\ E.atomicity \wedge$
$(\forall es \in (E.atomicity).\forall e_1\ e_2 \in es.(e_1.iiid = e_2.iiid)) \wedge$
$(\forall es \in (E.atomicity).\forall e_1 \in es.\forall e_2 \in (E.events).(e_1.iiid = e_2.iiid) \implies e_2\ \in\ es) \wedge$
$(\forall e \in (E.events).\forall p\ r.(\mathrm{loc}\ \ e = \textsc{Some}\ (\textsc{Location\_reg}\ p\ r)) \implies (p = \mathsf{proc}\ e))$

maximal_es $E\ X = maximal\_elements\ E.events(\mathrm{happens\_before}\ \ E\ X)$

deleteE $E\ e =$
$\langle\!|\ procs := E.procs;$
$\quad events := E.events\ DELETE\ e;$
$\quad intra\_causality := E.intra\_causality|_{(E.events\ DELETE\ e)};$
$\quad atomicity := PER\_RESTRICT\ E.atomicity(E.events\ DELETE\ e)|\!\rangle$

extend_so $E\ so\ e =$
$so\ \cup \{(e_1, e) \mid e_1 \mid e_1\ \in\ E.events\}$

locked_ready $E\ X\ es =$
$\forall e\ e'.e\ \in\ es \wedge (e, e') \in\ \mathrm{happens\_before}\ \ E\ X \implies e'\ \in\ es$

ind_hyp1 $E\ X\ so =$
$\forall es\ e.$
$es\ \in\ E.atomicity \wedge$
$(\exists e'.e'\ \in\ es \wedge \mathrm{locked\_ready}\ \ E\ X\ es) \wedge$
$e\ \in\ maximal\_elements\ E.events\ so$
$\implies$
$e\ \in\ es$

sequential_order_thm =
$\forall E\ X.$ wfes $E \wedge$ **finite** $E.events \wedge$
race_free $E\ X \wedge$ valid_execution $E\ X$
$\Longrightarrow \exists so.$
valid_sequential_execution $E\ X.initial\_state\ so \wedge$
ind_hyp1 $E\ X\ so \wedge$
(happens_before E(so_to_exec_witness $E\ X.initial\_state\ so$))
$\subseteq$ (strict so) $\wedge$
($X.write\_serialization$ = so_to_write_serialization so) $\wedge$
($X.lock\_serialization$ = so_to_lock_serialization $E\ so$) $\wedge$
($X.rfmap$ = so_to_rfmap $E\ so$)

pre $E\ X\ es$ = restrictE E(**biginter**$\{E''.events \mid E'' \mid es \subseteq E''.events \wedge E'' \in$ prefixes $E\ X\}$)

ind_concl $E'\ X'\ so$ =
valid_sequential_execution $E'\ X'.initial\_state\ so \wedge$
(happens_before E'(so_to_exec_witness $E'\ X'.initial\_state\ so$)) $\subseteq$ (strict so) $\wedge$
($X'.write\_serialization$ = so_to_write_serialization so) $\wedge$
($X'.lock\_serialization$ = so_to_lock_serialization $E'\ so$) $\wedge$
($X'.rfmap$ = so_to_rfmap $E'\ so$)

ind_concl2 $E'\ X'\ so$ = ind_concl $E'\ X'\ so \wedge$ ind_hyp1 $E'\ X'\ so$

ind_assum $E\ E'\ X'$ =
card $E'.events <$ **card** $E.events \wedge$
wfes $E' \wedge$
finite $E'.events \wedge$
sequential_race_free $E'\ X' \wedge$
valid_execution $E'\ X'$

data_race_free_thm =
$\forall E\ X.$ well_formed_event_structure $E \wedge$ **finite** $E.events \wedge$
sequential_race_free $E\ X \wedge$ valid_execution $E\ X$
$\Longrightarrow$
race_free $E\ X \wedge \exists so.$
valid_sequential_execution $E\ X.initial\_state\ so \wedge$
ind_hyp1 $E\ X\ so \wedge$
(happens_before E(so_to_exec_witness $E\ X.initial\_state\ so$))
$\subseteq$ (strict so) $\wedge$
($X.write\_serialization$ = so_to_write_serialization so) $\wedge$
($X.lock\_serialization$ = so_to_lock_serialization $E\ so$) $\wedge$
($X.rfmap$ = so_to_rfmap $E\ so$)

Part XIV

x86_hb_machine

clause_name $x = \mathbf{T}$

$f \oplus (x \mapsto y) = \lambda x'.\mathbf{if}\ x' = x\ \mathbf{then}\ y\ \mathbf{else}\ f\ x'$

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 → address → (value#(event option)) option)#
(*F*)(proc → proc → message list)#
(*G*)(address → event list)#
(* TODO: now we're using X, instead of having an erasure property, maybe we should lose the explicit G? *)
(*Rg*)(proc → reg → (value#(event option)) option)#
(*X*)execution_witness

(initial_machine_state : state_constraint → machine_state)*initialstate* =
(⟨| *events* :={}; *intra_causality* :={}; *atomicity* :={}|⟩,
($\lambda p.\lambda a.$**case** *initialstate*(LOCATION_MEM a) **of** NONE → NONE ∥ SOME v → SOME (v, NONE)),
($\lambda p_1.\lambda p_2.NIL$),
($\lambda a.NIL$),
($\lambda p.\lambda r.$**case** *initialstate*(LOCATION_REG p r) **of** NONE → NONE ∥ SOME v → SOME (v, NONE)),
⟨| *initial_state* := *initialstate*;
vo := $\lambda p.\{\}$;
write_serialization :={};
lock_serialization :={};
rfmap :={}|⟩
)

execution_witness_of_machine_state$(E, \mathsf{M}, \mathbf{F}, G, Rg, X) = X$

event_set_of_machine_state$(E, \mathsf{M}, \mathbf{F}, G, Rg, X) = E.events$

next_eiid_es $E\ e =$ next_eiid$\{e.\mathsf{eiid} \mid e \in E.events\}$

(**)

(* enqueing sub-writes *)
($\forall P\ M\ \mathbf{F}\ G\ Rg\ e\ F'\ a\ v\ p\ E\ X(X' :$ execution_witness$)$*iico mvl*.
(clause_name "enqueue-mem-write" ∧
$(e = mvl.mvl\_event)$ ∧

$(iico = mvl.mvl_{iico}) \wedge$
$(e.action = \text{ACCESS W}(\text{LOCATION\_MEM } a)v) \wedge$
$(p = e.iiid.\mathsf{proc}) \wedge$
$(\forall q. q \in P \implies ((F' \ p \ q) = ([(e, a, (v, iico))] + +(\mathbf{F} \ p \ q)))) \wedge$
$(\forall p' \ q. p' \in P \implies q \in P \implies \neg(p = p') \implies (F' \ p' \ q = \mathbf{F} \ p' \ q)) \wedge$
$(X' = X)$
$) \implies$
$machine\_trans \ P(E, M, \mathbf{F}, G, Rg, X)(\text{VIS } mvl)(E, M, F', G, Rg, X')$
$) \wedge$

(***)

(* 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" $\wedge$
$(e = mvl.mvl\_event) \wedge$
$(iico = mvl.mvl_{iico}) \wedge$
$(e.action = \text{ACCESS R}(\text{LOCATION\_MEM } a)v) \wedge$
$(p = e.iiid.\mathsf{proc}) \wedge$
$(M \ p \ a = \text{SOME } (v, eo)) \wedge$
$(\neg\exists e' \ v' \ iico'. \ \mathbf{mem} \ (e', a, v', iico') \ (\mathbf{F} \ p \ p)) \wedge$
$(E' = \langle\!| \ events := E.events \ \cup \{e\};$
$\quad intra\_causality := E.intra\_causality \ \cup \{(e', e) \mid e' \in iico\};$
$\quad atomicity := E.atomicity |\!\rangle) \wedge$
$(X' = X \ \langle\!| \ vo := X.vo \oplus (p \mapsto (\text{linear\_order\_extend}(X.vo \ p)e));$
$\quad rfmap := \mathbf{case} \ \ eo \ \ \mathbf{of} \ \text{NONE} \to X.rfmap$
$\quad\quad \| \ \text{SOME} \ \ ew \to X.rfmap \ \cup \{(ew, e)\} |\!\rangle) \wedge$
(* TODO: would it be cleaner to pull eo from the most recent write in our X.vo p, instead of recording eo in M?*)
$(\forall e'.(e', e) \in (\text{happens\_before} \ E' \ X')^{+} \implies e' \in \text{viewed\_events} \ E' \ p \implies (e', e') \in X.vo \ p)$
$) \implies$
$machine\_trans \ P(E, M, \mathbf{F}, G, Rg, X)(\text{VIS } mvl)(E', M, \mathbf{F}, G, Rg, X')$
$) \wedge$
(***)

(* 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") $\wedge$
$(e.action = \text{ACCESS W}(\text{LOCATION\_MEM } a)v) \wedge$
$(\mathbf{F} = \text{funupd2} \ F' \ p \ q((F' \ p \ q) + +[(e, a, (v, iico))])) \wedge$
$(G \ a = (G_0 + +[e] + +G_1)) \wedge$
$(\forall p' \ e' \ v'. p' \in P \implies \mathbf{mem} \ (e', a, v') \ (\mathbf{F} \ p' \ q) \implies \neg(\mathbf{mem} \ e' \ G_1)) \wedge$
$(M' = \text{funupd2} \ M \ q \ a(\text{SOME } (v, \text{SOME } \ e))) \wedge$
$(X' = X \ \langle\!| \ vo := X.vo \oplus (q \mapsto (\text{linear\_order\_extend}(X.vo \ q)e)) |\!\rangle) \wedge$
$(\forall e'.(e', e) \in (\text{happens\_before} \ E \ X')^{+} \implies e' \in \text{viewed\_events} \ E \ q \implies (e', e') \in X.vo \ q)$
$) \implies$
$machine\_trans \ P(E, M, \mathbf{F}, G, Rg, X)\text{TAU}(E, M', F', G, Rg, X')$
$) \wedge$

(***)

$(\forall P\ M\ \mathbf{F}\ G\ Rg\ M'\ F'\ G'\ p\ q\ a\ e\ v\ E\ E'\ X\ X'\ iico.$
$((\text{clause\_name}\text{“deliver-mem-write-2”}) \wedge$
$(e.action = \text{ACCESS W}(\text{LOCATION\_MEM}\ a)v) \wedge$
$(\mathbf{F} = (\text{funupd2}\ \ F'\ p\ q((F'\ p\ q) + +([(e, a, (v, iico))])))) \wedge$
$(\neg(\mathbf{mem}\ \ e\ (G\ a))) \wedge$
$(G' = G \oplus (a \mapsto ([e] + +G\ a))) \wedge$
$(\forall p'\ e'\ v'.p'\ \in\ P\ \implies\ \mathbf{mem}\ (e', a, v')\ (\mathbf{F}\ p'\ q)\ \implies\ \neg(\mathbf{mem}\ \ e'\ (G\ a))) \wedge$
$(M' = \text{funupd2}\ \ M\ q\ a(\text{SOME}\ (v, \text{SOME}\ \ e))) \wedge$
$(E' = \llparenthesis\ events := E.events\ \cup \{e\};$
$\quad intra\_causality := E.intra\_causality\ \cup \{(e', e) \mid e'\ \in\ iico\};$
$\quad atomicity := E.atomicity \rrparenthesis) \wedge$
$(X' = X\ \llparenthesis\ vo := X.vo \oplus (q \mapsto (\text{linear\_order\_extend}(X.vo\ q)e));$
$\quad write\_serialization := X.write\_serialization\ \cup \{(e', e) \mid \mathbf{mem}\ \ e'\ (G\ a)\}\rrparenthesis) \wedge$
$(\forall e'.(e', e) \in (\text{happens\_before}\ \ E\ X')^+\ \implies\ e'\ \in\ \text{viewed\_events}\ \ E\ q\ \implies\ (e', e') \in (X.vo\ q))$
$)\ \implies$
$machine\_trans\ P(E, M, \mathbf{F}, G, Rg, X)\text{TAU}(E', M', F', G', Rg, X')$
$) \wedge$

(***)

$(\forall P\ M\ \mathbf{F}\ G\ Rg\ e\ r\ v\ p\ E\ E'\ X\ X'\ eo\ iico\ mvl.$
$(\text{clause\_name}\text{“reg-read”} \wedge$
$(e = mvl.mvl\_event) \wedge$
$(iico = mvl.mvl_{iico}) \wedge$
$(e.action = \text{ACCESS R}(\text{LOCATION\_REG}\ p\ r)v) \wedge$
$(Rg\ p\ r = \text{SOME}\ (v, eo)) \wedge$
$(E' = \llparenthesis\ events := E.events\ \cup \{e\};$
$\quad intra\_causality := E.intra\_causality\ \cup \{(e', e) \mid e'\ \in\ iico\};$
$\quad atomicity := E.atomicity \rrparenthesis) \wedge$
$(X' = X\ \llparenthesis\ vo := X.vo \oplus (p \mapsto (\text{linear\_order\_extend}(X.vo\ p)e));$
$\quad rfmap := \mathbf{case}\ \ eo\ \ \mathbf{of}\ \text{NONE} \to X.rfmap$
$\qquad \parallel \text{SOME}\ \ ew \to X.rfmap\ \cup \{(ew, e)\}\rrparenthesis) \wedge$
$(\forall e'.(e', e) \in (\text{happens\_before}\ \ E'\ X')^+\ \implies\ e'\ \in\ \text{viewed\_events}\ \ E'\ p\ \implies\ (e', e') \in\ X.vo\ p)$
$)\ \implies$
$machine\_trans\ P(E, M, \mathbf{F}, G, Rg, X)(\text{VIS}\ mvl)(E', M, \mathbf{F}, G, Rg, X')$
$) \wedge$
(***)

$(\forall P\ M\ \mathbf{F}\ G\ Rg\ Rg'\ e\ r\ v\ p\ E\ E'\ X\ X'\ iico\ mvl.$
$(\text{clause\_name}\text{“reg-write”} \wedge$
$(e = mvl.mvl\_event) \wedge$
$(iico = mvl.mvl_{iico}) \wedge$
$(e.action = \text{ACCESS W}(\text{LOCATION\_REG}\ p\ r)v) \wedge$
$(Rg' = \text{funupd2}\ \ Rg\ p\ r(\text{SOME}\ (v, \text{SOME}\ \ e))) \wedge$
$(E' = \llparenthesis\ events := E.events\ \cup \{e\};$
$\quad intra\_causality := E.intra\_causality\ \cup \{(e', e) \mid e'\ \in\ iico\};$
$\quad atomicity := E.atomicity \rrparenthesis) \wedge$

$(X' = X \, \langle\!| \, vo := X.vo \oplus (p \mapsto (\text{linear\_order\_extend}(X.vo \; p) e)) |\!\rangle) \wedge$
$(\forall e'.(e', e) \in (\text{happens\_before} \;\; E' \; X')^{+} \implies e' \in \text{viewed\_events} \;\; E' \; p \implies (e', e') \in X.vo \; p)$
$) \implies$
$\mathit{machine\_trans} \; P(E, M, \mathbf{F}, G, Rg, X)(\textsc{Vis} \; mvl)(E', M, \mathbf{F}, G, Rg', X')$
$)$

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

Part XV

x86_lts_ops

$\mathsf{lts\_state} = \textsc{Leaf}\ \textbf{of}\ \mathsf{value} \mid \textsc{Left}\ \textbf{of}\ \mathsf{lts\_state} \mid \textsc{Right}\ \textbf{of}\ \mathsf{lts\_state} \mid \textsc{Pair}\ \textbf{of}\ \mathsf{lts\_state}\ \mathsf{lts\_state}$

$$
\begin{array}{l}
(\mathrm{eip\_tracked\_lts} : ('s,'a,\mathsf{lts\_monad\_visible\_label})\mathsf{LTS} \to \mathsf{address} \to \mathsf{program\_order\_index} \to (\mathsf{address}\#\mathsf{program\_order\_inde}\\
\langle\!|\ \mathit{states} := \{(\mathit{eip}, \mathrm{po}, s) \mid \mathit{eip}\ \in\ \mathit{UNIV} \wedge (\mathrm{po} = \mathit{po\_initial}) \wedge s\ \in\ \mathit{lts.states}\};\\
\quad \mathit{initial} := (\mathit{eip\_initial}, \mathit{po\_initial}, \mathit{lts.initial});\\
\quad \mathit{final} := \{((\mathit{eip}, \mathit{po\_initial}, s), x) \mid \mathit{eip}\ \in\ \mathit{UNIV} \wedge (s, x) \in\ \mathit{lts.final}\};\\
\quad \mathit{trans} := \{((\mathit{eip}, \mathit{po\_initial}, s), l, (\mathit{eip}', \mathit{po\_initial}, s')) \mid (s, l, s') \in\ \mathit{lts.trans} \wedge\\
\quad (\forall \mathit{lmvl}\ p\ v.((l = \textsc{Vis}\ \mathit{lmvl}) \wedge (\mathit{lmvl.lmvl}_{\mathit{action}} = \textsc{Access}\ \mathrm{R}(\textsc{Location\_reg}\ p\ \mathrm{RegEIP})v)) \implies ((\mathit{eip} = v) \wedge (\mathit{eip}' =\\
\quad (\forall \mathit{lmvl}\ p\ v.((l = \textsc{Vis}\ \mathit{lmvl}) \wedge (\mathit{lmvl.lmvl}_{\mathit{action}} = \textsc{Access}\ \mathrm{W}(\textsc{Location\_reg}\ p\ \mathrm{RegEIP})v)) \implies (\mathit{eip}' = \mathit{eip})) \wedge\\
\quad ((\neg(\exists \mathit{lmvl}\ p\ v\ d.(l = \textsc{Vis}\ \mathit{lmvl}) \wedge (\mathit{lmvl.lmvl}_{\mathit{action}} = \textsc{Access}\ d(\textsc{Location\_reg}\ p\ \mathrm{RegEIP})v))) \implies (\mathit{eip}' = \mathit{eip}))(\\
\quad \}|\!\rangle
\end{array}
$$

$$
\begin{array}{l}
(\mathrm{eip\_tracked\_lts\_initial} : \mathsf{address} \to (\mathsf{address}\#\mathsf{program\_order\_index}\#\mathsf{lts\_state}, \mathsf{unit}, \mathsf{lts\_monad\_visible\_label})\mathsf{LTS})\mathit{eip} =\\
\textbf{let}\ \ s = (\mathit{eip}, 0, \textsc{Leaf}\ 0w)\ \textbf{in}\\
\langle\!|\ \mathit{states} := \{s\};\\
\quad \mathit{initial} := s;\\
\quad \mathit{final} := \{(s, ())\};\\
\quad \mathit{trans} := \{\}|\!\rangle
\end{array}
$$

$$
\begin{array}{l}
(\mathrm{lts\_parallel} : ('s1,'a1,'vl)\mathsf{LTS} \to ('s2,'a2,'vl)\mathsf{LTS} \to (('s1\#'s2),('a1\#'a2),'vl)\mathsf{LTS})\mathit{lts1}\ \mathit{lts2} =\\
\langle\!|\ \mathit{states} := \{(s1, s2) \mid s1\ \in\ \mathit{lts1.states} \wedge s2\ \in\ \mathit{lts2.states}\};\\
\quad \mathit{initial} := (\mathit{lts1.initial}, \mathit{lts2.initial});\\
\quad \mathit{final} := \{((s1, s2), (x1, x2)) \mid (s1, x1) \in\ \mathit{lts1.final} \wedge (s2, x2) \in\ \mathit{lts2.final}\};\\
\quad \mathit{trans} := \{((s1, s2), \textsc{Tau}, (s1', s2)) \mid (s1, \textsc{Tau}, s1') \in\ \mathit{lts1.trans} \wedge s2\ \in\ \mathit{lts2.states}\} \cup\\
\qquad\quad \{((s1, s2), \textsc{Tau}, (s1, s2')) \mid (s2, \textsc{Tau}, s2') \in\ \mathit{lts2.trans} \wedge s1\ \in\ \mathit{lts1.states}\} \cup\\
\qquad\quad \{((s1, s2), l, (s1', s2')) \mid (s1, l, s1') \in\ \mathit{lts1.trans} \wedge (s2, l, s2') \in\ \mathit{lts2.trans} \wedge \neg(l = \textsc{Tau})\}\\
|\!\rangle
\end{array}
$$

$$
\begin{array}{l}
(\mathrm{traces\_of\_lts} : ('s,'v,'vl)\mathsf{LTS} \to ('s\#(\mathsf{num} \to (('vl\ \mathsf{label})\#'s)\ \mathsf{option}))\mathsf{set})\mathit{lts} =\\
\{(x, t) \mid (x = \mathit{lts.initial}) \wedge (\forall l'\ s'.(t\ 0 = \textsc{Some}\ (l', s')) \implies (\mathit{lts.initial}, l', s') \in\ \mathit{lts.trans}) \wedge\\
\qquad\qquad\qquad\qquad (\forall i\ l'\ s'.(t(i+1) = \textsc{Some}\ (l', s')) \implies \exists l\ s.(t\ i = \textsc{Some}\ (l, s)) \wedge (s, l', s') \in\ \mathit{lts.trans})\}
\end{array}
$$

$$
\begin{array}{l}
(\mathrm{completed\_traces\_of\_lts} : ('s,'v,'vl)\mathsf{LTS} \to ('s\#(\mathsf{num} \to ((('vl\ \mathsf{label})\#'s)\ \mathsf{option})))\mathsf{set})\mathit{lts} =\\
\{(x, t) \mid (x = \mathit{lts.initial}) \wedge\\
\qquad\qquad (\forall l'\ s'.(t\ 0 = \textsc{Some}\ (l', s')) \implies (\mathit{lts.initial}, l', s') \in\ \mathit{lts.trans}) \wedge\\
\qquad\qquad ((t\ 0 = \textsc{None}) \implies \neg(\exists l'\ s'.(\mathit{lts.initial}, l', s') \in\ \mathit{lts.trans})) \wedge\\
\qquad\qquad (\forall i\ l'\ s'.(t(i+1) = \textsc{Some}\ (l', s')) \implies \exists l\ s.(t\ i = \textsc{Some}\ (l, s)) \wedge (s, l', s') \in\ \mathit{lts.trans}) \wedge\\
\qquad\qquad (\forall i\ l\ s.(t\ i = \textsc{Some}\ (l, s)) \implies (t(i+1) = \textsc{None}) \implies \neg(\exists l'\ s'.(s, l', s') \in\ \mathit{lts.trans}))\}
\end{array}
$$

$$
\begin{array}{l}
(\mathrm{states\_of\_trace}(\mathit{trs} : ('s\#(\mathsf{num} \to ((('vl\ \mathsf{label})\#'s)\ \mathsf{option})))\mathsf{set}) : 's\ \mathsf{set}) =\\
\{s \mid \exists n\ l\ \mathit{tr}.\mathit{tr}\ \in\ \mathit{trs} \wedge ((\textbf{snd}\ \ \mathit{tr})n = \textsc{Some}\ (l, s))\}\\
\cup\\
\{\textbf{fst}\ \ \mathit{tr} \mid \mathit{tr}\ \in\ \mathit{trs}\}
\end{array}
$$

Part XVI

x86_hb_machine_thms

$(\text{machine\_lts} : \textsf{proc set} \to \textsf{state\_constraint} \to (\textsf{machine\_state}, \textsf{unit}, \textsf{machine\_visible\_label})\textsf{LTS})\mathit{ps}\ \mathit{initial\_state} =$
$\langle\!|\ \mathit{states} := (\mathit{UNIV} : \textsf{machine\_state set});$
$\quad \mathit{initial} := \text{initial\_machine\_state}\ \mathit{initial\_state};$
$\quad \mathit{final} := \{\};$
$\quad \mathit{trans} := \{(\mathit{s1}, l, \mathit{s2}) \mid \mathit{machine\_trans}\ \mathit{ps}\ \mathit{s1}\ l\ \mathit{s2}\}|\!\rangle$

$(\text{machine\_execution\_of\_event\_structure}\ E\ \mathit{initial\_state}) =$
$\textbf{let}\ \ \mathit{lts\_prog} = \text{lts\_po\_of\_event\_structure}\ E\ \ \textbf{in}$
$\textbf{let}\ \ \mathit{lts\_machine} = \text{machine\_lts}(E.\mathit{procs})\mathit{initial\_state}\ \ \textbf{in}$
$\textbf{let}\ \ \mathit{lts} = \text{lts\_parallel}\ \mathit{lts\_prog}\ \mathit{lts\_machine}\ \ \textbf{in}$
$\text{completed\_traces\_of\_lts}\ \mathit{lts}$

$\text{final\_states}\ \mathit{init\_state}\ E\ \mathit{trs} =$
$\{\mathit{st} \mid \exists \mathit{path}\ \mathit{lbl}.((\{\}, \mathit{init\_state}), \mathit{path}) \in\ \mathit{trs} \wedge$
$\qquad\qquad \text{final\_state}\ \mathit{path}(\mathit{lbl}, (E.\mathit{events}, \mathit{st}))\}$

$\text{hb\_equivalence\_thm1} =$
$\forall E\ X.$
$\text{well\_formed\_event\_structure}\ E \wedge$
$\textbf{finite}\ E.\mathit{events} \wedge$
$\text{valid\_execution}\ E\ X \wedge$
$\text{nice\_execution}\ E\ X$
$\implies$
$\exists M\ \mathbf{F}\ G\ \mathit{Rg}.$
$(E, M, \mathbf{F}, G, \mathit{Rg}, X) \in$
$\text{final\_states}(\text{initial\_machine\_state}\ X.\mathit{initial\_state})$
$\qquad E$
$\qquad (\text{machine\_execution\_of\_event\_structure}\ E\ X.\mathit{initial\_state})$

$\text{partial\_view\_orders\_well\_formed}\ E\ \mathit{vo} =$
$(\forall p \in (E.\mathit{procs}).$
$\quad (\exists \mathit{es}\ \mathit{es}'.$
$\qquad (\text{viewed\_events}\ E\ p = \mathit{es}\ \cup\ \mathit{es}') \wedge$
$\qquad (\forall e \in \mathit{es}'.\ \text{mem\_store}\ e) \wedge$
$\qquad \text{linear\_order}(\mathit{vo}\ p)\mathit{es}) \wedge$
$\quad \forall e \in (\text{viewed\_events}\ E\ p).\textbf{finite}\{e' \mid (e', e) \in (\mathit{vo}\ p)\}) \wedge$
$(\forall p.\neg(p\ \in\ E.\mathit{procs})\ \implies\ (\mathit{vo}\ p = \{\}))$

$\text{partial\_valid\_execution}\ E\ X =$
$\quad \text{partial\_view\_orders\_well\_formed}\ E\ X.\mathit{vo} \wedge$
$\quad X.\mathit{write\_serialization}\ \in\ \text{write\_serialization\_candidates}\ E \wedge$
$\quad X.\mathit{lock\_serialization}\ \in\ \text{lock\_serialization\_candidates}\ E \wedge$
$\quad X.\mathit{rfmap}\ \in\ \text{reads\_from\_map\_candidates}\ E \wedge$
$\quad \text{check\_causality}\ E\ X.\mathit{vo}(\text{happens\_before}\ E\ X) \wedge$
$\quad \text{check\_rfmap\_written}\ E\ X.\mathit{vo}\ X.\mathit{rfmap} \wedge$
$\quad \text{check\_rfmap\_initial}\ E\ X.\mathit{vo}\ X.\mathit{rfmap}\ X.\mathit{initial\_state} \wedge$
$\quad \text{check\_atomicity}\ E\ X.\mathit{vo}$

hb_equivalence_thm2 =
$\forall E\ M\ \mathbf{F}\ G\ Rg\ X.$
$\mathbf{finite}\ E.events \wedge$
well_formed_event_structure $E \wedge$
$(E, M, \mathbf{F}, G, Rg, X) \in$
final_states(initial_machine_state $X.initial\_state$)
E
(machine_execution_of_event_structure $E\ X.initial\_state$)
$\Longrightarrow$
partial_valid_execution $E\ X$

hb_machine_progress_thm =
$\forall E\ mst\ es\ path\ lbl\ init.$
$((\{\}, (\text{initial\_machine\_state}\ init)), path) \in$
traces_of_lts(lts_parallel(lts_po_of_event_structure E)(machine_lts $E.procs\ init$)) $\wedge$
final_state $path(lbl, (es, mst))$
$\Longrightarrow$
$\exists mst'.$
$(mst, \text{TAU}, mst') \in (\text{machine\_lts}\ E.procs\ init).trans \vee$
$(\exists es'\ l.$
$(es, l, es') \in (\text{lts\_po\_of\_event\_structure}\ E).trans \wedge$
$(mst, l, mst') \in (\text{machine\_lts}\ E.procs\ init).trans) \vee$
$((\forall p\ q.((\mathbf{fst}(\mathbf{snd}(\mathbf{snd}\ mst)))p\ q = [\,])) \wedge$
$(\forall es'\ l.\neg((es, l, es') \in (\text{lts\_po\_of\_event\_structure}\ E).trans)))$

Index