CHERI-MIPS capability monotonicity proof

Thomas Bauereiss

July 28, 2021

Contents

1 Abstract model of CHERI ISAs

1.1 Capability abstraction

Generated by Lem from *capabilities.lem*.

theory *Capabilities*

imports
Main
LEM.Lem-pervasives-extra

Sail.Sail2-values
Sail.Sail2-prompt-monad

begin

— *open import Pervasives-extra*
— *open import Sail2-values*
— *open import Sail2-prompt-monad*

record *perms* =

permit-ccall :: *bool*

permit-execute :: *bool*

permit-load :: *bool*

permit-load-capability :: *bool*

permit-seal :: *bool*

permit-store :: *bool*

permit-store-capability :: *bool*

permit-store-local-capability :: *bool*

permit-system-access :: *bool*

permit-unseal :: *bool*

record $'c$ *Capability-class*=

is-tagged-method :: $'c \Rightarrow$ *bool*

is-sealed-method :: $'c \Rightarrow$ *bool*

get-mem-region-method :: $'c \Rightarrow$ *nat set*

get-obj-type-method :: $'c \Rightarrow$ *nat*

get-perms-method :: $'c \Rightarrow$ *perms*

get-cursor-method :: $'c \Rightarrow$ *nat*

get-global-method :: $'c \Rightarrow$ *bool*

set-tag-method :: $'c \Rightarrow$ *bool* $\Rightarrow$ $'c$

set-seal-method :: *'c* ⇒ *bool* ⇒ *'c*

set-obj-type-method :: *'c* ⇒ *nat* ⇒ *'c*

set-perms-method :: *'c* ⇒ *perms* ⇒ *'c*

set-global-method :: *'c* ⇒ *bool* ⇒ *'c*

cap-of-mem-bytes-method :: *memory-byte list* ⇒ *bitU* ⇒ *'c option*

— *val seal : forall 'cap. Capability 'cap => 'cap -> nat -> 'cap*
definition *seal* :: *'cap Capability-class* ⇒ *'cap* ⇒ *nat* ⇒ *'cap* **where**
seal dict-Capabilities-Capability-cap c obj-type = (
(*set-seal-method dict-Capabilities-Capability-cap*) ((*set-obj-type-method dict-Capabilities-Capability-cap*) *c obj-type*) *True*)
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *c* :: *'cap*
and *obj-type* :: *nat*

— *val unseal : forall 'cap. Capability 'cap => 'cap -> bool -> 'cap*
definition *unseal* :: *'cap Capability-class* ⇒ *'cap* ⇒ *bool* ⇒ *'cap* **where**
unseal dict-Capabilities-Capability-cap c global1 = (
(*set-seal-method dict-Capabilities-Capability-cap*) ((*set-obj-type-method dict-Capabilities-Capability-cap*) ((*set-global-method dict-Capabilities-Capability-cap*) *c* (*global1* ∧ (*get-global-method dict-Capabilities-Capability-cap*) *c*))((*0* :: *nat*))) *False*)
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *c* :: *'cap*
and *global1* :: *bool*

— *val leq-perms : perms -> perms -> bool*
definition *leq-perms* :: *perms* ⇒ *perms* ⇒ *bool* **where**
leq-perms p1 p2 = (
((*permit-ccall p1*) ⟶(*permit-ccall p2*)) ∧
(((*permit-execute p1*) ⟶(*permit-execute p2*)) ∧
(((*permit-load p1*) ⟶(*permit-load p2*)) ∧
(((*permit-load-capability p1*) ⟶(*permit-load-capability p2*)) ∧
(((*permit-store p1*) ⟶(*permit-store p2*)) ∧
(((*permit-store-capability p1*) ⟶(*permit-store-capability p2*)) ∧
(((*permit-store-local-capability p1*) ⟶(*permit-store-local-capability p2*)) ∧
(((*permit-system-access p1*) ⟶(*permit-system-access p2*)) ∧
((*permit-unseal p1*) ⟶(*permit-unseal p2*))))))))))
for *p1* :: *perms*
and *p2* :: *perms*

— *val leq-cap : forall 'cap. Capability 'cap, Eq 'cap => 'cap −> 'cap −> bool*
definition *leq-cap* :: *'cap Capability-class ⇒ 'cap ⇒ 'cap ⇒ bool* **where**
leq-cap dict-Capabilities-Capability-cap c1 c2 = (
(*c1* = *c2*) ∨
((¬ ((*is-tagged-method dict-Capabilities-Capability-cap*) *c1*)) ∨
(((*is-tagged-method dict-Capabilities-Capability-cap*) *c2*) ∧
((¬ ((*is-sealed-method dict-Capabilities-Capability-cap*) *c1*) ∧ ¬ ((*is-sealed-method dict-Capabilities-Capability-cap*) *c2*)) ∧
((((*get-mem-region-method dict-Capabilities-Capability-cap*) *c1*) ⊆ ((*get-mem-region-method dict-Capabilities-Capability-cap*) *c2*)) ∧
(((*get-global-method dict-Capabilities-Capability-cap*) *c1* ⟶
(*get-global-method dict-Capabilities-Capability-cap*) *c2*) ∧
(*leq-perms* ((*get-perms-method dict-Capabilities-Capability-cap*) *c1*) ((*get-perms-method dict-Capabilities-Capability-cap*) *c2*))))))))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *c1* :: *'cap*
and *c2* :: *'cap*

— *val invokable : forall 'cap. Capability 'cap => 'cap −> 'cap −> bool*
definition *invokable* :: *'cap Capability-class ⇒ 'cap ⇒ 'cap ⇒ bool* **where**
invokable dict-Capabilities-Capability-cap cc cd1 = (
(*let pc* = ((*get-perms-method dict-Capabilities-Capability-cap*) *cc*) *in*
(*let pd* = ((*get-perms-method dict-Capabilities-Capability-cap*) *cd1*) *in* (*is-tagged-method dict-Capabilities-Capability-cap*) *cc* ∧ ((*is-tagged-method dict-Capabilities-Capability-cap*) *cd1* ∧
((*is-sealed-method dict-Capabilities-Capability-cap*) *cc* ∧ ((*is-sealed-method dict-Capabilities-Capability-cap*) *cd1* ∧
((*permit-ccall pc*) ∧ ((*permit-ccall pd*) ∧
(((*get-obj-type-method dict-Capabilities-Capability-cap*) *cc* =(*get-obj-type-method dict-Capabilities-Capability-cap*) *cd1*) ∧
((*permit-execute pc*) ∧ ((
(*get-cursor-method dict-Capabilities-Capability-cap*) *cc* ∈ (*get-mem-region-method dict-Capabilities-Capability-cap*) *cc*) ∧
¬(*permit-execute pd*)))))))))))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *cc* :: *'cap*
and *cd1* :: *'cap*

— *Derivation of capabilities, bounded by derivation depth to guarantee termination*
— *val cap-derivable-bounded : forall 'cap. Capability 'cap, SetType 'cap, Eq 'cap => nat −> set 'cap −> 'cap −> bool*
fun *cap-derivable-bounded* :: *'cap Capability-class ⇒ nat ⇒ 'cap set ⇒ 'cap ⇒ bool* **where**
cap-derivable-bounded dict-Capabilities-Capability-cap 0 C c = ((*c* ∈ *C*))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *C* :: *'cap set*

and *c* :: *'cap*
| *cap-derivable-bounded dict-Capabilities-Capability-cap* ((*Suc n*)) *C c* = (
((∃ *c'*. *cap-derivable-bounded*
dict-Capabilities-Capability-cap n C c' ∧ *leq-cap*
dict-Capabilities-Capability-cap c c')) ∨
((∃ *c'*. ∃ *c''*.
cap-derivable-bounded
dict-Capabilities-Capability-cap n C c' ∧
(*cap-derivable-bounded*
dict-Capabilities-Capability-cap n C c'' ∧
((*is-tagged-method dict-Capabilities-Capability-cap*) *c'* ∧ ((*is-tagged-method*
dict-Capabilities-Capability-cap) *c''* ∧ (¬ (
(*is-sealed-method dict-Capabilities-Capability-cap*) *c''*) ∧
(((*is-sealed-method dict-Capabilities-Capability-cap*) *c'* ∧ ((*permit-unseal* (
(*get-perms-method dict-Capabilities-Capability-cap*) *c''*)) ∧ (((*get-obj-type-method*
dict-Capabilities-Capability-cap) *c'* =(*get-cursor-method dict-Capabilities-Capability-cap*)
c'') ∧ (*unseal dict-Capabilities-Capability-cap c'* ((*get-global-method dict-Capabilities-Capability-cap*)
c'') = *c*)))) ∨
(¬ ((*is-sealed-method dict-Capabilities-Capability-cap*) *c'*) ∧ ((*permit-seal*
(
(*get-perms-method dict-Capabilities-Capability-cap*) *c''*)) ∧ (*seal dict-Capabilities-Capability-cap*
c' ((*get-cursor-method dict-Capabilities-Capability-cap*) *c''*) = *c*)))))))))))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *n* :: *nat*
and *C* :: *'cap set*
and *c* :: *'cap*

— *TODO*: *Prove an upper bound for the derivation depth. For a finite set of n capabilities, it seems a derivation depth of n+1 should be enough: If all but one capabilities in C are sealed, up to n−1 unsealing operations and possibly a restriction and a sealing operation might be necessary to derive the desired capability.*
definition *cap-derivable* :: *'a Capability-class* ⇒ *'a set* ⇒ *'a* ⇒ *bool* **where**
cap-derivable dict-Capabilities-Capability-a C c = (((∃ *n*. *cap-derivable-bounded*

dict-Capabilities-Capability-a n C c)))
for *dict-Capabilities-Capability-a* :: *'a Capability-class*
and *C* :: *'a set*
and *c* :: *'a*

— *val reads-from-reg* : *forall 'regval. event 'regval* −> *maybe register-name*
fun *reads-from-reg* :: *'regval event* ⇒(*string*)*option* **where**
reads-from-reg (*E-read-reg r* -) = (*Some r*)
for *r* :: *string*
| *reads-from-reg* - = (*None*)

— *val reads-reg-caps* : *forall 'regval 'cap. Capability 'cap, SetType 'cap* => (*'regval*

$-{>}$ *set 'cap*) $-{>}$ *event 'regval* $-{>}$ *set 'cap*
fun *reads-reg-caps* :: *'cap Capability-class* ⇒(*'regval* ⇒ *'cap set*)⇒ *'regval event* ⇒ *'cap set* **where**
reads-reg-caps dict-Capabilities-Capability-cap caps-of-regval1 (*E-read-reg - v*) = (*set-filter*
(*is-tagged-method dict-Capabilities-Capability-cap*) (*caps-of-regval1 v*))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *caps-of-regval1* :: *'regval* ⇒ *'cap set*
and *v* :: *'regval*
| *reads-reg-caps dict-Capabilities-Capability-cap caps-of-regval1 -* = ({})
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *caps-of-regval1* :: *'regval* ⇒ *'cap set*

— *val writes-to-reg* : *forall 'regval. event 'regval* $-{>}$ *maybe register-name*
fun *writes-to-reg* :: *'regval event* ⇒(*string*)*option* **where**
writes-to-reg (*E-write-reg r -*) = (*Some r*)
for *r* :: *string*
| *writes-to-reg -* = (*None*)

— *val writes-reg-caps* : *forall 'regval 'cap. Capability 'cap*, *SetType 'cap* $={>}$ (*'regval* $-{>}$ *set 'cap*) $-{>}$ *event 'regval* $-{>}$ *set 'cap*
fun *writes-reg-caps* :: *'cap Capability-class* ⇒(*'regval* ⇒ *'cap set*)⇒ *'regval event* ⇒ *'cap set* **where**
writes-reg-caps dict-Capabilities-Capability-cap caps-of-regval1 (*E-write-reg - v*) = (*set-filter*
(*is-tagged-method dict-Capabilities-Capability-cap*) (*caps-of-regval1 v*))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *caps-of-regval1* :: *'regval* ⇒ *'cap set*
and *v* :: *'regval*
| *writes-reg-caps dict-Capabilities-Capability-cap caps-of-regval1 -* = ({})
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *caps-of-regval1* :: *'regval* ⇒ *'cap set*

— *val reads-mem-val* : *forall 'regval. event 'regval* $-{>}$ *maybe* (*nat* ∗ *nat* ∗ *list memory-byte* ∗ *bitU*)
fun *reads-mem-val* :: *'regval event* ⇒(*nat*∗*nat*∗(*memory-byte*)*list*∗*bitU*)*option* **where**
reads-mem-val (*E-read-memt - addr sz* (*v*, *t*)) = (*Some* (*addr*, *sz*, *v*, *t*))
for *addr* :: *nat*
and *sz* :: *nat*
and *t* :: *bitU*
and *v* :: (*memory-byte*)*list*
| *reads-mem-val* (*E-read-mem - addr sz v*) = (*Some* (*addr*, *sz*, *v*, *B0*))
for *addr* :: *nat*
and *sz* :: *nat*
and *v* :: (*memory-byte*)*list*

```
| reads-mem-val - = ( None )


— val reads-mem-cap : forall 'regval 'cap. Capability 'cap => event 'regval ->
maybe (nat * nat * 'cap)
definition reads-mem-cap :: 'cap Capability-class ⇒ 'regval event ⇒(nat*nat*'cap) option
where
     reads-mem-cap dict-Capabilities-Capability-cap e = (
  Option.bind (reads-mem-val e) ( λx .
  (case  x of
     (addr, sz, v, t) =>
  Option.bind
   ((cap-of-mem-bytes-method   dict-Capabilities-Capability-cap) v t)
   (λ c .
    if(is-tagged-method   dict-Capabilities-Capability-cap) c then
      Some (addr, sz, c) else None)
  )))
  for  dict-Capabilities-Capability-cap  ::  'cap Capability-class
  and  e  ::  'regval event


— val writes-mem-val : forall 'regval. event 'regval -> maybe (nat * nat * list
memory-byte * bitU)
fun writes-mem-val  ::  'regval  event  ⇒(nat*nat*(memory-byte)list*bitU) option
where
     writes-mem-val (E-write-memt - addr sz v t -) = ( Some (addr, sz, v, t))
  for  addr  ::  nat
  and  sz  ::  nat
  and  t  ::  bitU
  and  v  :: (memory-byte)list
| writes-mem-val (E-write-mem - addr sz v -) = ( Some (addr, sz, v, B0))
  for  addr  ::  nat
  and  sz  ::  nat
  and  v  :: (memory-byte)list
| writes-mem-val - = ( None )


— val writes-mem-cap : forall 'regval 'cap. Capability 'cap => event 'regval ->
maybe (nat * nat * 'cap)
definition writes-mem-cap :: 'cap Capability-class ⇒ 'regval event ⇒(nat*nat*'cap) option
where
     writes-mem-cap dict-Capabilities-Capability-cap e = (
  Option.bind (writes-mem-val e) ( λx .
  (case  x of
     (addr, sz, v, t) =>
  Option.bind
   ((cap-of-mem-bytes-method   dict-Capabilities-Capability-cap) v t)
   (λ c .
    if(is-tagged-method   dict-Capabilities-Capability-cap) c then
```

Some (*addr*, *sz*, *c*) *else None*)
)))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *e* :: *'regval event*

end
theory *Cheri-axioms*

imports
Main
LEM.Lem-pervasives-extra
Sail.Sail2-values
Sail.Sail2-prompt-monad
Sail.Sail2-operators
Capabilities

begin

1.2 ISA abstraction

Generated by Lem from *cheri-axioms.lem*.

— *open import Pervasives-extra*
— *open import Sail2-values*
— *open import Sail2-prompt-monad*
— *open import Sail2-operators*
— *open import Capabilities*

— *TODO*: *Maybe add a read-kind for instruction fetches, so that we can distinguish loads from fetches in events and don't need to carry around the is-fetch parameter below*
datatype *acctype* = *Load* | *Store* | *Fetch*

record(*'cap*, *'regval*, *'instr*, *'e*) *isa* =

instr-sem :: *'instr* ⇒ (*'regval*, *unit*, *'e*) *monad*

instr-fetch :: (*'regval*, *'instr*, *'e*) *monad*

tag-granule :: *nat*

PCC :: *register-name* — *trace 'regval* −> *set*

KCC :: *register-name* — *trace 'regval* −> *set*

IDC :: *register-name* — *trace 'regval* −> *set*

caps-of-regval :: *'regval* ⇒ *'cap set*

invokes-caps :: *'instr* ⇒ *'regval trace* ⇒ *bool* *instr-raises-ex* :: *'instr* ⇒ *'regval trace* ⇒ *bool*

fetch-raises-ex :: *'regval trace* ⇒ *bool*

exception-targets :: *'regval set* ⇒ *'cap set*

privileged-regs :: *register-name* — *trace 'regval* −> *set*

translation-tables :: *'regval trace* ⇒ *nat set*

translate-address :: *nat* ⇒ *acctype* ⇒ *'regval trace* ⇒ *nat option*

definition *writes-mem-val-at-idx* :: *nat* ⇒(*'a event*)*list* ⇒(*nat*∗*nat*∗(*memory-byte*)*list*∗*bitU*)*option*
where
writes-mem-val-at-idx i t = (*Option.bind* (*index t i*) *writes-mem-val*)
for *i* :: *nat*
and *t* :: (*'a event*)*list*

definition *writes-mem-cap-at-idx* :: *'a Capability-class* ⇒ *nat* ⇒(*'b event*)*list* ⇒(*nat*∗*nat*∗*'a*)*option* **where**
writes-mem-cap-at-idx dict-Capabilities-Capability-a i t = (*Option.bind* (*index t i*)
(*writes-mem-cap dict-Capabilities-Capability-a*))
for *dict-Capabilities-Capability-a* :: *'a Capability-class*
and *i* :: *nat*
and *t* :: (*'b event*)*list*

definition *writes-to-reg-at-idx* :: *nat* ⇒(*'a event*)*list* ⇒(*string*)*option* **where**
writes-to-reg-at-idx i t = (*Option.bind* (*index t i*) *writes-to-reg*)
for *i* :: *nat*
and *t* :: (*'a event*)*list*

definition *writes-reg-caps-at-idx* :: *'d Capability-class* ⇒(*'d*,*'c*,*'b*,*'a*)*isa* ⇒ *nat* ⇒(*'c event*)*list* ⇒ *'d set* **where**
writes-reg-caps-at-idx dict-Capabilities-Capability-d ISA i t = (*case-option* {} (*writes-reg-caps*
dict-Capabilities-Capability-d(*caps-of-regval ISA*)) (*index t i*))
for *dict-Capabilities-Capability-d* :: *'d Capability-class*
and *ISA* :: (*'d*,*'c*,*'b*,*'a*)*isa*
and *i* :: *nat*
and *t* :: (*'c event*)*list*

definition *reads-mem-val-at-idx* :: *nat* ⇒(*'a event*)*list* ⇒(*nat*∗*nat*∗(*memory-byte*)*list*∗*bitU*)*option*
where
reads-mem-val-at-idx i t = (*Option.bind* (*index t i*) *reads-mem-val*)
for *i* :: *nat*
and *t* :: (*'a event*)*list*

definition *reads-mem-cap-at-idx* :: *'a Capability-class* ⇒ *nat* ⇒(*'b event*)*list* ⇒(*nat*∗*nat*∗*'a*)*option* **where**
reads-mem-cap-at-idx dict-Capabilities-Capability-a i t = (*Option.bind* (*index t i*)
(*reads-mem-cap dict-Capabilities-Capability-a*))
for *dict-Capabilities-Capability-a* :: *'a Capability-class*
and *i* :: *nat*
and *t* :: (*'b event*)*list*

definition *reads-from-reg-at-idx* :: *nat* ⇒(*'a event*)*list* ⇒(*string*)*option* **where**

reads-from-reg-at-idx i t = (*Option.bind* (*index t i*) *reads-from-reg*)
for *i* :: *nat*
and *t* :: (*'a event*)*list*

definition *reads-reg-caps-at-idx* :: *'d Capability-class* ⇒(*'d*,*'c*,*'b*,*'a*)*isa* ⇒ *nat* ⇒(*'c event*)*list* ⇒ *'d set* **where**
reads-reg-caps-at-idx dict-Capabilities-Capability-d ISA i t = (*case-option* {} (*reads-reg-caps*
dict-Capabilities-Capability-d(*caps-of-regval ISA*)) (*index t i*))
for *dict-Capabilities-Capability-d* :: *'d Capability-class*
and *ISA* :: (*'d*,*'c*,*'b*,*'a*)*isa*
and *i* :: *nat*
and *t* :: (*'c event*)*list*

— *val address-range : nat −> nat −> list nat*
definition *address-range* :: *nat* ⇒ *nat* ⇒(*nat*)*list* **where**
address-range start len = (*genlist* (λ *n* . *start* + *n*) *len*)
for *start* :: *nat*
and *len* :: *nat*

— *val address-tag-aligned : forall 'cap 'regval 'instr 'e. isa 'cap 'regval 'instr 'e −> nat −> bool*
definition *address-tag-aligned* :: (*'cap*,*'regval*,*'instr*,*'e*)*isa* ⇒ *nat* ⇒ *bool* **where**

address-tag-aligned ISA addr = (((*addr mod*(*tag-granule ISA*)) =(*0* :: *nat*)))
for *ISA* :: (*'cap*,*'regval*,*'instr*,*'e*)*isa*
and *addr* :: *nat*

— *val cap-reg-written-before-idx : forall 'cap 'regval 'instr 'e. Capability 'cap, Eq 'cap, SetType 'cap => isa 'cap 'regval 'instr 'e −> nat −> register-name −> trace 'regval −> bool*
definition *cap-reg-written-before-idx* :: *'cap Capability-class* ⇒(*'cap*,*'regval*,*'instr*,*'e*)*isa* ⇒ *nat* ⇒ *string* ⇒(*'regval event*)*list* ⇒ *bool* **where**

cap-reg-written-before-idx dict-Capabilities-Capability-cap ISA i r t = (((∃ *j*. (*j* < *i*) ∧ ((*writes-to-reg-at-idx j t* = *Some r*) ∧ ¬ (*writes-reg-caps-at-idx dict-Capabilities-Capability-cap ISA j t* = {})))))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *ISA* :: (*'cap*,*'regval*,*'instr*,*'e*)*isa*
and *i* :: *nat*
and *r* :: *string*
and *t* :: (*'regval event*)*list*

— *val system-access-permitted-before-idx : forall 'cap 'regval 'instr 'e. Capability 'cap, SetType 'cap, Eq 'cap => isa 'cap 'regval 'instr 'e -> nat -> trace 'regval -> bool*
definition *system-access-permitted-before-idx* :: *'cap Capability-class* ⇒(*'cap*,*'regval*,*'instr*,*'e*)*isa* ⇒ *nat* ⇒(*'regval event*)*list* ⇒ *bool* **where**
system-access-permitted-before-idx dict-Capabilities-Capability-cap ISA i t = (
((∃ *j*. ∃ *r*. ∃ *c*.
(*j* < *i*) ∧
((*reads-from-reg-at-idx j t* = *Some r*) ∧
(¬ (*cap-reg-written-before-idx dict-Capabilities-Capability-cap ISA j r t*) ∧
((*c* ∈ (*reads-reg-caps-at-idx dict-Capabilities-Capability-cap ISA j t*)) ∧
((*r* ∈(*PCC ISA*)) ∧ ((*r* ∉(*privileged-regs ISA*)) ∧
((*is-tagged-method dict-Capabilities-Capability-cap*) *c* ∧ (¬ ((*is-sealed-method dict-Capabilities-Capability-cap*) *c*) ∧(*permit-system-access*
((*get-perms-method dict-Capabilities-Capability-cap*) *c*))))))))))))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *ISA* :: (*'cap*,*'regval*,*'instr*,*'e*)*isa*
and *i* :: *nat*
and *t* :: (*'regval event*)*list*

— *val permits-cap-load : forall 'cap. Capability 'cap => 'cap -> nat -> nat -> bool*
definition *permits-cap-load* :: *'cap Capability-class* ⇒ *'cap* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
permits-cap-load dict-Capabilities-Capability-cap c vaddr sz = (
((*is-tagged-method dict-Capabilities-Capability-cap*) *c* ∧ (¬ ((*is-sealed-method dict-Capabilities-Capability-cap*) *c*) ∧
((*List.set* (*address-range vaddr sz*) ⊆ (
(*get-mem-region-method dict-Capabilities-Capability-cap*) *c*)) ∧(*permit-load-capability*
((*get-perms-method dict-Capabilities-Capability-cap*) *c*))))))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *c* :: *'cap*
and *vaddr* :: *nat*
and *sz* :: *nat*

— *val available-caps : forall 'cap 'regval 'instr 'e. Capability 'cap, Eq 'cap, SetType 'cap => isa 'cap 'regval 'instr 'e -> nat -> trace 'regval -> set 'cap*

fun *available-caps* :: *'cap Capability-class* ⇒(*'cap,'regval,'instr,'e*)*isa* ⇒ *nat* ⇒(*'regval event*)*list* ⇒ *'cap set* **where**

```
    available-caps dict-Capabilities-Capability-cap ISA 0 t = ( {} )
  for dict-Capabilities-Capability-cap :: 'cap Capability-class
  and ISA :: ('cap,'regval,'instr,'e)isa
  and t :: ('regval event)list
| available-caps dict-Capabilities-Capability-cap ISA ((Suc i)) t = (
  (let caps-of = (λ e .
                ((case reads-mem-cap dict-Capabilities-Capability-cap e of
                      Some (-, -, c) => {c}
                  | None => {}
                 )) ∪
                 ((case reads-from-reg e of
                        Some r =>
                  if (¬
                        (cap-reg-written-before-idx
                           dict-Capabilities-Capability-cap ISA i
                         r t) ∧
                        (system-access-permitted-before-idx
                           dict-Capabilities-Capability-cap ISA i
                         t ∨ ¬ (r ∈ (privileged-regs ISA))))
                  then
                    reads-reg-caps dict-Capabilities-Capability-cap
                      (caps-of-regval ISA) e else {}
                    | None => {}
                  ))) in
  (let new-caps = (case-option {} caps-of (index t i)) in
  (available-caps dict-Capabilities-Capability-cap ISA i t) ∪ new-caps)) )
  for dict-Capabilities-Capability-cap :: 'cap Capability-class
  and ISA :: ('cap,'regval,'instr,'e)isa
  and i :: nat
  and t :: ('regval event)list
```

1.3 Intra-instruction properties

— *val read-reg-axiom : forall 'cap 'regval 'instr 'e. Capability 'cap, SetType 'cap, Eq 'cap, Eq 'regval => isa 'cap 'regval 'instr 'e -> bool -> trace 'regval -> bool*

definition *read-reg-axiom* :: *'cap Capability-class* ⇒(*'cap,'regval,'instr,'e*)*isa* ⇒ *bool* ⇒(*'regval event*)*list* ⇒ *bool* **where**

```
    read-reg-axiom dict-Capabilities-Capability-cap ISA has-ex t = (
  ((∀ i. ∀ r. ∀ v.
     ((index t i = Some (E-read-reg r v)) ∧ (r ∈(privileged-regs ISA)))
     ⟶
     (system-access-permitted-before-idx
  dict-Capabilities-Capability-cap ISA i t ∨
```

$(has\text{-}ex$ — && r IN ISA.KCC$)))))$
for $dict\text{-}Capabilities\text{-}Capability\text{-}cap$ :: $'cap$ $Capability\text{-}class$
and ISA :: $('cap,'regval,'instr,'e)isa$
and $has\text{-}ex$ :: $bool$
and t :: $('regval\ event)list$

— *val store-cap-mem-axiom : forall 'cap 'regval 'instr 'e. Capability 'cap, SetType 'cap, Eq 'cap => isa 'cap 'regval 'instr 'e −> trace 'regval −> bool*
definition *store-cap-mem-axiom* :: $'cap$ $Capability\text{-}class$ $\Rightarrow('cap,'regval,'instr,'e)isa$ $\Rightarrow('regval\ event)list$ $\Rightarrow$ $bool$ **where**
store-cap-mem-axiom dict-Capabilities-Capability-cap ISA t = (
$((\forall\ i.\ \forall\ c.\ \forall\ addr.\ \forall\ sz.$
$(writes\text{-}mem\text{-}cap\text{-}at\text{-}idx$
$dict\text{-}Capabilities\text{-}Capability\text{-}cap\ i\ t = Some\ (addr,\ sz,\ c))$
$\longrightarrow$
$(cap\text{-}derivable\ dict\text{-}Capabilities\text{-}Capability\text{-}cap\ (available\text{-}caps\ dict\text{-}Capabilities\text{-}Capability\text{-}cap$
$ISA\ i\ t)\ c))))$
for $dict\text{-}Capabilities\text{-}Capability\text{-}cap$ :: $'cap$ $Capability\text{-}class$
and ISA :: $('cap,'regval,'instr,'e)isa$
and t :: $('regval\ event)list$

— *val store-cap-reg-axiom : forall 'cap 'regval 'instr 'e. Capability 'cap, SetType 'cap, SetType 'regval, Eq 'cap, Eq 'regval => isa 'cap 'regval 'instr 'e −> bool −> bool −> trace 'regval −> bool*
definition *store-cap-reg-axiom* :: $'cap$ $Capability\text{-}class$ $\Rightarrow('cap,'regval,'instr,'e)isa$ $\Rightarrow$ $bool$ $\Rightarrow$ $bool$ $\Rightarrow('regval\ event)list$ $\Rightarrow$ $bool$ **where**
store-cap-reg-axiom dict-Capabilities-Capability-cap ISA has-ex invokes-caps1 t = (
$((\forall\ i.\ \forall\ c.\ \forall\ r.$
$((writes\text{-}to\text{-}reg\text{-}at\text{-}idx\ i\ t = Some\ r) \wedge (c \in (writes\text{-}reg\text{-}caps\text{-}at\text{-}idx$
$dict\text{-}Capabilities\text{-}Capability\text{-}cap\ ISA\ i\ t)))$
$\longrightarrow$
$(cap\text{-}derivable\ dict\text{-}Capabilities\text{-}Capability\text{-}cap\ (available\text{-}caps\ dict\text{-}Capabilities\text{-}Capability\text{-}cap$
$ISA\ i\ t)\ c \vee$
$((has\text{-}ex \wedge$
(
(— *exists r′ v′ j. j < i && index t j = Just (E-read-reg r′ v′) &&c ∈(exception-targets ISA)* $\{v'\ .(\ \exists\ r'.\ \ \exists\ j.\ \ (j < i) \wedge ((index\ t\ j = Some\ (E\text{-}read\text{-}reg\ r'\ v')) \wedge (r' \in (KCC\ \ \ ISA))))\}) \wedge$
(
— *reads-from-reg-at-idx j t = Just r′ && c′ IN (reads-reg-caps-at-idx ISA j t) && leq-cap c c′ && r′ IN (ISA.KCC ⟨take j t⟩) &&r* $\in ((PCC\ \ \ ISA)$ — *take i t*$)))) \vee$
$((\exists\ cc.\ \exists\ cd0.$
$invokes\text{-}caps1 \wedge$
$(cap\text{-}derivable$
$dict\text{-}Capabilities\text{-}Capability\text{-}cap\ (available\text{-}caps\ dict\text{-}Capabilities\text{-}Capability\text{-}cap\ ISA$

$i\ t)\ cc \wedge$
$(cap\text{-}derivable$
$dict\text{-}Capabilities\text{-}Capability\text{-}cap\ (available\text{-}caps\ dict\text{-}Capabilities\text{-}Capability\text{-}cap\ ISA$
$i\ t)\ cd0 \wedge$
$(invokable\ dict\text{-}Capabilities\text{-}Capability\text{-}cap\ cc\ cd0\ \wedge$
$((leq\text{-}cap\ dict\text{-}Capabilities\text{-}Capability\text{-}cap\ c\ (unseal\ dict\text{-}Capabilities\text{-}Capability\text{-}cap$
$cc\ True) \wedge (r \in (PCC\quad ISA))) \vee$
$(leq\text{-}cap\ dict\text{-}Capabilities\text{-}Capability\text{-}cap\ c\ (unseal\ dict\text{-}Capabilities\text{-}Capability\text{-}cap$
$cd0\ True) \wedge (r \in (IDC\quad ISA)))))))))))))$
for $dict\text{-}Capabilities\text{-}Capability\text{-}cap\ ::\ 'cap\ Capability\text{-}class$
and $ISA\ ::\ ('cap,'regval,'instr,'e)isa$
and $has\text{-}ex\ ::\ bool$
and $invokes\text{-}caps1\ ::\ bool$
and $t\ ::\ ('regval\ event)list$

— *val load-mem-axiom : forall 'cap 'regval 'instr 'e. Capability 'cap, SetType 'cap, Eq 'cap => isa 'cap 'regval 'instr 'e -> bool -> trace 'regval -> bool*
definition $load\text{-}mem\text{-}axiom\ ::\ 'cap\ Capability\text{-}class \Rightarrow ('cap,'regval,'instr,'e)isa$
$\Rightarrow bool \Rightarrow ('regval\ event)list \Rightarrow bool$ **where**
$load\text{-}mem\text{-}axiom\ dict\text{-}Capabilities\text{-}Capability\text{-}cap\ ISA\ is\text{-}fetch\ t = ($
$((\forall\ i.\ \forall\ paddr.\ \forall\ sz.\ \forall\ v.\ \forall\ tag.$
$((reads\text{-}mem\text{-}val\text{-}at\text{-}idx\ i\ t = Some\ (paddr,\ sz,\ v,\ tag)) \wedge$
$\neg\ (paddr \in ((translation\text{-}tables\quad ISA)\ (List.take\ i\ t))))$
$\longrightarrow$
$((\exists\ c'.\ \exists\ vaddr.$
$cap\text{-}derivable$
$dict\text{-}Capabilities\text{-}Capability\text{-}cap\ (available\text{-}caps\ dict\text{-}Capabilities\text{-}Capability\text{-}cap\ ISA$
$i\ t)\ c' \wedge ($
$(is\text{-}tagged\text{-}method\quad dict\text{-}Capabilities\text{-}Capability\text{-}cap)\ c' \wedge (\neg\ ((is\text{-}sealed\text{-}method$
$dict\text{-}Capabilities\text{-}Capability\text{-}cap)\ c') \wedge$
$(((translate\text{-}address\quad ISA)\ vaddr\ (if\ is\text{-}fetch\ then\ Fetch\ else\ Load)\ (List.take$
$i\ t) = Some\ paddr) \wedge$
$((List.set\ (address\text{-}range\ vaddr\ sz) \subseteq ($
$(get\text{-}mem\text{-}region\text{-}method\quad dict\text{-}Capabilities\text{-}Capability\text{-}cap)\ c')) \wedge$
$((if\ is\text{-}fetch\ then(permit\text{-}execute\quad ($
$(get\text{-}perms\text{-}method\quad dict\text{-}Capabilities\text{-}Capability\text{-}cap)\ c'))\ else(permit\text{-}load\quad ($
$(get\text{-}perms\text{-}method\quad dict\text{-}Capabilities\text{-}Capability\text{-}cap)\ c'))) \wedge$
$((is\text{-}fetch \longrightarrow (tag = B0)) \wedge$
$((tag \neq B0) \longrightarrow ((permit\text{-}load\text{-}capability\quad ($
$(get\text{-}perms\text{-}method\quad dict\text{-}Capabilities\text{-}Capability\text{-}cap)\ c')) \wedge ((sz = (tag\text{-}granule$
$ISA)) \wedge address\text{-}tag\text{-}aligned\ ISA\ paddr))))))))))))))$
for $dict\text{-}Capabilities\text{-}Capability\text{-}cap\ ::\ 'cap\ Capability\text{-}class$
and $ISA\ ::\ ('cap,'regval,'instr,'e)isa$
and $is\text{-}fetch\ ::\ bool$
and $t\ ::\ ('regval\ event)list$

— *val mem-val-is-cap : forall 'cap 'regval 'instr 'e. Capability 'cap, SetType 'cap,*

Eq 'cap => isa 'cap 'regval 'instr 'e −> list memory-byte −> bitU −> bool
definition *mem-val-is-cap* :: *'cap Capability-class* ⇒(*'cap*,*'regval*,*'instr*,*'e*)*isa* ⇒(*memory-byte*)*list* ⇒ *bitU* ⇒ *bool* **where**
mem-val-is-cap dict-Capabilities-Capability-cap - v t = (((∃ *c*.
(*cap-of-mem-bytes-method dict-Capabilities-Capability-cap*) *v t* = *Some* (*c* :: *'cap*))))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *v* :: (*memory-byte*)*list*
and *t* :: *bitU*

— *val mem-val-is-local-cap : forall 'cap 'regval 'instr 'e. Capability 'cap, SetType 'cap, Eq 'cap => isa 'cap 'regval 'instr 'e −> list memory-byte −> bitU −> bool*
definition *mem-val-is-local-cap* :: *'cap Capability-class* ⇒(*'cap*,*'regval*,*'instr*,*'e*)*isa* ⇒(*memory-byte*)*list* ⇒ *bitU* ⇒ *bool* **where**
mem-val-is-local-cap dict-Capabilities-Capability-cap - v t = (((∃ *c*. (
(*cap-of-mem-bytes-method dict-Capabilities-Capability-cap*) *v t* = *Some* (*c* :: *'cap*)) ∧ ¬ (
(*get-global-method dict-Capabilities-Capability-cap*) *c*))))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *v* :: (*memory-byte*)*list*
and *t* :: *bitU*

— *val store-tag-axiom : forall 'cap 'regval 'instr 'e. Capability 'cap, SetType 'cap, Eq 'cap => isa 'cap 'regval 'instr 'e −> trace 'regval −> bool*
definition *store-tag-axiom* :: *'cap Capability-class* ⇒(*'cap*,*'regval*,*'instr*,*'e*)*isa* ⇒(*'regval event*)*list* ⇒ *bool* **where**
store-tag-axiom dict-Capabilities-Capability-cap ISA t = (
((∀ *i*. ∀ *paddr*. ∀ *sz*. ∀ *v*. ∀ *tag*.
(*writes-mem-val-at-idx i t* = *Some* (*paddr*, *sz*, *v*, *tag*))
⟶
((*List.length v* = *sz*) ∧
(((*tag* = *B0*) ∨ (*tag* = *B1*)) ∧
((*tag* = *B1*) ⟶ (*address-tag-aligned ISA paddr* ∧ (*sz* =(*tag-granule ISA*)))))))))

for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *ISA* :: (*'cap*,*'regval*,*'instr*,*'e*)*isa*
and *t* :: (*'regval event*)*list*

— *val store-mem-axiom : forall 'cap 'regval 'instr 'e. Capability 'cap, SetType 'cap, Eq 'cap => isa 'cap 'regval 'instr 'e −> trace 'regval −> bool*
definition *store-mem-axiom* :: *'cap Capability-class* ⇒(*'cap*,*'regval*,*'instr*,*'e*)*isa* ⇒(*'regval event*)*list* ⇒ *bool* **where**
store-mem-axiom dict-Capabilities-Capability-cap ISA t = (
((∀ *i*. ∀ *paddr*. ∀ *sz*. ∀ *v*. ∀ *tag*.
((*writes-mem-val-at-idx i t* = *Some* (*paddr*, *sz*, *v*, *tag*)) ∧
¬ (*paddr* ∈ ((*translation-tables ISA*) (*List.take i t*))))

$\longrightarrow$
$((\exists\ c'.\ \exists\ vaddr.$
cap-derivable
dict-Capabilities-Capability-cap (*available-caps dict-Capabilities-Capability-cap ISA*
i t) $c' \wedge$ (
(*is-tagged-method* *dict-Capabilities-Capability-cap*) $c' \wedge (\neg$ ((*is-sealed-method*
dict-Capabilities-Capability-cap) c') $\wedge$
(((*translate-address* *ISA*) *vaddr Store* (*List.take i t*) = *Some paddr*) $\wedge$
((*List.set* (*address-range vaddr sz*) $\subseteq$ (
(*get-mem-region-method* *dict-Capabilities-Capability-cap*) c')) $\wedge$
((*permit-store* (
(*get-perms-method* *dict-Capabilities-Capability-cap*) c')) $\wedge$
(((*mem-val-is-cap*
dict-Capabilities-Capability-cap ISA v tag $\wedge$ (*tag* = *B1*)) $\longrightarrow$(*permit-store-capability*
(
(*get-perms-method* *dict-Capabilities-Capability-cap*) c'))) $\wedge$
((*mem-val-is-local-cap*
dict-Capabilities-Capability-cap ISA v tag $\wedge$ (*tag* = *B1*)) $\longrightarrow$(*permit-store-local-capability*
(
(*get-perms-method* *dict-Capabilities-Capability-cap*) c'))))))))))))))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *ISA* :: (*'cap,'regval,'instr,'e*)*isa*
and *t* :: (*'regval event*)*list*

— *val cheri-axioms : forall 'cap 'regval 'instr 'e. Capability 'cap, SetType 'cap, SetType 'regval, Eq 'cap, Eq 'regval => isa 'cap 'regval 'instr 'e -> bool -> bool -> bool -> trace 'regval -> bool*
definition *cheri-axioms* :: *'cap Capability-class* $\Rightarrow$(*'cap,'regval,'instr,'e*)*isa* $\Rightarrow$ *bool* $\Rightarrow$ *bool* $\Rightarrow$ *bool* $\Rightarrow$(*'regval event*)*list* $\Rightarrow$ *bool* **where**
cheri-axioms dict-Capabilities-Capability-cap ISA is-fetch has-ex invokes-caps1
t = (
store-cap-mem-axiom
dict-Capabilities-Capability-cap ISA t $\wedge$
(*store-cap-reg-axiom*
dict-Capabilities-Capability-cap ISA has-ex invokes-caps1 t $\wedge$
(*read-reg-axiom dict-Capabilities-Capability-cap ISA has-ex t* $\wedge$
(*load-mem-axiom dict-Capabilities-Capability-cap ISA is-fetch t* $\wedge$
(*store-tag-axiom dict-Capabilities-Capability-cap ISA t* $\wedge$
store-mem-axiom dict-Capabilities-Capability-cap ISA t)))))
for *dict-Capabilities-Capability-cap* :: *'cap Capability-class*
and *ISA* :: (*'cap,'regval,'instr,'e*)*isa*
and *is-fetch* :: *bool*
and *has-ex* :: *bool*
and *invokes-caps1* :: *bool*
and *t* :: (*'regval event*)*list*

end
theory *Capabilities-lemmas*

imports *Capabilities*
begin

1.4 Helper definitions and lemmas

locale *Capabilities* =
fixes *CC* :: *'cap Capability-class*
assumes *is-tagged-set-tag*[*simp*]: $\bigwedge$*c tag. is-tagged-method CC* (*set-tag-method CC c tag*) = *tag*
and *is-tagged-set-seal*[*simp*]: $\bigwedge$*c s. is-tagged-method CC* (*set-seal-method CC c s*) = *is-tagged-method CC c*
and *is-tagged-set-obj-type*[*simp*]: $\bigwedge$*c t. is-tagged-method CC* (*set-obj-type-method CC c t*) = *is-tagged-method CC c*
and *is-tagged-set-perms*[*simp*]: $\bigwedge$*c p. is-tagged-method CC* (*set-perms-method CC c p*) = *is-tagged-method CC c*
and *is-tagged-cap-of-mem-bytes*[*simp*]: $\bigwedge$*c bytes tag. cap-of-mem-bytes-method CC bytes tag* = *Some c* $\Longrightarrow$ *is-tagged-method CC c* $\longleftrightarrow$ *tag* = *B1*
begin

inductive-set *derivable* :: *'cap set* $\Rightarrow$ *'cap set* **for** *C* :: *'cap set* **where**
Copy: *c* $\in$ *C* $\Longrightarrow$ *c* $\in$ *derivable C*
| *Restrict*: *c'* $\in$ *derivable C* $\Longrightarrow$ *leq-cap CC c c'* $\Longrightarrow$ *c* $\in$ *derivable C*
| *Unseal*:
⟦*c'* $\in$ *derivable C*; *c''* $\in$ *derivable C*; *is-tagged-method CC c'*; *is-tagged-method CC c''*;
¬*is-sealed-method CC c''*; *is-sealed-method CC c'*; *permit-unseal* (*get-perms-method CC c''*);
get-obj-type-method CC c' = *get-cursor-method CC c''*⟧ $\Longrightarrow$
unseal CC c' (*get-global-method CC c''*) $\in$ *derivable C*
| *Seal*:
⟦*c'* $\in$ *derivable C*; *c''* $\in$ *derivable C*; *is-tagged-method CC c'*; *is-tagged-method CC c''*;
¬*is-sealed-method CC c''*; ¬*is-sealed-method CC c'*; *permit-seal* (*get-perms-method CC c''*)⟧ $\Longrightarrow$
seal CC c' (*get-cursor-method CC c''*) $\in$ *derivable C*

lemma *leq-cap-refl*[*simp*, *intro*]:
leq-cap CC c c
by (*simp add*: *leq-cap-def*)

lemma *leq-cap-tag-imp*[*intro*]:
assumes *leq-cap CC c c'*
and *is-tagged-method CC c*
shows *is-tagged-method CC c'*
using *assms*
by (*auto simp*: *leq-cap-def*)

lemma *derivable-mono*:
assumes *C* $\subseteq$ *C'*

shows *derivable* $C \subseteq$ *derivable* C'
proof
fix c
assume $c \in$ *derivable* C
then show $c \in$ *derivable* C' **using** *assms* **by** *induction* (*auto intro*: *derivable.intros*)
qed

lemma *cap-derivable-bounded-gteq*:
assumes c: *cap-derivable-bounded CC n C c*
and m: $m \geq n$
shows *cap-derivable-bounded CC m C c*
proof $-$
from m **obtain** k **where** $m = n + k$ **using** *less-iff-Suc-add*[*of n m*] **by** *auto*
also have *cap-derivable-bounded CC* $(n + k)$ *C c* **using** c **by** (*induction k*) (*auto*)
finally show *?thesis* **.**
qed

lemma *derivable-refl*: $C \subseteq$ *derivable* C **by** (*auto intro*: *derivable.intros*)

lemma *derivable-union-subseteq-absorb*:
assumes $C' \subseteq$ *derivable* C
shows *derivable* $(C \cup C')$ = *derivable* C
proof
show *derivable* $(C \cup C') \subseteq$ *derivable* C
proof
fix c
assume $c \in$ *derivable* $(C \cup C')$
then show $c \in$ *derivable* C **using** *assms* **by** *induction* (*auto intro*: *derivable.intros*)
qed
show *derivable* $C \subseteq$ *derivable* $(C \cup C')$ **by** (*intro derivable-mono*) *auto*
qed

lemma *derivable-minus-subseteq*: *derivable* $(C - C') \subseteq$ *derivable* C
proof
fix c
assume $c \in$ *derivable* $(C - C')$
then show $c \in$ *derivable* C **by** *induction* (*auto intro*: *derivable.intros*)
qed

lemma *cap-derivable-iff-derivable*: *cap-derivable CC C c* $\longleftrightarrow$ $c \in$ *derivable* C
proof
assume *cap-derivable CC C c*
then obtain n **where** c: *cap-derivable-bounded CC n C c* **by** (*auto simp*: *cap-derivable-def*)
then show $c \in$ *derivable* C
by (*induction CC* $\equiv$ *CC n C c rule*: *cap-derivable-bounded.induct*)

(*auto intro*: *derivable.intros*)
next
assume *c*: *c* ∈ *derivable C*
then have ∃ *n*. *cap-derivable-bounded CC n C c*
proof (*induction rule*: *derivable.induct*)
case (*Copy c*)
then have *cap-derivable-bounded CC 0 C c* **by** *auto*
then show *?case* **by** *blast*
next
case (*Restrict c′ c*)
then obtain *n* **where** *cap-derivable-bounded CC n C c′* **by** *auto*
then have *cap-derivable-bounded CC* (*Suc n*) *C c* **using** *Restrict.hyps* **by** *auto*
then show *?case* **by** *blast*
next
case (*Unseal c′ c″*)
then obtain *n′ n″*
where *cap-derivable-bounded CC n′ C c′* **and** *cap-derivable-bounded CC n″ C c″*
by *blast*
then have *cap-derivable-bounded CC* (*max n′ n″*) *C c′*
and *cap-derivable-bounded CC* (*max n′ n″*) *C c″*
by (*auto intro*: *cap-derivable-bounded-gteq*)
then have *cap-derivable-bounded CC* (*Suc* (*max n′ n″*)) *C* (*unseal CC c′* (*get-global-method CC c″*))
using *Unseal.hyps*
by *auto*
then show *?case* **by** *blast*
next
case (*Seal c′ c″*)
then obtain *n′ n″*
where *cap-derivable-bounded CC n′ C c′* **and** *cap-derivable-bounded CC n″ C c″*
by *blast*
then have *cap-derivable-bounded CC* (*max n′ n″*) *C c′*
and *cap-derivable-bounded CC* (*max n′ n″*) *C c″*
by (*auto intro*: *cap-derivable-bounded-gteq*)
then have *cap-derivable-bounded CC* (*Suc* (*max n′ n″*)) *C* (*seal CC c′* (*get-cursor-method CC c″*))
using *Seal.hyps*
by *auto*
then show *?case* **by** *blast*
qed
then show *cap-derivable CC C c* **by** (*simp add*: *cap-derivable-def*)
qed

end

end
theory *Cheri-axioms-lemmas*

imports *Capabilities-lemmas Cheri-axioms*
begin

locale *Capability-ISA* = *Capabilities CC*
for *CC* :: *'cap Capability-class* +
fixes *ISA* :: (*'cap*, *'regval*, *'instr*, *'e*) *isa*

lemma *reads-from-reg-at-idx-Some-iff* [*simp*]:
reads-from-reg-at-idx i t = *Some r* $\longleftrightarrow$ *reads-from-reg* (*t* ! *i*) = *Some r* $\wedge$ *i* < *length t*
by (*auto simp*: *reads-from-reg-at-idx-def bind-eq-Some-conv*)

lemma *reads-from-reg-SomeE*[*elim*!]:
assumes *reads-from-reg e* = *Some r*
obtains *v* **where** *e* = *E-read-reg r v*
using *assms*
by (*cases e*) *auto*

lemma *reads-from-reg-Some-iff*:
reads-from-reg e = *Some r* $\longleftrightarrow$ ($\exists$ *v*. *e* = *E-read-reg r v*)
by (*cases e*) *auto*

lemma *member-reads-reg-caps-at-idx-iff* [*simp*]:
c $\in$ *reads-reg-caps-at-idx CC ISA i t* $\longleftrightarrow$
c $\in$ *reads-reg-caps CC* (*caps-of-regval ISA*) (*t* ! *i*) $\wedge$ *i* < *length t*
by (*auto simp*: *reads-reg-caps-at-idx-def split*: *option.splits*)

lemma *member-reads-reg-caps-iff*:
c $\in$ *reads-reg-caps CC c-of-r e* $\longleftrightarrow$
($\exists$ *r v*. *e* = *E-read-reg r v* $\wedge$ *c* $\in$ *c-of-r v* $\wedge$ *is-tagged-method CC c*)
by (*cases e*) *auto*

lemma *member-reads-reg-capsE*[*elim*!]:
assumes *c* $\in$ *reads-reg-caps CC c-of-r e*
obtains *r v* **where** *e* = *E-read-reg r v* **and** *c* $\in$ *c-of-r v* **and** *is-tagged-method CC c*
using *assms*
by (*auto simp*: *member-reads-reg-caps-iff*)

lemma *reads-reg-caps-Some-reads-mem-cap-None*[*simp*]:
assumes *c* $\in$ *reads-reg-caps CC cor e*
shows *reads-mem-cap CC e* = *None*
using *assms* **by** (*cases e*) (*auto simp*: *reads-mem-cap-def*)

lemma *writes-to-reg-at-idx-Some-iff* [*simp*]:
writes-to-reg-at-idx i t = *Some r* $\longleftrightarrow$ *writes-to-reg* (*t* ! *i*) = *Some r* $\wedge$ *i* < *length t*
by (*auto simp*: *writes-to-reg-at-idx-def bind-eq-Some-conv*)

lemma *writes-to-reg-SomeE*[*elim*!]:
  assumes *writes-to-reg e* = *Some r*
  obtains *v* **where** *e* = *E-write-reg r v*
  using *assms*
  by (*cases e*) *auto*

lemma *writes-to-reg-Some-iff*:
  writes-to-reg e = *Some r* $\longleftrightarrow$ ($\exists$ *v*. *e* = *E-write-reg r v*)
  by (*cases e*) *auto*

lemma *member-writes-reg-caps-at-idx-iff* [*simp*]:
  c $\in$ *writes-reg-caps-at-idx CC ISA i t* $\longleftrightarrow$
   c $\in$ *writes-reg-caps CC* (*caps-of-regval ISA*) (*t* ! *i*) $\wedge$ *i* < *length t*
  by (*auto simp*: *writes-reg-caps-at-idx-def split*: *option.splits*)

lemma *member-writes-reg-capsE*[*elim*!]:
  assumes *c* $\in$ *writes-reg-caps CC c-of-r e*
  obtains *r v* **where** *e* = *E-write-reg r v* **and** *c* $\in$ *c-of-r v* **and** *is-tagged-method CC c*
  using *assms*
  by (*cases e*) *auto*

lemma *writes-mem-cap-at-idx-Some-iff* [*simp*]:
  writes-mem-cap-at-idx CC i t = *Some* (*addr*, *sz*, *c*) $\longleftrightarrow$
   writes-mem-cap CC (*t* ! *i*) = *Some* (*addr*, *sz*, *c*) $\wedge$ *i* < *length t*
  by (*auto simp*: *writes-mem-cap-at-idx-def bind-eq-Some-conv*)

lemma *reads-mem-cap-at-idx-Some-iff* [*simp*]:
  reads-mem-cap-at-idx CC i t = *Some* (*addr*, *sz*, *c*) $\longleftrightarrow$
   reads-mem-cap CC (*t* ! *i*) = *Some* (*addr*, *sz*, *c*) $\wedge$ *i* < *length t*
  by (*auto simp*: *reads-mem-cap-at-idx-def bind-eq-Some-conv*)

lemma *nth-append-left*:
  assumes *i* < *length xs*
  shows (*xs* @ *ys*) ! *i* = *xs* ! *i*
  using *assms* **by** (*auto simp*: *nth-append*)

context *Capability-ISA*
begin

lemma *writes-mem-cap-SomeE*[*elim*!]:
  assumes *writes-mem-cap CC e* = *Some* (*addr*, *sz*, *c*)
  obtains *wk bytes r* **where** *e* = *E-write-memt wk addr sz bytes B1 r* **and**
    cap-of-mem-bytes-method CC bytes B1 = *Some c* **and** *is-tagged-method CC c*
  using *assms*
  by (*cases e*) (*auto simp*: *writes-mem-cap-def bind-eq-Some-conv split*: *if-splits*)

lemma *writes-mem-cap-Some-iff*:
  writes-mem-cap CC e = *Some* (*addr*, *sz*, *c*) $\longleftrightarrow$

(∃ *wk bytes r*. *e* = *E-write-memt wk addr sz bytes B1 r* ∧ *cap-of-mem-bytes-method CC bytes B1* = *Some c* ∧ *is-tagged-method CC c*)
by (*cases e*) (*auto simp*: *writes-mem-cap-def bind-eq-Some-conv*)

lemma *reads-mem-cap-SomeE*[*elim*!]:
assumes *reads-mem-cap CC e* = *Some* (*addr*, *sz*, *c*)
obtains *wk bytes r* **where** *e* = *E-read-memt wk addr sz* (*bytes*, *B1*) **and**
cap-of-mem-bytes-method CC bytes B1 = *Some c* **and** *is-tagged-method CC c*
using *assms*
by (*cases e*) (*auto simp*: *reads-mem-cap-def bind-eq-Some-conv split*: *if-splits*)

lemma *reads-mem-cap-Some-iff*:
reads-mem-cap CC e = *Some* (*addr*, *sz*, *c*) ⟷
(∃ *wk bytes*. *e* = *E-read-memt wk addr sz* (*bytes*, *B1*) ∧ *cap-of-mem-bytes-method CC bytes B1* = *Some c* ∧ *is-tagged-method CC c*)
by (*cases e*; *fastforce simp*: *reads-mem-cap-def bind-eq-Some-conv*)

lemma *available-caps-cases*:
assumes *c* ∈ *available-caps CC ISA i t*
obtains (*Reg*) *r v j* **where** *t* ! *j* = *E-read-reg r v*
and *c* ∈ *caps-of-regval ISA v*
and ¬*cap-reg-written-before-idx CC ISA j r t*
and *r* ∈ *privileged-regs ISA* ⟶ *system-access-permitted-before-idx CC ISA j t*
and *j* < *i* **and** *j* < *length t* **and** *is-tagged-method CC c*
| (*Mem*) *wk paddr bytes j sz* **where** *t* ! *j* = *E-read-memt wk paddr sz* (*bytes*, *B1*)
and *cap-of-mem-bytes-method CC bytes B1* = *Some c*
and *j* < *i* **and** *j* < *length t* **and** *is-tagged-method CC c*
using *assms*
by (*induction i*) (*auto split*: *option.splits if-splits*)

lemma *cap-reg-written-before-idx-0-False*[*simp*]:
cap-reg-written-before-idx CC ISA 0 r t ⟷ *False*
by (*auto simp*: *cap-reg-written-before-idx-def*)

lemma *cap-reg-written-before-idx-Suc-iff*[*simp*]:
cap-reg-written-before-idx CC ISA (*Suc i*) *r t* ⟷
(*cap-reg-written-before-idx CC ISA i r t* ∨
(∃ *v c*. *i* < *length t* ∧ *t* ! *i* = *E-write-reg r v* ∧ *c* ∈ *caps-of-regval ISA v* ∧ *is-tagged-method CC c*))
by (*fastforce simp*: *cap-reg-written-before-idx-def less-Suc-eq*)

definition *accessible-regs-at-idx* :: *nat* ⇒ *'regval trace* ⇒ *register-name set* **where**
accessible-regs-at-idx i t =
{*r*. ¬*cap-reg-written-before-idx CC ISA i r t* ∧
(*r* ∈ *privileged-regs ISA* ⟶ *system-access-permitted-before-idx CC ISA i t*)}

fun *accessed-reg-caps* :: *register-name set* ⇒ *'regval event* ⇒ *'cap set* **where**
accessed-reg-caps regs (*E-read-reg r v*) =

$\{c.\ r \in regs \wedge c \in caps\text{-}of\text{-}regval\ ISA\ v \wedge is\text{-}tagged\text{-}method\ CC\ c\}$
| *accessed-reg-caps regs - =* $\{\}$

lemma *member-accessed-reg-capsE*[*elim*!]:
assumes $c \in$ *accessed-reg-caps regs e*
obtains *r v* **where** *e = E-read-reg r v* **and** $r \in regs$
and $c \in$ *caps-of-regval ISA v* **and** *is-tagged-method CC c*
using *assms*
by (*cases e*) *auto*

fun *accessed-mem-caps* :: *'regval event* ⇒ *'cap set* **where**
accessed-mem-caps (*E-read-memt rk a sz val*) =
(*case cap-of-mem-bytes-method CC* (*fst val*) (*snd val*) *of*
Some c ⇒ *if is-tagged-method CC c then* $\{c\}$ *else* $\{\}$
| *None* ⇒ $\{\}$)
| *accessed-mem-caps - =* $\{\}$

lemma *member-accessed-mem-capsE*[*elim*!]:
assumes $c \in$ *accessed-mem-caps e*
obtains *rk a sz bytes tag* **where** *e = E-read-memt rk a sz* (*bytes*, *tag*)
and *cap-of-mem-bytes-method CC bytes tag = Some c* **and** *is-tagged-method CC c*
using *assms*
by (*cases e*) (*auto split*: *option.splits if-splits*)

fun *allows-system-reg-access* :: *register-name set* ⇒ *'regval event* ⇒ *bool* **where**
allows-system-reg-access accessible-regs (*E-read-reg r v*) =
($\exists\, c \in$ *caps-of-regval ISA v*.
is-tagged-method CC c ∧ ¬*is-sealed-method CC c* ∧
permit-system-access (*get-perms-method CC c*) ∧
$r \in$ *PCC ISA* ∩ *accessible-regs*)
| *allows-system-reg-access accessible-regs - = False*

lemma *system-access-permitted-before-idx-0*[*simp*]:
system-access-permitted-before-idx CC ISA 0 t = False
by (*auto simp*: *system-access-permitted-before-idx-def*)

lemma *system-access-permitted-before-idx-Suc*[*simp*]:
system-access-permitted-before-idx CC ISA (*Suc i*) *t* ⟷
(*system-access-permitted-before-idx CC ISA i t* ∨
(*i* < *length t* ∧ *allows-system-reg-access* (*accessible-regs-at-idx i t*) (*t* ! *i*)))
by (*fastforce simp*: *system-access-permitted-before-idx-def accessible-regs-at-idx-def less-Suc-eq*
elim!: *allows-system-reg-access.elims*)

lemma *accessible-regs-at-idx-0*[*simp*]:
accessible-regs-at-idx 0 t = (−*privileged-regs ISA*)
by (*auto simp*: *accessible-regs-at-idx-def*)

lemma *accessible-regs-at-idx-Suc*:
accessible-regs-at-idx (*Suc i*) *t* =
(*accessible-regs-at-idx i t* ∪
(*if i* < *length t* ∧ *allows-system-reg-access* (*accessible-regs-at-idx i t*) (*t* ! *i*)
then {*r* ∈ *privileged-regs ISA*. ¬*cap-reg-written-before-idx CC ISA i r t*} *else* {})) −
{*r*. ∃ *c v*. *i* < *length t* ∧ *t* ! *i* = *E-write-reg r v* ∧ *c* ∈ *caps-of-regval ISA v* ∧ *is-tagged-method CC c*}
by (*auto simp*: *accessible-regs-at-idx-def*)

declare *available-caps.simps*[*simp del*]

lemma *reads-from-reg-None-reads-reg-caps-empty*[*simp*]:
reads-from-reg e = *None* ⟹ *reads-reg-caps CC cor e* = {}
by (*cases e*) *auto*

lemma *available-caps-0*[*simp*]: *available-caps CC ISA 0 t* = {}
by (*auto simp*: *available-caps.simps*)

lemma *available-caps-Suc*:
available-caps CC ISA (*Suc i*) *t* =
available-caps CC ISA i t ∪
(*if i* < *length t*
then accessed-mem-caps (*t* ! *i*) ∪
accessed-reg-caps (*accessible-regs-at-idx i t*) (*t* ! *i*)
else {})
by (*cases t* ! *i*)
(*auto simp*: *available-caps.simps accessible-regs-at-idx-def reads-mem-cap-def bind-eq-Some-conv*
split: *option.splits*)

abbreviation *instr-sem-ISA* (⟦-⟧) **where** ⟦*instr*⟧ ≡ *instr-sem ISA instr*

end

end
theory *Properties*
imports *Cheri-axioms-lemmas Sail.Sail2-state-lemmas*
begin

locale *CHERI-ISA* = *Capability-ISA CC ISA*
for *CC* :: *'cap Capability-class* **and** *ISA* :: (*'cap*, *'regval*, *'instr*, *'e*) *isa* +
fixes *fetch-assms* :: *'regval trace* ⇒ *bool* **and** *instr-assms* :: *'regval trace* ⇒ *bool*
assumes *instr-cheri-axioms*: ⋀*t instr*. *hasTrace t* ⟦*instr*⟧ ⟹ *instr-assms t* ⟹ *cheri-axioms CC ISA False* (*instr-raises-ex ISA instr t*) (*invokes-caps ISA instr t*) *t*
and *fetch-cheri-axioms*: ⋀*t*. *hasTrace t* (*instr-fetch ISA*) ⟹ *fetch-assms t* ⟹

cheri-axioms CC ISA True (fetch-raises-ex ISA t) False t
and *instr-assms-appendE*: $\bigwedge t\ t'\ instr.\ instr\text{-}assms\ (t\ @\ t') \Longrightarrow Run\ [\![instr]\!]\ t\ () \Longrightarrow instr\text{-}assms\ t \wedge fetch\text{-}assms\ t'$
and *fetch-assms-appendE*: $\bigwedge t\ t'\ instr.\ fetch\text{-}assms\ (t\ @\ t') \Longrightarrow Run\ (instr\text{-}fetch\ ISA)\ t\ instr \Longrightarrow fetch\text{-}assms\ t \wedge instr\text{-}assms\ t'$

locale *Register-Accessors* =
fixes *read-regval* :: *register-name* $\Rightarrow$ *'regs* $\Rightarrow$ *'regval option*
and *write-regval* :: *register-name* $\Rightarrow$ *'regval* $\Rightarrow$ *'regs* $\Rightarrow$ *'regs option*
begin

abbreviation *s-emit-event e s* $\equiv$ *emitEventS* (*read-regval*, *write-regval*) *e s*
abbreviation *s-run-trace t s* $\equiv$ *runTraceS* (*read-regval*, *write-regval*) *t s*
abbreviation *s-allows-trace t s* $\equiv \exists s'.$ *s-run-trace t s* = *Some s'*

end

locale *CHERI-ISA-State* = *CHERI-ISA CC ISA* + *Register-Accessors read-regval write-regval*
for *ISA* :: (*'cap*, *'regval*, *'instr*, *'e*) *isa*
and *CC* :: *'cap Capability-class*
and *read-regval* :: *register-name* $\Rightarrow$ *'regs* $\Rightarrow$ *'regval option*
and *write-regval* :: *register-name* $\Rightarrow$ *'regval* $\Rightarrow$ *'regs* $\Rightarrow$ *'regs option* +

fixes *s-translation-tables* :: *'regs sequential-state* $\Rightarrow$ *nat set*
and *s-translate-address* :: *nat* $\Rightarrow$ *acctype* $\Rightarrow$ *'regs sequential-state* $\Rightarrow$ *nat option*
assumes *read-absorb-write*: $\bigwedge r\ v\ s\ s'.$ *write-regval r v s* = *Some s'* $\Longrightarrow$ *read-regval r s'* = *Some v*
and *read-ignore-write*: $\bigwedge r\ r'\ v\ s\ s'.$ *write-regval r v s* = *Some s'* $\Longrightarrow r' \neq r \Longrightarrow$ *read-regval r' s'* = *read-regval r' s*
and *translation-tables-sound*: $\bigwedge t\ s.$ *s-allows-trace t s* $\Longrightarrow$ *translation-tables ISA t* $\subseteq$ *s-translation-tables s*
and *translate-address-sound*: $\bigwedge t\ s\ vaddr\ paddr\ load.$
s-allows-trace t s $\Longrightarrow$
translate-address ISA vaddr load t = *Some paddr* $\Longrightarrow$
s-translate-address vaddr load s = *Some paddr*
and *translate-address-tag-aligned-iff*: $\bigwedge s\ vaddr\ paddr\ load.$
s-translate-address vaddr load s = *Some paddr* $\Longrightarrow$
address-tag-aligned ISA paddr $\longleftrightarrow$ *address-tag-aligned ISA vaddr*
begin

1.5 Reachable capabilities

fun *get-reg-val* :: *register-name* $\Rightarrow$ *'regs sequential-state* $\Rightarrow$ *'regval option* **where**
get-reg-val r s = *read-regval r* (*regstate s*)

fun *put-reg-val* :: *register-name* $\Rightarrow$ *'regval* $\Rightarrow$ *'regs sequential-state* $\Rightarrow$ *'regs sequential-state option* **where**
put-reg-val r v s = *map-option* ($\lambda rs'.\ s(\!|regstate := rs'|\!)$) (*write-regval r v* (*regstate*

s))

fun *get-reg-caps* :: *register-name* ⇒ *'regs sequential-state* ⇒ *'cap set* **where**
get-reg-caps r s = (*case read-regval r* (*regstate s*) *of Some v* ⇒ {*c* ∈ *caps-of-regval ISA v*. *is-tagged-method CC c*} | *None* ⇒ {})

fun *get-mem-cap* :: *nat* ⇒ *nat* ⇒ *'regs sequential-state* ⇒ *'cap option* **where**
get-mem-cap addr sz s =
Option.bind (*get-mem-bytes addr sz s*) (λ(*bytes*, *tag*).
Option.bind (*cap-of-mem-bytes-method CC bytes tag*) (λ*c*.
if is-tagged-method CC c then Some c else None))

fun *get-aligned-mem-cap* :: *nat* ⇒ *nat* ⇒ *'regs sequential-state* ⇒ *'cap option* **where**
get-aligned-mem-cap vaddr sz s =
(*if address-tag-aligned ISA vaddr* ∧ *sz* = *tag-granule ISA then get-mem-cap vaddr sz s else None*)

inductive-set *reachable-caps* :: *'regs sequential-state* ⇒ *'cap set* **for** *s* :: *'regs sequential-state* **where**
Reg: ⟦*c* ∈ *get-reg-caps r s*; *r* ∉ *privileged-regs ISA*; *is-tagged-method CC c*⟧ ⟹ *c* ∈ *reachable-caps s*
| *SysReg*:
⟦*c* ∈ *get-reg-caps r s*; *r* ∈ *privileged-regs ISA*; *c'* ∈ *reachable-caps s*;
permit-system-access (*get-perms-method CC c'*); ¬*is-sealed-method CC c'*;
is-tagged-method CC c⟧
⟹ *c* ∈ *reachable-caps s*
| *Mem*:
⟦*get-aligned-mem-cap addr* (*tag-granule ISA*) *s* = *Some c*;
s-translate-address vaddr Load s = *Some addr*;
c' ∈ *reachable-caps s*; *is-tagged-method CC c'*; ¬*is-sealed-method CC c'*;
set (*address-range vaddr* (*tag-granule ISA*)) ⊆ *get-mem-region-method CC c'*;
permit-load-capability (*get-perms-method CC c'*);
is-tagged-method CC c⟧
⟹ *c* ∈ *reachable-caps s*
| *Restrict*: ⟦*c* ∈ *reachable-caps s*; *leq-cap CC c' c*⟧ ⟹ *c'* ∈ *reachable-caps s*
| *Seal*:
⟦*c'* ∈ *reachable-caps s*; *c''* ∈ *reachable-caps s*; *is-tagged-method CC c'*; *is-tagged-method CC c''*;
¬*is-sealed-method CC c''*; ¬*is-sealed-method CC c'*; *permit-seal* (*get-perms-method CC c''*)⟧ ⟹
seal CC c' (*get-cursor-method CC c''*) ∈ *reachable-caps s*
| *Unseal*:
⟦*c'* ∈ *reachable-caps s*; *c''* ∈ *reachable-caps s*; *is-tagged-method CC c'*; *is-tagged-method CC c''*;
¬*is-sealed-method CC c''*; *is-sealed-method CC c'*; *permit-unseal* (*get-perms-method CC c''*);
get-obj-type-method CC c' = *get-cursor-method CC c''*⟧ ⟹
unseal CC c' (*get-global CC c''*) ∈ *reachable-caps s*

lemma *derivable-subseteq-reachableI*:
  assumes $C \subseteq$ *reachable-caps s*
  shows *derivable* $C \subseteq$ *reachable-caps s*
proof
  fix *c*
  assume $c \in$ *derivable C*
  then show $c \in$ *reachable-caps s* **using** *assms*
    by *induction* (*auto intro*: *reachable-caps.intros*)
qed

lemma *derivable-subseteq-reachableE*:
  assumes *derivable* $C \subseteq$ *reachable-caps s*
  shows $C \subseteq$ *reachable-caps s*
  using *assms* **by** (*auto intro*: *derivable.intros*)

lemma *derivable-reachable-caps-idem*[*simp*]: *derivable* (*reachable-caps s*) = *reachable-caps s*
  using *derivable-subseteq-reachableI*[*of reachable-caps s s*] *derivable-refl*
  by *auto*

lemma *runTraceS-rev-induct*[*consumes 1*, *case-names Init Step*]:
  assumes *s-run-trace t s* = *Some* s'
    and *Init*: *P* [] *s*
    and *Step*: $\bigwedge t\ e\ s''\ s'$. *s-run-trace t s* = *Some* $s'' \Longrightarrow$ *s-emit-event e* s'' = *Some* $s' \Longrightarrow P\ t\ s'' \Longrightarrow P\ (t\ @\ [e])\ s'$
  shows $P\ t\ s'$
  using *assms*
  by (*induction t arbitrary*: s' *rule*: *rev-induct*)
    (*auto elim*: *runTraceS-appendE runTraceS-ConsE simp*: *bind-eq-Some-conv*)

lemma *get-reg-val-s-run-trace-cases*:
  assumes *v*: *get-reg-val r* s' = *Some v* **and** *c*: $c \in$ *caps-of-regval ISA v*
    and s': *s-run-trace t s* = *Some* s'
  obtains (*Init*) *get-reg-val r s* = *Some v*
  | (*Update*) *j* v' **where** *t* ! *j* = *E-write-reg r* v' **and** $c \in$ *caps-of-regval ISA* v' **and** $j <$ *length t*
proof (*use* s' *v c* **in** ‹*induction rule*: *runTraceS-rev-induct*›)
  case (*Step t e* s'' s')
  note *Init* = *Step*(*4*)
  note *Update* = *Step*(*5*)
  note *c* = ‹$c \in$ *caps-of-regval ISA v*›
  show *?case*
  proof *cases*
    assume *v*-s'': *get-reg-val r* s'' = *Some v*
    show *?thesis*
    proof (*rule Step.IH*[*OF - - v*-s'' *c*])
      assume *get-reg-val r s* = *Some v*
      then show *thesis* **by** (*intro Init*)

next
fix *j v'*
assume $t ! j = E\text{-}write\text{-}reg\ r\ v'$ **and** $c \in caps\text{-}of\text{-}regval\ ISA\ v'$ **and** $j < length$
t
then show *thesis* **by** (*intro Update*[*of j v'*]) (*auto simp*: *nth-append-left*)
qed
next
assume *v-s''*: $get\text{-}reg\text{-}val\ r\ s'' \neq Some\ v$
note $e = \langle s\text{-}emit\text{-}event\ e\ s'' = Some\ s'\rangle$
note $v\text{-}s' = \langle get\text{-}reg\text{-}val\ r\ s' = Some\ v\rangle$
from *e v-s' v-s''* **have** $e = E\text{-}write\text{-}reg\ r\ v$
proof (*cases rule*: *emitEventS-update-cases*)
case (*Write-reg r' v' rs'*)
then show *?thesis*
using *v-s' v-s''*
by (*cases* $r' = r$) (*auto simp*: *read-ignore-write read-absorb-write*)
qed (*auto simp*: *put-mem-bytes-def Let-def*)
then show *thesis* **using** *c* **by** (*auto intro*: *Update*[*of length t v*])
qed
qed *auto*

lemma *reads-reg-cap-at-idx-provenance*[*consumes 5*]:
assumes *r*: $t ! i = E\text{-}read\text{-}reg\ r\ v$ **and** *c*: $c \in caps\text{-}of\text{-}regval\ ISA\ v$ **and** *tag*:
is-tagged-method CC c
and *s'*: $s\text{-}run\text{-}trace\ t\ s = Some\ s'$ **and** *i*: $i < length\ t$
obtains (*Initial*) $c \in get\text{-}reg\text{-}caps\ r\ s$
| (*Update*) *j* **where** $c \in writes\text{-}reg\text{-}caps\ CC\ (caps\text{-}of\text{-}regval\ ISA)\ (t ! j)$
and $writes\text{-}to\text{-}reg\ (t ! j) = Some\ r$ **and** $j < i$
proof −
from *s' i* **obtain** *s1 s2*
where *s1*: $s\text{-}run\text{-}trace\ (take\ i\ t)\ s = Some\ s1$
and *s2*: $s\text{-}emit\text{-}event\ (t ! i)\ s1 = Some\ s2$
by (*blast elim*: *runTraceS-nth-split*)
from *s2 c r tag* **have** $c \in get\text{-}reg\text{-}caps\ r\ s1$
by (*auto simp*: *bind-eq-Some-conv split*: *option.splits if-splits*)
with *s1 Update* **show** *thesis* **using** *i*
proof (*induction take i t s1 arbitrary*: *i t rule*: *runTraceS-rev-induct*)
case *Init*
then show *?case* **by** (*intro Initial*)
next
case (*Step t' e s'' s' i t*)
then obtain *j* **where** *j*: $i = Suc\ j$ **by** (*cases i*) *auto*
then have *t'*: $t' = take\ j\ t$ **and** *e*: $e = t ! j$
using *Step* **by** (*auto simp*: *take-hd-drop*[*symmetric*] *hd-drop-conv-nth*)
note $IH = Step(3)[of\ j\ t]$
note $Update = Step(5)$
note $i = \langle i < length\ t\rangle$
show *?case*
proof (*use* $\langle s\text{-}emit\text{-}event\ e\ s'' = Some\ s'\rangle$ **in** $\langle$*cases rule*: *emitEventS-update-cases*$\rangle$)

case (*Write-mem wk addr sz v tag res*)
then have *c*: $c \in$ *get-reg-caps r s''*
using *Step*
by (*auto simp*: *put-mem-bytes-def bind-eq-Some-conv Let-def*)
show *?thesis*
by (*rule IH*) (*use c t' i j Update* **in** ‹*auto*›)
next
case (*Write-reg r' v rs'*)
show *?thesis*
proof *cases*
assume $r' = r$
then show *?thesis*
using *Write-reg e j* ‹$c \in$ *get-reg-caps r s'*›
by (*intro Update*[*of j*]) (*auto simp*: *read-absorb-write*)
next
assume $r' \neq r$
show *?thesis*
by (*rule IH*)
(*use* ‹$r' \neq r$› *Write-reg e t' i j* ‹$c \in$ *get-reg-caps r s'*› *Update* **in**
‹*auto simp*: *read-ignore-write*›)
qed
next
case *Read*
show *?thesis*
by (*rule IH*) (*use Read Step.prems t' i j Update* **in** ‹*auto*›)
qed
qed
qed

lemma *reads-reg-cap-at-idx-from-initial*:
assumes *r*: *t ! i = E-read-reg r v* **and** *c*: $c \in$ *caps-of-regval ISA v* **and** *tag*: *is-tagged-method CC c*
and *s'*: *s-run-trace t s = Some s'* **and** *i*: $i <$ *length t*
and ¬ *cap-reg-written-before-idx CC ISA i r t*
shows $c \in$ *get-reg-caps r s*
using *assms*
by (*elim reads-reg-cap-at-idx-provenance*)
(*auto simp*: *cap-reg-written-before-idx-def writes-reg-caps-at-idx-def*)

1.6 Capability monotonicity

lemma *available-caps-mono*:
assumes *j*: $j < i$
shows *available-caps CC ISA j t* $\subseteq$ *available-caps CC ISA i t*
proof −
have *available-caps CC ISA j t* $\subseteq$ *available-caps CC ISA* (*Suc* (*j* + *k*)) *t* **for** *k*
by (*induction k*) (*auto simp*: *available-caps-Suc image-iff subset-iff*)
then show *?thesis* **using** *assms less-iff-Suc-add*[*of j i*] **by** *blast*
qed

lemma *reads-reg-cap-non-privileged-accessible*[*intro*]:
  assumes $c \in$ *caps-of-regval ISA v* **and** $t\ !\ j =$ *E-read-reg r v*
    and $\neg$*cap-reg-written-before-idx CC ISA j r t*
    and $r \notin$ *privileged-regs ISA*
    and *is-tagged-method CC c*
    and $j < i$
    and $j <$ *length t*
  shows $c \in$ *available-caps CC ISA i t*
proof −
  from *assms* **have** *c*: $c \in$ *available-caps CC ISA* (*Suc j*) *t*
    by (*auto simp*: *bind-eq-Some-conv image-iff available-caps.simps*)
  consider $i =$ *Suc j* | *Suc j* $< i$ **using** ‹$j < i$›
    by (*cases i* = *Suc j*) *auto*
  then show $c \in$ *available-caps CC ISA i t*
    using *c available-caps-mono*[*of Suc j i t*]
    by *cases auto*
qed

lemma *system-access-permitted-at-idx-available-caps*:
  assumes *system-access-permitted-before-idx CC ISA i t*
  obtains *c* **where** $c \in$ *available-caps CC ISA i t* **and** *is-tagged-method CC c*
    and $\neg$*is-sealed-method CC c* **and** *permit-system-access* (*get-perms-method CC c*)
  using *assms*
  by (*auto simp*: *system-access-permitted-before-idx-def*; *blast*)

lemma *writes-reg-cap-nth-provenance*[*consumes 4*]:
  assumes $t\ !\ i =$ *E-write-reg r v* **and** $c \in$ *caps-of-regval ISA v*
    and *cheri-axioms CC ISA is-fetch has-ex inv-caps t*
    and $i <$ *length t*
    and *tagged*: *is-tagged-method CC c*
  obtains (*Accessible*) $c \in$ *derivable* (*available-caps CC ISA i t*)
  | (*Exception*) $v'\ r'\ j$ **where** $c \in$ *exception-targets ISA* $\{v.\ \exists r\ j.\ j < i \wedge j <$ *length t* $\wedge\ t\ !\ j =$ *E-read-reg r v* $\wedge\ r \in$ *KCC ISA*$\}$

    and $r \in$ *PCC ISA* **and** *has-ex*
  | (*CCall*) *cc cd c'* **where** *inv-caps* **and** *invokable CC cc cd*
    and $cc \in$ *derivable* (*available-caps CC ISA i t*)
    and $cd \in$ *derivable* (*available-caps CC ISA i t*)
    and ($r \in$ *PCC ISA* $\wedge$ *leq-cap CC c* (*unseal CC cc True*)) $\vee$
        ($r \in$ *IDC ISA* $\wedge$ *leq-cap CC c* (*unseal CC cd True*))
  using *assms*
  unfolding *cheri-axioms-def store-cap-reg-axiom-def writes-reg-caps-at-idx-def cap-derivable-iff-derivable*
  by (*elim impE conjE allE*[**where** $x = i$] *allE*[**where** $x = c$])
    (*auto simp*: *eq-commute*[**where** $b = t\ !\ j$ **for** *t j*])

lemma *get-mem-cap-run-trace-cases*:
  assumes *c*: *get-mem-cap addr* (*tag-granule ISA*) $s' =$ *Some c*

and s': *s-run-trace t s* = *Some* s'
and *tagged*: *is-tagged-method CC c*
and *aligned*: *address-tag-aligned ISA addr*
and *axiom*: *store-tag-axiom CC ISA t*
obtains (*Initial*) *get-mem-cap addr* (*tag-granule ISA*) *s* = *Some c*
| (*Update*) *k wk bytes r* **where** $k <$ *length t*
and *t* ! *k* = *E-write-memt wk addr* (*tag-granule ISA*) *bytes B1 r*
and *cap-of-mem-bytes-method CC bytes B1* = *Some c*
proof (*use* s' *c axiom* **in** ‹*induction rule*: *runTraceS-rev-induct*›)
case (*Step t e* s'' s')
note *Update* = *Step.prems*(*2*)
have *axiom*: *store-tag-axiom CC ISA t*
using ‹*store-tag-axiom CC ISA* (*t* @ [*e*])›
by (*auto simp*: *store-tag-axiom-def writes-mem-val-at-idx-def nth-append bind-eq-Some-conv split*: *if-splits*; *metis less-SucI*)
have *IH*: *thesis* **if** *get-mem-cap addr* (*tag-granule ISA*) s'' = *Some c*
proof (*rule Step.IH*[*OF - - that axiom*])
assume *get-mem-cap addr* (*tag-granule ISA*) *s* = *Some c*
then show *thesis* **by** (*rule Initial*)
next
fix *k wk bytes r*
assume $k <$ *length t* **and** *t* ! *k* = *E-write-memt wk addr* (*tag-granule ISA*) *bytes B1 r*
and *cap-of-mem-bytes-method CC bytes B1* = *Some c*
then show *thesis*
by (*intro Update*[*of k wk bytes r*]) (*auto simp*: *nth-append*)
qed
obtain *v tag*
where *v*: *get-mem-bytes addr* (*tag-granule ISA*) s' = *Some* (*v*, *tag*)
and *cv*: *cap-of-mem-bytes-method CC v tag* = *Some c*
using ‹*get-mem-cap addr* (*tag-granule ISA*) s' = *Some c*›
by (*auto simp*: *bind-eq-Some-conv bool-of-bitU-def split*: *if-splits*)
then have *tag*: *tag* = *B1* **using** *tagged* **by** *auto*
from ‹*s-emit-event e* s'' = *Some* s'› **show** *thesis*
proof (*cases rule*: *emitEventS-update-cases*)
case (*Write-mem wk* $addr'$ sz' v' tag' *r*)
have sz': tag' = *B1* ⟶ (*address-tag-aligned ISA* $addr'$ ∧ sz' = *tag-granule ISA*)
and *len-*v': *length* v' = sz'
using ‹*store-tag-axiom CC ISA* (*t* @ [*e*])› *Write-mem*
by (*auto simp*: *store-tag-axiom-def writes-mem-val-at-idx-def bind-eq-Some-conv nth-append split*: *if-splits*)
show *?thesis*
proof *cases*
assume *addr-disj*: {*addr*..<*tag-granule ISA* + *addr*} ∩ {$addr'$..<sz' + $addr'$} = {}
then have *get-mem-bytes addr* (*tag-granule ISA*) s'' = *get-mem-bytes addr* (*tag-granule ISA*) s'
using *Write-mem len-*v'

```
      by (intro get-mem-bytes-cong) (auto simp: memstate-put-mem-bytes tagstate-put-mem-bytes)
      then show thesis
        using v cv tag
        by (intro IH) auto
    next
      assume addr-overlap: {addr..<tag-granule ISA + addr} ∩ {addr'..<sz' +
addr'} ≠ {}
      then have tag': tag' = B1
      proof –
        obtain addr''
          where addr-orig: addr'' ∈ {addr..<tag-granule ISA + addr}
            and addr-prime: addr'' ∈ {addr'..<sz' + addr'}
          using addr-overlap
          by blast
        have tagstate s' addr'' = Some B1
          using addr-orig get-mem-bytes-tagged-tagstate[OF v[unfolded tag]]
          by auto
        then show tag' = B1
          using addr-prime Write-mem len-v'
          by (auto simp: tagstate-put-mem-bytes)
      qed
      with addr-overlap aligned sz' have addr': addr' = addr
        by (auto simp: address-tag-aligned-def dvd-def mult-Suc-right[symmetric]
                 simp del: mult-Suc-right)
      then have v': v' = v
        using v tag tag' sz' len-v' Write-mem
        by (auto simp: get-mem-bytes-put-mem-bytes-same-addr)
      then show thesis
        using Write-mem cv tag' addr' sz' tag
        by (intro Step.prems(2)[of length t]) (auto simp: writes-mem-cap-def)
    qed
  next
    case (Write-reg r v rs')
    with ‹get-mem-cap addr (tag-granule ISA) s' = Some c› show thesis
      by (auto intro: IH simp: get-mem-bytes-def)
  next
    case Read
    with ‹get-mem-cap addr (tag-granule ISA) s' = Some c› show thesis
      by (auto intro: IH)
  qed
qed auto

lemma reads-mem-cap-at-idx-provenance:
  assumes read: t ! i = E-read-memt rk addr (tag-granule ISA) (bytes, B1)
    and c: cap-of-mem-bytes-method CC bytes B1 = Some c
    and s': s-run-trace t s = Some s'
    and axioms: cheri-axioms CC ISA is-fetch has-ex inv-caps t
    and i: i < length t
    and tagged: is-tagged-method CC c
```

and *aligned*: *address-tag-aligned ISA addr*
obtains (*Initial*) *get-mem-cap addr* (*tag-granule ISA*) *s* = *Some c*
| (*Update*) *k wk bytes′ r* **where** *k* < *i*
and *t* ! *k* = *E-write-memt wk addr* (*tag-granule ISA*) *bytes′ B1 r*
and *cap-of-mem-bytes-method CC bytes′ B1* = *Some c*
proof −
obtain *s′′*
where *s′′*: *s-run-trace* (*take i t*) *s* = *Some s′′*
and *c′*: *get-mem-cap addr* (*tag-granule ISA*) *s′′* = *Some c*
using *s′ i read c tagged*
by (*cases rule*: *runTraceS-nth-split*; *cases t* ! *i*)
(*auto simp*: *bind-eq-Some-conv reads-mem-cap-def split*: *if-splits*)
have *store-tag-axiom CC ISA* (*take i t*)
using *axioms*
by (*fastforce simp*: *cheri-axioms-def store-tag-axiom-def writes-mem-val-at-idx-def bind-eq-Some-conv*)
with *c′ s′′ tagged aligned* **show** *thesis*
by (*cases rule*: *get-mem-cap-run-trace-cases*) (*auto intro*: *that*)
qed

fun *s-invariant* :: (*′regs sequential-state* ⇒ *′a*) ⇒ *′regval trace* ⇒ *′regs sequential-state* ⇒ *bool* **where**
s-invariant f [] *s* = *True*
| *s-invariant f* (*e* # *t*) *s* = (*case s-emit-event e s of Some s′* ⇒ *f s′* = *f s* ∧ *s-invariant f t s′* | *None* ⇒ *False*)

abbreviation *s-invariant-holds* :: (*′regs sequential-state* ⇒ *bool*) ⇒ *′regval trace* ⇒ *′regs sequential-state* ⇒ *bool* **where**
s-invariant-holds P t s ≡ *P s* ∧ *s-invariant P t s*

lemma *s-invariant-append*:
s-invariant f (*β* @ *α*) *s* ⟷
(∃ *s′*. *s-invariant f β s* ∧ *s-run-trace β s* = *Some s′* ∧ *s-invariant f α s′*)
by (*induction β arbitrary*: *s*) (*auto split*: *option.splits simp*: *runTraceS-Cons-tl*)

lemma *s-invariant-takeI*:
assumes *s-invariant f t s*
shows *s-invariant f* (*take n t*) *s*
proof −
from *assms* **have** *s-invariant f* (*take n t* @ *drop n t*) *s* **by** *auto*
then show *?thesis* **unfolding** *s-invariant-append* **by** *auto*
qed

lemma *s-invariant-run-trace-eq*:
assumes *s-invariant f t s* **and** *s-run-trace t s* = *Some s′*
shows *f s′* = *f s*
using *assms*
by (*induction f t s rule*: *s-invariant.induct*)
(*auto split*: *option.splits elim*: *runTraceS-ConsE*)

definition *no-caps-in-translation-tables* :: *'regs sequential-state* ⇒ *bool* **where**
no-caps-in-translation-tables s ≡
∀ *addr sz c*. *get-mem-cap addr sz s* = *Some c* ∧ *is-tagged-method CC c* ⟶
addr ∉ *s-translation-tables s*

lemma *derivable-available-caps-subseteq-reachable-caps*:
assumes *axioms*: *cheri-axioms CC ISA is-fetch has-ex inv-caps t*
and *t*: *s-run-trace t s* = *Some s'*
and *translation-table-addrs-invariant*: *s-invariant s-translation-tables t s*
and *no-caps-in-translation-tables*: *s-invariant-holds no-caps-in-translation-tables t s*
shows *derivable* (*available-caps CC ISA i t*) ⊆ *reachable-caps s*
proof (*induction i rule*: *less-induct*)
case (*less i*)
show *?case* **proof**
fix *c*
assume *c* ∈ *derivable* (*available-caps CC ISA i t*)
then show *c* ∈ *reachable-caps s*
proof *induction*
fix *c*
assume *c* ∈ *available-caps CC ISA i t*
then show *c* ∈ *reachable-caps s*
proof (*cases rule*: *available-caps-cases*)
case (*Reg r v j*)
with *t* **have** *initial*: *c* ∈ *get-reg-caps r s*
by (*blast intro*: *reads-reg-cap-at-idx-from-initial*)
show *?thesis*
proof *cases*
assume *r*: *r* ∈ *privileged-regs ISA*
then obtain *c'* **where** *c'*: *c'* ∈ *reachable-caps s* **and** *is-tagged-method CC c'*
and ¬*is-sealed-method CC c'* **and** *p*: *permit-system-access* (*get-perms-method CC c'*)
using *Reg less.IH*[*OF* ‹*j* < *i*›] *derivable-refl*[*of available-caps CC ISA j t*]
by (*auto elim*!: *system-access-permitted-at-idx-available-caps*)
then show *?thesis*
using *Reg*
by (*auto intro*: *reachable-caps.SysReg*[*OF initial r c'*])
next
assume *r* ∉ *privileged-regs ISA*
then show *?thesis* **using** *initial Reg* **by** (*auto intro*: *reachable-caps.Reg*)
qed
next
case (*Mem wk paddr bytes j sz*)
note *read* = ‹*t* ! *j* = *E-read-memt wk paddr sz* (*bytes*, *B1*)›
note *bytes* = ‹*cap-of-mem-bytes-method CC bytes B1* = *Some c*›
have *addr*: *paddr* ∉ *translation-tables ISA* (*take j t*)
proof

assume *paddr-j*: *paddr* ∈ *translation-tables ISA* (*take j t*)
then have *paddr* ∈ *s-translation-tables s*
using *translation-tables-sound*[*of take j t s*] *t* ‹*j* < *length t*›
by (*auto elim*: *runTraceS-nth-split*)
moreover have *paddr* ∉ *s-translation-tables s*
proof −
obtain *s''*
where *s''*: *s-run-trace* (*take j t*) *s* = *Some s''*
and *c-s''*: *get-mem-cap paddr sz s''* = *Some c*
using *t* ‹*j* < *length t*› *read bytes* ‹*is-tagged-method CC c*›
by (*cases rule*: *runTraceS-nth-split*; *cases t* ! *j*)
(*auto simp*: *bind-eq-Some-conv reads-mem-cap-def split*: *if-splits*)
moreover have *no-caps-in-translation-tables s''*
using *no-caps-in-translation-tables s''*
using *s-invariant-takeI*[*of no-caps-in-translation-tables t s j*]
using *s-invariant-run-trace-eq*[*of no-caps-in-translation-tables take j t*
s s'']
by *auto*
moreover have *s-translation-tables s''* = *s-translation-tables s*
using *translation-table-addrs-invariant s''*
by (*intro s-invariant-run-trace-eq*) (*auto intro*: *s-invariant-takeI*)
ultimately show *?thesis*
using ‹*is-tagged-method CC c*›
by (*fastforce simp*: *no-caps-in-translation-tables-def bind-eq-Some-conv*)
qed
ultimately show *False* **by** *blast*
qed
then obtain *vaddr c'*
where *vaddr*: *translate-address ISA vaddr Load* (*take j t*) = *Some paddr*
and *c'*: *c'* ∈ *derivable* (*available-caps CC ISA j t*)
is-tagged-method CC c' ¬*is-sealed-method CC c'*
set (*address-range vaddr sz*) ⊆ *get-mem-region-method CC c'*
permit-load (*get-perms-method CC c'*)
permit-load-capability (*get-perms-method CC c'*)
and *sz*: *sz* = *tag-granule ISA*
and *aligned*: *address-tag-aligned ISA paddr*
using *read t axioms* ‹*j* < *length t*› ‹*is-tagged-method CC c*›
unfolding *cheri-axioms-def load-mem-axiom-def reads-mem-cap-def*
by (*fastforce simp*: *reads-mem-val-at-idx-def bind-eq-Some-conv cap-derivable-iff-derivable*
split: *if-splits*)
have *s-vaddr*: *s-translate-address vaddr Load s* = *Some paddr*
using *vaddr t* ‹*j* < *length t*›
by (*blast intro*: *translate-address-sound*[*of take j t*] *elim*: *runTraceS-nth-split*)
from *read*[*unfolded sz*] *bytes t axioms* ‹*j* < *length t*› ‹*is-tagged-method CC*
c› *aligned*
show *?thesis*
proof (*cases rule*: *reads-mem-cap-at-idx-provenance*)
case *Initial*
then show *?thesis*

using *Mem s-vaddr less.IH[of j] c′ aligned sz*
by (*intro reachable-caps.Mem[of paddr s c vaddr c′]*)
(*auto simp*: *bind-eq-Some-conv translate-address-tag-aligned-iff permits-cap-load-def*)
next
case (*Update k wk bytes′ r*)
then show *?thesis*
using *axioms* ‹*is-tagged-method CC c*› ‹*j* < *length t*› ‹*j* < *i*› *less.IH[of k]*
unfolding *cheri-axioms-def store-cap-mem-axiom-def*
by (*auto simp*: *writes-mem-cap-at-idx-def writes-mem-cap-Some-iff bind-eq-Some-conv cap-derivable-iff-derivable*)
qed
qed
qed (*auto intro*: *reachable-caps.intros*)
qed
qed

lemma *put-regval-get-mem-cap*:
assumes *s′*: *put-reg-val r v s* = *Some s′*
and *s-translate-address addr acctype s′* = *s-translate-address addr acctype s*
shows *get-mem-cap addr sz s′* = *get-mem-cap addr sz s*
using *assms* **by** (*auto cong*: *bind-option-cong simp*: *get-mem-bytes-def*)

definition *system-access-reachable* :: *′regs sequential-state* ⇒ *bool* **where**
system-access-reachable s ≡ ∃ *c* ∈ *reachable-caps s*.
permit-system-access (*get-perms-method CC c*) ∧ ¬*is-sealed-method CC c*

lemma *get-reg-cap-intra-domain-trace-reachable*:
assumes *r*: *c* ∈ *get-reg-caps r s′*
and *s′*: *s-run-trace t s* = *Some s′*
and *axioms*: *cheri-axioms CC ISA is-fetch False False t*

and *translation-table-addrs-invariant*: *s-invariant s-translation-tables t s*
and *no-caps-in-translation-tables*: *s-invariant-holds no-caps-in-translation-tables t s*
and *tag*: *is-tagged-method CC c*
and *priv*: *r* ∈ *privileged-regs ISA* ⟶ *system-access-reachable s*
shows *c* ∈ *reachable-caps s*
proof −
from *r* **obtain** *v* **where** *v*: *get-reg-val r s′* = *Some v* **and** *c*: *c* ∈ *caps-of-regval ISA v*
by (*auto simp*: *bind-eq-Some-conv split*: *option.splits*)
from *v c s′* **show** *c* ∈ *reachable-caps s*
proof (*cases rule*: *get-reg-val-s-run-trace-cases*)
case *Init*
show *?thesis*
proof *cases*
assume *r*: *r* ∈ *privileged-regs ISA*
with *priv* **obtain** *c′* **where** *c′*: *c′* ∈ *reachable-caps s*

and *permit-system-access* (*get-perms-method CC c′*) **and** ¬*is-sealed-method CC c′*
by (*auto simp*: *system-access-reachable-def*)
then show *?thesis* **using** *Init c tag* **by** (*intro reachable-caps.SysReg*[*OF - r c′*]) *auto*
next
assume $r \notin$ *privileged-regs ISA*
then show *?thesis* **using** *Init c tag* **by** (*intro reachable-caps.Reg*) *auto*
qed
next
case (*Update j v′*)
then have ∗: $c \in$ *writes-reg-caps CC* (*caps-of-regval ISA*) (*t ! j*)
and *writes-to-reg* (*t ! j*) = *Some r*
using *c tag* **by** *auto*
then have $c \in$ *derivable* (*available-caps CC ISA j t*)
using *axioms tag* ‹*j* < *length t*›
unfolding *cheri-axioms-def store-cap-reg-axiom-def*
by (*fastforce simp*: *cap-derivable-iff-derivable*)
moreover have *derivable* (*available-caps CC ISA j t*) ⊆ *reachable-caps s*
using *axioms s′ translation-table-addrs-invariant no-caps-in-translation-tables*
by (*intro derivable-available-caps-subseteq-reachable-caps*)
ultimately show *?thesis* **by** *auto*
qed
qed

lemma *reachable-caps-trace-intradomain-monotonicity*:
assumes *axioms*: *cheri-axioms CC ISA is-fetch False False t*
and *s′*: *s-run-trace t s* = *Some s′*
and *addr-trans-inv*: *s-invariant* (λ*s′ addr load*. *s-translate-address addr load s′*) *t s*
and *translation-table-addrs-invariant*: *s-invariant s-translation-tables t s*
and *no-caps-in-translation-tables*: *s-invariant-holds no-caps-in-translation-tables t s*
shows *reachable-caps s′* ⊆ *reachable-caps s*
proof
fix *c*
assume $c \in$ *reachable-caps s′*
then show $c \in$ *reachable-caps s*
proof *induction*
case (*Reg r c*)
then show *?case*
using *axioms s′ translation-table-addrs-invariant no-caps-in-translation-tables*
by (*intro get-reg-cap-intra-domain-trace-reachable*) *auto*
next
case (*SysReg r c c′*)
then show *?case*
using *axioms s′ translation-table-addrs-invariant no-caps-in-translation-tables*
by (*intro get-reg-cap-intra-domain-trace-reachable*) (*auto simp*: *system-access-reachable-def*)
next

case (*Mem addr c vaddr c'*)
then have *c*: *get-mem-cap addr* (*tag-granule ISA*) *s'* = *Some c*
and *aligned*: *address-tag-aligned ISA addr*
by (*auto split*: *if-splits*)
have *axiom*: *store-tag-axiom CC ISA t*
using *axioms*
by (*auto simp*: *cheri-axioms-def*)
from *c s'* ‹*is-tagged-method CC c*› *aligned axiom* **show** *?case*
proof (*cases rule*: *get-mem-cap-run-trace-cases*)
case *Initial*
have *s-translate-address vaddr Load s'* = *s-translate-address vaddr Load s*
using *s-invariant-run-trace-eq*[*OF addr-trans-inv s'*]
by *meson*
then show *?thesis*
using *Initial Mem*
by (*intro reachable-caps.Mem*[*of addr s c vaddr c'*]) (*auto split*: *if-splits*)
next
case (*Update k wk bytes r*)
have *derivable* (*available-caps CC ISA k t*) ⊆ *reachable-caps s*
using *assms axioms*
by (*intro derivable-available-caps-subseteq-reachable-caps*)
then show *?thesis*
using *Update* ‹*is-tagged-method CC c*› *axioms*
unfolding *cheri-axioms-def store-cap-mem-axiom-def cap-derivable-iff-derivable*
by (*auto simp*: *writes-mem-cap-at-idx-def writes-mem-cap-Some-iff*)
qed
qed (*auto intro*: *reachable-caps.intros*)
qed

lemma *reachable-caps-instr-trace-intradomain-monotonicity*:
assumes *t*: *hasTrace t* (*instr-sem ISA instr*)
and *ta*: *instr-assms t*
and *s'*: *s-run-trace t s* = *Some s'*
and *no-exception*: ¬*instr-raises-ex ISA instr t*
and *no-ccall*: ¬*invokes-caps ISA instr t*
and *addr-trans-inv*: *s-invariant* (λ*s' addr load*. *s-translate-address addr load s'*)
t s
and *translation-table-addrs-invariant*: *s-invariant s-translation-tables t s*
and *no-caps-in-translation-tables*: *s-invariant-holds no-caps-in-translation-tables*
t s
shows *reachable-caps s'* ⊆ *reachable-caps s*
using *assms instr-cheri-axioms*[*OF t ta*]
by (*intro reachable-caps-trace-intradomain-monotonicity*) *auto*

lemma *reachable-caps-fetch-trace-intradomain-monotonicity*:
assumes *t*: *hasTrace t* (*instr-fetch ISA*)
and *ta*: *fetch-assms t*
and *s'*: *s-run-trace t s* = *Some s'*
and *no-exception*: ¬*fetch-raises-ex ISA t*

and *addr-trans-inv*: *s-invariant* ($\lambda s'$ *addr load*. *s-translate-address addr load* s') *t s*
and *translation-table-addrs-invariant*: *s-invariant s-translation-tables t s*
and *no-caps-in-translation-tables*: *s-invariant-holds no-caps-in-translation-tables t s*
shows *reachable-caps* $s' \subseteq$ *reachable-caps s*
using *assms fetch-cheri-axioms*[*OF t ta*]
by (*intro reachable-caps-trace-intradomain-monotonicity*) *auto*

end

Multi-instruction sequences

fun *fetch-execute-loop* :: ('*cap*, '*regval*, '*instr*, '*e*) *isa* $\Rightarrow$ *nat* $\Rightarrow$ ('*regval*, *unit*, '*e*) *monad* **where**
fetch-execute-loop ISA (*Suc bound*) = (*instr-fetch ISA* $\gg=$ *instr-sem ISA*) $\gg$ *fetch-execute-loop ISA bound*
| *fetch-execute-loop ISA 0* = *return* ()

fun *instrs-raise-ex* :: ('*cap*, '*regval*, '*instr*, '*e*) *isa* $\Rightarrow$ *nat* $\Rightarrow$ '*regval trace* $\Rightarrow$ *bool* **where**
instrs-raise-ex ISA (*Suc bound*) *t* =
($\exists$ *tf* t'. $t = tf \mathbin{@} t' \wedge$ *hasTrace tf* (*instr-fetch ISA*) $\wedge$
(*fetch-raises-ex ISA tf* $\vee$
($\exists$ *instr ti* t''. $t' = ti \mathbin{@} t'' \wedge$
runTrace tf (*instr-fetch ISA*) = *Some* (*Done instr*) $\wedge$
hasTrace ti (*instr-sem ISA instr*) $\wedge$
(*instr-raises-ex ISA instr ti* $\vee$
instrs-raise-ex ISA bound t''))))
| *instrs-raise-ex ISA 0 t* = *False*

fun *instrs-invoke-caps* :: ('*cap*, '*regval*, '*instr*, '*e*) *isa* $\Rightarrow$ *nat* $\Rightarrow$ '*regval trace* $\Rightarrow$ *bool* **where**
instrs-invoke-caps ISA (*Suc bound*) *t* =
($\exists$ *tf* t'. $t = tf \mathbin{@} t' \wedge$ *hasTrace tf* (*instr-fetch ISA*) $\wedge$
($\exists$ *instr ti* t''. $t' = ti \mathbin{@} t'' \wedge$
runTrace tf (*instr-fetch ISA*) = *Some* (*Done instr*) $\wedge$
hasTrace ti (*instr-sem ISA instr*) $\wedge$
(*invokes-caps ISA instr ti* $\vee$
instrs-invoke-caps ISA bound t'')))
| *instrs-invoke-caps ISA 0 t* = *False*

context *CHERI-ISA-State*
begin

lemma *reachable-caps-instrs-trace-intradomain-monotonicity*:
assumes *t*: *hasTrace t* (*fetch-execute-loop ISA n*)
and *ta*: *fetch-assms t*
and s': *s-run-trace t s* = *Some* s'
and *no-exception*: $\neg$*instrs-raise-ex ISA n t*

and *no-ccall*: ¬*instrs-invoke-caps ISA n t*
and *addr-trans-inv*: *s-invariant* (λ*s′ addr load*. *s-translate-address addr load s′*) *t s*
and *translation-table-addrs-invariant*: *s-invariant s-translation-tables t s*
and *no-caps-in-translation-tables*: *s-invariant-holds no-caps-in-translation-tables t s*
shows *reachable-caps s′* ⊆ *reachable-caps s*
proof (*use assms* **in** ‹*induction n arbitrary*: *s t*›)
case *0*
then show *?case* **by** (*auto simp*: *return-def hasTrace-iff-Traces-final*)
next
case (*Suc n*)
then obtain *m′*
where (*instr-fetch ISA* ⤜ (λ*instr*. ⟦*instr*⟧ ≫ *fetch-execute-loop ISA n*), *t*, *m′*) ∈ *Traces*
and *m′*: *final m′*
by (*auto simp*: *hasTrace-iff-Traces-final*)
then show *?case*
proof (*cases rule*: *bind-Traces-cases*)
case (*Left m″*)
then have *hasTrace t* (*instr-fetch ISA*)
using *m′*
by (*auto elim*!: *final-bind-cases*) (*auto simp*: *hasTrace-iff-Traces-final final-def*)
then show *?thesis*
using *Suc.prems*
by (*intro reachable-caps-fetch-trace-intradomain-monotonicity*) *auto*
next
case (*Bind tf instr t′*)
obtain *s″* **where** *s″*: *s-run-trace tf s* = *Some s″* **and** *t′*: *s-run-trace t′ s″* = *Some s′*
using *Bind Suc*
by (*auto elim*: *runTraceS-appendE*)
have *tf*: *hasTrace tf* (*instr-fetch ISA*)
using *Bind*
by (*auto simp*: *hasTrace-iff-Traces-final final-def*)
have *invs′*:
s-invariant (λ*s′ addr load*. *s-translate-address addr load s′*) *t′ s″*
s-invariant s-translation-tables t′ s″
s-invariant-holds no-caps-in-translation-tables t′ s″
using *tf s″ Bind Suc.prems*
using *s-invariant-run-trace-eq*[*of no-caps-in-translation-tables tf s s″*]
by (*auto simp*: *s-invariant-append*)
have *ta′*: *fetch-assms tf instr-assms t′*
using *Bind Suc.prems fetch-assms-appendE*
by *auto*
from ‹(⟦*instr*⟧ ≫ *fetch-execute-loop ISA n*, *t′*, *m′*) ∈ *Traces*›
have *reachable-caps s′* ⊆ *reachable-caps s″*
proof (*cases rule*: *bind-Traces-cases*)
case (*Left m″*)

then have *hasTrace* t' $[\![instr]\!]$
using m'
by (*auto elim*!: *final-bind-cases*) (*auto simp*: *hasTrace-iff-Traces-final final-def*)
then show *?thesis*
using *tf* t' s'' *Bind Suc.prems* $invs'$ ta'
by (*intro reachable-caps-instr-trace-intradomain-monotonicity*)
(*auto simp*: *runTrace-iff-Traces*)
next
case (*Bind ti am* t'')
obtain s''' **where** s''': *s-run-trace ti* $s'' = Some\ s'''$ **and** t'': *s-run-trace* t'' $s''' = Some\ s'$
using *Bind* t'
by (*auto elim*: *runTraceS-appendE*)
have *ti*: *hasTrace ti* $[\![instr]\!]$
using *Bind*
by (*auto simp*: *hasTrace-iff-Traces-final final-def*)
have $invs''$:
s-invariant ($\lambda s'$ *addr load*. *s-translate-address addr load* s') t'' s'''
s-invariant s-translation-tables t'' s'''
s-invariant-holds no-caps-in-translation-tables t'' s'''
using $invs'$ s''' *Bind*
using *s-invariant-run-trace-eq*[*of no-caps-in-translation-tables ti* s'' s''']
by (*auto simp*: *s-invariant-append*)
have ta'': *instr-assms ti fetch-assms* t''
using *Bind* ta' *instr-assms-appendE*
by *auto*
have *no-exception*$'$: ¬*fetch-raises-ex ISA tf* ¬*instr-raises-ex ISA instr ti*
and *no-ccall*$'$: ¬*invokes-caps ISA instr ti*
and *no-exception*$''$: ¬*instrs-raise-ex ISA n* t''
and *no-ccall*$''$: ¬*instrs-invoke-caps ISA n* t''
using *ti tf Suc.prems Bind* $\langle t = tf\ @\ t' \rangle$
using ⟨*Run* (*instr-fetch ISA*) *tf instr*⟩
by (*auto simp*: *runTrace-iff-Traces*)
then have *reachable-caps* $s' \subseteq$ *reachable-caps* s'''
using *Bind* m' t'' $invs''$ ta''
by (*intro Suc.IH*) (*auto simp*: *hasTrace-iff-Traces-final final-def*)
also have *reachable-caps* $s''' \subseteq$ *reachable-caps* s''
using *ti* s''' *no-exception*$'$ *no-ccall*$'$ $invs'$ $\langle t' = ti\ @\ t'' \rangle$ ta''
by (*intro reachable-caps-instr-trace-intradomain-monotonicity*)
(*auto simp*: *s-invariant-append*)
finally show *?thesis* **.**
qed
also have *reachable-caps* $s'' \subseteq$ *reachable-caps s*
using *tf* s'' *Bind Suc.prems* ta'
by (*intro reachable-caps-fetch-trace-intradomain-monotonicity*)
(*auto simp*: *s-invariant-append*)
finally show *?thesis* **.**
qed
qed

end

end
theory *Trace-Assumptions*
imports *Sail.Sail2-state-lemmas HOL−Eisbach.Eisbach-Tools*
begin

2 Verification infrastructure

lemma *return-Traces-iff* [*simp*]:
$(return\ x,\ t,\ m') \in Traces \longleftrightarrow t = [] \wedge m' = Done\ x$
by (*auto simp*: *return-def*)

lemma *Run-read-regE*:
assumes *Run* (*read-reg r*) *t v*
obtains (*Read*) *rv* **where** $t = [E\text{-}read\text{-}reg\ (name\ r)\ rv]$ **and** *of-regval r rv* = *Some v*
using *assms*
by (*auto simp*: *read-reg-def elim*!: *Read-reg-TracesE split*: *option.splits*)

lemmas *Run-elims* = *Run-bindE Run-or-boolM-E Run-returnE Run-letE Run-and-boolM-E Run-ifE*

lemma *Run-assert-exp-iff* [*simp*]: $Run\ (assert\text{-}exp\ c\ m)\ t\ a \longleftrightarrow c \wedge t = [] \wedge a = ()$
by (*auto simp*: *assert-exp-def*)

lemma *Run-liftR-assert-exp-iff* [*simp*]:
$Run\ (liftR\ (assert\text{-}exp\ c\ msg :: ('r,\ unit,\ 'ex)\ monad))\ t\ a \longleftrightarrow Run\ (assert\text{-}exp\ c\ msg :: ('r,\ unit,\ 'ex)\ monad)\ t\ a$
by (*auto simp*: *assert-exp-def liftR-def*)

lemma *Run-foreachM-appendE*:
assumes *Run* (*foreachM* (*xs* @ *ys*) *vars body*) *t vars'*
obtains *txs tys vars''*
where *t* = *txs* @ *tys*
and *Run* (*foreachM xs vars body*) *txs vars''*
and *Run* (*foreachM ys vars'' body*) *tys vars'*
proof −
have ∃ *txs tys vars''*.
t = *txs* @ *tys* ∧
Run (*foreachM xs vars body*) *txs vars''* ∧
Run (*foreachM ys vars'' body*) *tys vars'*
proof (*use assms* **in** ‹*induction xs arbitrary*: *vars t*›)
case (*Cons x xs*)
then obtain *vars'' tx t'*
where *tx*: *Run* (*body x vars*) *tx vars''*
and *t'*: *Run* (*foreachM* (*xs* @ *ys*) *vars'' body*) *t' vars'*

and t: $t = tx @ t'$
by (*auto elim*: *Run-bindE*)
from *Cons.IH*[*OF* t'] **obtain** $vars'''$ *txs tys*
where $t' = txs @ tys$
and *Run* (*foreachM xs* $vars''$ *body*) *txs* $vars'''$
and *tys*: *Run* (*foreachM ys* $vars'''$ *body*) *tys* $vars'$
by *blast*
then have *Run* (*foreachM* (x # *xs*) *vars body*) (tx @ *txs*) $vars'''$
using *tx*
by (*auto intro*: *Traces-bindI*)
then show *?case*
using *tys*
unfolding ‹$t = tx @ t'$› **and** ‹$t' = txs @ tys$› **and** *append-assoc*[*symmetric*]
by *blast*
qed *auto*
then show *thesis*
using *that*
by *blast*
qed

lemma *Run-foreachM-elim*:
assumes *Run* (*foreachM xs vars body*) *t* $vars'$
and $\bigwedge n$ *tl tn tr* $vars'$ $vars''$.
⟦$t = tl @ tn @ tr$;
P tl $vars'$;
Run (*body* (*xs* ! *n*) $vars'$) *tn* $vars''$;
$n <$ *length xs*⟧
$\Longrightarrow$ *P* (*tl* @ *tn*) $vars''$
and *P* [] *vars*
shows *P t* $vars'$
using *assms*
proof (*use assms* **in** ‹*induction xs arbitrary*: *t* $vars'$ *rule*: *rev-induct*›)
case (*snoc x xs*)
then obtain *txs tx* $vars''$
where *t*: $t = txs @ tx$
and *txs*: *Run* (*foreachM xs vars body*) *txs* $vars''$
and *tx*: *Run* (*body x* $vars''$) *tx* $vars'$
by (*elim Run-foreachM-appendE*) *auto*
then have *P txs* $vars''$
using ‹*P* [] *vars*›
by (*intro snoc.IH*[*OF txs*]) (*auto simp*: *nth-append intro*!: *snoc.prems*(*2*))
then show *?case*
using *t txs tx*
using *snoc.prems*(*2*)[**where** *tl* = *txs* **and** *tn* = *tx* **and** *tr* = [] **and** *n* = *length*
xs]
by *auto*
qed *auto*

lemma *Run-try-catchE*:

assumes *Run* (*try-catch m h*) *t a*
obtains (*Run*) *Run m t a*
| (*Catch*) *tm e th* **where** (*m*, *tm*, *Exception e*) ∈ *Traces* **and** *Run* (*h e*) *th a* **and** *t* = *tm* @ *th*
proof (*use assms* **in** ‹*cases rule*: *try-catch-Traces-cases*›)
case (*NoEx m′*)
then show *?thesis*
by (*cases* (*m′*, *h*) *rule*: *try-catch.cases*) (*auto elim*!: *Run Catch*)
next
case (*Ex tm ex th*)
show *?thesis* **using** *Catch*[*OF Ex*] **.**
qed

lemma *throw-Traces-iff* [*simp*]:
(*throw e*, *t*, *m′*) ∈ *Traces* ⟷ *t* = [] ∧ *m′* = *Exception e*
by (*auto simp*: *throw-def*)

lemma *early-return-Traces-iff* [*simp*]:
(*early-return a*, *t*, *m′*) ∈ *Traces* ⟷ *t* = [] ∧ *m′* = *Exception* (*Inl a*)
by (*auto simp*: *early-return-def*)

lemma *Run-catch-early-returnE*:
assumes *Run* (*catch-early-return m*) *t a*
obtains (*Run*) *Run m t a*
| (*Early*) (*m*, *t*, *Exception* (*Inl a*)) ∈ *Traces*
using *assms*
unfolding *catch-early-return-def*
by (*elim Run-try-catchE*) (*auto split*: *sum.splits*)

2.1 Assumptions about register reads and writes

definition *no-reg-writes-to Rs m* ≡ (∀ *t m′ r v*. (*m*, *t*, *m′*) ∈ *Traces* ∧ *r* ∈ *Rs* ⟶ *E-write-reg r v* ∉ *set t*)
definition *runs-no-reg-writes-to Rs m* ≡ (∀ *t a r v*. *Run m t a* ∧ *r* ∈ *Rs* ⟶ *E-write-reg r v* ∉ *set t*)

locale *Register-State* =
fixes *get-regval* :: *string* ⇒ *′regstate* ⇒ *′regval option*
and *set-regval* :: *string* ⇒ *′regval* ⇒ *′regstate* ⇒ *′regstate option*
begin

fun *updates-regs* :: *string set* ⇒ *′regval trace* ⇒ *′regstate* ⇒ *′regstate option* **where**
updates-regs R [] *s* = *Some s*
| *updates-regs R* (*E-write-reg r v* # *t*) *s* =
(*if r* ∈ *R*
then Option.bind (*set-regval r v s*) (*updates-regs R t*)
else updates-regs R t s)
| *updates-regs R* (- # *t*) *s* = *updates-regs R t s*

fun *reads-regs-from* :: *string set* ⇒ *'regval trace* ⇒ *'regstate* ⇒ *bool* **where**
reads-regs-from R [] *s* = *True*
| *reads-regs-from R* (*E-read-reg r v* # *t*) *s* =
(*if r* ∈ *R*
then get-regval r s = *Some v* ∧ *reads-regs-from R t s*
else reads-regs-from R t s)
| *reads-regs-from R* (*E-write-reg r v* # *t*) *s* =
(*if r* ∈ *R*
then (*case set-regval r v s of Some s'* ⇒ *reads-regs-from R t s'* | *None* ⇒ *False*)
else reads-regs-from R t s)
| *reads-regs-from R* (- # *t*) *s* = *reads-regs-from R t s*

lemma *reads-regs-from-updates-regs-Some*:
assumes *reads-regs-from R t s*
obtains *s'* **where** *updates-regs R t s* = *Some s'*
using *assms*
by (*induction R t s rule*: *reads-regs-from.induct*) (*auto split*: *if-splits option.splits*)

named-theorems *regstate-simp*

lemma *updates-regs-append-iff* [*regstate-simp*]:
updates-regs R (*t* @ *t'*) *s* = *Option.bind* (*updates-regs R t s*) (*updates-regs R t'*)
by (*induction R t s rule*: *updates-regs.induct*) (*auto split*: *bind-splits*)

lemma *reads-regs-from-append-iff* [*regstate-simp*]:
reads-regs-from R (*t* @ *t'*) *s* ⟷ (*reads-regs-from R t s* ∧ *reads-regs-from R t'* (*the* (*updates-regs R t s*)))
by (*induction R t s rule*: *reads-regs-from.induct*) (*auto split*: *option.splits*)

lemma *reads-regs-from-appendE-simp*:
assumes *reads-regs-from Rs t regs* **and** *t* = *t1* @ *t2*
and *the* (*updates-regs Rs t1 regs*) = *regs'*
obtains *reads-regs-from Rs t1 regs* **and** *reads-regs-from Rs t2 regs'*
using *assms*
by (*auto simp*: *reads-regs-from-append-iff*)

lemma *no-reg-writes-to-updates-regs-inv*[*simp*]:
assumes (*m*, *t*, *m'*) ∈ *Traces*
and *no-reg-writes-to Rs m*
shows *updates-regs Rs t s* = *Some s*
using *assms*
proof −
have ∀ *r* ∈ *Rs*. ∀ *v*. *E-write-reg r v* ∉ *set t*
using *assms*
by (*auto simp*: *no-reg-writes-to-def*)
then show *updates-regs Rs t s* = *Some s*
by (*induction Rs t s rule*: *updates-regs.induct*) *auto*
qed

lemma *no-reg-writes-to-updates-regsE*:
  assumes $(m, t, m') \in$ *Traces*
    and *no-reg-writes-to Rs m*
  obtains *updates-regs Rs t s* = *Some s*
  using *assms*
  by *auto*

named-theorems *no-reg-writes-toI*
named-theorems *runs-no-reg-writes-toI*

lemma *no-reg-writes-runs-no-reg-writes*:
  no-reg-writes-to Rs m $\Longrightarrow$ *runs-no-reg-writes-to Rs m*
  by (*auto simp*: *no-reg-writes-to-def runs-no-reg-writes-to-def*)

lemma *no-reg-writes-to-bindI*[*intro*, *simp*, *no-reg-writes-toI*]:
  assumes *no-reg-writes-to Rs m* **and** $\bigwedge t\ a$. *Run m t a* $\Longrightarrow$ *no-reg-writes-to Rs* (*f a*)
  shows *no-reg-writes-to Rs* ($m \gg= f$)
  using *assms*
  by (*auto simp*: *no-reg-writes-to-def elim*: *bind-Traces-cases*)

lemma *runs-no-reg-writes-to-bindI*[*intro*, *simp*, *runs-no-reg-writes-toI*]:
  assumes *runs-no-reg-writes-to Rs m* **and** $\bigwedge t\ a$. *Run m t a* $\Longrightarrow$ *runs-no-reg-writes-to Rs* (*f a*)
  shows *runs-no-reg-writes-to Rs* ($m \gg= f$)
  using *assms*
  by (*auto simp*: *runs-no-reg-writes-to-def elim*: *Run-bindE*)

lemma *no-reg-writes-to-return*[*simp*, *no-reg-writes-toI*]:
  no-reg-writes-to Rs (*return a*)
  by (*auto simp*: *no-reg-writes-to-def*)

lemma *no-reg-writes-to-throw*[*simp*, *no-reg-writes-toI*]:
  no-reg-writes-to Rs (*throw e*)
  by (*auto simp*: *no-reg-writes-to-def*)

lemma *no-reg-writes-to-Fail*[*simp*, *no-reg-writes-toI*]:
  no-reg-writes-to Rs (*Fail msg*)
  by (*auto simp*: *no-reg-writes-to-def*)

lemma *no-reg-writes-to-try-catchI*[*intro*, *simp*, *no-reg-writes-toI*]:
  assumes *no-reg-writes-to Rs m* **and** $\bigwedge e$. *no-reg-writes-to Rs* (*h e*)
  shows *no-reg-writes-to Rs* (*try-catch m h*)
  using *assms*
  by (*auto simp*: *no-reg-writes-to-def elim!*: *try-catch-Traces-cases*)

lemma *no-reg-writes-to-catch-early-returnI*[*intro*, *simp*, *no-reg-writes-toI*]:
  assumes *no-reg-writes-to Rs m*

shows *no-reg-writes-to Rs* (*catch-early-return m*)
using *assms*
by (*auto simp*: *catch-early-return-def split*: *sum.splits*)

lemma *no-reg-writes-to-early-return*[*intro*, *simp*, *no-reg-writes-toI*]:
shows *no-reg-writes-to Rs* (*early-return a*)
by (*auto simp*: *early-return-def*)

lemma *no-reg-writes-to-liftR-I*[*intro*, *simp*, *no-reg-writes-toI*]:
assumes *no-reg-writes-to Rs m*
shows *no-reg-writes-to Rs* (*liftR m*)
using *assms*
by (*auto simp*: *liftR-def*)

lemma *no-reg-writes-to-let*[*simp*, *no-reg-writes-toI*]:
no-reg-writes-to Rs (*f x*) $\Longrightarrow$ *no-reg-writes-to Rs* (*let a* = *x in f a*)
by *auto*

lemma *no-reg-writes-to-if*[*simp*, *no-reg-writes-toI*]:
assumes *c* $\Longrightarrow$ *no-reg-writes-to Rs m1* **and** $\neg$*c* $\Longrightarrow$ *no-reg-writes-to Rs m2*
shows *no-reg-writes-to Rs* (*if c then m1 else m2*)
using *assms*
by *auto*

lemma *runs-no-reg-writes-to-if*[*simp*, *runs-no-reg-writes-toI*]:
assumes *c* $\Longrightarrow$ *runs-no-reg-writes-to Rs m1* **and** $\neg$*c* $\Longrightarrow$ *runs-no-reg-writes-to Rs m2*
shows *runs-no-reg-writes-to Rs* (*if c then m1 else m2*)
using *assms*
by *auto*

lemma *no-reg-writes-to-case-prod*[*intro*, *simp*, *no-reg-writes-toI*]:
assumes $\bigwedge$*x y*. *no-reg-writes-to Rs* (*f x y*)
shows *no-reg-writes-to Rs* (*case z of* (*x*, *y*) $\Rightarrow$ *f x y*)
using *assms*
by (*cases z*) *auto*

lemma *runs-no-reg-writes-to-case-prod*[*intro*, *simp*, *runs-no-reg-writes-toI*]:
assumes $\bigwedge$*x y*. *runs-no-reg-writes-to Rs* (*f x y*)
shows *runs-no-reg-writes-to Rs* (*case z of* (*x*, *y*) $\Rightarrow$ *f x y*)
using *assms*
by (*cases z*) *auto*

lemma *no-reg-writes-to-choose-bool*[*simp*, *no-reg-writes-toI*]:
no-reg-writes-to Rs (*choose-bool desc*)
by (*auto simp*: *choose-bool-def no-reg-writes-to-def elim*: *Traces-cases*)

lemma *no-reg-writes-to-undefined-bool*[*simp*, *no-reg-writes-toI*]:
no-reg-writes-to Rs (*undefined-bool* ())

by (*auto simp*: *undefined-bool-def*)

lemma *no-reg-writes-to-foreachM-inv*[*simp*, *no-reg-writes-toI*]:
assumes $\bigwedge$*x vars*. *no-reg-writes-to Rs* (*body x vars*)
shows *no-reg-writes-to Rs* (*foreachM xs vars body*)
using *assms*
by (*induction xs vars body rule*: *foreachM.induct*) *auto*

lemma *no-reg-writes-to-bool-of-bitU-nondet*[*simp*, *no-reg-writes-toI*]:
no-reg-writes-to Rs (*bool-of-bitU-nondet b*)
by (*cases b*) (*auto simp*: *bool-of-bitU-nondet-def*)

lemma *no-reg-writes-to-and-boolM*[*intro*, *simp*, *no-reg-writes-toI*]:
assumes *no-reg-writes-to Rs m1* **and** *no-reg-writes-to Rs m2*
shows *no-reg-writes-to Rs* (*and-boolM m1 m2*)
using *assms*
by (*auto simp*: *and-boolM-def*)

lemma *no-reg-writes-to-or-boolM*[*intro*, *simp*, *no-reg-writes-toI*]:
assumes *no-reg-writes-to Rs m1* **and** *no-reg-writes-to Rs m2*
shows *no-reg-writes-to Rs* (*or-boolM m1 m2*)
using *assms*
by (*auto simp*: *or-boolM-def*)

lemma *no-reg-writes-to-assert-exp*[*simp*, *no-reg-writes-toI*]:
no-reg-writes-to Rs (*assert-exp c m*)
by (*auto simp*: *assert-exp-def no-reg-writes-to-def*)

lemma *no-reg-writes-to-exit*[*simp*, *no-reg-writes-toI*]:
no-reg-writes-to Rs (*exit0* ())
by (*auto simp*: *exit0-def no-reg-writes-to-def*)

lemma *no-reg-writes-to-read-reg*[*simp*, *no-reg-writes-toI*]:
no-reg-writes-to Rs (*read-reg r*)
by (*auto simp*: *no-reg-writes-to-def read-reg-def elim*: *Read-reg-TracesE split*: *option.splits*)

lemma *no-reg-writes-to-write-reg*[*simp*, *no-reg-writes-toI*]:
assumes *name r* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*write-reg r v*)
using *assms*
by (*auto simp*: *no-reg-writes-to-def write-reg-def elim*!: *Write-reg-TracesE*)

lemma *no-reg-writes-to-read-mem-bytes*[*simp*, *no-reg-writes-toI*]:
no-reg-writes-to Rs (*read-mem-bytes BC BC' rk addr bytes*)
by (*auto simp*: *read-mem-bytes-def no-reg-writes-to-def maybe-fail-def*
elim: *Traces-cases split*: *option.splits*)

lemma *no-reg-writes-to-read-mem*[*simp*, *no-reg-writes-toI*]:

no-reg-writes-to Rs (*read-mem BC BC′ rk addr-length addr bytes*)
by (*auto simp*: *read-mem-def split*: *option.splits*)

lemma *no-reg-writes-to-write-mem-ea*[*simp*, *no-reg-writes-toI*]:
no-reg-writes-to Rs (*write-mem-ea BC wk addr-length addr bytes*)
by (*auto simp*: *write-mem-ea-def no-reg-writes-to-def maybe-fail-def split*: *option.splits elim*: *Traces-cases*)

lemma *no-reg-writes-to-write-mem*[*simp*, *no-reg-writes-toI*]:
no-reg-writes-to Rs (*write-mem BC BC′ wk addr-length addr bytes value*)
by (*auto simp*: *write-mem-def no-reg-writes-to-def split*: *option.splits elim*: *Traces-cases*)

lemma *no-reg-writes-to-genlistM*[*simp*, *no-reg-writes-toI*]:
assumes $\bigwedge i$. *no-reg-writes-to Rs* (*f i*)
shows *no-reg-writes-to Rs* (*genlistM f n*)
using *assms*
by (*auto simp*: *genlistM-def*)

lemma *no-reg-writes-to-choose-bools*[*simp*, *no-reg-writes-toI*]:
shows *no-reg-writes-to Rs* (*choose-bools desc n*)
by (*auto simp*: *choose-bools-def*)

lemma *no-reg-writes-to-chooseM*[*simp*, *no-reg-writes-toI*]:
shows *no-reg-writes-to Rs* (*chooseM desc xs*)
by (*auto simp*: *chooseM-def split*: *option.splits*)

lemma *no-reg-writes-to-internal-pick*[*simp*, *no-reg-writes-toI*]:
shows *no-reg-writes-to Rs* (*internal-pick xs*)
by (*auto simp*: *internal-pick-def*)

lemma *no-reg-writes-to-bools-of-bits-nondet*[*simp*, *no-reg-writes-toI*]:
shows *no-reg-writes-to Rs* (*bools-of-bits-nondet bits*)
by (*auto simp*: *bools-of-bits-nondet-def*)

lemma *no-reg-writes-to-of-bits-nondet*[*simp*, *no-reg-writes-toI*]:
shows *no-reg-writes-to Rs* (*of-bits-nondet BC bits*)
by (*auto simp*: *of-bits-nondet-def*)

lemmas *no-reg-write-builtins* =
no-reg-writes-to-return no-reg-writes-to-throw no-reg-writes-to-Fail
no-reg-writes-to-early-return no-reg-writes-to-assert-exp
no-reg-writes-to-read-reg no-reg-writes-to-chooseM no-reg-writes-to-internal-pick
no-reg-writes-to-choose-bool no-reg-writes-to-undefined-bool
no-reg-writes-to-bool-of-bitU-nondet no-reg-writes-to-bools-of-bits-nondet
no-reg-writes-to-of-bits-nondet no-reg-writes-to-choose-bools no-reg-writes-to-exit
no-reg-writes-to-read-mem-bytes no-reg-writes-to-read-mem
no-reg-writes-to-write-mem-ea no-reg-writes-to-write-mem

method *no-reg-writes-toI* **uses** *simp* =

(*intro runs-no-reg-writes-toI no-reg-writes-runs-no-reg-writes no-reg-writes-toI conjI impI*;
auto simp: *simp split del*: *if-split split*: *option.splits*)

lemma *Run-choose-bool-updates-regs*[*regstate-simp*]:
assumes *Run* (*choose-bool desc*) *t b*
shows *updates-regs Rs t regs* = *Some regs*
using *assms*
by (*auto simp*: *choose-bool-def elim*!: *Traces-cases*[**where** *t* = *t*])

lemma *Run-choose-bools-updates-regs*[*regstate-simp*]:
assumes *Run* (*choose-bools desc n*) *t b*
shows *updates-regs Rs t regs* = *Some regs*
using *assms*
by (*auto simp*: *choose-bools-def genlistM-def regstate-simp elim*!: *Run-foreachM-elim Run-bindE*)

lemma *Run-undefined-bool-updates-regs*[*regstate-simp*]:
assumes *Run* (*undefined-bool u*) *t b*
shows *updates-regs Rs t regs* = *Some regs*
using *assms*
unfolding *undefined-bool-def*
by (*elim Run-choose-bool-updates-regs*)

lemma *Run-internal-pick-updates-regs*[*regstate-simp*]:
assumes *Run* (*internal-pick xs*) *t a*
shows *updates-regs Rs t regs* = *Some regs*
using *assms*
by (*auto simp*: *internal-pick-def chooseM-def regstate-simp elim*!: *Run-elims split*: *option.splits*)

named-theorems *RunE*

method *RunE* **uses** *elim* =
(*match* **premises in** *R*[*thin*]: ‹*Run m t a*› **and** *regs*[*thin*]: *reads-regs-from Rs t regs* **for** *m t a Rs regs* ⇒
‹*match elim RunE in E*: ‹*R'* ⟹ *regs'* ⟹ -› *for R' regs'* ⇒
‹*match* (‹*Run m t a*›) *in R'* ⇒
‹*match* (‹*reads-regs-from Rs t regs*›) *in regs'* ⇒
‹*rule E*[*OF R regs*]; (*RunE elim*: *elim*)?››››)

end

2.2 State invariants

locale *State-Invariant* = *Register-State get-regval set-regval*
for *get-regval* :: *string* ⇒ *'regstate* ⇒ *'regval option*
and *set-regval* :: *string* ⇒ *'regval* ⇒ *'regstate* ⇒ *'regstate option*
+ **fixes** *invariant* :: *'regstate* ⇒ *bool* **and** *inv-regs* :: *register-name set*

begin

definition
Run-inv m t a regs ≡ *Run m t a* ∧ *reads-regs-from inv-regs t regs* ∧ *invariant regs*

definition *trace-preserves-invariant* :: *'regval trace* ⇒ *bool* **where**
trace-preserves-invariant t ≡
(∀ *s*. *invariant s* ∧ *reads-regs-from inv-regs t s* ⟶ *invariant* (*the* (*updates-regs inv-regs t s*)))

lemma *trace-preserves-invariantE*:
assumes *trace-preserves-invariant t* **and** *reads-regs-from inv-regs t s* **and** *invariant s*
obtains *s'* **where** *updates-regs inv-regs t s* = *Some s'* **and** *invariant s'*
using *assms*
by (*fastforce simp*: *trace-preserves-invariant-def elim*: *reads-regs-from-updates-regs-Some*)

lemma *trace-preserves-invariant-appendI*:
assumes *t1*: *trace-preserves-invariant t1* **and** *t2*: *trace-preserves-invariant t2*
shows *trace-preserves-invariant* (*t1* @ *t2*)
using *t2*
by (*auto simp*: *trace-preserves-invariant-def regstate-simp elim*: *trace-preserves-invariantE*[*OF t1*])

definition *traces-preserve-invariant* :: (*'regval*, *'a*, *'e*) *monad* ⇒ *bool* **where**
traces-preserve-invariant m ≡ (∀ *t m'*. (*m*, *t*, *m'*) ∈ *Traces* ⟶ *trace-preserves-invariant t*)

definition *runs-preserve-invariant* :: (*'regval*, *'a*, *'e*) *monad* ⇒ *bool* **where**
runs-preserve-invariant m ≡ (∀ *t a*. *Run m t a* ⟶ *trace-preserves-invariant t*)

definition *exceptions-preserve-invariant* :: (*'regval*, *'a*, *'e*) *monad* ⇒ *bool* **where**
exceptions-preserve-invariant m ≡ (∀ *t e*. (*m*, *t*, *Exception e*) ∈ *Traces* ⟶ *trace-preserves-invariant t*)

lemma *traces-runs-preserve-invariantI*:
assumes *traces-preserve-invariant m*
shows *runs-preserve-invariant m*
using *assms*
by (*auto simp*: *traces-preserve-invariant-def runs-preserve-invariant-def*)

lemma *traces-exceptions-preserve-invariantI*:
assumes *traces-preserve-invariant m*
shows *exceptions-preserve-invariant m*
using *assms*
by (*auto simp*: *traces-preserve-invariant-def exceptions-preserve-invariant-def*)

lemma *traces-preserve-invariantE*:

```
  assumes traces-preserve-invariant m
    and (m, t, m') ∈ Traces and invariant s and reads-regs-from inv-regs t s
  obtains s' where updates-regs inv-regs t s = Some s' and invariant s'
  using assms
  by (auto simp: traces-preserve-invariant-def elim: trace-preserves-invariantE)

lemma runs-preserve-invariantE:
  assumes runs-preserve-invariant m
    and Run m t a and invariant s and reads-regs-from inv-regs t s
  obtains s' where updates-regs inv-regs t s = Some s' and invariant s'
  using assms
  by (auto simp: runs-preserve-invariant-def elim: trace-preserves-invariantE)

lemma Run-inv-bindE:
  assumes Run-inv (m ≫= f) t a regs and runs-preserve-invariant m
  obtains tm am tf where t = tm @ tf and Run-inv m tm am regs
    and Run-inv (f am) tf a (the (updates-regs inv-regs tm regs))
proof −
  from assms
  obtain tm am tf
    where t: t = tm @ tf and tm: Run m tm am and tf: Run (f am) tf a
      and regs: reads-regs-from inv-regs tm regs
      and regs': reads-regs-from inv-regs tf (the (updates-regs inv-regs tm regs))
      and inv: invariant regs
    using assms unfolding Run-inv-def
    by (auto simp: regstate-simp elim!: Run-bindE)
  moreover obtain regs' where regs': updates-regs inv-regs tm regs = Some regs'
and inv': invariant regs'
    using assms tm inv regs
    by (elim runs-preserve-invariantE)
  ultimately show thesis
    using that
    unfolding Run-inv-def
    by auto
qed

named-theorems preserves-invariantI
named-theorems trace-preserves-invariantI

lemma no-reg-writes-trace-preserves-invariantI:
  assumes no-reg-writes-to inv-regs m
    and (m, t, m') ∈ Traces
  shows trace-preserves-invariant t
  using assms
  by (auto simp: trace-preserves-invariant-def)

lemma no-reg-writes-traces-preserve-invariantI:
  assumes no-reg-writes-to inv-regs m
  shows traces-preserve-invariant m
```

using *assms*
by (*auto simp*: *traces-preserve-invariant-def intro*: *no-reg-writes-trace-preserves-invariantI*)

method *preserves-invariantI* **uses** *intro simp elim* =
(*intro intro preserves-invariantI conjI allI impI traces-runs-preserve-invariantI traces-exceptions-preserve-invariantI*;
auto simp: *simp elim*!: *elim*)

lemma *traces-preserve-invariant-bindI*[*preserves-invariantI*]:
assumes *m*: *traces-preserve-invariant m*
and *f*: $\bigwedge$*s t a*. *Run-inv m t a s* $\Longrightarrow$ *traces-preserve-invariant* (*f a*)
shows *traces-preserve-invariant* (*m* $\gg\!=$ *f*)
proof −
{ fix *s t m'*
assume (*m* $\gg\!=$ *f*, *t*, *m'*) ∈ *Traces* **and** *s*: *invariant s* **and** *regs*: *reads-regs-from inv-regs t s*
then have *invariant* (*the* (*updates-regs inv-regs t s*))
proof (*cases rule*: *bind-Traces-cases*)
case (*Left m''*)
with *m s regs* **show** *?thesis*
by (*auto simp*: *traces-preserve-invariant-def trace-preserves-invariant-def*)
next
case (*Bind tm am tf*)
then obtain *s'*
where *regs'*: *reads-regs-from inv-regs tm s*
and *s'*: *updates-regs inv-regs tm s* = *Some s'*
using *regs*
by (*auto simp*: *regstate-simp elim*: *reads-regs-from-updates-regs-Some*)
then have *invariant s'* **and** *Run-inv m tm am s*
using *s m* ‹*Run m tm am*›
by (*fastforce simp*: *traces-preserve-invariant-def trace-preserves-invariant-def Run-inv-def*)+
then show *?thesis*
using *Bind s' regs f*[*OF* ‹*Run-inv m tm am s*›]
by (*auto simp*: *traces-preserve-invariant-def trace-preserves-invariant-def regstate-simp*)
qed
}
then show *?thesis*
by (*simp add*: *traces-preserve-invariant-def trace-preserves-invariant-def*)
qed

lemma *runs-preserve-invariant-bindI*[*preserves-invariantI*]:
assumes *runs-preserve-invariant m* **and** $\bigwedge$*t a*. *Run m t a* $\Longrightarrow$ *runs-preserve-invariant* (*f a*)
shows *runs-preserve-invariant* (*m* $\gg\!=$ *f*)
using *assms*

by (*fastforce simp*: *runs-preserve-invariant-def elim*!: *Run-bindE intro*: *trace-preserves-invariant-appendI*)

lemma *runs-preserve-invariant-try-catchI*[*preserves-invariantI*]:
assumes *runs-preserve-invariant m*
and *exceptions-preserve-invariant m*
and $\bigwedge$*t e*. (*m*, *t*, *Exception e*) ∈ *Traces* ⟹ *runs-preserve-invariant* (*h e*)
shows *runs-preserve-invariant* (*try-catch m h*)
using *assms*
by (*fastforce simp*: *runs-preserve-invariant-def exceptions-preserve-invariant-def elim*!: *Run-try-catchE intro*: *trace-preserves-invariant-appendI*)

lemma *preserves-invariant-case-sum*[*preserves-invariantI*]:
assumes $\bigwedge$*a*. *traces-preserve-invariant* (*f a*) **and** $\bigwedge$*b*. *traces-preserve-invariant* (*g b*)
shows *traces-preserve-invariant* (*case x of Inl a* ⇒ *f a* | *Inr b* ⇒ *g b*)
using *assms*
by (*auto split*: *sum.splits*)

lemma *preserves-invariant-case-option*[*preserves-invariantI*]:
assumes $\bigwedge$*a*. *traces-preserve-invariant* (*f a*) **and** *traces-preserve-invariant g*
shows *traces-preserve-invariant* (*case x of Some a* ⇒ *f a* | *None* ⇒ *g*)
using *assms*
by (*auto split*: *option.splits*)

lemma *preserves-invariant-case-prod*[*preserves-invariantI*]:
assumes $\bigwedge$*x y*. *traces-preserve-invariant* (*f x y*)
shows *traces-preserve-invariant* (*case z of* (*x*, *y*) ⇒ *f x y*)
using *assms*
by *auto*

lemmas *no-reg-write-builtins-preserve-invariant*[*preserves-invariantI*] =
no-reg-write-builtins[*THEN no-reg-writes-traces-preserve-invariantI*]

lemma *preserves-invariant-if*[*preserves-invariantI*]:
assumes *c* ⟹ *traces-preserve-invariant m1* **and** ¬*c* ⟹ *traces-preserve-invariant m2*
shows *traces-preserve-invariant* (*if c then m1 else m2*)
using *assms*
by *auto*

lemma *preserves-invariant-try-catch*[*preserves-invariantI*]:
assumes *traces-preserve-invariant m*
and $\bigwedge$*t e*. (*m*, *t*, *Exception e*) ∈ *Traces* ⟹ *traces-preserve-invariant* (*h e*)
shows *traces-preserve-invariant* (*try-catch m h*)
using *assms*
by (*fastforce simp*: *traces-preserve-invariant-def elim*!: *try-catch-Traces-cases intro*: *trace-preserves-invariant-appendI*)

lemma *preserves-invariant-catch-early-return*[*preserves-invariantI*]:

assumes *traces-preserve-invariant m*
shows *traces-preserve-invariant* (*catch-early-return m*)
using *assms*
by (*auto simp*: *catch-early-return-def intro*: *preserves-invariantI*)

lemma *runs-preserve-invariant-catch-early-returnI*[*preserves-invariantI*]:
assumes *runs-preserve-invariant m*
and *exceptions-preserve-invariant m*
shows *runs-preserve-invariant* (*catch-early-return m*)
using *assms*
unfolding *catch-early-return-def*
by (*auto intro*!: *preserves-invariantI no-reg-write-builtins-preserve-invariant*[*THEN traces-runs-preserve-invariantI*] *split*: *sum.splits*)

lemma *preserves-invariant-liftR*[*preserves-invariantI*]:
assumes *traces-preserve-invariant m*
shows *traces-preserve-invariant* (*liftR m*)
using *assms*
by (*auto simp*: *liftR-def intro*: *preserves-invariantI*)

lemma *Nil-preserves-invariant*[*intro*, *simp*]:
trace-preserves-invariant []
by (*auto simp*: *trace-preserves-invariant-def*)

lemma *preserves-invariant-and-boolM*[*preserves-invariantI*]:
assumes *traces-preserve-invariant m1* **and** *traces-preserve-invariant m2*
shows *traces-preserve-invariant* (*and-boolM m1 m2*)
using *assms*
by (*auto simp*: *and-boolM-def intro*: *preserves-invariantI*)

lemma *preserves-invariant-or-boolM*[*preserves-invariantI*]:
assumes *traces-preserve-invariant m1* **and** *traces-preserve-invariant m2*
shows *traces-preserve-invariant* (*or-boolM m1 m2*)
using *assms*
by (*auto simp*: *or-boolM-def intro*: *preserves-invariantI*)

lemma *preserves-invariant-let*[*preserves-invariantI*]:
assumes *traces-preserve-invariant* (*f y*)
shows *traces-preserve-invariant* (*let x* = *y in f x*)
using *assms*
by *auto*

lemma *runs-preserve-invariant-foreachM*[*preserves-invariantI*]:
assumes $\bigwedge$*x vars*. *runs-preserve-invariant* (*body x vars*)
shows *runs-preserve-invariant* (*foreachM xs vars body*)
using *assms*
by (*induction xs arbitrary*: *vars*) (*auto intro*: *preserves-invariantI traces-runs-preserve-invariantI*)

lemma *preserves-invariant-foreachM*[*preserves-invariantI*]:

assumes $\bigwedge x\ vars.\ traces\text{-}preserve\text{-}invariant\ (body\ x\ vars)$
shows *traces-preserve-invariant* (*foreachM xs vars body*)
using *assms*
by (*induction xs arbitrary*: *vars*) (*auto intro*: *preserves-invariantI*)

end

2.3 Deterministic expressions

context *State-Invariant*
begin

definition *Traces-inv regs* $\equiv \{(m, t, m') \mid m\ t\ m'.\ (m, t, m') \in Traces \wedge reads\text{-}regs\text{-}from\ inv\text{-}regs\ t\ regs \wedge invariant\ regs \wedge final\ m'\}$
definition *determ-traces m* $\equiv (\forall t1\ m1'\ regs1\ t2\ m2'\ regs2.\ (m, t1, m1') \in Traces\text{-}inv\ regs1 \wedge (m, t2, m2') \in Traces\text{-}inv\ regs2 \longrightarrow m1' = m2')$
definition *determ-runs m* $\equiv (\forall t1\ a1\ regs1\ t2\ a2\ regs2.\ Run\text{-}inv\ m\ t1\ a1\ regs1 \wedge Run\text{-}inv\ m\ t2\ a2\ regs2 \longrightarrow a1 = a2)$
definition *the-outcome m* $\equiv (THE\ m'.\ \exists t\ regs.\ (m, t, m') \in Traces\text{-}inv\ regs)$
definition *the-result m* $\equiv (THE\ a.\ \exists t\ regs.\ Run\text{-}inv\ m\ t\ a\ regs)$

lemma *in-Traces-inv-iff*:
$(m, t, m') \in Traces\text{-}inv\ regs \longleftrightarrow (m, t, m') \in Traces \wedge reads\text{-}regs\text{-}from\ inv\text{-}regs\ t\ regs \wedge invariant\ regs \wedge final\ m'$
by (*auto simp*: *Traces-inv-def*)

lemma *Run-inv-iff-Traces-inv*:
$Run\text{-}inv\ m\ t\ a\ regs \longleftrightarrow (m, t, Done\ a) \in Traces\text{-}inv\ regs$
unfolding *Run-inv-def in-Traces-inv-iff*
by (*auto simp*: *final-def*)

lemma *determ-tracesI*:
assumes $\bigwedge t\ m''\ regs.\ (m, t, m'') \in Traces\text{-}inv\ regs \Longrightarrow m'' = m'$
shows *determ-traces m*
using *assms*
unfolding *determ-traces-def*
by *blast*

lemma *determ-runsI*:
assumes $\bigwedge t\ a\ regs.\ Run\text{-}inv\ m\ t\ a\ regs \Longrightarrow a = c$
shows *determ-runs m*
using *assms*
unfolding *determ-runs-def*
by *blast*

named-theorems *determ*

lemma *determ-the-outcome-eq*:
  assumes *determ-traces m* **and** $(m, t, m') \in$ *Traces-inv regs*
  shows *the-outcome* $m = m'$
  using *assms*
  unfolding *the-outcome-def determ-traces-def in-Traces-inv-iff*
  by *blast*

lemma *determ-the-result-eq*:
  assumes *determ-runs m* **and** *Run-inv m t a regs*
  shows *the-result* $m = a$
  using *assms*
  unfolding *the-result-def determ-runs-def*
  by *blast*

lemma *determ-traces-runs*:
  assumes *determ-traces m*
  shows *determ-runs m*
  using *assms*
  unfolding *determ-traces-def determ-runs-def Run-inv-iff-Traces-inv*
  by *blast*

lemma *determ-runs-return*[*determ*]: *determ-runs* (*return a*)
  by (*auto simp*: *determ-runs-def Run-inv-def*)

lemma *determ-traces-return*[*determ*]: *determ-traces* (*return a*)
  by (*auto simp*: *determ-traces-def in-Traces-inv-iff*)

lemma *determ-traces-throw*[*determ*]: *determ-traces* (*throw e*)
  by (*auto simp*: *determ-traces-def in-Traces-inv-iff*)

lemma *determ-runs-bindI*:
  assumes *determ-runs m* **and** *determ-runs* (*f* (*the-result m*)) **and** *runs-preserve-invariant m*
  shows *determ-runs* ($m \gg= f$)
  using *assms*
  by (*intro determ-runsI*[**where** *c* = *the-result* (*f* (*the-result m*))])
    (*auto elim*!: *Run-inv-bindE simp*: *determ-the-result-eq*)

lemma *final-simps*[*intro*, *simp*]:
  final (*Exception e*)
  final (*Fail msg*)
  by (*auto simp*: *final-def*)

lemma *runs-preserve-invariant-Run-invariant*[*simp*]:
  assumes *runs-preserve-invariant m*
    and *Run m t a* **and** *invariant s* **and** *reads-regs-from inv-regs t s*
  shows *invariant* (*the* (*updates-regs inv-regs t s*))
  using *assms*
  by (*auto elim*!: *runs-preserve-invariantE*)

lemma *traces-preserve-invariant-Traces-invariant*[*simp*]:
assumes *traces-preserve-invariant m*
and $(m, t, m') \in$ *Traces* **and** *invariant s* **and** *reads-regs-from inv-regs t s*
shows *invariant* (*the* (*updates-regs inv-regs t s*))
using *assms*
by (*auto elim*!: *traces-preserve-invariantE*)

lemma *bind-Traces-inv-cases*:
assumes $(m \gg= f, t, m') \in$ *Traces-inv regs* **and** *runs-preserve-invariant m*
obtains (*Ex*) *e* **where** $(m, t,$ *Exception e*$) \in$ *Traces-inv regs* **and** $m' =$ *Exception e*
| (*Fail*) *msg* **where** $(m, t,$ *Fail msg*$) \in$ *Traces-inv regs* **and** $m' =$ *Fail msg*
| (*Bind*) *tm am tf* **where** $t = tm$ @ *tf* **and** *Run-inv m tm am regs*
and $(f\ am, tf, m') \in$ *Traces-inv* (*the* (*updates-regs inv-regs tm regs*))
using *assms Bind*[*of t* []]
unfolding *in-Traces-inv-iff*
by (*auto elim*!: *bind-Traces-cases final-bind-cases simp*: *Run-inv-def regstate-simp elim*: *final-cases*)

lemma *determ-traces-bindI*:
assumes *determ-traces m* **and** *runs-preserve-invariant m*
and $\bigwedge t\ a\ regs.$ *Run-inv m t a regs* $\Longrightarrow$ *determ-traces* $(f\ a)$
shows *determ-traces* $(m \gg= f)$
unfolding *determ-traces-def*
using *assms*
by (*auto simp*: *Run-inv-iff-Traces-inv elim*!: *bind-Traces-inv-cases final-bind-cases dest*!: *determ-the-outcome-eq*[*OF assms*(*1*), *rotated*] *determ-the-outcome-eq*[*OF assms*(*3*), *rotated*])

lemma *try-catch-eq-iff*:
(*try-catch m h* = *Done a*) $\longleftrightarrow$ (m = *Done a* $\vee$ ($\exists e.$ m = *Exception e* $\wedge$ $h\ e$ = *Done a*))
(*try-catch m h* = *Exception e*) $\longleftrightarrow$ ($\exists e'.$ m = *Exception e'* $\wedge$ $h\ e'$ = *Exception e*)
(*try-catch m h* = *Fail msg*) $\longleftrightarrow$ (m = *Fail msg* $\vee$ ($\exists e.$ m = *Exception e* $\wedge$ $h\ e$ = *Fail msg*))
by (*cases m*; *auto*)+

lemma *try-catch-Traces-inv-cases*:
assumes (*try-catch m h*, *t*, *mtc*) $\in$ *Traces-inv regs* **and** *traces-preserve-invariant m*
obtains (*Done*) *a* **where** *Run-inv m t a regs* **and** *mtc* = *Done a*
| (*Fail*) *msg* **where** $(m, t,$ *Fail msg*$) \in$ *Traces-inv regs* **and** *mtc* = *Fail msg*
| (*Ex*) *tm ex th* **where** $(m, tm,$ *Exception ex*$) \in$ *Traces-inv regs*
and (*h ex*, *th*, *mtc*) $\in$ *Traces-inv* (*the* (*updates-regs inv-regs tm regs*)) **and** t = *tm* @ *th*
using *assms*
unfolding *in-Traces-inv-iff Run-inv-def*

by (*auto elim*!: *try-catch-Traces-cases final-cases simp*: *regstate-simp try-catch-eq-iff*; *fastforce*)

lemma *determ-traces-try-catchI*:
assumes *determ-traces m* **and** *traces-preserve-invariant m*
and $\bigwedge e$. *determ-traces* (*h e*)
shows *determ-traces* (*try-catch m h*)
unfolding *determ-traces-def*
using *assms determ-the-outcome-eq*[*OF assms*(*3*)] *determ-the-outcome-eq*[*OF assms*(*1*)]
by (*fastforce simp*: *Run-inv-iff-Traces-inv elim*!: *try-catch-Traces-inv-cases*
dest!: *determ-the-outcome-eq*[*OF assms*(*1*), *rotated*] *determ-the-outcome-eq*[*OF assms*(*3*), *rotated*])

lemma *determ-traces-liftR*[*determ*]:
assumes *determ-traces m* **and** *traces-preserve-invariant m*
shows *determ-traces* (*liftR m*)
using *assms*
unfolding *liftR-def*
by (*auto intro*!: *determ-traces-try-catchI determ*)

lemma *determ-traces-catch-early-return*[*determ*]:
assumes *determ-traces m* **and** *traces-preserve-invariant m*
shows *determ-traces* (*catch-early-return m*)
using *assms*
unfolding *catch-early-return-def*
by (*auto intro*!: *determ-traces-try-catchI determ split*: *sum.splits*)

lemma *determ-traces-early-return*[*determ*]:
determ-traces (*early-return a*)
by (*auto simp*: *early-return-def intro*: *determ*)

lemma *determ-traces-foreachM*:
assumes $\bigwedge x$ *vars*. $x \in$ *set xs* $\Longrightarrow$ *determ-traces* (*body x vars*)
and $\bigwedge x$ *vars*. $x \in$ *set xs* $\Longrightarrow$ *runs-preserve-invariant* (*body x vars*)
shows *determ-traces* (*foreachM xs vars body*)
using *assms*
by (*induction xs arbitrary*: *vars*) (*auto intro*: *determ determ-traces-bindI*)

lemma *determ-runs-if*:
determ-runs (*if c then m1 else m2*) **if** $c \Longrightarrow$ *determ-runs m1* **and** $\neg c \Longrightarrow$ *determ-runs m2*
using *that*
by *auto*

lemma *determ-traces-if*:
determ-traces (*if c then m1 else m2*) **if** $c \Longrightarrow$ *determ-traces m1* **and** $\neg c \Longrightarrow$ *determ-traces m2*
using *that*

by *auto*

lemma *determ-traces-read-inv-reg*:
assumes *name r* $\in$ *inv-regs*
and $\forall$ *regs*. *invariant regs* $\longrightarrow$ *get-regval* (*name r*) *regs* = *Some v* $\wedge$ *of-regval r v* = *Some* (*read-from r regs*)
shows *determ-traces* (*read-reg r*)
using *assms*
by (*intro determ-tracesI*[**where** m' = *Done* (*the* (*of-regval r v*))])
(*auto simp*: *Traces-inv-def read-reg-def elim*!: *Read-reg-TracesE final-cases split*: *option.splits*)

lemma *determ-runs-read-inv-reg*:
determ-runs (*read-reg r*) **if** *name r* $\in$ *inv-regs* **and** $\bigwedge$*regs*. *invariant regs* $\Longrightarrow$ *get-regval* (*name r*) *regs* = *Some v*
using *that*
by (*intro determ-runsI*[**where** *c* = *the* (*of-regval r v*)])
(*auto simp*: *determ-runs-def Run-inv-def elim*!: *Run-read-regE*)

lemma *determ-runs-or-boolM*[*determ*]:
determ-runs (*or-boolM m1 m2*) **if** *determ-runs m1* **and** *determ-runs m2* **and** *runs-preserve-invariant m1*
using *that*
unfolding *or-boolM-def*
by (*auto intro*!: *determ-runs-bindI determ-runs-return*)

lemma *determ-runs-assert-exp*[*determ*]: *determ-runs* (*assert-exp e msg*)
by (*intro determ-runsI*) *auto*

lemma *determ-runs-case-prod*[*determ*]:
determ-runs (*case x of* (*y*, *z*) $\Rightarrow$ *f y z*) **if** $\bigwedge$*y z*. *x* = (*y*, *z*) $\Longrightarrow$ *determ-runs* (*f y z*)
using *that*
by *auto*

lemma *determ-runs-case-option*[*determ*]:
determ-runs (*case x of Some y* $\Rightarrow$ *f y* | *None* $\Rightarrow$ *g*) **if** $\bigwedge$*y*. *x* = *Some y* $\Longrightarrow$ *determ-runs* (*f y*) **and** *determ-runs g*
using *that*
by (*cases x*) *auto*

lemma *determ-traces-exit*[*determ*]: *determ-traces* (*exit0 u*)
by (*intro determ-tracesI*) (*auto simp*: *exit0-def in-Traces-inv-iff*)

lemmas *determ-runs-exit0* = *determ-traces-exit*[*THEN determ-traces-runs*, *determ*]

end

(Conditionally) deterministic monadic expressions

definition *determ-exp-if P m c* ≡ (∀ *t a*. *Run m t a* ∧ *P t* ⟶ *a* = *c*)
definition *prefix-closed P* ≡ (∀ *t1 t2*. *P* (*t1* @ *t2*) ⟶ *P t1*)

lemma *Run-bind-determ-exp-ifE*:
  assumes *prefix-closed P*
    and *determ-exp-if P m c*
    and *Run* (*m* ≫= *f*) *t a*
    and *P t*
  obtains *tm tf* **where** *Run m tm c* **and** *Run* (*f c*) *tf a* **and** *t* = *tm* @ *tf*
  using *assms*
  by (*elim Run-bindE*) (*auto simp*: *determ-exp-if-def prefix-closed-def*)

abbreviation *determ-exp* ≡ *determ-exp-if* (λ-. *True*)

lemma *Run-bind-determ-expE*:
  assumes *determ-exp m c*
    and *Run* (*m* ≫= *f*) *t a*
  obtains *tm tf* **where** *Run m tm c* **and** *Run* (*f c*) *tf a* **and** *t* = *tm* @ *tf*
  using *assms*
  by (*elim Run-bindE*) (*auto simp*: *determ-exp-if-def*)

end
theory *Recognising-Automata*
imports *Cheri-axioms-lemmas Sail.Sail2-state-lemmas Trace-Assumptions*
begin

2.4 Verification tools for CHERI properties

For proving that a concrete ISA satisfies the CHERI axioms, we define an automaton for each axiom that only accepts traces satisfying the axiom. The state of the automaton keeps track of relevant information, e.g. the capabilities read so far.

This makes it easy to decompose proofs about complete instruction traces into proofs about parts of a trace, e.g. corresponding to calls to auxiliary functions.

locale *Deterministic-Automaton* =
  fixes *enabled* :: ′*s* ⇒ ′*rv event* ⇒ *bool*
    and *step* :: ′*s* ⇒ ′*rv event* ⇒ ′*s*
    and *initial* :: ′*s*
    and *final* :: ′*s* ⇒ *bool*
begin

fun *trace-enabled* :: ′*s* ⇒ ′*rv trace* ⇒ *bool* **where**
  trace-enabled s [] = *True*
| *trace-enabled s* (*e* # *t*) = (*enabled s e* ∧ *trace-enabled* (*step s e*) *t*)

abbreviation *run* :: ′*s* ⇒ ′*rv trace* ⇒ ′*s* **where** *run s t* ≡ *foldl step s t*

definition *accepts-from* :: $'s \Rightarrow 'rv$ *trace* $\Rightarrow$ *bool* **where**
accepts-from s t $\equiv$ *trace-enabled s t* $\wedge$ *final* (*run s t*)

abbreviation *accepts* $\equiv$ *accepts-from initial*

lemma *trace-enabled-append-iff*: *trace-enabled s* (t @ t') $\longleftrightarrow$ *trace-enabled s t* $\wedge$ *trace-enabled* (*run s t*) t'
by (*induction t arbitrary*: *s*) *auto*

lemma *accepts-from-append-iff*: *accepts-from s* (t @ t') $\longleftrightarrow$ *trace-enabled s t* $\wedge$ *accepts-from* (*run s t*) t'
by (*auto simp*: *accepts-from-def trace-enabled-append-iff*)

lemma *accepts-from-Cons*[*simp*]: *accepts-from s* (e # t) $\longleftrightarrow$ *enabled s e* $\wedge$ *accepts-from* (*step s e*) *t*
by (*auto simp*: *accepts-from-def*)

lemma *accepts-from-id-take-nth-drop*:
assumes $i <$ *length t*
shows *accepts-from s t* = *accepts-from s* (*take i t* @ *t* ! *i* # *drop* (*Suc i*) *t*)
using *assms*
by (*auto simp*: *id-take-nth-drop*[*symmetric*])

lemma *accepts-from-trace-enabledI*:
assumes *accepts-from s t*
shows *trace-enabled s t*
using *assms*
by (*auto simp*: *accepts-from-def*)

lemma *accepts-from-trace-enabled-takeI*:
assumes *accepts-from s t*
shows *trace-enabled s* (*take i t*)
using *assms*
by (*cases i* $<$ *length t*)
(*auto simp*: *accepts-from-id-take-nth-drop accepts-from-append-iff intro*: *accepts-from-trace-enabledI*)

lemma *accepts-from-nth-enabledI*:
assumes *accepts-from s t*
and $i <$ *length t*
shows *enabled* (*run s* (*take i t*)) (*t* ! *i*)
using *assms*
by (*auto simp*: *accepts-from-id-take-nth-drop accepts-from-append-iff*)

lemma *accepts-from-iff-all-enabled-final*:
accepts-from s t $\longleftrightarrow$ ($\forall\, i <$ *length t*. *enabled* (*run s* (*take i t*)) (*t* ! *i*)) $\wedge$ *final* (*run s t*)
by (*induction t arbitrary*: *s*)
(*auto simp*: *accepts-from-def nth-Cons split*: *nat.splits*)

lemma *trace-enabled-acceptI*:
assumes *trace-enabled s t* **and** *final* (*run s t*)
shows *accepts-from s t*
using *assms*
by (*auto simp*: *accepts-from-def*)

named-theorems *trace-simp*
named-theorems *trace-elim*

lemma *Nil-trace-enabled*[*trace-elim*]:
assumes $t = []$
shows *trace-enabled s t*
using *assms*
by *auto*

lemma *bind-TracesE*:
assumes $(m \gg= f,\ t,\ m') \in \mathit{Traces}$
and $\bigwedge tm\ tf\ m''.\ (m,\ tm,\ m'') \in \mathit{Traces} \Longrightarrow t = tm\ @\ tf \Longrightarrow P\ tm$
and $\bigwedge tm\ am\ tf.\ (f\ am,\ tf,\ m') \in \mathit{Traces} \Longrightarrow \mathit{Run}\ m\ tm\ am \Longrightarrow t = tm\ @\ tf \Longrightarrow P\ tm \Longrightarrow P\ (tm\ @\ tf)$
shows $P\ t$
proof (*use assms* **in** ‹*cases rule*: *bind-Traces-cases*›)
case (*Left* m'')
then show *?thesis* **using** *assms*(*2*)[**where** $tm = t$ **and** $tf = []$] **by** *auto*
next
case (*Bind tm am tf*)
then show *?thesis* **using** *assms*(*2*) *assms*(*3*) **by** *auto*
qed

lemma *Run-bind-trace-enabled*[*trace-elim*]:
assumes $\mathit{Run}\ (m \gg= f)\ t\ a$
and $\bigwedge tm\ tf\ am.\ t = tm\ @\ tf \Longrightarrow \mathit{Run}\ m\ tm\ am \Longrightarrow \mathit{trace\text{-}enabled}\ s\ tm$
and $\bigwedge tm\ tf\ am.\ t = tm\ @\ tf \Longrightarrow \mathit{Run}\ m\ tm\ am \Longrightarrow \mathit{Run}\ (f\ am)\ tf\ a \Longrightarrow \mathit{trace\text{-}enabled}\ (\mathit{run}\ s\ tm)\ tf$
shows *trace-enabled s t*
using *assms*
by (*elim Run-bindE*) (*auto simp*: *trace-enabled-append-iff*)

lemma *Exception-bind-trace-enabled*:
assumes $(m \gg= f,\ t,\ \mathit{Exception}\ e) \in \mathit{Traces}$
and $(m,\ t,\ \mathit{Exception}\ e) \in \mathit{Traces} \Longrightarrow \mathit{trace\text{-}enabled}\ s\ t$
and $\bigwedge tm\ tf\ am.\ t = tm\ @\ tf \Longrightarrow \mathit{Run}\ m\ tm\ am \Longrightarrow \mathit{trace\text{-}enabled}\ s\ tm$
and $\bigwedge tm\ tf\ am.\ t = tm\ @\ tf \Longrightarrow \mathit{Run}\ m\ tm\ am \Longrightarrow (f\ am,\ tf,\ \mathit{Exception}\ e) \in \mathit{Traces} \Longrightarrow \mathit{trace\text{-}enabled}\ (\mathit{run}\ s\ tm)\ tf$
shows *trace-enabled s t*
proof (*use assms* **in** ‹*cases rule*: *bind-Traces-cases*›)
case (*Left* m'')
then consider (*Ex*) $m'' = \mathit{Exception}\ e$ | (*Done*) *a* **where** $m'' = \mathit{Done}\ a$ **and** $f\ a = \mathit{Exception}\ e$

by (*cases* m'') *auto*
then show *?thesis*
using ‹(m, t, m'') ∈ *Traces*› *assms*
by *cases auto*
next
case (*Bind tm am tf*)
then show *?thesis*
using *assms*
by (*auto simp*: *trace-enabled-append-iff*)
qed

lemma *bind-Traces-trace-enabled*[*trace-elim*]:
assumes ($m \ggg= f$, t, m') ∈ *Traces*
and $\bigwedge$*tm tf* m''. (m, *tm*, m'') ∈ *Traces* ⟹ t = *tm* @ *tf* ⟹ *trace-enabled s tm*
and $\bigwedge$*tm am tf*. (*f am*, *tf*, m') ∈ *Traces* ⟹ *Run m tm am* ⟹ t = *tm* @ *tf* ⟹ *trace-enabled* (*run s tm*) *tf*
shows *trace-enabled s t*
using *assms*
by (*elim bind-TracesE*) (*auto simp*: *trace-enabled-append-iff*)

lemma *try-catch-trace-enabled*[*trace-elim*]:
assumes (*try-catch m h*, t, m') ∈ *Traces*
and $\bigwedge$$n$ m''. (m, *take n t*, m'') ∈ *Traces* ⟹ *trace-enabled s* (*take n t*)
and $\bigwedge$*tm ex th*. (*h ex*, *th*, m') ∈ *Traces* ⟹ (m, *tm*, *Exception ex*) ∈ *Traces* ⟹ t = *tm* @ *th* ⟹ *trace-enabled* (*run s tm*) *th*
shows *trace-enabled s t*
proof (*use assms* **in** ‹*cases rule*: *try-catch-Traces-cases*›)
case (*NoEx* m'')
then show *?thesis* **using** *assms*(*2*)[*of length t* m''] **by** *auto*
next
case (*Ex tm ex th*)
then show *?thesis* **using** *assms*(*2*)[*of length tm*] *assms*(*3*) **by** (*auto simp*: *trace-enabled-append-iff*)
qed

lemma *if-Traces-trace-enabled*[*trace-elim*]:
assumes (*if b then m1 else m2*, t, m') ∈ *Traces*
and b ⟹ (*m1*, t, m') ∈ *Traces* ⟹ *trace-enabled s t*
and ¬b ⟹ (*m2*, t, m') ∈ *Traces* ⟹ *trace-enabled s t*
shows *trace-enabled s t*
using *assms* **by** (*cases b*) *auto*

lemma *let-Traces-trace-enabled*[*trace-elim*]:
assumes (*let x = y in f x*, t, m') ∈ *Traces*
and (*f y*, t, m') ∈ *Traces* ⟹ *trace-enabled s t*
shows *trace-enabled s t*
using *assms* **by** *auto*

lemma *case-prod-Traces-trace-enabled*[*trace-elim*]:
  assumes (*case p of* (*a*, *b*) ⇒ *f a b*, *t*, *m′*) ∈ *Traces*
    and $\bigwedge$*x y*. *p* = (*x*, *y*) ⟹ (*f x y*, *t*, *m′*) ∈ *Traces* ⟹ *trace-enabled s t*
  shows *trace-enabled s t*
  using *assms* **by** (*cases p*) *auto*

lemma *case-option-Traces-trace-enabled*[*trace-elim*]:
  assumes (*case x of Some y* ⇒ *f y* | *None* ⇒ *m*, *t*, *m′*) ∈ *Traces*
    and $\bigwedge$*y*. (*f y*, *t*, *m′*) ∈ *Traces* ⟹ *x* = *Some y* ⟹ *trace-enabled s t*
    and (*m*, *t*, *m′*) ∈ *Traces* ⟹ *x* = *None* ⟹ *trace-enabled s t*
  shows *trace-enabled s t*
  using *assms* **by** (*cases x*) *auto*

lemma *return-trace-enabled*[*trace-elim*]:
  assumes (*return a*, *t*, *m′*) ∈ *Traces*
  shows *trace-enabled s t*
  using *assms*
  by (*auto simp*: *return-def*)

lemma *throw-trace-enabled*[*trace-elim*]:
  assumes (*throw e*, *t*, *m′*) ∈ *Traces*
  shows *trace-enabled s t*
  using *assms*
  by (*auto simp*: *throw-def*)

lemma *early-return-trace-enabled*[*trace-elim*]:
  assumes (*early-return a*, *t*, *m′*) ∈ *Traces*
  shows *trace-enabled s t*
  using *assms*
  by (*auto simp*: *early-return-def elim*!: *trace-elim*)

lemma *catch-early-return-trace-enabled*[*trace-elim*]:
  assumes (*catch-early-return m*, *t*, *m′*) ∈ *Traces*
    and $\bigwedge$*n m″*. (*m*, *take n t*, *m″*) ∈ *Traces* ⟹ *trace-enabled s* (*take n t*)
  shows *trace-enabled s t*
  using *assms*
  by (*auto simp*: *catch-early-return-def elim*!: *trace-elim split*: *sum.splits*)

lemma *liftR-trace-enabled*[*trace-elim*]:
  assumes (*liftR m*, *t*, *m′*) ∈ *Traces*
    and $\bigwedge$*n m″*. (*m*, *take n t*, *m″*) ∈ *Traces* ⟹ *trace-enabled s* (*take n t*)
  shows *trace-enabled s t*
  using *assms*
  by (*auto simp*: *liftR-def elim*!: *trace-elim*)

lemma *foreachM-inv-trace-enabled*:
  assumes (*foreachM xs vars body*, *t*, *m′*) ∈ *Traces*
    and $\bigwedge$*s x vars t m′*. (*body x vars*, *t*, *m′*) ∈ *Traces* ⟹ *P s* ⟹ *x* ∈ *set xs* ⟹
trace-enabled s t

and $\bigwedge s\ x\ vars\ t\ vars'.\ Run\ (body\ x\ vars)\ t\ vars' \Longrightarrow P\ s \Longrightarrow x \in set\ xs \Longrightarrow P\ (run\ s\ t)$
and $P\ s$
shows *trace-enabled s t*
using *assms*
by (*induction xs arbitrary*: *s t vars*) (*auto simp*: *trace-enabled-append-iff elim*!: *trace-elim*)

lemma *foreachM-const-trace-enabled*[*trace-elim*]:
assumes $(foreachM\ xs\ vars\ body,\ t,\ m') \in Traces$
and $\bigwedge x\ vars\ t\ m'.\ (body\ x\ vars,\ t,\ m') \in Traces \Longrightarrow x \in set\ xs \Longrightarrow trace\text{-}enabled\ s\ t$
and $\bigwedge x\ vars\ t\ vars'.\ Run\ (body\ x\ vars)\ t\ vars' \Longrightarrow x \in set\ xs \Longrightarrow run\ s\ t = s$
shows *trace-enabled s t*
using *assms*
by (*elim foreachM-inv-trace-enabled*[**where** $P = \lambda s'.\ s' = s$]) *auto*

lemma *Run-and-boolM-trace-enabled*[*trace-elim*]:
assumes *Run* (*and-boolM l r*) *t a*
and $\bigwedge tl\ tr\ al.\ t = tl\ @\ tr \Longrightarrow Run\ l\ tl\ al \Longrightarrow trace\text{-}enabled\ s\ tl$
and $\bigwedge tl\ tr.\ t = tl\ @\ tr \Longrightarrow Run\ l\ tl\ True \Longrightarrow Run\ r\ tr\ a \Longrightarrow trace\text{-}enabled\ (run\ s\ tl)\ tr$
shows *trace-enabled s t*
using *assms*
unfolding *and-boolM-def*
by (*elim Run-bind-trace-enabled*) (*auto simp*: *return-def split*: *if-splits*)

lemma *and-boolM-trace-enabled*[*trace-elim*]:
assumes $(and\text{-}boolM\ m1\ m2,\ t,\ m') \in Traces$
and $\bigwedge tm\ tf\ m''.\ (m1,\ tm,\ m'') \in Traces \Longrightarrow t = tm\ @\ tf \Longrightarrow trace\text{-}enabled\ s\ tm$
and $\bigwedge tm\ tf.\ (m2,\ tf,\ m') \in Traces \Longrightarrow Run\ m1\ tm\ True \Longrightarrow t = tm\ @\ tf \Longrightarrow trace\text{-}enabled\ (run\ s\ tm)\ tf$
shows *trace-enabled s t*
using *assms*
by (*auto simp*: *and-boolM-def elim*!: *trace-elim*)

lemma *Run-or-boolM-trace-enabled*[*trace-elim*]:
assumes *Run* (*or-boolM l r*) *t a*
and $\bigwedge tl\ tr\ al.\ t = tl\ @\ tr \Longrightarrow Run\ l\ tl\ al \Longrightarrow trace\text{-}enabled\ s\ tl$
and $\bigwedge tl\ tr.\ t = tl\ @\ tr \Longrightarrow Run\ l\ tl\ False \Longrightarrow Run\ r\ tr\ a \Longrightarrow trace\text{-}enabled\ (run\ s\ tl)\ tr$
shows *trace-enabled s t*
using *assms*
unfolding *or-boolM-def*
by (*elim Run-bind-trace-enabled*) (*auto simp*: *return-def split*: *if-splits*)

lemma *or-boolM-trace-enabled*[*trace-elim*]:
assumes $(or\text{-}boolM\ m1\ m2,\ t,\ m') \in Traces$

and $\bigwedge$*tm tf m''*. (*m1*, *tm*, *m''*) ∈ *Traces* ⟹ *t* = *tm* @ *tf* ⟹ *trace-enabled s tm*
and $\bigwedge$*tm tf*. (*m2*, *tf*, *m'*) ∈ *Traces* ⟹ *Run m1 tm False* ⟹ *t* = *tm* @ *tf* ⟹ *trace-enabled* (*run s tm*) *tf*
shows *trace-enabled s t*
using *assms*
by (*auto simp*: *or-boolM-def elim*!: *trace-elim*)

end

An automaton for the axiom that capabilities stored to memory must be derivable from accessible capabilities

record (*'cap*, *'regval*) *axiom-state* =
accessed-caps :: *'cap set*
system-reg-access :: *bool*
read-from-KCC :: *'regval set*
written-regs :: *string set*

locale *Cap-Axiom-Automaton* = *Capability-ISA CC ISA*
for *CC* :: *'cap Capability-class* **and** *ISA* :: (*'cap*, *'regval*, *'instr*, *'e*) *isa* +
fixes *enabled* :: (*'cap*, *'regval*) *axiom-state* ⇒ *'regval event* ⇒ *bool*
begin

definition *accessible-regs* :: (*'cap*, *'regval*) *axiom-state* ⇒ *register-name set* **where**
accessible-regs s = {*r*. *r* ∉ *written-regs s* ∧ (*r* ∈ *privileged-regs ISA* ⟶ *system-reg-access s*)}

definition *axiom-step* :: (*'cap*, *'regval*) *axiom-state* ⇒ *'regval event* ⇒ (*'cap*, *'regval*) *axiom-state* **where**
axiom-step s e = (|*accessed-caps* = *accessed-caps s* ∪ *accessed-mem-caps e* ∪ *accessed-reg-caps* (*accessible-regs s*) *e*,
system-reg-access = *system-reg-access s* ∨ *allows-system-reg-access* (*accessible-regs s*) *e*,
read-from-KCC = *read-from-KCC s* ∪ {*v*. ∃ *r* ∈ *KCC ISA*. *e* = *E-read-reg r v*},
written-regs = *written-regs s* ∪ {*r*. ∃ *v c*. *e* = *E-write-reg r v* ∧ *c* ∈ *caps-of-regval ISA v* ∧ *is-tagged-method CC c*}|)

lemma *step-selectors*[*simp*]:
accessed-caps (*axiom-step s e*) = *accessed-caps s* ∪ *accessed-mem-caps e* ∪ *accessed-reg-caps* (*accessible-regs s*) *e*
system-reg-access (*axiom-step s e*) ⟷ *system-reg-access s* ∨ *allows-system-reg-access* (*accessible-regs s*) *e*
read-from-KCC (*axiom-step s e*) = *read-from-KCC s* ∪ {*v*. ∃ *r* ∈ *KCC ISA*. *e* = *E-read-reg r v*}
written-regs (*axiom-step s e*) = *written-regs s* ∪ {*r*. ∃ *v c*. *e* = *E-write-reg r v* ∧ *c* ∈ *caps-of-regval ISA v* ∧ *is-tagged-method CC c*}
by (*auto simp*: *axiom-step-def*)

abbreviation *initial* ≡ (|*accessed-caps* = {}, *system-reg-access* = *False*, *read-from-KCC* = {}, *written-regs* = {}|)

lemma *accessible-regs-initial-iff* [*simp*]:
  r ∈ *accessible-regs initial* ⟷ *r* ∉ *privileged-regs ISA*
  by (*auto simp*: *accessible-regs-def*)

sublocale *Deterministic-Automaton enabled axiom-step initial* λ-. *True* **.**

lemma *cap-reg-written-before-idx-written-regs*:
  cap-reg-written-before-idx CC ISA i r t ⟷ *r* ∈ *written-regs* (*run initial* (*take i t*))
proof (*induction i*)
  case (*Suc i*)
  then show *?case*
    by (*cases i* < *length t*) (*auto simp*: *take-Suc-conv-app-nth*)
qed *auto*

lemma *accessible-regs-axiom-step*:
  accessible-regs (*axiom-step s e*) =
     accessible-regs s ∪
      (*if allows-system-reg-access* (*accessible-regs s*) *e then privileged-regs ISA else* {}) −
     written-regs (*axiom-step s e*)
  by (*auto simp*: *accessible-regs-def*)

lemma *system-reg-access-run-take-eq*[*simp*]:
  system-access-permitted-before-idx CC ISA i t ⟷ *system-reg-access* (*run initial* (*take i t*))
    (**is** *?sys-reg-access i*)
  accessible-regs-at-idx i t = *accessible-regs* (*run initial* (*take i t*))
    (**is** *?accessible-regs i*)
proof (*induction i*)
  case (*Suc i*)
  show *?accessible-regs* (*Suc i*)
    by (*cases i* < *length t*)
       (*auto simp*: *Suc.IH accessible-regs-def accessible-regs-at-idx-def*
                 cap-reg-written-before-idx-written-regs take-Suc-conv-app-nth)
  show *?sys-reg-access* (*Suc i*)
    by (*cases i* < *length t*) (*auto simp*: *Suc.IH take-Suc-conv-app-nth*)
qed (*auto simp*: *accessible-regs-def*)

lemma *accessed-caps-run-take-eq*[*simp*]:
  available-caps CC ISA i t = *accessed-caps* (*run initial* (*take i t*))
proof (*induction i*)
  case (*Suc i*)
  then show *?case*
   by (*cases i* < *length t*) (*auto simp add*: *available-caps-Suc take-Suc-conv-app-nth*)
qed *auto*

lemma *read-from-KCC-run-take-eq*:
read-from-KCC (*run initial* (*take i t*)) = {*v*. ∃ *r j*. *j* < *i* ∧ *j* < *length t* ∧ *t* ! *j* = *E-read-reg r v* ∧ *r* ∈ *KCC ISA*}
proof (*induction i*)
case (*Suc i*)
then show *?case*
using *system-reg-access-run-take-eq*(*1*)[*of i t*]
by (*cases i* < *length t*) (*auto simp*: *take-Suc-conv-app-nth less-Suc-eq*)
qed *auto*

lemma *write-only-regs-run-take-eq*:
written-regs (*run initial* (*take i t*)) = {*r*. ∃ *v c j*. *t* ! *j* = *E-write-reg r v* ∧ *j* < *i* ∧ *j* < *length t* ∧ *c* ∈ *caps-of-regval ISA v* ∧ *is-tagged-method CC c*}
proof (*induction i*)
case (*Suc i*)
then show *?case*
by (*cases i* < *length t*) (*auto simp*: *take-Suc-conv-app-nth less-Suc-eq*)
qed *auto*

lemmas *step-defs* = *axiom-step-def reads-mem-cap-def*

abbreviation *special-reg-names* ≡ *PCC ISA* ∪ *IDC ISA* ∪ *KCC ISA* ∪ *privileged-regs ISA*

definition *non-cap-reg* :: ('*regstate*, '*regval*, '*a*) *register-ref* ⇒ *bool* **where**
non-cap-reg r ≡
name r ∉ *PCC ISA* ∪ *IDC ISA* ∪ *KCC ISA* ∪ *privileged-regs ISA* ∧
(∀ *rv v*. *of-regval r rv* = *Some v* ⟶ *caps-of-regval ISA rv* = {}) ∧
(∀ *v*. *caps-of-regval ISA* (*regval-of r v*) = {})

fun *non-cap-event* :: '*regval event* ⇒ *bool* **where**
non-cap-event (*E-read-reg r v*) = (*r* ∉ *PCC ISA* ∪ *IDC ISA* ∪ *KCC ISA* ∪ *privileged-regs ISA* ∧ *caps-of-regval ISA v* = {})
| *non-cap-event* (*E-write-reg r v*) = (*r* ∉ *PCC ISA* ∪ *IDC ISA* ∪ *KCC ISA* ∪ *privileged-regs ISA* ∧ *caps-of-regval ISA v* = {})
| *non-cap-event* (*E-read-memt* - - - -) = *False*
| *non-cap-event* (*E-read-mem* - - - -) = *False*
| *non-cap-event* (*E-write-memt* - - - - - -) = *False*
| *non-cap-event* (*E-write-mem* - - - - -) = *False*
| *non-cap-event* - = *True*

fun *non-mem-event* :: '*regval event* ⇒ *bool* **where**
non-mem-event (*E-read-memt* - - - -) = *False*
| *non-mem-event* (*E-read-mem* - - - -) = *False*
| *non-mem-event* (*E-write-memt* - - - - - -) = *False*
| *non-mem-event* (*E-write-mem* - - - - -) = *False*
| *non-mem-event* - = *True*

definition *non-cap-trace* :: *'regval trace* ⇒ *bool* **where**
non-cap-trace t ≡ (∀ *e* ∈ *set t*. *non-cap-event e*)

definition *non-mem-trace* :: *'regval trace* ⇒ *bool* **where**
non-mem-trace t ≡ (∀ *e* ∈ *set t*. *non-mem-event e*)

lemma *non-cap-trace-Nil*[*intro*, *simp*]:
non-cap-trace []
by (*auto simp*: *non-cap-trace-def*)

lemma *non-cap-trace-Cons*[*iff*]:
non-cap-trace (*e* # *t*) ⟷ *non-cap-event e* ∧ *non-cap-trace t*
by (*auto simp*: *non-cap-trace-def*)

lemma *non-cap-trace-append*[*iff*]:
non-cap-trace (*t1* @ *t2*) ⟷ *non-cap-trace t1* ∧ *non-cap-trace t2*
by (*induction t1*) *auto*

lemma *non-mem-trace-Nil*[*intro*, *simp*]:
non-mem-trace []
by (*auto simp*: *non-mem-trace-def*)

lemma *non-mem-trace-Cons*[*iff*]:
non-mem-trace (*e* # *t*) ⟷ *non-mem-event e* ∧ *non-mem-trace t*
by (*auto simp*: *non-mem-trace-def*)

lemma *non-mem-trace-append*[*iff*]:
non-mem-trace (*t1* @ *t2*) ⟷ *non-mem-trace t1* ∧ *non-mem-trace t2*
by (*induction t1*) *auto*

lemma *non-cap-event-non-mem-event*:
non-mem-event e **if** *non-cap-event e*
using *that*
by (*cases e*) *auto*

lemma *non-cap-trace-non-mem-trace*:
non-mem-trace t **if** *non-cap-trace t*
using *that*
by (*auto simp*: *non-mem-trace-def non-cap-trace-def intro*: *non-cap-event-non-mem-event*)

lemma *non-cap-event-axiom-step-inv*:
assumes *non-cap-event e*
shows *axiom-step s e* = *s*
using *assms*
by (*elim non-cap-event.elims*) (*auto simp*: *step-defs bind-eq-Some-conv split*: *option.splits*)

lemma *non-cap-trace-run-inv*:
assumes *non-cap-trace t*

shows *run s t = s*
using *assms*
by (*induction t*) (*auto simp*: *non-cap-event-axiom-step-inv*)

definition *non-cap-exp* :: (*'regval*, *'a*, *'exception*) *monad* $\Rightarrow$ *bool* **where**
non-cap-exp m = ($\forall$ *t m'*. (*m*, *t*, *m'*) $\in$ *Traces* $\longrightarrow$ (*non-cap-trace t* $\vee$ ($\exists$ *t' r v msg*. *t* = *t'* @ [*E-read-reg r v*] $\wedge$ *r* $\notin$ *special-reg-names* $\wedge$ *non-cap-trace t'* $\wedge$ *m'* = *Fail msg*)))

definition *non-mem-exp* :: (*'regval*, *'a*, *'exception*) *monad* $\Rightarrow$ *bool* **where**
non-mem-exp m = ($\forall$ *t m'*. (*m*, *t*, *m'*) $\in$ *Traces* $\longrightarrow$ *non-mem-trace t*)

lemma *non-cap-exp-Traces-cases*:
assumes *non-cap-exp m*
and (*m*, *t*, *m'*) $\in$ *Traces*
obtains (*Non-cap*) *non-cap-trace t*
| (*Fail*) *t' r v msg* **where** *t* = *t'* @ [*E-read-reg r v*] **and** *r* $\notin$ *special-reg-names* **and** *m'* = *Fail msg* **and** *non-cap-trace t'*
using *assms*
unfolding *non-cap-exp-def*
by *blast*

lemma *non-cap-exp-non-mem-exp*:
non-mem-exp m **if** *non-cap-exp m*
by (*auto simp*: *non-mem-exp-def elim!*: *non-cap-exp-Traces-cases*[*OF that*] *intro*: *non-cap-trace-non-mem-trace*)

lemma *non-cap-exp-Run-non-cap-trace*:
assumes *m*: *non-cap-exp m*
and *t*: *Run m t a*
shows *non-cap-trace t*
using *t*
by (*elim non-cap-exp-Traces-cases*[*OF m*]) *auto*

lemmas *non-cap-exp-Run-run-invI* = *non-cap-exp-Run-non-cap-trace*[*THEN non-cap-trace-run-inv*]

named-theorems *non-cap-expI*
named-theorems *non-mem-expI*

lemma *non-cap-exp-return*[*non-cap-expI*]:
non-cap-exp (*return a*)
by (*auto simp*: *non-cap-exp-def return-def*)

lemma *non-cap-exp-bindI*[*intro!*]:
assumes *m*: *non-cap-exp m*
and *f*: $\bigwedge$*t a*. *Run m t a* $\Longrightarrow$ *non-cap-exp* (*f a*)
shows *non-cap-exp* (*m* $\gg=$ *f*)
proof (*unfold non-cap-exp-def*, *intro allI impI*)
fix *t m'*

  assume $(m \ggg f,\ t,\ m') \in$ *Traces*
  then show *non-cap-trace* $t \vee (\exists t'\ r\ v\ msg.\ t = t' @ [$*E-read-reg* $r\ v] \wedge r \notin$ *special-reg-names* $\wedge$ *non-cap-trace* $t' \wedge m' =$ *Fail msg*)
  proof (*cases rule*: *bind-Traces-cases*)
    case (*Left* m'')
    then show *?thesis*
      by (*elim non-cap-exp-Traces-cases*[*OF m*]) *auto*
  next
    case (*Bind tm am tf*)
    then show *?thesis*
      using *non-cap-exp-Run-non-cap-trace*[*OF m* ‹*Run m tm am*›]
      by (*elim f*[*OF* ‹*Run m tm am*›, *THEN non-cap-exp-Traces-cases*]) *auto*
  qed
qed

lemma *non-mem-exp-bindI*[*intro*!]:
  assumes *non-mem-exp m*
    and $\bigwedge t\ a.$ *Run m t a* $\Longrightarrow$ *non-mem-exp* (*f a*)
  shows *non-mem-exp* $(m \ggg f)$
  using *assms*
  by (*fastforce simp*: *non-mem-exp-def elim*!: *bind-Traces-cases*)

lemma *non-cap-exp-try-catch*[*intro*!]:
  assumes *m*: *non-cap-exp m*
    and *h*: $\bigwedge t\ ex.\ (m,\ t,$ *Exception ex*$) \in$ *Traces* $\Longrightarrow$ *non-cap-exp* (*h ex*)
  shows *non-cap-exp* (*try-catch m h*)
proof (*unfold non-cap-exp-def*, *intro allI impI*)
  fix $t\ m'$
  assume (*try-catch m h*, $t,\ m') \in$ *Traces*
  then show *non-cap-trace* $t \vee (\exists t'\ r\ v\ msg.\ t = t' @ [$*E-read-reg* $r\ v] \wedge r \notin$ *special-reg-names* $\wedge$ *non-cap-trace* $t' \wedge m' =$ *Fail msg*)
  proof (*cases rule*: *try-catch-Traces-cases*)
    case (*NoEx* m'')
    then show *?thesis*
      by (*elim non-cap-exp-Traces-cases*[*OF m*]) *auto*
  next
    case (*Ex tm ex th*)
    then show *?thesis*
      by (*elim non-cap-exp-Traces-cases*[*OF m*]
          h[*OF* ‹(*m*, *tm*, *Exception ex*) $\in$ *Traces*›, *THEN non-cap-exp-Traces-cases*])
        auto
  qed
qed

lemma *non-mem-exp-try-catch*:
  assumes *non-mem-exp m*
    and $\bigwedge t\ ex.\ (m,\ t,$ *Exception ex*$) \in$ *Traces* $\Longrightarrow$ *non-mem-exp* (*h ex*)
  shows *non-mem-exp* (*try-catch m h*)
  using *assms*

by (*fastforce simp*: *non-mem-exp-def elim*!: *try-catch-Traces-cases*)

lemma *non-cap-exp-throw*[*non-cap-expI*]:
non-cap-exp (*throw e*)
by (*auto simp*: *non-cap-exp-def*)

lemma *non-cap-exp-early-return*[*non-cap-expI*]:
non-cap-exp (*early-return a*)
by (*auto simp*: *early-return-def intro*!: *non-cap-expI*)

lemma *non-cap-exp-catch-early-return*[*intro*!]:
non-cap-exp (*catch-early-return m*) **if** *non-cap-exp m*
by (*auto simp*: *catch-early-return-def intro*!: *that non-cap-expI split*: *sum.splits*)

lemma *non-mem-exp-catch-early-return*:
non-mem-exp (*catch-early-return m*) **if** *non-mem-exp m*
by (*auto simp*: *catch-early-return-def intro*!: *that non-mem-exp-try-catch non-cap-expI*[*THEN non-cap-exp-non-mem-exp*] *split*: *sum.splits*)

lemma *non-cap-exp-liftR*[*intro*!]:
non-cap-exp (*liftR m*) **if** *non-cap-exp m*
by (*auto simp*: *liftR-def intro*!: *that non-cap-expI*)

lemma *non-mem-exp-liftR*:
non-mem-exp (*liftR m*) **if** *non-mem-exp m*
by (*auto simp*: *liftR-def intro*!: *that non-mem-exp-try-catch non-cap-expI*[*THEN non-cap-exp-non-mem-exp*])

lemma *non-cap-exp-assert-exp*[*non-cap-expI*]:
non-cap-exp (*assert-exp c msg*)
by (*auto simp*: *assert-exp-def non-cap-exp-def*)

lemma *non-cap-exp-foreachM*[*intro*]:
assumes $\bigwedge x$ *vars*. $x \in$ *set xs* $\Longrightarrow$ *non-cap-exp* (*body x vars*)
shows *non-cap-exp* (*foreachM xs vars body*)
using *assms*
by (*induction xs vars body rule*: *foreachM.induct*) (*auto intro*: *non-cap-expI*)

lemma *non-mem-exp-foreachM*:
assumes $\bigwedge x$ *vars*. $x \in$ *set xs* $\Longrightarrow$ *non-mem-exp* (*body x vars*)
shows *non-mem-exp* (*foreachM xs vars body*)
using *assms*
by (*induction xs vars body rule*: *foreachM.induct*) (*auto intro*: *non-cap-expI*[*THEN non-cap-exp-non-mem-exp*])

lemma *non-cap-exp-choose-bool*[*non-cap-expI*]:
non-cap-exp (*choose-bool desc*)
by (*auto simp*: *choose-bool-def non-cap-exp-def elim*: *Traces-cases*)

lemma *non-cap-exp-undefined-bool*[*non-cap-expI*]:
non-cap-exp (*undefined-bool* ())
by (*auto simp*: *undefined-bool-def intro*: *non-cap-expI*)

lemma *non-cap-exp-bool-of-bitU-nondet*[*non-cap-expI*]:
non-cap-exp (*bool-of-bitU-nondet b*)
unfolding *bool-of-bitU-nondet-def*
by (*cases b*) (*auto intro*: *non-cap-expI*)

lemma *non-cap-exp-genlistM*:
assumes $\bigwedge n$. *non-cap-exp* (*f n*)
shows *non-cap-exp* (*genlistM f n*)
using *assms*
by (*auto simp*: *genlistM-def intro*!: *non-cap-expI*)

lemma *non-cap-exp-choose-bools*[*non-cap-expI*]:
non-cap-exp (*choose-bools desc n*)
by (*auto simp*: *choose-bools-def intro*: *non-cap-expI non-cap-exp-genlistM*)

lemma *non-cap-exp-Fail*[*non-cap-expI*]:
non-cap-exp (*Fail msg*)
by (*auto simp*: *non-cap-exp-def*)

lemma *non-cap-exp-exit*[*non-cap-expI*]:
non-cap-exp (*exit0* ())
unfolding *exit0-def*
by (*rule non-cap-exp-Fail*)

lemma *non-cap-exp-chooseM*[*non-cap-expI*]:
non-cap-exp (*chooseM desc xs*)
by (*auto simp*: *chooseM-def intro*!: *non-cap-expI split*: *option.splits*)

lemma *non-cap-exp-internal-pick*[*non-cap-expI*]:
non-cap-exp (*internal-pick xs*)
by (*auto simp*: *internal-pick-def intro*!: *non-cap-expI*)

lemma *non-cap-exp-and-boolM*[*intro*!]:
non-cap-exp (*and-boolM m1 m2*) **if** *non-cap-exp m1* **and** *non-cap-exp m2*
by (*auto simp*: *and-boolM-def intro*!: *that non-cap-expI*)

lemma *non-mem-exp-and-boolM*:
non-mem-exp (*and-boolM m1 m2*) **if** *non-mem-exp m1* **and** *non-mem-exp m2*
by (*auto simp*: *and-boolM-def intro*!: *that non-cap-expI*[*THEN non-cap-exp-non-mem-exp*])

lemma *non-cap-exp-or-boolM*[*intro*!]:
non-cap-exp (*or-boolM m1 m2*) **if** *non-cap-exp m1* **and** *non-cap-exp m2*
by (*auto simp*: *or-boolM-def intro*!: *that non-cap-expI*)

lemma *non-mem-exp-or-boolM*:

non-mem-exp (*or-boolM m1 m2*) **if** *non-mem-exp m1* **and** *non-mem-exp m2*
by (*auto simp*: *or-boolM-def intro*!: *that non-cap-expI*[*THEN non-cap-exp-non-mem-exp*])

lemma *non-cap-exp-let*[*intro*!]:
non-cap-exp (*let x = a in m x*) **if** *non-cap-exp* (*m a*)
by (*auto intro*: *that*)

lemma *non-mem-exp-let*:
non-mem-exp (*let x = a in m x*) **if** *non-mem-exp* (*m a*)
by (*auto intro*: *that*)

lemma *non-cap-exp-if*:
assumes $c \Longrightarrow$ *non-cap-exp m1* **and** $\neg c \Longrightarrow$ *non-cap-exp m2*
shows *non-cap-exp* (*if c then m1 else m2*)
using *assms*
by *auto*

lemma *non-mem-exp-if*:
assumes $c \Longrightarrow$ *non-mem-exp m1* **and** $\neg c \Longrightarrow$ *non-mem-exp m2*
shows *non-mem-exp* (*if c then m1 else m2*)
using *assms*
by *auto*

lemma *non-cap-exp-read-non-cap-reg*:
assumes *non-cap-reg r*
shows *non-cap-exp* (*read-reg r* :: (*'regval*, *'r*, *'exception*) *monad*)
proof −
have *non-cap-trace t* ∨ (∃ *v msg*. *t* = [*E-read-reg* (*name r*) *v*] ∧ *name r* ∉ *special-reg-names* ∧ *m'* = *Fail msg*)
if (*read-reg r*, *t*, *m'* :: (*'regval*, *'r*, *'exception*) *monad*) ∈ *Traces* **for** *t m'*
using *that assms*
by (*auto simp*: *read-reg-def non-cap-exp-def non-cap-reg-def elim*!: *Read-reg-TracesE split*: *option.splits*)
then show *?thesis*
unfolding *non-cap-exp-def*
by *blast*
qed

lemma
non-mem-exp-read-reg[*non-mem-expI*]: *non-mem-exp* (*read-reg r*) **and**
non-mem-exp-write-reg[*non-mem-expI*]: *non-mem-exp* (*write-reg r v*)
unfolding *non-mem-exp-def read-reg-def write-reg-def*
by (*auto elim*!: *Read-reg-TracesE Write-reg-TracesE split*: *option.splits*)

lemma *non-cap-exp-write-non-cap-reg*:
assumes *non-cap-reg r*
shows *non-cap-exp* (*write-reg r v*)
using *assms*
unfolding *write-reg-def*

by (*auto simp*: *non-cap-exp-def non-cap-reg-def elim*!: *Write-reg-TracesE*)

method *non-cap-expI* **uses** *simp* =
(*auto simp*: *simp intro*!: *non-cap-expI non-cap-exp-if non-cap-exp-read-non-cap-reg non-cap-exp-write-non-cap-reg*
split del: *if-split split*: *option.split sum.split prod.split*)

lemmas *non-mem-exp-combinators* =
non-mem-exp-bindI non-mem-exp-if non-mem-exp-let non-mem-exp-and-boolM non-mem-exp-or-boolM
non-mem-exp-foreachM non-mem-exp-try-catch non-mem-exp-catch-early-return non-mem-exp-liftR

method *non-mem-expI* **uses** *simp* =
(*auto simp*: *simp intro*!: *non-mem-expI non-mem-exp-combinators non-cap-expI*[*THEN non-cap-exp-non-mem-exp*]
split del: *if-split split*: *option.split sum.split prod.split*)

lemma *Run-write-reg-no-cap*[*trace-simp*]:
assumes *Run* (*write-reg r v*) *t a*
and *non-cap-reg r*
shows *run s t* = *s*
using *assms*
by (*cases s*) (*auto simp*: *write-reg-def step-defs non-cap-reg-def elim*!: *Write-reg-TracesE*)

lemma *Run-write-reg-run-gen*:
assumes *Run* (*write-reg r v*) *t a*
shows *run s t* =
$s(|$*written-regs* := *written-regs s* ∪
(*if* (∃ *c* ∈ *caps-of-regval ISA* (*regval-of r v*). *is-tagged-method CC c*)
then {*name r*} *else* {})$|)$
using *assms*
by (*cases s*) (*auto simp*: *write-reg-def step-defs elim*!: *Write-reg-TracesE*)

lemma *Run-read-non-cap-reg-run*[*trace-simp*]:
assumes *Run* (*read-reg r*) *t v*
and *non-cap-reg r*
shows *run s t* = *s*
using *assms*
by (*auto simp*: *step-defs non-cap-reg-def elim*!: *Run-read-regE*)

lemma *no-reg-writes-to-written-regs-run-inv*[*trace-simp*]:
assumes *Run m t a*
and *no-reg-writes-to UNIV m*
shows *written-regs* (*run s t*) = *written-regs s*
proof −
have *E-write-reg r v* ∉ *set t* **for** *r v*
using *assms*

by (*auto simp*: *no-reg-writes-to-def*)
then show *?thesis*
by (*induction t rule*: *rev-induct*) *auto*
qed

method *trace-enabledI* **uses** *simp elim* =
(*auto simp*: *simp trace-simp elim*!: *elim trace-elim*)

end

locale *Write-Cap-Automaton* = *Capability-ISA CC ISA*
for *CC* :: *'cap Capability-class* **and** *ISA* :: (*'cap*, *'regval*, *'instr*, *'e*) *isa* +
fixes *ex-traces* :: *bool* **and** *invocation-traces* :: *bool*
begin

fun *enabled* :: (*'cap*, *'regval*) *axiom-state* ⇒ *'regval event* ⇒ *bool* **where**
enabled s (*E-write-reg r v*) =
(∀ *c*. (*c* ∈ *caps-of-regval ISA v* ∧ *is-tagged-method CC c*)
⟶
(*c* ∈ *derivable* (*accessed-caps s*) ∨
(*c* ∈ *exception-targets ISA* (*read-from-KCC s*) ∧ *ex-traces* ∧ *r* ∈ *PCC ISA*)
∨
(∃ *cc cd*. *invocation-traces* ∧ *cc* ∈ *derivable* (*accessed-caps s*) ∧ *cd* ∈
derivable (*accessed-caps s*) ∧
invokable CC cc cd ∧ (*r* ∈ *PCC ISA* ∧ *leq-cap CC c* (*unseal CC cc*
True) ∨ *r* ∈ *IDC ISA* ∧ *leq-cap CC c* (*unseal CC cd True*)))))
| *enabled s* (*E-read-reg r v*) = (*r* ∈ *privileged-regs ISA* ⟶ (*system-reg-access s* ∨
ex-traces))
| *enabled s* (*E-write-memt - addr sz bytes tag -*) =
(∀ *c*. *cap-of-mem-bytes-method CC bytes tag* = *Some c* ∧ *is-tagged-method CC*
c ⟶ *c* ∈ *derivable* (*accessed-caps s*))
| *enabled s -* = *True*

lemma *enabled-E-write-reg-cases*:
assumes *enabled s* (*E-write-reg r v*)
and *c* ∈ *caps-of-regval ISA v*
and *is-tagged-method CC c*
obtains (*Derivable*) *c* ∈ *derivable* (*accessed-caps s*)
| (*KCC*) *c* ∈ *exception-targets ISA* (*read-from-KCC s*) **and** *ex-traces* **and**
r ∈ *PCC ISA* **and** *c* ∉ *derivable* (*accessed-caps s*)
| (*CCall*) *cc cd* **where** *invocation-traces* **and** *invokable CC cc cd* **and**
cc ∈ *derivable* (*accessed-caps s*) **and** *cd* ∈ *derivable* (*accessed-caps s*) **and**
r ∈ *PCC ISA* ∧ *leq-cap CC c* (*unseal CC cc True*) ∨ *r* ∈ *IDC ISA* ∧ *leq-cap*
CC c (*unseal CC cd True*) **and**
c ∉ *derivable* (*accessed-caps s*)
using *assms* **by** (*cases c* ∈ *derivable* (*accessed-caps s*)) *auto*

sublocale *Cap-Axiom-Automaton CC ISA enabled* **..**

lemma *non-cap-event-enabledI*:
  assumes *non-cap-event e*
  shows *enabled s e*
  using *assms*
  by (*elim non-cap-event.elims*) *auto*

lemma *non-cap-trace-enabledI*:
  assumes *non-cap-trace t*
  shows *trace-enabled s t*
  using *assms*
  by (*induction t*) (*auto simp*: *non-cap-event-enabledI non-cap-event-axiom-step-inv*)

lemma *non-cap-exp-trace-enabledI*:
  assumes *non-cap-exp m*
    and $(m, t, m') \in$ *Traces*
  shows *trace-enabled s t*
  by (*cases rule*: *non-cap-exp-Traces-cases*[*OF assms*])
    (*auto intro*: *non-cap-trace-enabledI simp*: *trace-enabled-append-iff*)

lemma *index-eq-some'*: (*index l n* = *Some x*) = ($n <$ *length l* $\wedge$ *l* ! $n = x$)
  by *auto*

lemma *recognises-store-cap-reg-read-reg-axioms*:
  assumes *t*: *accepts t*
  shows *store-cap-reg-axiom CC ISA ex-traces invocation-traces t*
    and *store-cap-mem-axiom CC ISA t*
    and *read-reg-axiom CC ISA ex-traces t*
proof −
  show *read-reg-axiom CC ISA ex-traces t*
    using *assms*
    unfolding *accepts-from-iff-all-enabled-final read-reg-axiom-def*
    by (*auto elim*!: *enabled.elims*)
  show *store-cap-reg-axiom CC ISA ex-traces invocation-traces t*
  proof (*unfold store-cap-reg-axiom-def*, *intro allI impI*, *goal-cases Idx*)
    case (*Idx i c r*)
    then show *?case*
    proof *cases*
      assume *i*: *i* < *length t*
      then obtain *v* **where** *e*: *index t i* = *Some* (*E-write-reg r v*)
        and *c*: $c \in$ *caps-of-regval ISA v*
        and *tag*: *is-tagged-method CC c*
        using *Idx*
        by (*cases t* ! *i*) *auto*
      then have *enabled* (*run initial* (*take i t*)) (*E-write-reg r v*)
        using *accepts-from-nth-enabledI*[*OF t i*]

```
      by auto
    from this c tag
    show ?thesis
    proof (cases rule: enabled-E-write-reg-cases)
      case Derivable
      then show ?thesis
        by (auto simp: cap-derivable-iff-derivable)
    next
      case KCC

      show ?thesis
        using  KCC
        unfolding index-eq-some'
        by (auto simp: cap-derivable-iff-derivable read-from-KCC-run-take-eq)
    next
      case (CCall cc cd)
      then show ?thesis
        by (auto simp: cap-derivable-iff-derivable)
    qed
  qed auto
 qed
 show store-cap-mem-axiom CC ISA t
  using assms
  unfolding accepts-from-iff-all-enabled-final store-cap-mem-axiom-def
  by (auto simp: cap-derivable-iff-derivable writes-mem-cap-Some-iff)
qed

end

locale Cap-Axiom-Inv-Automaton = Cap-Axiom-Automaton CC ISA enabled +
 State-Invariant get-regval set-regval invariant inv-regs
 for CC :: 'cap Capability-class and ISA :: ('cap, 'regval, 'instr, 'e) isa
  and enabled :: ('cap, 'regval) axiom-state ⇒ 'regval event ⇒ bool
  and get-regval :: string ⇒ 'regstate ⇒ 'regval option
  and set-regval :: string ⇒ 'regval ⇒ 'regstate ⇒ 'regstate option
  and invariant :: 'regstate ⇒ bool and inv-regs :: register-name set +
 fixes ex-traces :: bool
 assumes non-cap-event-enabled: ⋀e. non-cap-event e ⟹ enabled s e
  and read-non-special-regs-enabled: ⋀r v. r ∉ PCC ISA ∪ IDC ISA ∪ KCC ISA
∪ privileged-regs ISA ⟹ enabled s (E-read-reg r v)
begin

definition isException m ≡ ((∃ e. m = Exception e) ∨ (∃ msg. m = Fail msg)) ∧
ex-traces

definition finished :: ('regval,'a,'ex) monad ⇒ bool where
 finished m = ((∃ a. m = Done a) ∨ isException m)

lemma finishedE:
```

assumes *finished m*
obtains (*Done*) *a* **where** $m = Done\ a$
| (*Ex*) *isException m*
using *assms*
by (*auto simp*: *finished-def*)

lemma *finished-cases*:
assumes *finished m*
obtains (*Done*) *a* **where** $m = Done\ a$ | (*Fail*) *msg* **where** $m = Fail\ msg$ | (*Ex*) *e* **where** $m = Exception\ e$
using *assms*
by (*auto simp*: *finished-def isException-def*)

lemma *finished-Done*[*intro*, *simp*]:
finished (*Done a*)
by (*auto simp*: *finished-def*)

lemma *finished-Fail*[*intro*, *simp*]:
finished (*Fail msg*) $\Longrightarrow$ *finished* (*Fail msg'*)
unfolding *finished-def isException-def*
by *auto*

lemma *finished-Exception*[*intro*, *simp*]:
finished (*Exception e*) $\Longrightarrow$ *finished* (*Exception e'*)
unfolding *finished-def isException-def*
by *auto*

lemma *finished-isException*[*intro*, *simp*]:
isException m $\Longrightarrow$ *finished m*
by (*auto simp*: *finished-def*)

lemma *finished-bind-left*:
assumes *finished* ($m \gg= f$)
shows *finished m*
using *assms*
unfolding *finished-def isException-def*
by (*cases m*) *auto*

definition
traces-enabled m s regs $\equiv$
$\forall t\ m'.\ (m,\ t,\ m') \in Traces \wedge finished\ m' \wedge reads\text{-}regs\text{-}from\ inv\text{-}regs\ t\ regs \wedge invariant\ regs \longrightarrow trace\text{-}enabled\ s\ t$

lemma *traces-enabled-accepts-fromI*:
assumes *hasTrace t m* **and** *traces-enabled m s regs* **and** *hasException t m* $\vee$ *hasFailure t m* $\longrightarrow$ *ex-traces*
and *reads-regs-from inv-regs t regs* **and** *invariant regs*
shows *accepts-from s t*
using *assms*

unfolding *traces-enabled-def finished-def isException-def*
unfolding *hasTrace-iff-Traces-final hasException-iff-Traces-Exception hasFailure-iff-Traces-Fail*
unfolding *runTrace-iff-Traces*[*symmetric*]
by (*intro trace-enabled-acceptI*) (*auto elim*!: *final-cases*)

named-theorems *traces-enabledI*

lemma *traces-enabled-bind*[*traces-enabledI*]:
assumes *runs-preserve-invariant m* **and** *traces-enabled m s regs*
and $\bigwedge$*t a. Run-inv m t a regs* $\Longrightarrow$ *traces-enabled* (*f a*) (*run s t*) (*the* (*updates-regs inv-regs t regs*))
shows *traces-enabled* (*m* $\gg=$ *f*) *s regs*
using *assms*
by (*auto simp*: *traces-enabled-def Run-inv-def regstate-simp trace-enabled-append-iff*
dest!: *finished-bind-left elim*!: *bind-Traces-cases elim*!: *runs-preserve-invariantE*; *fastforce*)

lemma *non-cap-trace-enabledI*:
assumes *non-cap-trace t*
shows *trace-enabled s t*
using *assms*
by (*induction t*) (*auto simp*: *non-cap-event-enabled non-cap-event-axiom-step-inv*)

lemma *non-cap-exp-trace-enabledI*:
assumes *m*: *non-cap-exp m*
and *t*: (*m*, *t*, *m'*) $\in$ *Traces*
shows *trace-enabled s t*
by (*cases rule*: *non-cap-exp-Traces-cases*[*OF m t*])
(*auto intro*: *non-cap-trace-enabledI read-non-special-regs-enabled simp*: *trace-enabled-append-iff*)

lemma *non-cap-exp-traces-enabledI*:
assumes *non-cap-exp m*
shows *traces-enabled m s regs*
using *assms*
by (*auto simp*: *traces-enabled-def intro*: *non-cap-exp-trace-enabledI*)

lemma *Run-inv-RunI*[*simp*]: *Run-inv m t a regs* $\Longrightarrow$ *Run m t a*
by (*simp add*: *Run-inv-def*)

lemma *traces-enabled-let*[*traces-enabledI*]:
assumes *traces-enabled* (*f y*) *s regs*
shows *traces-enabled* (*let x* = *y in f x*) *s regs*
using *assms*
by *auto*

lemma *traces-enabled-case-prod*[*traces-enabledI*]:
assumes $\bigwedge$*x y. z* = (*x*, *y*) $\Longrightarrow$ *traces-enabled* (*f x y*) *s regs*
shows *traces-enabled* (*case z of* (*x*, *y*) $\Rightarrow$ *f x y*) *s regs*
using *assms*

by *auto*

lemma *traces-enabled-if*[*traces-enabledI*]:
assumes $c \Longrightarrow$ *traces-enabled m1 s regs* **and** $\neg c \Longrightarrow$ *traces-enabled m2 s regs*
shows *traces-enabled* (*if c then m1 else m2*) *s regs*
using *assms*
by *auto*

lemma *traces-enabled-if-ignore-cond*:
assumes *traces-enabled m1 s regs* **and** *traces-enabled m2 s regs*
shows *traces-enabled* (*if c then m1 else m2*) *s regs*
using *assms*
by *auto*

lemma *traces-enabled-and-boolM*[*traces-enabledI*]:
assumes *runs-preserve-invariant m1* **and** *traces-enabled m1 s regs*
and $\bigwedge$*t*. *Run-inv m1 t True regs* $\Longrightarrow$ *traces-enabled m2* (*run s t*) (*the* (*updates-regs inv-regs t regs*))
shows *traces-enabled* (*and-boolM m1 m2*) *s regs*
using *assms*
by (*auto simp*: *and-boolM-def intro*!: *traces-enabledI intro*: *non-cap-exp-traces-enabledI non-cap-expI*)

lemma *traces-enabled-or-boolM*[*traces-enabledI*]:
assumes *runs-preserve-invariant m1* **and** *traces-enabled m1 s regs*
and $\bigwedge$*t*. *Run-inv m1 t False regs* $\Longrightarrow$ *traces-enabled m2* (*run s t*) (*the* (*updates-regs inv-regs t regs*))
shows *traces-enabled* (*or-boolM m1 m2*) *s regs*
using *assms*
by (*auto simp*: *or-boolM-def intro*!: *traces-enabledI intro*: *non-cap-exp-traces-enabledI non-cap-expI*)

lemma *traces-enabled-foreachM-inv*:
assumes $\bigwedge$*x vars s regs*. *P vars s regs* $\Longrightarrow$ *x* $\in$ *set xs* $\Longrightarrow$ *traces-enabled* (*body x vars*) *s regs*
and $\bigwedge$*x vars*. *x* $\in$ *set xs* $\Longrightarrow$ *runs-preserve-invariant* (*body x vars*)
and $\bigwedge$*x vars s regs t vars'*. *P vars s regs* $\Longrightarrow$ *x* $\in$ *set xs* $\Longrightarrow$ *Run-inv* (*body x vars*) *t vars' regs* $\Longrightarrow$ *P vars'* (*run s t*) (*the* (*updates-regs inv-regs t regs*))
and *P vars s regs*
shows *traces-enabled* (*foreachM xs vars body*) *s regs*
by (*use assms* **in** ‹*induction xs arbitrary*: *vars s regs*›;
fastforce intro!: *traces-enabledI intro*: *non-cap-exp-traces-enabledI non-cap-expI*)

lemma *traces-enabled-try-catch*:
assumes *traces-enabled m s regs*
and $\bigwedge$*tm e th m'*.
(*m*, *tm*, *Exception e*) $\in$ *Traces* $\Longrightarrow$ (*h e*, *th*, *m'*) $\in$ *Traces* $\Longrightarrow$ *finished m'* $\Longrightarrow$
reads-regs-from inv-regs (*tm* @ *th*) *regs* $\Longrightarrow$ *invariant regs* $\Longrightarrow$

trace-enabled s (*tm* @ *th*)
shows *traces-enabled* (*try-catch m h*) *s regs*
proof −

have ∗: *finished* (*try-catch m h*) ⟷ (∃ *a*. *m* = *Done a*) ∨ (∃ *msg*. *m* = *Fail msg* ∧ *finished m*) ∨ (∃ *e*. *m* = *Exception e* ∧ (*h e*, [], *h e*) ∈ *Traces* ∧ *finished* (*h e*))
for *m*
by (*cases m*) (*auto simp*: *finished-def isException-def*)
show *?thesis*
using *assms*
by (*fastforce simp*: *traces-enabled-def regstate-simp trace-enabled-append-iff Run-inv-def* ∗
elim!: *try-catch-Traces-cases elim*: *traces-preserve-invariantE*)
qed

lemma *traces-enabled-liftR*[*traces-enabledI*]:
assumes *traces-enabled m s regs*
shows *traces-enabled* (*liftR m*) *s regs*
using *assms*
unfolding *liftR-def*
by (*intro traces-enabled-try-catch*) (*auto simp*: *traces-enabled-def Run-inv-def*)

definition
early-returns-enabled m s regs ≡
traces-enabled m s regs ∧
(∀ *t a*. (*m*, *t*, *Exception* (*Inl a*)) ∈ *Traces* ∧ *reads-regs-from inv-regs t regs* ∧ *invariant regs* ⟶ *trace-enabled s t*)

lemma *traces-enabled-catch-early-return*[*traces-enabledI*]:
assumes *early-returns-enabled m s regs*
shows *traces-enabled* (*catch-early-return m*) *s regs*
using *assms*
unfolding *catch-early-return-def*
by (*intro traces-enabled-try-catch*)
(*auto simp*: *traces-enabled-def early-returns-enabled-def Run-inv-def split*: *sum.splits*)

lemma *liftR-no-early-return*[*simp*]:
shows (*liftR m*, *t*, *Exception* (*Inl e*)) ∈ *Traces* ⟷ *False*
by (*induction m arbitrary*: *t*) (*auto simp*: *liftR-def elim*: *Traces-cases*)

lemma *early-returns-enabled-liftR*[*traces-enabledI*]:
assumes *traces-enabled m s regs*
shows *early-returns-enabled* (*liftR m*) *s regs*
using *assms*
by (*auto simp*: *early-returns-enabled-def intro*: *traces-enabled-liftR*)

lemma *early-returns-enabled-return*[*traces-enabledI*]:
early-returns-enabled (*return a*) *s regs*

by (*auto simp*: *early-returns-enabled-def traces-enabled-def*)

lemma *early-returns-enabled-bind*[*traces-enabledI*]:
assumes *inv*: *traces-preserve-invariant m*
and *m*: *early-returns-enabled m s regs*
and *f*: $\bigwedge t\ a$. *Run-inv m t a regs* $\Longrightarrow$ *early-returns-enabled* (*f a*) (*run s t*) (*the* (*updates-regs inv-regs t regs*))
shows *early-returns-enabled* (*m* $\gg=$ *f*) *s regs*
proof −
{ fix *t a*
assume (*m* $\gg=$ *f*, *t*, *Exception* (*Inl a*)) $\in$ *Traces* **and** *t*: *reads-regs-from inv-regs t regs* **and** *regs*: *invariant regs*
then have *trace-enabled s t*
proof (*cases rule*: *bind-Traces-cases*)
case (*Left m''*)
then consider *m''* = *Exception* (*Inl a*) | *a'* **where** *m''* = *Done a'* **and** *f a'* = *Exception* (*Inl a*)
by (*cases m''*) *auto*
then show *?thesis*
using *Left m t regs*
by *cases* (*auto simp*: *early-returns-enabled-def traces-enabled-def*)
next
case (*Bind tm am tf*)
then obtain *regs'*
where *updates-regs inv-regs tm regs* = *Some regs'* **and** *invariant regs'*
and *reads-regs-from inv-regs tm regs* **and** *reads-regs-from inv-regs tf regs'*
using *t regs*
by (*elim traces-preserve-invariantE*[*OF inv*]) (*auto simp*: *regstate-simp*)
then show *?thesis*
using *Bind m f*[*of tm am*] *regs*
by (*auto simp*: *trace-enabled-append-iff early-returns-enabled-def traces-enabled-def Run-inv-def*)
qed
}
then show *?thesis*
using *assms*
by (*auto intro*: *traces-enabled-bind traces-runs-preserve-invariantI simp*: *early-returns-enabled-def*)
qed

lemma *early-returns-enabled-early-return*[*traces-enabledI*]:
early-returns-enabled (*early-return a*) *s regs*
by (*auto simp*: *early-returns-enabled-def early-return-def throw-def traces-enabled-def*)

lemma *early-returns-enabled-let*[*traces-enabledI*]:
assumes *early-returns-enabled* (*f y*) *s regs*
shows *early-returns-enabled* (*let x* = *y in f x*) *s regs*
using *assms*
by *auto*

lemma *early-returns-enabled-case-prod*[*traces-enabledI*]:
assumes $\bigwedge x\ y.\ z = (x,\ y) \Longrightarrow$ *early-returns-enabled* (*f x y*) *s regs*
shows *early-returns-enabled* (*case z of* (*x*, *y*) ⇒ *f x y*) *s regs*
using *assms*
by *auto*

lemma *early-returns-enabled-if*[*traces-enabledI*]:
assumes $c \Longrightarrow$ *early-returns-enabled m1 s regs* **and** $\neg c \Longrightarrow$ *early-returns-enabled m2 s regs*
shows *early-returns-enabled* (*if c then m1 else m2*) *s regs*
using *assms*
by *auto*

lemma *early-returns-enabled-if-ignore-cond*:
assumes *early-returns-enabled m1 s regs* **and** *early-returns-enabled m2 s regs*
shows *early-returns-enabled* (*if c then m1 else m2*) *s regs*
using *assms*
by *auto*

lemma *early-returns-enabled-and-boolM*[*traces-enabledI*]:
assumes *traces-preserve-invariant m1* **and** *early-returns-enabled m1 s regs*
and $\bigwedge t.$ *Run-inv m1 t True regs* $\Longrightarrow$ *early-returns-enabled m2* (*run s t*) (*the* (*updates-regs inv-regs t regs*))
shows *early-returns-enabled* (*and-boolM m1 m2*) *s regs*
using *assms*
by (*auto simp*: *and-boolM-def intro*!: *traces-enabledI intro*: *non-cap-exp-traces-enabledI non-cap-expI*)

lemma *early-returns-enabled-or-boolM*[*traces-enabledI*]:
assumes *traces-preserve-invariant m1* **and** *early-returns-enabled m1 s regs*
and $\bigwedge t.$ *Run-inv m1 t False regs* $\Longrightarrow$ *early-returns-enabled m2* (*run s t*) (*the* (*updates-regs inv-regs t regs*))
shows *early-returns-enabled* (*or-boolM m1 m2*) *s regs*
using *assms*
by (*auto simp*: *or-boolM-def intro*!: *traces-enabledI intro*: *non-cap-exp-traces-enabledI non-cap-expI*)

lemma *early-returns-enabled-foreachM-inv*:
assumes $\bigwedge x$ *vars s regs*. *P vars s regs* $\Longrightarrow x \in$ *set xs* $\Longrightarrow$ *early-returns-enabled* (*body x vars*) *s regs*
and $\bigwedge x$ *vars*. $x \in$ *set xs* $\Longrightarrow$ *traces-preserve-invariant* (*body x vars*)
and $\bigwedge x$ *vars s regs t vars'*. *P vars s regs* $\Longrightarrow x \in$ *set xs* $\Longrightarrow$ *Run-inv* (*body x vars*) *t vars' regs* $\Longrightarrow$ *P vars'* (*run s t*) (*the* (*updates-regs inv-regs t regs*))
and *P vars s regs*
shows *early-returns-enabled* (*foreachM xs vars body*) *s regs*
by (*use assms* **in** ‹*induction xs arbitrary*: *vars s regs*›;
fastforce intro!: *traces-enabledI intro*: *non-cap-exp-traces-enabledI non-cap-expI*)

lemma *non-cap-exp-Run-inv-traces-enabled-runE*:

assumes *Run-inv m1 t a regs* **and** *non-cap-exp m1* **and** *traces-enabled m2 s regs'*
shows *traces-enabled m2 (run s t) regs'*
using *assms*
by (*auto simp*: *Run-inv-def non-cap-exp-Run-run-invI*)

lemma *no-reg-writes-Run-inv-traces-enabled-updates-regsE*:
assumes *Run-inv m1 t a regs* **and** *no-reg-writes-to inv-regs m1* **and** *traces-enabled m2 s regs*
shows *traces-enabled m2 s (the (updates-regs inv-regs t regs))*
using *assms*
by (*auto simp*: *Run-inv-def*)

lemma *non-cap-exp-Run-inv-early-returns-enabled-runE*:
assumes *Run-inv m1 t a regs* **and** *non-cap-exp m1* **and** *early-returns-enabled m2 s regs'*
shows *early-returns-enabled m2 (run s t) regs'*
using *assms*
by (*auto simp*: *Run-inv-def non-cap-exp-Run-run-invI*)

lemma *no-reg-writes-Run-inv-early-returns-enabled-updates-regsE*:
assumes *Run-inv m1 t a regs* **and** *no-reg-writes-to inv-regs m1* **and** *early-returns-enabled m2 s regs*
shows *early-returns-enabled m2 s (the (updates-regs inv-regs t regs))*
using *assms*
by (*auto simp*: *Run-inv-def*)

lemma *accessible-regs-no-writes-run*:
assumes *t*: *Run m t a*
and *m*: *runs-no-reg-writes-to* $\{r\}$ *m*
and *s*: $r \in$ *accessible-regs s*
shows $r \in$ *accessible-regs (run s t)*
proof −
have *no-write*: $\forall v$. *E-write-reg r v* $\notin$ *set t*
using *m t*
by (*auto simp*: *runs-no-reg-writes-to-def Run-inv-def*)
show *?thesis*
proof (*use s no-write* **in** ‹*induction t arbitrary*: *s*›)
case (*Cons e t*)
then have $r \in$ *accessible-regs (axiom-step s e)* **and** $\forall v$. *E-write-reg r v* $\notin$ *set t*
by (*auto simp*: *accessible-regs-def*)
from *Cons.IH[OF this]* **show** *?case* **by** *auto*
qed *auto*
qed

lemma *no-reg-writes-to-mono*:
assumes *runs-no-reg-writes-to Rs m*
and *Rs'* $\subseteq$ *Rs*
shows *runs-no-reg-writes-to Rs' m*
using *assms*

by (*auto simp*: *runs-no-reg-writes-to-def*)

lemma *accessible-regs-no-writes-run-subset*:
assumes *t*: *Run m t a* **and** *m*: *runs-no-reg-writes-to Rs m*
and *Rs*: $Rs \subseteq$ *accessible-regs s*
shows $Rs \subseteq$ *accessible-regs* (*run s t*)
using *t Rs no-reg-writes-to-mono*[*OF m*]
by (*auto intro*: *accessible-regs-no-writes-run*)

lemma *accessible-regs-no-writes-run-inv-subset*:
assumes *t*: *Run-inv m t a regs* **and** *m*: *runs-no-reg-writes-to Rs m*
and *Rs*: $Rs \subseteq$ *accessible-regs s*
shows $Rs \subseteq$ *accessible-regs* (*run s t*)
using *assms*
by (*intro accessible-regs-no-writes-run-subset*) (*auto simp*: *Run-inv-def*)

named-theorems *accessible-regsE*
named-theorems *accessible-regsI*

method *accessible-regs-step* **uses** *simp assms* =
((*erule accessible-regsE eqTrueE*)
| (*rule accessible-regsI preserves-invariantI TrueI*)
| (*erule accessible-regs-no-writes-run-inv-subset accessible-regs-no-writes-run-subset*,
solves ‹*use assms in* ‹*no-reg-writes-toI simp*: *simp*››))

method *accessible-regsI-with* **methods** *solve* **uses** *simp assms* =
((*erule accessible-regsE eqTrueE*; *accessible-regsI-with solve simp*: *simp assms*: *assms*)
| (*rule accessible-regsI preserves-invariantI TrueI*; *accessible-regsI-with solve simp*: *simp assms*: *assms*)
| (*erule accessible-regs-no-writes-run-inv-subset accessible-regs-no-writes-run-subset*,
solves ‹*use assms in* ‹*no-reg-writes-toI simp*: *simp*››,
accessible-regsI-with solve simp: *simp assms*: *assms*)
| *solve*)

method *accessible-regsI* **uses** *simp assms* =
(*accessible-regsI-with*
‹(*use assms in* ‹*no-reg-writes-toI simp*: *simp*›)
| (*use assms in* ‹*auto simp*: *simp*›)›
simp: *simp assms*: *assms*)

definition *derivable-caps s* $\equiv$ {*c*. *is-tagged-method CC c* $\longrightarrow$ *c* $\in$ *derivable* (*accessed-caps s*)}

named-theorems *derivable-capsI*
named-theorems *derivable-capsE*

lemma *accessed-caps-run-mono*:
accessed-caps s ⊆ *accessed-caps* (*run s t*)
by (*rule subsetI*) (*induction t arbitrary*: *s*; *auto*)

lemma *derivable-caps-run-mono*:
derivable-caps s ⊆ *derivable-caps* (*run s t*)
using *derivable-mono*[*OF accessed-caps-run-mono*]
by (*auto simp*: *derivable-caps-def*)

lemma *derivable-caps-run-imp*:
c ∈ *derivable-caps s* ⟹ *c* ∈ *derivable-caps* (*run s t*)
using *derivable-caps-run-mono*
by *auto*

method *derivable-caps-step* =
(*rule derivable-capsI preserves-invariantI TrueI*
| *erule derivable-capsE eqTrueE*
| *rule derivable-caps-run-imp*)

method *derivable-capsI-with* **methods** *solve* **uses** *simp assms* =
((*rule derivable-capsI preserves-invariantI TrueI*
| *erule derivable-capsE eqTrueE*
| *rule derivable-caps-run-imp*
| *solve*);
derivable-capsI-with solve simp: *simp assms*: *assms*)

method *derivable-capsI* **uses** *simp assms* =
(*derivable-capsI-with* ‹*solves* ‹*accessible-regsI simp*: *simp assms*: *assms*›› *simp*: *simp assms*: *assms*)

method *try-simp-traces-enabled* =
((*match* **conclusion in** ‹*traces-enabled m2* (*run s t*) (*the* (*updates-regs inv-regs t regs*))› **for** *m2 s t regs* ⇒
‹*match premises in m1*: ‹*Run-inv m1 t a regs*› *for m1 a* ⇒
‹(*rule non-cap-exp-Run-inv-traces-enabled-runE*[*OF m1*], *solves* ‹*non-cap-expI*›)?,
(*rule no-reg-writes-Run-inv-traces-enabled-updates-regsE*[*OF m1*], *solves* ‹*no-reg-writes-toI*›)?››
| ‹*early-returns-enabled m2* (*run s t*) (*the* (*updates-regs inv-regs t regs*))› **for** *m2 s t regs* ⇒
‹*match premises in m1*: ‹*Run-inv m1 t a regs*› *for m1 a* ⇒
‹(*rule non-cap-exp-Run-inv-early-returns-enabled-runE*[*OF m1*], *solves* ‹*non-cap-expI*›)?,
(*rule no-reg-writes-Run-inv-early-returns-enabled-updates-regsE*[*OF m1*], *solves* ‹*no-reg-writes-toI*›)?››)?)

named-theorems *traces-enabled-combinatorI*

lemmas *traces-enabled-builtin-combinatorsI* =
traces-enabled-bind traces-enabled-and-boolM traces-enabled-or-boolM
early-returns-enabled-bind early-returns-enabled-and-boolM early-returns-enabled-or-boolM

named-theorems *traces-enabled-split*
declare *option.split*[**where** $P = \lambda m$. *traces-enabled m s regs* **for** *s regs*, *traces-enabled-split*]
declare *prod.split*[**where** $P = \lambda m$. *traces-enabled m s regs* **for** *s regs*, *traces-enabled-split*]

method *traces-enabled-step* **uses** *intro elim* =
((*rule intro TrueI*)
| (*erule elim eqTrueE*)
| ((*rule traces-enabled-combinatorI traces-enabled-builtin-combinatorsI*[*rotated 2*], *try-simp-traces-enabled*))
| (*rule traces-enabledI preserves-invariantI*)
| (*rule traces-enabled-split*[*THEN iffD2*]; *intro conjI impI*))

method *traces-enabledI-with* **methods** *solve* **uses** *intro elim* =
((*rule intro TrueI*; *traces-enabledI-with solve intro*: *intro elim*: *elim*)
| (*erule elim eqTrueE*; *traces-enabledI-with solve intro*: *intro elim*: *elim*)
| ((*rule traces-enabled-combinatorI traces-enabled-builtin-combinatorsI*[*rotated 2*], *try-simp-traces-enabled*); *traces-enabledI-with solve intro*: *intro elim*: *elim*)
| (*rule traces-enabledI*; *traces-enabledI-with solve intro*: *intro elim*: *elim*)
| (*preserves-invariantI intro*: *intro elim*: *elim*; *traces-enabledI-with solve intro*: *intro elim*: *elim*)
| (*rule traces-enabled-split*[*THEN iffD2*]; *intro conjI impI*; *traces-enabledI-with solve intro*: *intro elim*: *elim*)
| *solve*)

method *traces-enabledI* **uses** *simp intro elim assms* =
((*traces-enabled-step intro*: *intro elim*: *elim*; *traces-enabledI simp*: *simp intro*: *intro elim*: *elim assms*: *assms*)
| (*accessible-regs-step simp*: *simp assms*: *assms*; *solves* ‹*traces-enabledI simp*: *simp intro*: *intro elim*: *elim assms*: *assms*›)
| (*derivable-caps-step*; *solves* ‹*traces-enabledI simp*: *simp intro*: *intro elim*: *elim assms*: *assms*›)
| (*solves* ‹*no-reg-writes-toI simp*: *simp*›)
| (*solves* ‹*preserves-invariantI simp*: *simp*›)
| (*use assms* **in** ‹*auto intro*!: *intro elim*!: *elim simp*: *simp*›)?)

lemma *if-derivable-capsI*[*derivable-capsI*]:
assumes *cond* $\Longrightarrow$ *c1* $\in$ *derivable-caps s* **and** $\neg$*cond* $\Longrightarrow$ *c2* $\in$ *derivable-caps s*
shows (*if cond then c1 else c2*) $\in$ *derivable-caps s*
using *assms*
by *auto*

end

locale *Write-Cap-Inv-Automaton* =

Write-Cap-Automaton CC ISA ex-traces invocation-traces +
State-Invariant get-regval set-regval invariant inv-regs
for *CC* :: *'cap Capability-class* **and** *ISA* :: (*'cap*, *'regval*, *'instr*, *'e*) *isa*
and *ex-traces* :: *bool* **and** *invocation-traces* :: *bool*
and *get-regval* :: *string* ⇒ *'regstate* ⇒ *'regval option*
and *set-regval* :: *string* ⇒ *'regval* ⇒ *'regstate* ⇒ *'regstate option*
and *invariant* :: *'regstate* ⇒ *bool* **and** *inv-regs* :: *register-name set*
begin

sublocale *Cap-Axiom-Inv-Automaton* **where** *enabled* = *enabled*
proof
fix *s e*
assume *non-cap-event e*
then show *enabled s e*
by (*cases e*) *auto*
next
fix *s r v*
assume *r* ∉ *special-reg-names*
then show *enabled s* (*E-read-reg r v*)
by *auto*
qed

lemma *read-reg-trace-enabled*:
assumes *t*: (*read-reg r*, *t*, *m'*) ∈ *Traces*
and *r*: *name r* ∈ *privileged-regs ISA* ⟶ *system-reg-access s* ∨ *ex-traces*
shows *trace-enabled s t*
by (*use t* **in** ‹*auto simp*: *read-reg-def elim*!: *Read-reg-TracesE split*: *option.splits*›)
(*use r* **in** ‹*auto*›)

lemma *traces-enabled-read-reg*:
assumes *name r* ∈ *privileged-regs ISA* ⟶ (*system-reg-access s* ∨ *ex-traces*)
shows *traces-enabled* (*read-reg r*) *s regs*
using *assms*
unfolding *traces-enabled-def*
by (*blast intro*: *read-reg-trace-enabled*)

lemma *write-reg-trace-enabled*:
assumes (*write-reg r v*, *t*, *m'*) ∈ *Traces*
and *enabled s* (*E-write-reg* (*name r*) (*regval-of r v*))
shows *trace-enabled s t*
using *assms*
by (*auto simp add*: *write-reg-def simp del*: *enabled.simps elim*!: *Write-reg-TracesE*)

lemma *traces-enabled-write-reg*:
assumes *enabled s* (*E-write-reg* (*name r*) (*regval-of r v*))
shows *traces-enabled* (*write-reg r v*) *s regs*
using *assms*
unfolding *traces-enabled-def*
by (*blast intro*: *write-reg-trace-enabled*)

lemma *traces-enabled-reg-axioms*:
  assumes *traces-enabled m initial regs* **and** *hasTrace t m*
    and *reads-regs-from inv-regs t regs* **and** *invariant regs*
    and *hasException t m* $\vee$ *hasFailure t m* $\longrightarrow$ *ex-traces*
  shows *store-cap-reg-axiom CC ISA ex-traces invocation-traces t*
    and *store-cap-mem-axiom CC ISA t*
    and *read-reg-axiom CC ISA ex-traces t*
  using *assms*
  by (*intro recognises-store-cap-reg-read-reg-axioms*;
      elim traces-enabled-accepts-fromI[**where** *regs* = *regs*];
      auto)+

end

locale *Capability-ISA-Fixed-Translation* = *Capability-ISA CC ISA*
  for *CC* :: *'cap Capability-class* **and** *ISA* :: (*'cap*, *'regval*, *'instr*, *'e*) *isa* +
  fixes *translation-assm* :: *'regval trace* $\Rightarrow$ *bool*
  assumes *fixed-translation-tables*: $\bigwedge$*i t*. *translation-assm t* $\Longrightarrow$ *translation-tables ISA* (*take i t*) = *translation-tables ISA* []
    and *fixed-translation*: $\bigwedge$*i t addr load*. *translation-assm t* $\Longrightarrow$ *translate-address ISA addr load* (*take i t*) = *translate-address ISA addr load* []

fun *non-store-event* :: *'regval event* $\Rightarrow$ *bool* **where**
  non-store-event (*E-write-mem - paddr sz v -*) = *False*
| *non-store-event* (*E-write-memt - paddr sz v tag -*) = *False*
| *non-store-event - = True*

abbreviation *non-store-trace* :: *'regval trace* $\Rightarrow$ *bool* **where**
  non-store-trace t $\equiv$ ($\forall$ *e* $\in$ *set t*. *non-store-event e*)

lemma (**in** *Cap-Axiom-Automaton*) *non-mem-trace-mem-axiomsI*:
  assumes *non-mem-trace t*
  shows *store-mem-axiom CC ISA t* **and** *store-tag-axiom CC ISA t* **and** *load-mem-axiom CC ISA is-fetch t*
proof −
  have *i*: *non-mem-event* (*t* ! *i*) **if** *i* < *length t* **for** *i*
    using *assms that*
    by (*auto simp*: *non-mem-trace-def*)
  show *store-mem-axiom CC ISA t*
    using *i*
    by (*fastforce simp*: *store-mem-axiom-def writes-mem-val-at-idx-def bind-eq-Some-conv elim*!: *writes-mem-val.elims*)
  show *store-tag-axiom CC ISA t*
    using *i*
    by (*fastforce simp*: *store-tag-axiom-def writes-mem-val-at-idx-def bind-eq-Some-conv elim*!: *writes-mem-val.elims*)
  show *load-mem-axiom CC ISA is-fetch t*
    using *i*

by (*fastforce simp*: *load-mem-axiom-def reads-mem-val-at-idx-def bind-eq-Some-conv elim*!: *reads-mem-val.elims*)
qed

locale *Mem-Automaton* = *Capability-ISA-Fixed-Translation* **where** *CC* = *CC* **and** *ISA* = *ISA*
for *CC* :: *'cap Capability-class* **and** *ISA* :: (*'cap*, *'regval*, *'instr*, *'e*) *isa* +
fixes *is-fetch* :: *bool*
begin

definition *paddr-in-mem-region* :: *'cap* ⇒ *acctype* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
paddr-in-mem-region c acctype paddr sz =
(∃ *vaddr*. *set* (*address-range vaddr sz*) ⊆ *get-mem-region-method CC c* ∧
translate-address ISA vaddr acctype [] = *Some paddr*)

definition *has-access-permission* :: *perms* ⇒ *acctype* ⇒ *bool* ⇒ *bool* ⇒ *bool* **where**
has-access-permission perms acctype is-cap is-local-cap =
(*case acctype of*
Fetch ⇒ *permit-execute perms*
| *Load* ⇒ *permit-load perms* ∧ (*is-cap* ⟶ *permit-load-capability perms*)
| *Store* ⇒ *permit-store perms* ∧ (*is-cap* ⟶ *permit-store-capability perms*) ∧ (*is-local-cap* ⟶ *permit-store-local-capability perms*))

definition *authorises-access* :: *'cap* ⇒ *acctype* ⇒ *bool* ⇒ *bool* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
authorises-access c acctype is-cap is-local-cap paddr sz =
(*is-tagged-method CC c* ∧ ¬*is-sealed-method CC c* ∧ *paddr-in-mem-region c acctype paddr sz* ∧
has-access-permission (*get-perms-method CC c*) *acctype is-cap is-local-cap*)

definition *access-enabled* :: (*'cap*, *'regval*) *axiom-state* ⇒ *acctype* ⇒ *nat* ⇒ *nat* ⇒ *memory-byte list* ⇒ *bitU* ⇒ *bool* **where**
access-enabled s acctype paddr sz v tag =
((*tag* ≠ *B0* ⟶ *address-tag-aligned ISA paddr* ∧ *sz* = *tag-granule ISA*) ∧
(*case acctype of Load* ⇒ *True*
| *Store* ⇒ (*tag* = *B0* ∨ *tag* = *B1*) ∧ *length v* = *sz*
| *Fetch* ⇒ *tag* = *B0*) ∧
(*paddr* ∈ *translation-tables ISA* [] ∨
(∃ *c'* ∈ *derivable* (*accessed-caps s*).
let is-cap = *tag* ≠ *B0 in*
let is-local-cap = *mem-val-is-local-cap CC ISA v tag* ∧ *tag* = *B1 in*
authorises-access c' acctype is-cap is-local-cap paddr sz)))

lemmas *access-enabled-defs* = *access-enabled-def authorises-access-def paddr-in-mem-region-def has-access-permission-def*

fun *enabled* :: (*'cap*, *'regval*) *axiom-state* ⇒ *'regval event* ⇒ *bool* **where**
enabled s (*E-write-mem - paddr sz v -*) = *access-enabled s Store paddr sz v B0*
| *enabled s* (*E-write-memt - paddr sz v tag -*) = *access-enabled s Store paddr sz v*

tag
| *enabled s* (*E-read-mem* - *paddr sz v*) = *access-enabled s* (*if is-fetch then Fetch else Load*) *paddr sz v B0*
| *enabled s* (*E-read-memt* - *paddr sz v-tag*) = *access-enabled s* (*if is-fetch then Fetch else Load*) *paddr sz* (*fst v-tag*) (*snd v-tag*)
| *enabled s* - = *True*

sublocale *Cap-Axiom-Automaton* **where** *enabled* = *enabled* **..**

lemma *accepts-store-mem-axiom*:
  assumes ∗: *translation-assm t* **and** ∗∗: *accepts t*
  shows *store-mem-axiom CC ISA t*
  using *accepts-from-nth-enabledI*[*OF* ∗∗]
  unfolding *store-mem-axiom-def*
  unfolding *writes-mem-val-at-idx-def cap-derivable-iff-derivable*
  unfolding *fixed-translation-tables*[*OF* ∗] *fixed-translation*[*OF* ∗]
 by (*fastforce simp*: *access-enabled-defs bind-eq-Some-conv elim*!: *writes-mem-val.elims*)

lemma *accepts-store-tag-axiom*:
  assumes *accepts t*
  shows *store-tag-axiom CC ISA t*
  using *accepts-from-nth-enabledI*[*OF assms*]
  unfolding *store-tag-axiom-def writes-mem-val-at-idx-def*
 by (*fastforce simp*: *access-enabled-defs bind-eq-Some-conv elim*!: *writes-mem-val.elims*)

lemma *accepts-load-mem-axiom*:
  assumes ∗: *translation-assm t* **and** ∗∗: *accepts t*
  shows *load-mem-axiom CC ISA is-fetch t*
  unfolding *load-mem-axiom-def*
  unfolding *reads-mem-val-at-idx-def cap-derivable-iff-derivable*
  unfolding *fixed-translation-tables*[*OF* ∗] *fixed-translation*[*OF* ∗]
 by (*auto simp*: *bind-eq-Some-conv elim*!: *reads-mem-val.elims dest*!: *accepts-from-nth-enabledI*[*OF* ∗∗] *split del*: *if-split*;
      cases is-fetch; *fastforce simp*: *access-enabled-defs*)

lemma *non-mem-event-enabledI*:
  enabled s e **if** *non-mem-event e*
  using *that*
  by (*auto elim*: *non-mem-event.elims*)

lemma *non-mem-trace-enabledI*:
  trace-enabled s t **if** *non-mem-trace t*
  using *that*
  by (*induction t arbitrary*: *s*) (*auto intro*: *non-mem-event-enabledI*)

end

locale *Mem-Inv-Automaton* =
  Mem-Automaton translation-assm CC ISA is-fetch +

State-Invariant get-regval set-regval invariant inv-regs
for *CC* :: *'cap Capability-class* **and** *ISA* :: (*'cap*, *'regval*, *'instr*, *'e*) *isa*
and *translation-assm* :: *'regval event list* ⇒ *bool*
and *is-fetch* :: *bool* **and** *ex-traces* :: *bool*
and *get-regval* :: *string* ⇒ *'regstate* ⇒ *'regval option*
and *set-regval* :: *string* ⇒ *'regval* ⇒ *'regstate* ⇒ *'regstate option*
and *invariant* :: *'regstate* ⇒ *bool* **and** *inv-regs* :: *register-name set*
begin

sublocale *Cap-Axiom-Inv-Automaton* **where** *enabled* = *enabled* **and** *ex-traces* = *ex-traces*
proof
fix *s e*
assume *non-cap-event e*
then show *enabled s e*
by (*cases e*) *auto*
next
fix *s r v*
assume *r* ∉ *special-reg-names*
then show *enabled s* (*E-read-reg r v*)
by *auto*
qed

lemma *non-mem-exp-trace-enabledI*:
trace-enabled s t **if** *non-mem-exp m* **and** (*m*, *t*, *m'*) ∈ *Traces*
using *that*
by (*auto simp*: *non-mem-exp-def intro*: *non-mem-trace-enabledI*)

lemma *non-mem-exp-traces-enabledI*:
traces-enabled m s regs **if** *non-mem-exp m*
using *that*
by (*auto simp*: *traces-enabled-def intro*: *non-mem-exp-trace-enabledI*)

lemma *traces-enabled-mem-axioms*:
assumes *traces-enabled m initial regs* **and** *hasTrace t m*
and *reads-regs-from inv-regs t regs* **and** *invariant regs*
and *hasException t m* ∨ *hasFailure t m* ⟶ *ex-traces*
and *translation-assm t*
shows *store-mem-axiom CC ISA t*
and *store-tag-axiom CC ISA t*
and *load-mem-axiom CC ISA is-fetch t*
using *assms*
by (*intro accepts-store-mem-axiom accepts-store-tag-axiom accepts-load-mem-axiom traces-enabled-accepts-fromI*[**where** *m* = *m* **and** *regs* = *regs*];
auto)+

end

end
theory *Cheri-reg-lemmas*
imports *Sail−CHERI−MIPS.Cheri-lemmas*
begin

termination *execute* **by** *size-change*

definition
register-names ≡
{''InstCount'', ''CID'', ''ErrorEPCC'', ''KDC'', ''KR2C'', ''KR1C'', ''CPLR'',

''CULR'', ''C31'', ''C30'', ''C29'', ''C28'', ''C27'', ''C26'', ''C25'',
''C24'', ''C23'', ''C22'', ''C21'', ''C20'', ''C19'', ''C18'', ''C17'',
''C16'', ''C15'', ''C14'', ''C13'', ''C12'', ''C11'', ''C10'', ''C09'',
''C08'', ''C07'', ''C06'', ''C05'', ''C04'', ''C03'', ''C02'', ''C01'',
''DDC'', ''CapCause'', ''NextPCC'', ''DelayedPCC'', ''PCC'', ''KCC'', ''EPCC'',
''UART-RVALID'', ''UART-RDATA'', ''UART-WRITTEN'', ''UART-WDATA'',
''GPR'',
''LO'', ''HI'', ''DelayedPC'', ''BranchPending'', ''InBranchDelay'',
''NextInBranchDelay'', ''CP0Status'', ''CP0ConfigK0'', ''CP0UserLocal'',
''CP0HWREna'', ''CP0Count'', ''CP0BadInstrP'', ''CP0BadInstr'',
''LastInstrBits'', ''CurrentInstrBits'', ''CP0BadVAddr'', ''CP0LLAddr'',
''CP0LLBit'', ''CP0Cause'', ''CP0Compare'', ''TLBEntry63'', ''TLBEntry62'',
''TLBEntry61'', ''TLBEntry60'', ''TLBEntry59'', ''TLBEntry58'', ''TLBEntry57'',
''TLBEntry56'', ''TLBEntry55'', ''TLBEntry54'', ''TLBEntry53'', ''TLBEntry52'',
''TLBEntry51'', ''TLBEntry50'', ''TLBEntry49'', ''TLBEntry48'', ''TLBEntry47'',
''TLBEntry46'', ''TLBEntry45'', ''TLBEntry44'', ''TLBEntry43'', ''TLBEntry42'',
''TLBEntry41'', ''TLBEntry40'', ''TLBEntry39'', ''TLBEntry38'', ''TLBEntry37'',
''TLBEntry36'', ''TLBEntry35'', ''TLBEntry34'', ''TLBEntry33'', ''TLBEntry32'',
''TLBEntry31'', ''TLBEntry30'', ''TLBEntry29'', ''TLBEntry28'', ''TLBEntry27'',
''TLBEntry26'', ''TLBEntry25'', ''TLBEntry24'', ''TLBEntry23'', ''TLBEntry22'',
''TLBEntry21'', ''TLBEntry20'', ''TLBEntry19'', ''TLBEntry18'', ''TLBEntry17'',
''TLBEntry16'', ''TLBEntry15'', ''TLBEntry14'', ''TLBEntry13'', ''TLBEntry12'',
''TLBEntry11'', ''TLBEntry10'', ''TLBEntry09'', ''TLBEntry08'', ''TLBEntry07'',
''TLBEntry06'', ''TLBEntry05'', ''TLBEntry04'', ''TLBEntry03'', ''TLBEntry02'',
''TLBEntry01'', ''TLBEntry00'', ''TLBXContext'', ''TLBEntryHi'', ''TLBWired'',
''TLBPageMask'', ''TLBContext'', ''TLBEntryLo1'', ''TLBEntryLo0'',
''TLBRandom'', ''TLBIndex'', ''TLBProbe'', ''NextPC'', ''PC''}

lemma *register-name-cases*:
obtains *r* = ''InstCount''
| *r* = ''CID''
| *r* = ''ErrorEPCC''
| *r* = ''KDC''
| *r* = ''KR2C''
| *r* = ''KR1C''
| *r* = ''CPLR''
| *r* = ''CULR''

$| \; r = ''C31''$
$| \; r = ''C30''$
$| \; r = ''C29''$
$| \; r = ''C28''$
$| \; r = ''C27''$
$| \; r = ''C26''$
$| \; r = ''C25''$
$| \; r = ''C24''$
$| \; r = ''C23''$
$| \; r = ''C22''$
$| \; r = ''C21''$
$| \; r = ''C20''$
$| \; r = ''C19''$
$| \; r = ''C18''$
$| \; r = ''C17''$
$| \; r = ''C16''$
$| \; r = ''C15''$
$| \; r = ''C14''$
$| \; r = ''C13''$
$| \; r = ''C12''$
$| \; r = ''C11''$
$| \; r = ''C10''$
$| \; r = ''C09''$
$| \; r = ''C08''$
$| \; r = ''C07''$
$| \; r = ''C06''$
$| \; r = ''C05''$
$| \; r = ''C04''$
$| \; r = ''C03''$
$| \; r = ''C02''$
$| \; r = ''C01''$
$| \; r = ''DDC''$
$| \; r = ''CapCause''$
$| \; r = ''NextPCC''$
$| \; r = ''DelayedPCC''$
$| \; r = ''PCC''$
$| \; r = ''KCC''$
$| \; r = ''EPCC''$
$| \; r = ''UART\text{-}RVALID''$
$| \; r = ''UART\text{-}RDATA''$
$| \; r = ''UART\text{-}WRITTEN''$
$| \; r = ''UART\text{-}WDATA''$
$| \; r = ''GPR''$
$| \; r = ''LO''$
$| \; r = ''HI''$
$| \; r = ''DelayedPC''$
$| \; r = ''BranchPending''$
$| \; r = ''InBranchDelay''$
$| \; r = ''NextInBranchDelay''$

$| \ r = "CP0Status"$
$| \ r = "CP0ConfigK0"$
$| \ r = "CP0UserLocal"$
$| \ r = "CP0HWREna"$
$| \ r = "CP0Count"$
$| \ r = "CP0BadInstrP"$
$| \ r = "CP0BadInstr"$
$| \ r = "LastInstrBits"$
$| \ r = "CurrentInstrBits"$
$| \ r = "CP0BadVAddr"$
$| \ r = "CP0LLAddr"$
$| \ r = "CP0LLBit"$
$| \ r = "CP0Cause"$
$| \ r = "CP0Compare"$
$| \ r = "TLBEntry63"$
$| \ r = "TLBEntry62"$
$| \ r = "TLBEntry61"$
$| \ r = "TLBEntry60"$
$| \ r = "TLBEntry59"$
$| \ r = "TLBEntry58"$
$| \ r = "TLBEntry57"$
$| \ r = "TLBEntry56"$
$| \ r = "TLBEntry55"$
$| \ r = "TLBEntry54"$
$| \ r = "TLBEntry53"$
$| \ r = "TLBEntry52"$
$| \ r = "TLBEntry51"$
$| \ r = "TLBEntry50"$
$| \ r = "TLBEntry49"$
$| \ r = "TLBEntry48"$
$| \ r = "TLBEntry47"$
$| \ r = "TLBEntry46"$
$| \ r = "TLBEntry45"$
$| \ r = "TLBEntry44"$
$| \ r = "TLBEntry43"$
$| \ r = "TLBEntry42"$
$| \ r = "TLBEntry41"$
$| \ r = "TLBEntry40"$
$| \ r = "TLBEntry39"$
$| \ r = "TLBEntry38"$
$| \ r = "TLBEntry37"$
$| \ r = "TLBEntry36"$
$| \ r = "TLBEntry35"$
$| \ r = "TLBEntry34"$
$| \ r = "TLBEntry33"$
$| \ r = "TLBEntry32"$
$| \ r = "TLBEntry31"$
$| \ r = "TLBEntry30"$
$| \ r = "TLBEntry29"$

| $r = ''TLBEntry28''$
| $r = ''TLBEntry27''$
| $r = ''TLBEntry26''$
| $r = ''TLBEntry25''$
| $r = ''TLBEntry24''$
| $r = ''TLBEntry23''$
| $r = ''TLBEntry22''$
| $r = ''TLBEntry21''$
| $r = ''TLBEntry20''$
| $r = ''TLBEntry19''$
| $r = ''TLBEntry18''$
| $r = ''TLBEntry17''$
| $r = ''TLBEntry16''$
| $r = ''TLBEntry15''$
| $r = ''TLBEntry14''$
| $r = ''TLBEntry13''$
| $r = ''TLBEntry12''$
| $r = ''TLBEntry11''$
| $r = ''TLBEntry10''$
| $r = ''TLBEntry09''$
| $r = ''TLBEntry08''$
| $r = ''TLBEntry07''$
| $r = ''TLBEntry06''$
| $r = ''TLBEntry05''$
| $r = ''TLBEntry04''$
| $r = ''TLBEntry03''$
| $r = ''TLBEntry02''$
| $r = ''TLBEntry01''$
| $r = ''TLBEntry00''$
| $r = ''TLBXContext''$
| $r = ''TLBEntryHi''$
| $r = ''TLBWired''$
| $r = ''TLBPageMask''$
| $r = ''TLBContext''$
| $r = ''TLBEntryLo1''$
| $r = ''TLBEntryLo0''$
| $r = ''TLBRandom''$
| $r = ''TLBIndex''$
| $r = ''TLBProbe''$
| $r = ''NextPC''$
| $r = ''PC''$
| $r \notin$ *register-names*
proof *cases*
assume $r \in$ *register-names*
then show *?thesis*
unfolding *register-names-def*
by (*elim insertE*) (*auto elim*: *that*)
qed

lemma *set-regval-non-register-name*[*simp*]:
$r \notin$ *register-names* $\Longrightarrow$ *set-regval* $r\ v\ s$ = *None*
by (*auto simp*: *register-names-def set-regval-def*)

lemma *get-regval-non-register-name*[*simp*]:
$r \notin$ *register-names* $\Longrightarrow$ *get-regval* $r\ s$ = *None*
by (*auto simp*: *register-names-def get-regval-def*)

lemma *set-regval-cases*:
assumes *set-regval* $r\ v\ s$ = *Some* s'
obtains v' **where** $r = ''InstCount''$ **and** *int-of-regval* v = *Some* v' **and** $s' = s(|InstCount := v'|)$
| v' **where** $r = ''CID''$ **and** *bitvector-64-dec-of-regval* v = *Some* v' **and** $s' = s(|CID := v'|)$
| v' **where** $r = ''ErrorEPCC''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|ErrorEPCC := v'|)$
| v' **where** $r = ''KDC''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|KDC := v'|)$
| v' **where** $r = ''KR2C''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|KR2C := v'|)$
| v' **where** $r = ''KR1C''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|KR1C := v'|)$
| v' **where** $r = ''CPLR''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|CPLR := v'|)$
| v' **where** $r = ''CULR''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|CULR := v'|)$
| v' **where** $r = ''C31''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C31 := v'|)$
| v' **where** $r = ''C30''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C30 := v'|)$
| v' **where** $r = ''C29''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C29 := v'|)$
| v' **where** $r = ''C28''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C28 := v'|)$
| v' **where** $r = ''C27''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C27 := v'|)$
| v' **where** $r = ''C26''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C26 := v'|)$
| v' **where** $r = ''C25''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C25 := v'|)$
| v' **where** $r = ''C24''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C24 := v'|)$
| v' **where** $r = ''C23''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C23 := v'|)$
| v' **where** $r = ''C22''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C22 := v'|)$
| v' **where** $r = ''C21''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C21 := v'|)$
| v' **where** $r = ''C20''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s(|C20$

$:= v'$)
| v' **where** $r = ''C19''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C19* $:= v'$|)
| v' **where** $r = ''C18''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C18* $:= v'$|)
| v' **where** $r = ''C17''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C17* $:= v'$|)
| v' **where** $r = ''C16''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C16* $:= v'$|)
| v' **where** $r = ''C15''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C15* $:= v'$|)
| v' **where** $r = ''C14''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C14* $:= v'$|)
| v' **where** $r = ''C13''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C13* $:= v'$|)
| v' **where** $r = ''C12''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C12* $:= v'$|)
| v' **where** $r = ''C11''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C11* $:= v'$|)
| v' **where** $r = ''C10''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C10* $:= v'$|)
| v' **where** $r = ''C09''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C09* $:= v'$|)
| v' **where** $r = ''C08''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C08* $:= v'$|)
| v' **where** $r = ''C07''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C07* $:= v'$|)
| v' **where** $r = ''C06''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C06* $:= v'$|)
| v' **where** $r = ''C05''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C05* $:= v'$|)
| v' **where** $r = ''C04''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C04* $:= v'$|)
| v' **where** $r = ''C03''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C03* $:= v'$|)
| v' **where** $r = ''C02''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C02* $:= v'$|)
| v' **where** $r = ''C01''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*C01* $:= v'$|)
| v' **where** $r = ''DDC''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*DDC* $:= v'$|)
| v' **where** $r = ''CapCause''$ **and** *CapCauseReg-of-regval* v = *Some* v' **and** $s' = s$(|*CapCause* $:= v'$|)
| v' **where** $r = ''NextPCC''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*NextPCC* $:= v'$|)
| v' **where** $r = ''DelayedPCC''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*DelayedPCC* $:= v'$|)
| v' **where** $r = ''PCC''$ **and** *Capability-of-regval* v = *Some* v' **and** $s' = s$(|*regstate.PCC* $:= v'$|)

| v' **where** $r = ''KCC''$ **and** *Capability-of-regval* $v =$ *Some* v' **and** $s' = s(\!| regstate.KCC := v' |\!)$
| v' **where** $r = ''EPCC''$ **and** *Capability-of-regval* $v =$ *Some* v' **and** $s' = s(\!| EPCC := v' |\!)$
| v' **where** $r = ''UART\text{-}RVALID''$ **and** *bitvector-1-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| UART\text{-}RVALID := v' |\!)$
| v' **where** $r = ''UART\text{-}RDATA''$ **and** *bitvector-8-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| UART\text{-}RDATA := v' |\!)$
| v' **where** $r = ''UART\text{-}WRITTEN''$ **and** *bitvector-1-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| UART\text{-}WRITTEN := v' |\!)$
| v' **where** $r = ''UART\text{-}WDATA''$ **and** *bitvector-8-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| UART\text{-}WDATA := v' |\!)$
| v' **where** $r = ''GPR''$ **and** *vector-of-regval bitvector-64-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| GPR := v' |\!)$
| v' **where** $r = ''LO''$ **and** *bitvector-64-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| LO := v' |\!)$
| v' **where** $r = ''HI''$ **and** *bitvector-64-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| HI := v' |\!)$
| v' **where** $r = ''DelayedPC''$ **and** *bitvector-64-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| DelayedPC := v' |\!)$
| v' **where** $r = ''BranchPending''$ **and** *bitvector-1-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| BranchPending := v' |\!)$
| v' **where** $r = ''InBranchDelay''$ **and** *bitvector-1-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| InBranchDelay := v' |\!)$
| v' **where** $r = ''NextInBranchDelay''$ **and** *bitvector-1-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| NextInBranchDelay := v' |\!)$
| v' **where** $r = ''CP0Status''$ **and** *StatusReg-of-regval* $v =$ *Some* v' **and** $s' = s(\!| CP0Status := v' |\!)$
| v' **where** $r = ''CP0ConfigK0''$ **and** *bitvector-3-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| CP0ConfigK0 := v' |\!)$
| v' **where** $r = ''CP0UserLocal''$ **and** *bitvector-64-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| CP0UserLocal := v' |\!)$
| v' **where** $r = ''CP0HWREna''$ **and** *bitvector-32-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| CP0HWREna := v' |\!)$
| v' **where** $r = ''CP0Count''$ **and** *bitvector-32-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| CP0Count := v' |\!)$
| v' **where** $r = ''CP0BadInstrP''$ **and** *bitvector-32-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| CP0BadInstrP := v' |\!)$
| v' **where** $r = ''CP0BadInstr''$ **and** *bitvector-32-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| CP0BadInstr := v' |\!)$
| v' **where** $r = ''LastInstrBits''$ **and** *bitvector-32-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| LastInstrBits := v' |\!)$
| v' **where** $r = ''CurrentInstrBits''$ **and** *bitvector-32-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| CurrentInstrBits := v' |\!)$
| v' **where** $r = ''CP0BadVAddr''$ **and** *bitvector-64-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| CP0BadVAddr := v' |\!)$
| v' **where** $r = ''CP0LLAddr''$ **and** *bitvector-64-dec-of-regval* $v =$ *Some* v' **and** $s' = s(\!| CP0LLAddr := v' |\!)$
| v' **where** $r = ''CP0LLBit''$ **and** *bitvector-1-dec-of-regval* $v =$ *Some* v' **and** s'

$= s(\!|\, CP0LLBit := v' \,|\!)$
$\mid v'$ **where** $r = ''CP0Cause''$ **and** $CauseReg\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, CP0Cause := v' \,|\!)$
$\mid v'$ **where** $r = ''CP0Compare''$ **and** $bitvector\text{-}32\text{-}dec\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, CP0Compare := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry63''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry63 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry62''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry62 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry61''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry61 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry60''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry60 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry59''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry59 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry58''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry58 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry57''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry57 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry56''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry56 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry55''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry55 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry54''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry54 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry53''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry53 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry52''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry52 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry51''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry51 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry50''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry50 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry49''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry49 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry48''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry48 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry47''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry47 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry46''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry46 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry45''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry45 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry44''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry44 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry43''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry43 := v' \,|\!)$
$\mid v'$ **where** $r = ''TLBEntry42''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\, TLBEntry42 := v' \,|\!)$

| v' **where** $r = ''TLBEntry41''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry41 := v'|\!)$
| v' **where** $r = ''TLBEntry40''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry40 := v'|\!)$
| v' **where** $r = ''TLBEntry39''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry39 := v'|\!)$
| v' **where** $r = ''TLBEntry38''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry38 := v'|\!)$
| v' **where** $r = ''TLBEntry37''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry37 := v'|\!)$
| v' **where** $r = ''TLBEntry36''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry36 := v'|\!)$
| v' **where** $r = ''TLBEntry35''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry35 := v'|\!)$
| v' **where** $r = ''TLBEntry34''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry34 := v'|\!)$
| v' **where** $r = ''TLBEntry33''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry33 := v'|\!)$
| v' **where** $r = ''TLBEntry32''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry32 := v'|\!)$
| v' **where** $r = ''TLBEntry31''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry31 := v'|\!)$
| v' **where** $r = ''TLBEntry30''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry30 := v'|\!)$
| v' **where** $r = ''TLBEntry29''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry29 := v'|\!)$
| v' **where** $r = ''TLBEntry28''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry28 := v'|\!)$
| v' **where** $r = ''TLBEntry27''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry27 := v'|\!)$
| v' **where** $r = ''TLBEntry26''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry26 := v'|\!)$
| v' **where** $r = ''TLBEntry25''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry25 := v'|\!)$
| v' **where** $r = ''TLBEntry24''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry24 := v'|\!)$
| v' **where** $r = ''TLBEntry23''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry23 := v'|\!)$
| v' **where** $r = ''TLBEntry22''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry22 := v'|\!)$
| v' **where** $r = ''TLBEntry21''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry21 := v'|\!)$
| v' **where** $r = ''TLBEntry20''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry20 := v'|\!)$
| v' **where** $r = ''TLBEntry19''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry19 := v'|\!)$
| v' **where** $r = ''TLBEntry18''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' = s(\!|\ TLBEntry18 := v'|\!)$
| v' **where** $r = ''TLBEntry17''$ **and** $TLBEntry\text{-}of\text{-}regval\ v = Some\ v'$ **and** $s' =$

$s(\!|\mathit{TLBEntry17} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry16}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry16} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry15}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry15} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry14}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry14} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry13}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry13} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry12}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry12} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry11}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry11} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry10}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry10} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry09}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry09} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry08}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry08} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry07}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry07} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry06}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry06} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry05}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry05} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry04}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry04} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry03}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry03} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry02}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry02} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry01}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry01} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntry00}''$ **and** $\mathit{TLBEntry\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntry00} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBXContext}''$ **and** $\mathit{XContextReg\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBXContext} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntryHi}''$ **and** $\mathit{TLBEntryHiReg\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntryHi} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBWired}''$ **and** $\mathit{bitvector\text{-}6\text{-}dec\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBWired} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBPageMask}''$ **and** $\mathit{bitvector\text{-}16\text{-}dec\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBPageMask} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBContext}''$ **and** $\mathit{ContextReg\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBContext} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntryLo1}''$ **and** $\mathit{TLBEntryLoReg\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntryLo1} := v'|\!)$
$|$ v' **where** $r = ''\mathit{TLBEntryLo0}''$ **and** $\mathit{TLBEntryLoReg\text{-}of\text{-}regval}\ v = \mathit{Some}\ v'$ **and** $s' = s(\!|\mathit{TLBEntryLo0} := v'|\!)$

| v' **where** $r = ''TLBRandom''$ **and** *bitvector-6-dec-of-regval* $v =$ *Some* v' **and** $s' = s(|TLBRandom := v'|)$
| v' **where** $r = ''TLBIndex''$ **and** *bitvector-6-dec-of-regval* $v =$ *Some* v' **and** $s' = s(|TLBIndex := v'|)$
| v' **where** $r = ''TLBProbe''$ **and** *bitvector-1-dec-of-regval* $v =$ *Some* v' **and** $s' = s(|TLBProbe := v'|)$
| v' **where** $r = ''NextPC''$ **and** *bitvector-64-dec-of-regval* $v =$ *Some* v' **and** $s' = s(|NextPC := v'|)$
| v' **where** $r = ''PC''$ **and** *bitvector-64-dec-of-regval* $v =$ *Some* v' **and** $s' = s(|PC := v'|)$
proof −
from *assms* **have** $r \in$ *register-names*
by (*cases* $r \in$ *register-names*) *auto*
with *assms* **show** *thesis*
by (*cases r rule*: *register-name-cases*) (*auto simp*: *register-defs elim*: *that*)
qed

lemma *get-regval-simps*:
get-regval ''InstCount'' s = Some (regval-of-int (InstCount s))
get-regval ''CID'' s = Some (regval-of-bitvector-64-dec (CID s))
get-regval ''ErrorEPCC'' s = Some (regval-of-Capability (ErrorEPCC s))
get-regval ''KDC'' s = Some (regval-of-Capability (KDC s))
get-regval ''KR2C'' s = Some (regval-of-Capability (KR2C s))
get-regval ''KR1C'' s = Some (regval-of-Capability (KR1C s))
get-regval ''CPLR'' s = Some (regval-of-Capability (CPLR s))
get-regval ''CULR'' s = Some (regval-of-Capability (CULR s))
get-regval ''C31'' s = Some (regval-of-Capability (C31 s))
get-regval ''C30'' s = Some (regval-of-Capability (C30 s))
get-regval ''C29'' s = Some (regval-of-Capability (C29 s))
get-regval ''C28'' s = Some (regval-of-Capability (C28 s))
get-regval ''C27'' s = Some (regval-of-Capability (C27 s))
get-regval ''C26'' s = Some (regval-of-Capability (C26 s))
get-regval ''C25'' s = Some (regval-of-Capability (C25 s))
get-regval ''C24'' s = Some (regval-of-Capability (C24 s))
get-regval ''C23'' s = Some (regval-of-Capability (C23 s))
get-regval ''C22'' s = Some (regval-of-Capability (C22 s))
get-regval ''C21'' s = Some (regval-of-Capability (C21 s))
get-regval ''C20'' s = Some (regval-of-Capability (C20 s))
get-regval ''C19'' s = Some (regval-of-Capability (C19 s))
get-regval ''C18'' s = Some (regval-of-Capability (C18 s))
get-regval ''C17'' s = Some (regval-of-Capability (C17 s))
get-regval ''C16'' s = Some (regval-of-Capability (C16 s))
get-regval ''C15'' s = Some (regval-of-Capability (C15 s))
get-regval ''C14'' s = Some (regval-of-Capability (C14 s))
get-regval ''C13'' s = Some (regval-of-Capability (C13 s))
get-regval ''C12'' s = Some (regval-of-Capability (C12 s))
get-regval ''C11'' s = Some (regval-of-Capability (C11 s))
get-regval ''C10'' s = Some (regval-of-Capability (C10 s))
get-regval ''C09'' s = Some (regval-of-Capability (C09 s))

$get\text{-}regval\ ''C08''\ s = Some\ (regval\text{-}of\text{-}Capability\ (C08\ s))$
$get\text{-}regval\ ''C07''\ s = Some\ (regval\text{-}of\text{-}Capability\ (C07\ s))$
$get\text{-}regval\ ''C06''\ s = Some\ (regval\text{-}of\text{-}Capability\ (C06\ s))$
$get\text{-}regval\ ''C05''\ s = Some\ (regval\text{-}of\text{-}Capability\ (C05\ s))$
$get\text{-}regval\ ''C04''\ s = Some\ (regval\text{-}of\text{-}Capability\ (C04\ s))$
$get\text{-}regval\ ''C03''\ s = Some\ (regval\text{-}of\text{-}Capability\ (C03\ s))$
$get\text{-}regval\ ''C02''\ s = Some\ (regval\text{-}of\text{-}Capability\ (C02\ s))$
$get\text{-}regval\ ''C01''\ s = Some\ (regval\text{-}of\text{-}Capability\ (C01\ s))$
$get\text{-}regval\ ''DDC''\ s = Some\ (regval\text{-}of\text{-}Capability\ (DDC\ s))$
$get\text{-}regval\ ''CapCause''\ s = Some\ (regval\text{-}of\text{-}CapCauseReg\ (CapCause\ s))$
$get\text{-}regval\ ''NextPCC''\ s = Some\ (regval\text{-}of\text{-}Capability\ (NextPCC\ s))$
$get\text{-}regval\ ''DelayedPCC''\ s = Some\ (regval\text{-}of\text{-}Capability\ (DelayedPCC\ s))$
$get\text{-}regval\ ''PCC''\ s = Some\ (regval\text{-}of\text{-}Capability\ (regstate.PCC\ s))$
$get\text{-}regval\ ''KCC''\ s = Some\ (regval\text{-}of\text{-}Capability\ (regstate.KCC\ s))$
$get\text{-}regval\ ''EPCC''\ s = Some\ (regval\text{-}of\text{-}Capability\ (EPCC\ s))$
$get\text{-}regval\ ''UART\text{-}RVALID''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}1\text{-}dec\ (UART\text{-}RVALID\ s))$
$get\text{-}regval\ ''UART\text{-}RDATA''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}8\text{-}dec\ (UART\text{-}RDATA\ s))$
$get\text{-}regval\ ''UART\text{-}WRITTEN''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}1\text{-}dec\ (UART\text{-}WRITTEN\ s))$
$get\text{-}regval\ ''UART\text{-}WDATA''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}8\text{-}dec\ (UART\text{-}WDATA\ s))$
$get\text{-}regval\ ''GPR''\ s = Some\ (regval\text{-}of\text{-}vector\ regval\text{-}of\text{-}bitvector\text{-}64\text{-}dec\ (GPR\ s))$
$get\text{-}regval\ ''LO''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}64\text{-}dec\ (LO\ s))$
$get\text{-}regval\ ''HI''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}64\text{-}dec\ (HI\ s))$
$get\text{-}regval\ ''DelayedPC''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}64\text{-}dec\ (DelayedPC\ s))$
$get\text{-}regval\ ''BranchPending''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}1\text{-}dec\ (BranchPending\ s))$
$get\text{-}regval\ ''InBranchDelay''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}1\text{-}dec\ (InBranchDelay\ s))$
$get\text{-}regval\ ''NextInBranchDelay''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}1\text{-}dec\ (NextInBranchDelay\ s))$
$get\text{-}regval\ ''CP0Status''\ s = Some\ (regval\text{-}of\text{-}StatusReg\ (CP0Status\ s))$
$get\text{-}regval\ ''CP0ConfigK0''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}3\text{-}dec\ (CP0ConfigK0\ s))$
$get\text{-}regval\ ''CP0UserLocal''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}64\text{-}dec\ (CP0UserLocal\ s))$
$get\text{-}regval\ ''CP0HWREna''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}32\text{-}dec\ (CP0HWREna\ s))$
$get\text{-}regval\ ''CP0Count''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}32\text{-}dec\ (CP0Count\ s))$
$get\text{-}regval\ ''CP0BadInstrP''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}32\text{-}dec\ (CP0BadInstrP\ s))$
$get\text{-}regval\ ''CP0BadInstr''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}32\text{-}dec\ (CP0BadInstr\ s))$
$get\text{-}regval\ ''LastInstrBits''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}32\text{-}dec\ (LastInstrBits\ s))$
$get\text{-}regval\ ''CurrentInstrBits''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}32\text{-}dec\ (CurrentInstrBits\ s))$
$get\text{-}regval\ ''CP0BadVAddr''\ s = Some\ (regval\text{-}of\text{-}bitvector\text{-}64\text{-}dec\ (CP0BadVAddr$

s))
get-regval ''CP0LLAddr'' s = Some (regval-of-bitvector-64-dec (CP0LLAddr s))
get-regval ''CP0LLBit'' s = Some (regval-of-bitvector-1-dec (CP0LLBit s))
get-regval ''CP0Cause'' s = Some (regval-of-CauseReg (CP0Cause s))
get-regval ''CP0Compare'' s = Some (regval-of-bitvector-32-dec (CP0Compare s))
get-regval ''TLBEntry63'' s = Some (regval-of-TLBEntry (TLBEntry63 s))
get-regval ''TLBEntry62'' s = Some (regval-of-TLBEntry (TLBEntry62 s))
get-regval ''TLBEntry61'' s = Some (regval-of-TLBEntry (TLBEntry61 s))
get-regval ''TLBEntry60'' s = Some (regval-of-TLBEntry (TLBEntry60 s))
get-regval ''TLBEntry59'' s = Some (regval-of-TLBEntry (TLBEntry59 s))
get-regval ''TLBEntry58'' s = Some (regval-of-TLBEntry (TLBEntry58 s))
get-regval ''TLBEntry57'' s = Some (regval-of-TLBEntry (TLBEntry57 s))
get-regval ''TLBEntry56'' s = Some (regval-of-TLBEntry (TLBEntry56 s))
get-regval ''TLBEntry55'' s = Some (regval-of-TLBEntry (TLBEntry55 s))
get-regval ''TLBEntry54'' s = Some (regval-of-TLBEntry (TLBEntry54 s))
get-regval ''TLBEntry53'' s = Some (regval-of-TLBEntry (TLBEntry53 s))
get-regval ''TLBEntry52'' s = Some (regval-of-TLBEntry (TLBEntry52 s))
get-regval ''TLBEntry51'' s = Some (regval-of-TLBEntry (TLBEntry51 s))
get-regval ''TLBEntry50'' s = Some (regval-of-TLBEntry (TLBEntry50 s))
get-regval ''TLBEntry49'' s = Some (regval-of-TLBEntry (TLBEntry49 s))
get-regval ''TLBEntry48'' s = Some (regval-of-TLBEntry (TLBEntry48 s))
get-regval ''TLBEntry47'' s = Some (regval-of-TLBEntry (TLBEntry47 s))
get-regval ''TLBEntry46'' s = Some (regval-of-TLBEntry (TLBEntry46 s))
get-regval ''TLBEntry45'' s = Some (regval-of-TLBEntry (TLBEntry45 s))
get-regval ''TLBEntry44'' s = Some (regval-of-TLBEntry (TLBEntry44 s))
get-regval ''TLBEntry43'' s = Some (regval-of-TLBEntry (TLBEntry43 s))
get-regval ''TLBEntry42'' s = Some (regval-of-TLBEntry (TLBEntry42 s))
get-regval ''TLBEntry41'' s = Some (regval-of-TLBEntry (TLBEntry41 s))
get-regval ''TLBEntry40'' s = Some (regval-of-TLBEntry (TLBEntry40 s))
get-regval ''TLBEntry39'' s = Some (regval-of-TLBEntry (TLBEntry39 s))
get-regval ''TLBEntry38'' s = Some (regval-of-TLBEntry (TLBEntry38 s))
get-regval ''TLBEntry37'' s = Some (regval-of-TLBEntry (TLBEntry37 s))
get-regval ''TLBEntry36'' s = Some (regval-of-TLBEntry (TLBEntry36 s))
get-regval ''TLBEntry35'' s = Some (regval-of-TLBEntry (TLBEntry35 s))
get-regval ''TLBEntry34'' s = Some (regval-of-TLBEntry (TLBEntry34 s))
get-regval ''TLBEntry33'' s = Some (regval-of-TLBEntry (TLBEntry33 s))
get-regval ''TLBEntry32'' s = Some (regval-of-TLBEntry (TLBEntry32 s))
get-regval ''TLBEntry31'' s = Some (regval-of-TLBEntry (TLBEntry31 s))
get-regval ''TLBEntry30'' s = Some (regval-of-TLBEntry (TLBEntry30 s))
get-regval ''TLBEntry29'' s = Some (regval-of-TLBEntry (TLBEntry29 s))
get-regval ''TLBEntry28'' s = Some (regval-of-TLBEntry (TLBEntry28 s))
get-regval ''TLBEntry27'' s = Some (regval-of-TLBEntry (TLBEntry27 s))
get-regval ''TLBEntry26'' s = Some (regval-of-TLBEntry (TLBEntry26 s))
get-regval ''TLBEntry25'' s = Some (regval-of-TLBEntry (TLBEntry25 s))
get-regval ''TLBEntry24'' s = Some (regval-of-TLBEntry (TLBEntry24 s))
get-regval ''TLBEntry23'' s = Some (regval-of-TLBEntry (TLBEntry23 s))
get-regval ''TLBEntry22'' s = Some (regval-of-TLBEntry (TLBEntry22 s))
get-regval ''TLBEntry21'' s = Some (regval-of-TLBEntry (TLBEntry21 s))

get-regval ''TLBEntry20'' s = Some (regval-of-TLBEntry (TLBEntry20 s))
get-regval ''TLBEntry19'' s = Some (regval-of-TLBEntry (TLBEntry19 s))
get-regval ''TLBEntry18'' s = Some (regval-of-TLBEntry (TLBEntry18 s))
get-regval ''TLBEntry17'' s = Some (regval-of-TLBEntry (TLBEntry17 s))
get-regval ''TLBEntry16'' s = Some (regval-of-TLBEntry (TLBEntry16 s))
get-regval ''TLBEntry15'' s = Some (regval-of-TLBEntry (TLBEntry15 s))
get-regval ''TLBEntry14'' s = Some (regval-of-TLBEntry (TLBEntry14 s))
get-regval ''TLBEntry13'' s = Some (regval-of-TLBEntry (TLBEntry13 s))
get-regval ''TLBEntry12'' s = Some (regval-of-TLBEntry (TLBEntry12 s))
get-regval ''TLBEntry11'' s = Some (regval-of-TLBEntry (TLBEntry11 s))
get-regval ''TLBEntry10'' s = Some (regval-of-TLBEntry (TLBEntry10 s))
get-regval ''TLBEntry09'' s = Some (regval-of-TLBEntry (TLBEntry09 s))
get-regval ''TLBEntry08'' s = Some (regval-of-TLBEntry (TLBEntry08 s))
get-regval ''TLBEntry07'' s = Some (regval-of-TLBEntry (TLBEntry07 s))
get-regval ''TLBEntry06'' s = Some (regval-of-TLBEntry (TLBEntry06 s))
get-regval ''TLBEntry05'' s = Some (regval-of-TLBEntry (TLBEntry05 s))
get-regval ''TLBEntry04'' s = Some (regval-of-TLBEntry (TLBEntry04 s))
get-regval ''TLBEntry03'' s = Some (regval-of-TLBEntry (TLBEntry03 s))
get-regval ''TLBEntry02'' s = Some (regval-of-TLBEntry (TLBEntry02 s))
get-regval ''TLBEntry01'' s = Some (regval-of-TLBEntry (TLBEntry01 s))
get-regval ''TLBEntry00'' s = Some (regval-of-TLBEntry (TLBEntry00 s))
get-regval ''TLBXContext'' s = Some (regval-of-XContextReg (TLBXContext s))
get-regval ''TLBEntryHi'' s = Some (regval-of-TLBEntryHiReg (TLBEntryHi s))
get-regval ''TLBWired'' s = Some (regval-of-bitvector-6-dec (TLBWired s))
get-regval ''TLBPageMask'' s = Some (regval-of-bitvector-16-dec (TLBPageMask s))
get-regval ''TLBContext'' s = Some (regval-of-ContextReg (TLBContext s))
get-regval ''TLBEntryLo1'' s = Some (regval-of-TLBEntryLoReg (TLBEntryLo1 s))
get-regval ''TLBEntryLo0'' s = Some (regval-of-TLBEntryLoReg (TLBEntryLo0 s))
get-regval ''TLBRandom'' s = Some (regval-of-bitvector-6-dec (TLBRandom s))
get-regval ''TLBIndex'' s = Some (regval-of-bitvector-6-dec (TLBIndex s))
get-regval ''TLBProbe'' s = Some (regval-of-bitvector-1-dec (TLBProbe s))
get-regval ''NextPC'' s = Some (regval-of-bitvector-64-dec (NextPC s))
get-regval ''PC'' s = Some (regval-of-bitvector-64-dec (PC s))
by (*auto simp*: *register-defs*)

lemma *get-ignore-set-regval*:
assumes *s'*: *set-regval r v s = Some s'* **and** *r'*: $r' \neq r$
shows *get-regval r' s' = get-regval r' s*
using *assms*
apply (*elim set-regval-cases*)
subgoal by (*cases r' rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases r' rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases r' rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases r' rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases r' rule*: *register-name-cases*; *simp add*: *get-regval-simps*)

subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)

subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)

subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
subgoal by (*cases* r' *rule*: *register-name-cases*; *simp add*: *get-regval-simps*)
done

fun-cases *of-regval-SomeE*:
int-of-regval v = *Some* v'
bit-of-regval v = *Some* v'

bitvector-1-dec-of-regval v = *Some* v'
bitvector-3-dec-of-regval v = *Some* v'
bitvector-6-dec-of-regval v = *Some* v'
bitvector-8-dec-of-regval v = *Some* v'
bitvector-16-dec-of-regval v = *Some* v'
bitvector-32-dec-of-regval v = *Some* v'
bitvector-64-dec-of-regval v = *Some* v'
Capability-of-regval v = *Some* v'
CapCauseReg-of-regval v = *Some* v'
CauseReg-of-regval v = *Some* v'
ContextReg-of-regval v = *Some* v'
XContextReg-of-regval v = *Some* v'
StatusReg-of-regval v = *Some* v'
TLBEntry-of-regval v = *Some* v'
TLBEntryHiReg-of-regval v = *Some* v'
TLBEntryLoReg-of-regval v = *Some* v'

lemmas *regval-of-defs* =
regval-of-int-def regval-of-bitvector-64-dec-def regval-of-Capability-def regval-of-CapCauseReg-def regval-of-CauseReg-def regval-of-ContextReg-def regval-of-XContextReg-def regval-of-StatusReg-def regval-of-TLBEntry-def regval-of-TLBEntryHiReg-def regval-of-TLBEntryLoReg-def regval-of-bit-def regval-of-bitvector-1-dec-def regval-of-bitvector-3-dec-def regval-of-bitvector-6-dec-def regval-of-bitvector-8-dec-def regval-of-bitvector-16-dec-def regval-of-bitvector-32-dec-def regval-of-vector-def

lemma *vector-of-regval-SomeE*:
assumes $*$: *vector-of-regval of-rv* v = *Some xs* **and** $**$: $\bigwedge v\ v'$. *of-rv* v = *Some* $v' \Longrightarrow$ *rv-of* $v' = v$
obtains v = *Regval-vector* (*map rv-of xs*)
proof −
from $*$ **obtain** *vs* **where** v = *Regval-vector vs* **and** $***$: *map of-rv vs* = *map Some xs*
by (*auto simp*: *vector-of-regval-def split*: *register-value.splits*)
moreover have *map rv-of xs* = *vs*
using $**$ $***$
by (*induction xs arbitrary*: *vs*) *auto*
ultimately
show *?thesis*
using *that*
by *blast*
qed

lemma *get-absorb-set-regval*:
assumes *set-regval r v s* = *Some* s'
shows *get-regval r* s' = *Some* v
using *assms*
by (*elim set-regval-cases*)
(*auto simp*: *get-regval-simps regval-of-defs elim*: *of-regval-SomeE vector-of-regval-SomeE*)

end
theory *CHERI-MIPS-Instantiation*
imports *Sail−CHERI−MIPS.Cheri-lemmas Cheri-reg-lemmas Recognising-Automata Sail.Sail2-operators-mwords-lemmas Word-Extra*
begin

3 Capability monotonicity in CHERI-MIPS

lemma *more-and-or-boolM-simps*[*simp*]:
and-boolM (*return True*) *m* = *m*
and-boolM (*return False*) *m* = *return False*
or-boolM (*return True*) *m* = *return True*
or-boolM (*return False*) *m* = *m*
by (*auto simp*: *and-boolM-def or-boolM-def*)

lemma *final-Done*[*intro*, *simp*]: *final* (*Done a*)
by (*auto simp*: *final-def*)

lemma *bitU-of-bool-simps*[*simp*]: *bitU-of-bool True* = *B1 bitU-of-bool False* = *B0*
by (*auto simp*: *bitU-of-bool-def*)

lemma *nat-of-mword-unat*[*simp*]: *nat-of-bv BC-mword w* = *Some* (*unat w*)
by (*auto simp*: *nat-of-bv-def unat-def*)

lemma *pow2-simp*[*simp*]: *pow2 n* = *2* ˆ *nat n*
by (*auto simp*: *pow2-def pow-def*)

lemma *to-bits-mult*[*simp*]:
n = *int* (*LENGTH*(*'a*)) ⟹ *to-bits n* (*a* ∗ *b*) = (*to-bits n a* ∗ *to-bits n b* :: *'a*::*len word*)
by (*auto simp*: *to-bits-def of-bl-bin-word-of-int wi-hom-syms*)

lemma *to-bits-64-32*[*simp*]: *to-bits 64 32* = (*32* :: *64 word*)
by *eval*

lemma *mult-32-shiftl-5*[*simp*]: *32* ∗ (*w* :: *'a*::*len word*) = *w* << *5*
by (*auto simp*: *shiftl-t2n*)

lemma *shiftl-AND-mask-0*[*simp*]: (*w* << *n*) *AND mask n* = *0*
by (*intro word-eqI*) (*auto simp*: *word-ao-nth nth-shiftl*)

lemma *unat-to-bits*[*simp*]:
len = *int* (*LENGTH*(*'a*)) ⟹ *unat* (*to-bits len i* :: *'a*::*len word*) = *nat* (*i mod 2* ˆ *LENGTH*(*'a*))
by (*auto simp*: *to-bits-def of-bl-bin-word-of-int unat-def uint-word-of-int*)

lemma *uint-to-bits*[*simp*]:
len = *int* (*LENGTH*(*'a*)) ⟹ *uint* (*to-bits len i* :: *'a*::*len word*) = *i mod 2* ˆ *LENGTH*(*'a*)

by (*auto simp*: *to-bits-def of-bl-bin-word-of-int uint-word-of-int*)

lemma *length-take-chunks*[*simp*]:
n dvd length xs $\Longrightarrow$ *length* (*take-chunks n xs*) = *length xs div n*
by (*induction n xs rule*: *take-chunks.induct*) (*auto simp*: *le-div-geq*[*symmetric*] *dvd-imp-le*)

lemma *length-mem-bytes-of-word*[*simp*]:
fixes *w* :: *'a*::*len word*
assumes *8 dvd LENGTH*(*'a*)
shows *length* (*mem-bytes-of-word w*) = *LENGTH*(*'a*) *div 8*
using *assms*
by (*auto simp add*: *mem-bytes-of-word-def simp del*: *take-chunks.simps*)

lemma (**in** *State-Invariant*) *Run-inv-assert-exp-iff* [*iff*]:
Run-inv (*assert-exp c msg*) *t a regs* $\longleftrightarrow$ *c* $\wedge$ *t* = [] $\wedge$ *invariant regs*
unfolding *Run-inv-def*
by *auto*

lemma (**in** *Cap-Axiom-Automaton*) *Run-runs-no-reg-writes-written-regs-eq*:
assumes *Run m t a* **and** *runs-no-reg-writes-to* {*r*} *m*
shows *r* $\in$ *written-regs* (*run s t*) $\longleftrightarrow$ *r* $\in$ *written-regs s*
proof −
from *assms* **have** *E-write-reg r v* $\notin$ *set t* **for** *v*
unfolding *runs-no-reg-writes-to-def*
by *auto*
then show *?thesis*
by (*induction t arbitrary*: *s*) *auto*
qed

3.1 Instantiation of the abstract model for CHERI-MIPS

definition *get-cap-perms* :: *Capability* $\Rightarrow$ *perms* **where**
get-cap-perms c =
(|*permit-ccall* = *Capability-permit-ccall c*,
permit-execute = *Capability-permit-execute c*,
permit-load = *Capability-permit-load c*,
permit-load-capability = *Capability-permit-load-cap c*,
permit-seal = *Capability-permit-seal c*,
permit-store = *Capability-permit-store c*,
permit-store-capability = *Capability-permit-store-cap c*,
permit-store-local-capability = *Capability-permit-store-local-cap c*,
permit-system-access = *Capability-access-system-regs c*,
permit-unseal = *Capability-permit-unseal c*|)

definition *set-cap-perms* :: *Capability* $\Rightarrow$ *perms* $\Rightarrow$ *Capability* **where**
set-cap-perms c p =
c(|*Capability-permit-ccall* := *permit-ccall p*,
Capability-permit-execute := *permit-execute p*,

Capability-permit-load := *permit-load p*,
Capability-permit-load-cap := *permit-load-capability p*,
Capability-permit-seal := *permit-seal p*,
Capability-permit-store := *permit-store p*,
Capability-permit-store-cap := *permit-store-capability p*,
Capability-permit-store-local-cap := *permit-store-local-capability p*,
Capability-access-system-regs := *permit-system-access p*,
Capability-permit-unseal := *permit-unseal p*⦈

fun *cap-of-mem-bytes* :: *memory-byte list* ⇒ *bitU* ⇒ *Capability option* **where**
cap-of-mem-bytes bs t =
Option.bind (*bool-of-bitU t*) (λ*t*.
map-option (λ*bs*. *memBitsToCapability t bs*) (*of-bits-method BC-mword* (*bits-of-mem-bytes bs*)))

abbreviation
CC ≡
⦇*is-tagged-method* = (λ*c*. *Capability-tag c*),
is-sealed-method = (λ*c*. *Capability-sealed c*),
get-mem-region-method = (λ*c*. {*nat* (*getCapBase c*) ..< *nat* (*getCapTop c*)}),
get-obj-type-method = (λ*c*. *unat* (*Capability-otype c*)),
get-perms-method = *get-cap-perms*,
get-cursor-method = (λ*c*. *nat* (*getCapCursor c*)),
get-global-method = (λ*c*. *Capability-global c*),
set-tag-method = (λ*c t*. *c*⦇*Capability-tag* := *t*⦈),
set-seal-method = (λ*c s*. *c*⦇*Capability-sealed* := *s*⦈),
set-obj-type-method = (λ*c t*. *c*⦇*Capability-otype* := *of-nat t*⦈),
set-perms-method = *set-cap-perms*,
set-global-method = (λ*c g*. *c*⦇*Capability-global* := *g*⦈),
cap-of-mem-bytes-method = *cap-of-mem-bytes*⦈

interpretation *Capabilities CC*
by *unfold-locales*
(*auto simp*: *bool-of-bitU-def memBitsToCapability-def capBitsToCapability-def get-cap-perms-def set-cap-perms-def split*: *bitU.splits*)

abbreviation *privileged-CHERI-regs* ≡ {''*EPCC*'', ''*ErrorEPCC*'', ''*KDC*'', ''*KCC*'', ''*KR1C*'', ''*KR2C*'', ''*CapCause*'', ''*CPLR*''}

definition *TLBEntries-names* ≡ *name* ‘ (*set TLBEntries*)

locale *CHERI-MIPS-ISA* =
fixes *translate-address* :: *nat* ⇒ *acctype* ⇒ *register-value trace* ⇒ *nat option*
begin

abbreviation *fetch-and-decode* ≡ (*fetch* () ≫= (λ*res*. *case res of Some ast* ⇒ *return ast* | *None* ⇒ *Fail* ''*decode*''))

definition

$ISA \equiv$
$(\!|$*instr-sem* = *execute*,
instr-fetch = *fetch-and-decode*,
tag-granule = *32*,
PCC = {''*PCC*'', ''*NextPCC*'', ''*DelayedPCC*''},
KCC = {''*KCC*''},
IDC = {''*C26*''},
caps-of-regval = (λ*rv*. *case rv of Regval-Capability c* $\Rightarrow$ {*c*} | - $\Rightarrow$ {}),
invokes-caps = (λ*instr t*. *case instr of CCall* - $\Rightarrow$ *True* | - $\Rightarrow$ *False*),
instr-raises-ex = (λ*instr t*. *hasException t* (*execute instr*) $\vee$ *hasFailure t* (*execute instr*)),
fetch-raises-ex = (λ*t*. *hasException t* (*fetch-and-decode*) $\vee$ *hasFailure t* (*fetch-and-decode*)),
exception-targets = (λ*rvs*. $\bigcup$*rv* $\in$ *rvs*. *case rv of Regval-Capability c* $\Rightarrow$ {*c*} | - $\Rightarrow$ {}),
privileged-regs = *privileged-CHERI-regs*,
translation-tables = (λ*t*. {}),
translate-address = *translate-address*$|\!)$

interpretation *Capability-ISA CC ISA* **by** *unfold-locales*

sublocale *Register-State get-regval set-regval* **.**

lemma *ISA-simps*[*simp*]:
PCC ISA = {''*PCC*'', ''*NextPCC*'', ''*DelayedPCC*''}
KCC ISA = {''*KCC*''}
IDC ISA = {''*C26*''}
privileged-regs ISA = *privileged-CHERI-regs*
instr-sem ISA = *execute*
instr-fetch ISA = (*fetch* () $\gg=$ (λ*res*. *case res of Some ast* $\Rightarrow$ *return ast* | *None* $\Rightarrow$ *Fail* ''*decode*''))
by (*auto simp*: *ISA-def*)

lemma *invokes-caps-iff-CCall*[*simp*]:
invokes-caps ISA instr t $\longleftrightarrow$ ($\exists$ *cs cb sel*. *instr* = *CCall* (*cs*, *cb*, *sel*))
by (*cases instr*) (*auto simp*: *ISA-def*)

lemma *instr-raises-ex-iff*[*simp*]:
instr-raises-ex ISA instr t $\longleftrightarrow$ *hasException t* (*execute instr*) $\vee$ *hasFailure t* (*execute instr*)
by (*auto simp*: *ISA-def*)

lemma *fetch-raises-ex-iff*[*simp*]:
fetch-raises-ex ISA t $\longleftrightarrow$ *hasException t* (*fetch-and-decode*) $\vee$ *hasFailure t* (*fetch-and-decode*)
by (*auto simp*: *ISA-def*)

lemma *TLBEntries-no-cap*:
assumes *r* $\in$ *set TLBEntries*
shows $\bigwedge$*c*. *of-regval r* (*Regval-Capability c*) = *None* **and** *name r* $\neq$ ''*KCC*''
using *assms*

unfolding *TLBEntries-def register-defs*
by *auto*

lemma [*simp*]: *length TLBEntries* = *64*
by (*auto simp*: *TLBEntries-def*)

lemma *vector-of-regval-Regval-Capability-None*[*simp*]:
vector-of-regval or (*Regval-Capability c*) = *None*
by (*auto simp*: *vector-of-regval-def*)

definition *is-cap-reg* :: (*'s*, *register-value*, *Capability*) *register-ref* ⇒ *bool* **where**
is-cap-reg r = (∀ *v c*. *of-regval r v* = *Some c* ⟷ *v* = *Regval-Capability c*)

lemma *Capability-of-regval-Some-iff-Regval-Capability*[*simp*]:
Capability-of-regval v = *Some c* ⟷ *v* = *Regval-Capability c*
by (*cases v*) *auto*

lemma *caps-of-regval-of-Capability*[*simp*]:
caps-of-regval ISA (*regval-of-Capability c*) = {*c*}
by (*auto simp*: *regval-of-Capability-def ISA-def*)

lemma *CapRegs-is-cap-reg*: *r* ∈ *set CapRegs* ⟹ *is-cap-reg r*
unfolding *register-defs CapRegs-def*
by (*auto simp*: *is-cap-reg-def*)

lemma [*simp*]: *length CapRegs* = *32*
by (*auto simp*: *CapRegs-def*)

definition *CapRegs-names* ≡ *name* ‘ (*set CapRegs*)

lemma *CapRegs-names-unfold*[*simp*]:
CapRegs-names =
{*''C31''*, *''C30''*, *''C29''*, *''C28''*, *''C27''*, *''C26''*, *''C25''*, *''C24''*, *''C23''*, *''C22''*, *''C21''*,
''C20'', *''C19''*, *''C18''*, *''C17''*, *''C16''*, *''C15''*, *''C14''*, *''C13''*, *''C12''*, *''C11''*, *''C10''*,
''C09'', *''C08''*, *''C07''*, *''C06''*, *''C05''*, *''C04''*, *''C03''*, *''C02''*, *''C01''*, *''DDC''*}
unfolding *CapRegs-names-def CapRegs-def register-defs*
by *auto*

lemma *name-CapRegs-CapRegs-names*: *r* ∈ *set CapRegs* ⟹ *name r* ∈ *CapRegs-names*
unfolding *CapRegs-names-def*
by *auto*

lemma *name-CapRegs-not-privileged*[*simp*]:
assumes *r* ∈ *set CapRegs*
shows *name r* ≠ *''PCC''*
name r ≠ *''EPCC''*

$name\ r \neq ''ErrorEPCC''$
$name\ r \neq ''KDC''$
$name\ r \neq ''KCC''$
$name\ r \neq ''KR1C''$
$name\ r \neq ''KR2C''$
$name\ r \neq ''CapCause''$
$name\ r \neq ''CPLR''$
using *assms*
by (*auto dest*: *name-CapRegs-CapRegs-names*)

lemma *TLBEntries-names-unfold*[*simp*]:
TLBEntries-names =
{''TLBEntry63'', ''TLBEntry62'', ''TLBEntry61'', ''TLBEntry60'', ''TLBEntry59'',
''TLBEntry58'', ''TLBEntry57'', ''TLBEntry56'', ''TLBEntry55'', ''TLBEntry54'',
''TLBEntry53'', ''TLBEntry52'', ''TLBEntry51'', ''TLBEntry50'', ''TLBEntry49'',
''TLBEntry48'', ''TLBEntry47'', ''TLBEntry46'', ''TLBEntry45'', ''TLBEntry44'',
''TLBEntry43'', ''TLBEntry42'', ''TLBEntry41'', ''TLBEntry40'', ''TLBEntry39'',
''TLBEntry38'', ''TLBEntry37'', ''TLBEntry36'', ''TLBEntry35'', ''TLBEntry34'',
''TLBEntry33'', ''TLBEntry32'', ''TLBEntry31'', ''TLBEntry30'', ''TLBEntry29'',
''TLBEntry28'', ''TLBEntry27'', ''TLBEntry26'', ''TLBEntry25'', ''TLBEntry24'',
''TLBEntry23'', ''TLBEntry22'', ''TLBEntry21'', ''TLBEntry20'', ''TLBEntry19'',
''TLBEntry18'', ''TLBEntry17'', ''TLBEntry16'', ''TLBEntry15'', ''TLBEntry14'',
''TLBEntry13'', ''TLBEntry12'', ''TLBEntry11'', ''TLBEntry10'', ''TLBEntry09'',
''TLBEntry08'', ''TLBEntry07'', ''TLBEntry06'', ''TLBEntry05'', ''TLBEntry04'',
''TLBEntry03'', ''TLBEntry02'', ''TLBEntry01'', ''TLBEntry00''}
unfolding *TLBEntries-def register-defs TLBEntries-names-def*
by *auto*

lemma *ref-name-not-PCC*[*simp*]:
$name\ CapCause\text{-}ref \neq ''PCC''$
$name\ CP0Cause\text{-}ref \neq ''PCC''$
$name\ CP0Status\text{-}ref \neq ''PCC''$
$name\ TLBEntryHi\text{-}ref \neq ''PCC''$
$name\ TLBEntryLo0\text{-}ref \neq ''PCC''$
$name\ TLBEntryLo1\text{-}ref \neq ''PCC''$
$name\ TLBContext\text{-}ref \neq ''PCC''$
$name\ TLBXContext\text{-}ref \neq ''PCC''$
by (*auto simp*: *register-defs*)

lemma *uint6-upper-bound*[*simp*]: $uint\ (idx :: 6\ word) \leq 63$
using *uint-bounded*[*of idx*]
by *auto*

lemma *upto-63-unfold*:
$\{0..63\} = \{0 :: int, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,$
$20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,$
$40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,$

57, *58*, *59*,
        60, *61*, *62*, *63*}
  by *eval*

lemma *TLBEntry-name-not-PCC*[*simp*]:
  assumes *idx* ∈ {*0..63*}
  shows *name* (*TLBEntries* ! (*64* − *nat* (*idx* + *1*))) ≠ *''PCC''*
  using *assms*
  unfolding *upto-63-unfold*
  by (*auto simp*: *TLBEntries-def register-defs*)

lemma *upto-31-unfold*: {*0..31*} = {*0* :: *int*, *1*, *2*, *3*, *4*, *5*, *6*, *7*, *8*, *9*, *10*, *11*, *12*, *13*, *14*, *15*, *16*, *17*, *18*, *19*, *20*, *21*, *22*, *23*, *24*, *25*, *26*, *27*, *28*, *29*, *30*, *31*}
  by *eval*

lemma [*simp*]: *uint* (*idx* :: *5 word*) ≤ *31*
  using *uint-bounded*[*of idx*]
  by *auto*

lemma [*simp*]: *caps-of-regval ISA* (*Regval-Capability c*) = {*c*}
  by (*auto simp*: *ISA-def*)

lemma [*simp*]: *bits-of-mem-bytes* (*mem-bytes-of-word* (*capToMemBits c*)) = *map bitU-of-bool* (*to-bl* (*capToMemBits c*))
  unfolding *mem-bytes-of-word-def bits-of-mem-bytes-def bits-of-bytes-def*
  by (*auto simp*: *append-assoc*[*symmetric*] *take-add*[*symmetric*] *simp del*: *append-assoc*)

lemma [*simp*]: *of-bits-method BC-mword* (*bits-of-mem-bytes* (*mem-bytes-of-word* (*capToMemBits c*))) = *Some* (*capToMemBits c*)
  by *auto*

lemma *Capability-tag-memBitsToCapability*[*simp*]:
  Capability-tag (*memBitsToCapability tag c*) = *tag*
  by (*auto simp*: *memBitsToCapability-def capBitsToCapability-def*)

lemma *Run-throw-False*[*simp*]: *Run* (*throw e*) *t a* ⟷ *False*
  by (*auto simp*: *throw-def*)

lemma *Run-SignalException-False*[*simp*]:
  Run (*SignalException e*) *t a* ⟷ *False*
  by (*auto simp*: *SignalException-def elim*!: *Run-bindE*)

lemma *Run-SignalException-wrappers-False*[*simp*]:
  Run (*SignalExceptionTLB ex badAddr*) *t a* ⟷ *False*
  Run (*SignalExceptionBadAddr ex badAddr*) *t a* ⟷ *False*
  by (*auto simp*: *SignalExceptionTLB-def SignalExceptionBadAddr-def elim*!: *Run-bindE*)

lemma *Run-raise-c2-exception-False*[*simp*]:
  Run (*raise-c2-exception8 capEx reg8*) *t a* ⟷ *False*

$Run\ (raise\text{-}c2\text{-}exception\ capEx\ reg5)\ t\ a \longleftrightarrow False$
$Run\ (raise\text{-}c2\text{-}exception\text{-}noreg\ capEx)\ t\ a \longleftrightarrow False$
by (*auto simp*: *raise-c2-exception8-def raise-c2-exception-def raise-c2-exception-noreg-def elim*!: *Run-bindE*)

lemma *Done-eq-bind-iff*:
$Done\ a = (m \gg= f) \longleftrightarrow (\exists\, a'.\ m = Done\ a' \wedge f\ a' = Done\ a)$
$(m \gg= f) = Done\ a \longleftrightarrow (\exists\, a'.\ m = Done\ a' \wedge f\ a' = Done\ a)$
by (*cases m*; *auto*)+

lemma *Exception-eq-bind-iff*:
$Exception\ e = (m \gg= f) \longleftrightarrow (m = Exception\ e \vee (\exists\, a.\ m = Done\ a \wedge f\ a = Exception\ e))$
$(m \gg= f) = Exception\ e \longleftrightarrow (m = Exception\ e \vee (\exists\, a.\ m = Done\ a \wedge f\ a = Exception\ e))$
by (*cases m*; *auto*)+

lemma *read-reg-no-ex*: $(read\text{-}reg\ r,\ t,\ Exception\ e) \in Traces \longleftrightarrow False$
by (*auto simp*: *read-reg-def elim*: *Read-reg-TracesE split*: *option.splits*)

lemma [*simp*]: $bit\text{-}to\text{-}bool\ (bitU\text{-}of\text{-}bool\ b) = b$
by (*auto simp*: *bitU-of-bool-def*)

lemma *to-bl-bool-to-bits*: $to\text{-}bl\ (bool\text{-}to\text{-}bits\ b) = [b]$
by (*auto simp*: *bool-to-bits-def*) *eval*

lemma *memBitsToCapability-capToMemBits*[*simp*]:
$memBitsToCapability\ tag\ (capToMemBits\ c) = c(\!|Capability\text{-}tag := tag|\!)$
unfolding *memBitsToCapability-def capToMemBits-def capToBits-def capBitsToCapability-def*
by (*auto simp*: *word-bw-assocs subrange-vec-dec-subrange-list-dec slice-take word-cat-bl of-bl-append-same getCapPerms-def getCapHardPerms-def test-bit-of-bl nth-append append-assoc*[*symmetric*]
simp del: *append-assoc*)
(*auto simp*: *to-bl-bool-to-bits*)

lemma [*simp*]: $Capability\text{-}tag\ c \Longrightarrow c(\!|Capability\text{-}tag := True|\!) = c$
by (*cases c*) *auto*

end

locale *CHERI-MIPS-Axiom-Automaton* = *CHERI-MIPS-ISA* +
fixes *enabled* :: (*Capability*, *register-value*) *axiom-state* ⇒ *register-value event* ⇒ *bool*
begin

sublocale *Cap-Axiom-Automaton CC ISA enabled* ..

lemma *non-cap-exp-undefineds*[*non-cap-expI*]:
non-cap-exp (*undefined-unit u*)
non-cap-exp (*undefined-string u*)
non-cap-exp (*undefined-int u*)
non-cap-exp (*undefined-range x y*)
non-cap-exp (*undefined-bitvector n*)
unfolding *undefined-unit-def undefined-string-def undefined-int-def undefined-bitvector-def undefined-range-def*
by *non-cap-expI*

lemma *non-cap-exp-barrier*[*non-cap-expI*]:
non-cap-exp (*barrier b*)
unfolding *barrier-def non-cap-exp-def*
by (*auto elim*: *Traces-cases*)

lemma *non-cap-exp-skip*[*non-cap-expI*]:
non-cap-exp (*skip u*)
unfolding *skip-def*
by *non-cap-expI*

lemma *non-cap-exp-maybe-fail*[*non-cap-expI*]:
non-cap-exp (*maybe-fail msg x*)
unfolding *maybe-fail-def non-cap-exp-def*
by (*auto split*: *option.splits*)

lemma *non-cap-exp-shift-bits*[*non-cap-expI*]:
non-cap-exp (*shift-bits-left BCa BCb BCd v n*)
non-cap-exp (*shift-bits-right BCa BCb BCd v n*)
non-cap-exp (*shift-bits-right-arith BCa BCb BCd v n*)
unfolding *shift-bits-left-def shift-bits-right-def shift-bits-right-arith-def*
by *non-cap-expI*

lemma *no-cap-regvals*[*simp*]:
$\bigwedge v.$ *bitvector-1-dec-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *bitvector-3-dec-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *bitvector-6-dec-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *bitvector-8-dec-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *bitvector-16-dec-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *bitvector-32-dec-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *bitvector-64-dec-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *CauseReg-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *StatusReg-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *ContextReg-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *XContextReg-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *int-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *TLBEntry-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *TLBEntryHiReg-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge v.$ *TLBEntryLoReg-of-regval rv* = *Some v* $\Longrightarrow$ *caps-of-regval ISA rv* = {}
$\bigwedge xs.$ *vector-of-regval of-rv rv* = *Some xs* $\Longrightarrow$ *caps-of-regval ISA rv* = {}

$\bigwedge xs.$ *caps-of-regval ISA* (*regval-of-vector rv-of xs*) = {}
by (*cases rv*; *auto simp*: *ISA-def vector-of-regval-def regval-of-vector-def*)+

lemma *non-cap-reg-nth-TLBEntries*[*intro*, *simp*]:
assumes *idx* ∈ {*0..63*}
shows *non-cap-reg* (*TLBEntries* ! (*64* − *nat* (*idx* + *1*)))
using *assms*
unfolding *upto-63-unfold*
by (*elim insertE*) (*auto simp*: *TLBEntries-def register-defs non-cap-reg-def*)

lemma *non-cap-exp-read-reg-access-TLBEntries*[*non-cap-expI*]:
assumes *idx* ∈ {*0..63*}
shows *non-cap-exp* (*read-reg* (*access-list-dec TLBEntries idx*))
using *assms*
by *non-cap-expI*

lemma *no-reg-writes-to-case-option*[*no-reg-writes-toI*]:
assumes $\bigwedge a.$ *no-reg-writes-to Rs* (*f a*)
and *no-reg-writes-to Rs m*
shows *no-reg-writes-to Rs* (*case x of Some a* ⇒ *f a* | *None* ⇒ *m*)
using *assms*
by (*cases x*) *auto*

lemma *no-reg-writes-to-undefineds*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-unit u*)
no-reg-writes-to Rs (*undefined-string u*)
no-reg-writes-to Rs (*undefined-int u*)
no-reg-writes-to Rs (*undefined-range x y*)
no-reg-writes-to Rs (*undefined-bitvector n*)
unfolding *undefined-unit-def undefined-string-def undefined-int-def undefined-range-def undefined-bitvector-def*
by (*no-reg-writes-toI*)+

lemma *no-reg-writes-to-barrier*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*barrier b*)
unfolding *barrier-def no-reg-writes-to-def*
by (*auto elim*: *Traces-cases*)

lemma *no-reg-writes-to-skip*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*skip u*)
unfolding *skip-def*
by *no-reg-writes-toI*

lemma *no-reg-writes-to-maybe-fail*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*maybe-fail msg x*)
unfolding *maybe-fail-def non-cap-exp-def*
by (*auto split*: *option.splits*)

lemma *no-reg-writes-to-shift-bits*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*shift-bits-left BCa BCb BCd v n*)
no-reg-writes-to Rs (*shift-bits-right BCa BCb BCd v n*)
no-reg-writes-to Rs (*shift-bits-right-arith BCa BCb BCd v n*)
unfolding *shift-bits-left-def shift-bits-right-def shift-bits-right-arith-def*
by *no-reg-writes-toI*+

lemma *no-reg-writes-to-write-ram*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*write-ram arg0 arg1 arg2 arg3 arg4*)
unfolding *write-ram-def MEMea-def MEMval-def*
by *no-reg-writes-toI*

lemma *no-reg-writes-to-read-ram*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*read-ram arg0 arg1 arg2 arg3*)
unfolding *read-ram-def MEMr-def*
by *no-reg-writes-toI*

lemma *no-reg-writes-to-read-memt-bytes*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*read-memt-bytes BCa BCb rk addr sz*)
unfolding *read-memt-bytes-def maybe-fail-def*
by (*auto simp*: *no-reg-writes-to-def elim*: *bind-Traces-cases Traces-cases split*: *option.splits*)

lemma *no-reg-writes-to-read-memt*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*read-memt BCa BCb rk addr sz*)
unfolding *read-memt-def*
by *no-reg-writes-toI*

lemma *no-reg-writes-to-write-memt*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*write-memt BCa BCb wk addr sz v t*)
unfolding *write-memt-def*
by (*auto simp*: *no-reg-writes-to-def elim*: *Traces-cases split*: *option.splits*)

lemma *no-reg-writes-to-MEMval-tagged*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMval-tagged addr sz t v*)
unfolding *MEMval-tagged-def*
by *no-reg-writes-toI*

lemma *no-reg-writes-to-MEMval-tagged-conditional*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMval-tagged-conditional addr sz t v*)
unfolding *MEMval-tagged-conditional-def*
by *no-reg-writes-toI*

lemma *runs-no-reg-writes-to-SignalException*[*runs-no-reg-writes-toI*]:
runs-no-reg-writes-to Rs (*SignalException ex*)
unfolding *runs-no-reg-writes-to-def*
by *auto*

lemma *runs-no-reg-writes-to-raise-c2-exception*[*runs-no-reg-writes-toI*]:
runs-no-reg-writes-to Rs (*raise-c2-exception8 capEx reg8*)
runs-no-reg-writes-to Rs (*raise-c2-exception capEx reg5*)
runs-no-reg-writes-to Rs (*raise-c2-exception-noreg capEx*)
by (*auto simp*: *runs-no-reg-writes-to-def*)

lemma *runs-no-reg-writes-to-checkCP0AccessHook*[*runs-no-reg-writes-toI*]:
runs-no-reg-writes-to Rs (*checkCP0AccessHook u*)
unfolding *checkCP0AccessHook-def pcc-access-system-regs-def*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-writeCapReg*[*no-reg-writes-toI*, *simp*]:
assumes *CapRegs-names* $\cap$ *Rs* = {}
shows *no-reg-writes-to Rs* (*writeCapReg arg0 arg1*)
using *assms name-CapRegs-CapRegs-names*[*of access-list-dec CapRegs* (*uint arg0*)]
unfolding *writeCapReg-def bind-assoc capToString-def*
by (*intro no-reg-writes-toI*) (*auto simp del*: *CapRegs-names-unfold*)

lemma *no-reg-writes-to-readCapReg*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*readCapReg arg0*)
unfolding *readCapReg-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-readCapRegDDC*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*readCapRegDDC arg0*)
unfolding *readCapRegDDC-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *non-mem-exp-rwCapReg*[*non-mem-expI*]:
non-mem-exp (*readCapReg r*)
non-mem-exp (*readCapRegDDC r*)
non-mem-exp (*writeCapReg r v*)
by (*non-mem-expI simp*: *readCapReg-def readCapRegDDC-def writeCapReg-def capToString-def*)

declare *MemAccessType.split*[**where** P = λm. *no-reg-writes-to Rs m* **for** *Rs*, *THEN iffD2*, *no-reg-writes-toI*]
declare *MemAccessType.split*[*split*]
declare *WordType.split*[**where** P = λm. *no-reg-writes-to Rs m* **for** *Rs*, *THEN iffD2*, *no-reg-writes-toI*]
declare *WordType.split*[*split*]
declare *ClearRegSet.split*[**where** P = λm. *no-reg-writes-to Rs m* **for** *Rs*, *THEN iffD2*, *no-reg-writes-toI*]
declare *ClearRegSet.split*[*split*]

end

locale *CHERI-MIPS-Axiom-Inv-Automaton* = *CHERI-MIPS-Axiom-Automaton* +

Cap-Axiom-Inv-Automaton **where** $CC = CC$ **and** $ISA = ISA$ **and** *get-regval* = *get-regval* **and** *set-regval* = *set-regval*
begin

lemma *preserve-invariant-undefineds*[*preserves-invariantI*]:
  traces-preserve-invariant (*undefined-unit u*)
  traces-preserve-invariant (*undefined-string u*)
  traces-preserve-invariant (*undefined-int u*)
  traces-preserve-invariant (*undefined-range x y*)
  traces-preserve-invariant (*undefined-bitvector n*)
 by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*; *simp*)+

lemma *preserves-invariant-barrier*[*no-reg-writes-toI*, *simp*]:
  traces-preserve-invariant (*barrier b*)
 by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*; *simp*)+

lemma *preserves-invariant-skip*[*no-reg-writes-toI*, *simp*]:
  traces-preserve-invariant (*skip u*)
 by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*; *simp*)+

lemma *preserves-invariant-maybe-fail*[*no-reg-writes-toI*, *simp*]:
  traces-preserve-invariant (*maybe-fail msg x*)
 by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*; *simp*)+

lemma *preserves-invariant-shift-bits*[*no-reg-writes-toI*, *simp*]:
  traces-preserve-invariant (*shift-bits-left BCa BCb BCd v n*)
  traces-preserve-invariant (*shift-bits-right BCa BCb BCd v n*)
  traces-preserve-invariant (*shift-bits-right-arith BCa BCb BCd v n*)
 by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*; *simp*)+

lemma *preserves-invariant-write-ram*[*preserves-invariantI*]:
  traces-preserve-invariant (*write-ram arg0 arg1 arg2 arg3 arg4*)
 by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*; *simp*)

lemma *preserves-invariant-read-ram*[*preserves-invariantI*]:
  traces-preserve-invariant (*read-ram arg0 arg1 arg2 arg3*)
 by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*; *simp*)

lemma *traces-enabled-case-option*[*traces-enabledI*]:
  assumes $\bigwedge a.\ x = Some\ a \Longrightarrow$ *traces-enabled* (*f a*) *s regs*
   and $x = None \Longrightarrow$ *traces-enabled m s regs*
  shows *traces-enabled* (*case x of Some a* $\Rightarrow$ *f a* | *None* $\Rightarrow$ *m*) *s regs*
  using *assms*
  by (*cases x*) *auto*

lemma *Run-inv-ifE*:
  assumes *Run-inv* (*if c then m1 else m2*) *t a regs*

obtains *Run-inv m1 t a regs* **and** *c* | *Run-inv m2 t a regs* **and** $\neg$*c*
using *assms*
by (*auto split*: *if-splits*)

lemma *Run-inv-letE*:
assumes *Run-inv* (*let x* = *y in f x*) *t a regs*
obtains *Run-inv* (*f y*) *t a regs*
using *assms*
by *auto*

declare *Run-inv-ifE*[**where** *t* = *t* **and** *thesis* = *c* $\in$ *derivable-caps* (*run s t*) **for** *s t c*, *derivable-capsE*]
declare *Run-inv-letE*[**where** *t* = *t* **and** *thesis* = *c* $\in$ *derivable-caps* (*run s t*) **for** *s t c*, *derivable-capsE*]

lemma *Run-inv-return*[*simp*]: *Run-inv* (*return a*) *t a' regs* $\longleftrightarrow$ (*a'* = *a* $\wedge$ *t* = [] $\wedge$ *invariant regs*)
unfolding *Run-inv-def*
by *auto*

lemma *null-cap-derivable*[*intro*, *simp*]: *null-cap* $\in$ *derivable-caps s*
unfolding *null-cap-def derivable-caps-def*
by *auto*

lemma *read-reg-access-CapRegs-derivable-caps*[*derivable-capsE*]:
assumes *Run-inv* (*read-reg* (*access-list-dec CapRegs idx*)) *t c regs*
and *idx* $\in$ {*0..31*} **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *c* $\in$ *derivable-caps* (*run s t*)
using *assms*
unfolding *Run-inv-def upto-31-unfold*
by (*elim insertE conjE Run-read-regE*)
(*auto simp*: *CapRegs-def CapRegs-names-def derivable-caps-def register-defs intro*!: *derivable.Copy*)

lemma *memt-builtins-preserve-invariant*[*preserves-invariantI*]:
$\bigwedge$*BCa BCb rk addr sz*. *traces-preserve-invariant* (*read-memt-bytes BCa BCb rk addr sz*)
$\bigwedge$*BCa BCb rk addr sz*. *traces-preserve-invariant* (*read-memt BCa BCb rk addr sz*)
$\bigwedge$*BCa BCb wk addr sz v t*. *traces-preserve-invariant* (*write-memt BCa BCb wk addr sz v t*)
$\bigwedge$*addr sz t v*. *traces-preserve-invariant* (*MEMval-tagged addr sz t v*)
$\bigwedge$*addr sz t v*. *traces-preserve-invariant* (*MEMval-tagged-conditional addr sz t v*)
by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*; *simp*)+

lemma *dvd-8-Suc-iffs*[*simp*]:
8 dvd Suc (*Suc 0*) $\longleftrightarrow$ *False*
8 dvd Suc (*Suc* (*Suc 0*)) $\longleftrightarrow$ *False*
8 dvd Suc (*Suc* (*Suc* (*Suc 0*))) $\longleftrightarrow$ *False*

$8\ dvd\ Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0)))) \longleftrightarrow False$
$8\ dvd\ Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))) \longleftrightarrow False$
$8\ dvd\ Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0)))))) \longleftrightarrow False$
$8\ dvd\ Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ x))))))) \longleftrightarrow 8\ dvd\ x$
by *presburger+*

lemma *byte-chunks-eq-Some-iff* [*simp*]:
shows *byte-chunks xs = Some ys* ⟷ *ys = take-chunks 8 xs* ∧ *8 dvd length xs*
by (*induction xs arbitrary*: *ys rule*: *byte-chunks.induct*) (*auto simp*: *bind-eq-Some-conv*)

lemma *mem-bytes-of-bits-mword-eq-Some-iff* [*simp*]:
fixes *w* :: *'a::len word*
shows *mem-bytes-of-bits BC-mword w = Some bytes* ⟷ *bytes = mem-bytes-of-word w* ∧ *8 dvd LENGTH('a)*
by (*auto simp*: *mem-bytes-of-bits-def bytes-of-bits-def mem-bytes-of-word-def BC-mword-defs*)

lemma *concat-take-chunks*[*simp*]:
assumes $n > 0$
shows *List.concat* (*take-chunks n xs*) = *xs*
using *assms*
by (*induction n xs rule*: *take-chunks.induct*) *auto*

lemma *bits-of-mem-bytes-of-word*[*simp*]:
fixes *w* :: *'a::len word*
assumes *8 dvd LENGTH('a)*
shows *bits-of-mem-bytes* (*mem-bytes-of-word w*) = *map bitU-of-bool* (*to-bl w*)
using *assms*
by (*auto simp add*: *bits-of-mem-bytes-def bits-of-bytes-def mem-bytes-of-word-def simp del*: *take-chunks.simps*)

lemma *bitU-of-bool-eq-iff* [*simp*]:
bitU-of-bool b = B1 ⟷ *b* *bitU-of-bool b = B0* ⟷ ¬*b*
by (*auto simp*: *bitU-of-bool-def*)

lemma *memBitsToCapability-False-derivable-caps*[*intro*, *simp*, *derivable-capsI*]:
shows *memBitsToCapability False w* ∈ *derivable-caps s*
by (*auto simp*: *derivable-caps-def*)

lemma *memBitsToCapability-ucast-256-derivable-caps*[*intro*, *simp*, *derivable-capsI*]:
assumes *memBitsToCapability tag w* ∈ *derivable-caps s*
shows *memBitsToCapability tag* (*ucast w*) ∈ *derivable-caps s*
using *assms*
by *auto*

lemma *memBitsToCapability-capToMemBits-derivable-caps*[*intro*, *derivable-capsI*]:
assumes *c*: *c* ∈ *derivable-caps s* **and** *tag*: *tag* ⟶ *Capability-tag c*
shows *memBitsToCapability tag* (*capToMemBits c*) ∈ *derivable-caps s*
using *assms*
by (*cases tag*) (*auto simp*: *derivable-caps-def*)

lemma *read-from-KCC-run-mono*: *read-from-KCC s* $\subseteq$ *read-from-KCC* (*run s t*)
proof (*induction t arbitrary*: *s*)
  case (*Cons e t*)
  have *read-from-KCC s* $\subseteq$ *read-from-KCC* (*axiom-step s e*)
    by *auto*
  also have ... $\subseteq$ *read-from-KCC* (*run* (*axiom-step s e*) *t*)
    by (*rule Cons.IH*)
  finally show *?case*
    unfolding *foldl-Cons* **.**
qed *auto*

lemma *exception-targets-run-imp*:
  assumes *c* $\in$ *exception-targets ISA* (*read-from-KCC s*)
  shows *c* $\in$ *exception-targets ISA* (*read-from-KCC* (*run s t*))
  using *assms read-from-KCC-run-mono*
  by (*auto simp*: *ISA-def*)

lemma *exception-targets-insert*[*simp*]:
  exception-targets ISA (*insert* (*Regval-Capability c*) *C*) = *insert c* (*exception-targets ISA C*)
  by (*auto simp*: *ISA-def*)

lemma *read-reg-KCC-exception-targets*:
  assumes *Run-inv* (*read-reg KCC-ref*) *t c regs*
  shows *c* $\in$ *exception-targets ISA* (*read-from-KCC* (*run s t*))
  using *assms*
  unfolding *Run-inv-def*
  by (*auto elim*!: *Run-read-regE simp*: *KCC-ref-def*)

lemma *leq-perms-refl*[*intro*, *simp*]: *leq-perms p p*
  unfolding *leq-perms-def*
  by *auto*

lemma *setCapOffset-getCapOffset-idem*:
  assumes *setCapOffset c offset* = (*representable*, *c'*)
    and *uint offset* = *getCapOffset c*
  shows *c'* = *c*
  using *assms uint-bounded*[*of Capability-address c*]
  by (*cases c*)
    (*auto simp add*: *setCapOffset-def getCapOffset-def uint-word-ariths mod-add-right-eq simp flip*: *uint-inject*)

lemma *setCapOffset-derivable-caps*[*derivable-capsE*]:
  assumes *setCapOffset c offset* = (*representable*, *c'*)
   and *Capability-tag c* $\Longrightarrow$ *Capability-tag c'* $\Longrightarrow$ *Capability-sealed c* $\wedge$ *Capability-sealed c'* $\Longrightarrow$ *uint offset* = *getCapOffset c*
    and *c* $\in$ *derivable-caps s*

shows $c' \in$ *derivable-caps s*
proof −
have *leq-cap CC* c' *c*
using *assms setCapOffset-getCapOffset-idem*[*OF assms*(*1*)]
by (*auto simp*: *leq-cap-def setCapOffset-def getCapBase-def getCapTop-def get-cap-perms-def*)
then show *?thesis*
using *assms*
by (*auto simp*: *derivable-caps-def setCapOffset-def elim*: *derivable.Restrict*)
qed

lemma *Run-inv-return-derivable-caps*[*derivable-capsE*]:
assumes *Run-inv* (*return a*) *t* a' *regs* **and** $a \in$ *derivable-caps s*
shows $a' \in$ *derivable-caps* (*run s t*) **and** $a' \in$ *derivable-caps s*
using *assms*
by *auto*

lemma *Run-inv-bind-derivable-caps*[*derivable-capsE*]:
assumes *Run-inv* ($m \gg= f$) *t a regs* **and** *runs-preserve-invariant m*
and $\bigwedge$*tm am tf*. *t* = *tm* @ *tf* $\Longrightarrow$ *Run-inv m tm am regs* $\Longrightarrow$ *Run-inv* (*f am*) *tf a* (*the* (*updates-regs inv-regs tm regs*)) $\Longrightarrow$ $c \in$ *derivable-caps* (*run* (*run s tm*) *tf*)
shows $c \in$ *derivable-caps* (*run s t*)
using *assms*
by (*elim Run-inv-bindE*) *auto*

lemma *int-to-cap-derivable-caps*[*derivable-capsI*]:
unrepCap c $\in$ *derivable-caps s*
by (*auto simp*: *unrepCap-def derivable-caps-def*)

lemma *update-Capability-tag-derivable-caps*[*derivable-capsI*]:
assumes *t* $\Longrightarrow$ $c \in$ *derivable-caps s* **and** *t* $\Longrightarrow$ *Capability-tag c*
shows *c*⦇*Capability-tag* := *t*⦈ $\in$ *derivable-caps s*
using *assms*
by (*cases Capability-tag c*) (*auto simp*: *derivable-caps-def*)

lemma *preserves-invariant-readCapReg*[*preserves-invariantI*]:
$\bigwedge$*arg0*. *traces-preserve-invariant* (*readCapReg arg0*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*readCapRegDDC arg0*)
by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-toI*; *simp*)+

lemma *readCapReg-derivable*[*derivable-capsE*]:
assumes *Run-inv* (*readCapReg arg0*) *t c regs* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows $c \in$ *derivable-caps* (*run s t*)
using *assms*
unfolding *readCapReg-def*
by (−) (*derivable-capsI assms*: *assms*)

lemma *readCapRegDDC-derivable*[*derivable-capsE*]:
assumes *Run-inv* (*readCapRegDDC arg0*) *t c regs* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows $c \in$ *derivable-caps* (*run s t*)
using *assms*
unfolding *readCapRegDDC-def*
by (−) (*derivable-capsI assms*: *assms*)

lemma *caps-of-CapCauseReg-empty*[*simp*]: *caps-of-regval ISA* (*regval-of-CapCauseReg r*) = {}
by (*auto simp*: *ISA-def regval-of-CapCauseReg-def*)

lemma *letI*: *P* (*let x* = *y in f x*) **if** *P* (*f y*)
using *that*
by *auto*

declare *if-split*[**where** *P* = λm. *runs-preserve-invariant m*, *THEN iffD2*, *preserves-invariantI*]
declare *option.split*[**where** *P* = λm. *runs-preserve-invariant m*, *THEN iffD2*, *preserves-invariantI*]
declare *prod.split*[**where** *P* = λm. *runs-preserve-invariant m*, *THEN iffD2*, *preserves-invariantI*]
declare *sum.split*[**where** *P* = λm. *runs-preserve-invariant m*, *THEN iffD2*, *preserves-invariantI*]
declare *letI*[**where** *P* = λm. *runs-preserve-invariant m*, *preserves-invariantI*]

declare *MemAccessType.split*[**where** *P* = λm. *traces-enabled m s regs* **for** *s regs*, *traces-enabled-split*]
declare *MemAccessType.split*[**where** *P* = λm. *runs-preserve-invariant m* **for** *Rs*, *THEN iffD2*, *preserves-invariantI*]
declare *MemAccessType.split*[**where** *P* = λm. *traces-preserve-invariant m* **for** *Rs*, *THEN iffD2*, *preserves-invariantI*]
declare *WordType.split*[**where** *P* = λm. *traces-enabled m s regs* **for** *s regs*, *traces-enabled-split*]
declare *WordType.split*[**where** *P* = λm. *runs-preserve-invariant m* **for** *Rs*, *THEN iffD2*, *preserves-invariantI*]
declare *WordType.split*[**where** *P* = λm. *traces-preserve-invariant m* **for** *Rs*, *THEN iffD2*, *preserves-invariantI*]
declare *ClearRegSet.split*[**where** *P* = λm. *traces-enabled m s regs* **for** *s regs*, *traces-enabled-split*]
declare *ClearRegSet.split*[**where** *P* = λm. *runs-preserve-invariant m* **for** *Rs*, *THEN iffD2*, *preserves-invariantI*]
declare *ClearRegSet.split*[**where** *P* = λm. *traces-preserve-invariant m* **for** *Rs*, *THEN iffD2*, *preserves-invariantI*]

lemma *preserves-invariant-SignalException*[*preserves-invariantI*]:
runs-preserve-invariant (*SignalException ex*)
runs-preserve-invariant (*SignalExceptionBadAddr ex badAddr*)
runs-preserve-invariant (*SignalExceptionTLB ex badAddr*)
by (*auto simp*: *runs-preserve-invariant-def*)

lemma *Run-raise-c2-exception-False*[*simp*]:

Run-inv (*raise-c2-exception8 capEx reg8*) *t a regs* ⟷ *False*
Run-inv (*raise-c2-exception capEx reg5*) *t a regs* ⟷ *False*
Run-inv (*raise-c2-exception-noreg capEx*) *t a regs* ⟷ *False*
by (*auto simp*: *Run-inv-def*)

lemma *runs-preserve-invariant-raise-c2-exception*[*preserves-invariantI*]:
runs-preserve-invariant (*raise-c2-exception8 capEx reg8*)
runs-preserve-invariant (*raise-c2-exception capEx reg5*)
runs-preserve-invariant (*raise-c2-exception-noreg capEx*)
by (*auto simp*: *runs-preserve-invariant-def*)

end

locale *CHERI-MIPS-Reg-Automaton* = *CHERI-MIPS-ISA* +
fixes *ex-traces* :: *bool* **and** *invocation-traces* :: *bool*
begin

abbreviation *invariant* **where** *invariant regs* ≡ *Capability-tag* (*regstate.PCC regs*) ∧ ¬*Capability-sealed* (*regstate.PCC regs*)
abbreviation *inv-regs* :: *register-name set* **where** *inv-regs* ≡ {*''PCC''*}

sublocale *Write-Cap-Inv-Automaton CC ISA ex-traces invocation-traces get-regval set-regval invariant inv-regs* **..**

sublocale *CHERI-MIPS-Axiom-Inv-Automaton* **where** *enabled* = *enabled* **and** *invariant* = *invariant* **and** *inv-regs* = *inv-regs* **..**

lemma *traces-enabled-read-reg-nth-CapRegs*[*traces-enabledI*]:
assumes *idx* ∈ {*0..31*}
shows *traces-enabled* (*read-reg* (*access-list-dec CapRegs idx*)) *s regs*
using *assms*
unfolding *upto-31-unfold*
by (*elim insertE*) (*auto simp*: *CapRegs-def intro*!: *traces-enabled-read-reg*)

lemma *preserves-invariant-writeCapReg*[*preserves-invariantI*]:
⋀*arg0 arg1*. *traces-preserve-invariant* (*writeCapReg arg0 arg1*)
by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-toI*; *simp*)+

lemma *traces-enabled-read-mem*[*traces-enabledI*]:
traces-enabled (*read-mem BCa BCb rk addr-sz addr sz*) *s regs*
unfolding *read-mem-def read-mem-bytes-def traces-enabled-def maybe-fail-def*
by (*auto elim*: *bind-Traces-cases Traces-cases split*: *option.splits*)

lemma *traces-enabled-read-memt*[*traces-enabledI*]:
traces-enabled (*read-memt BCa BCb rk addr sz*) *s regs*
unfolding *read-memt-def read-memt-bytes-def traces-enabled-def maybe-fail-def*
by (*auto elim*: *bind-Traces-cases Traces-cases split*: *option.splits*)

lemma *traces-enabled-write-mem-ea*[*traces-enabledI*]:

traces-enabled (*write-mem-ea BCa wk a1 a2 a3*) *s regs*
unfolding *write-mem-ea-def traces-enabled-def maybe-fail-def*
by (*auto elim*: *bind-Traces-cases Traces-cases split*: *option.splits*)

lemma *traces-enabled-write-mem*[*traces-enabledI*]:
traces-enabled (*write-mem BCa BCb wk a1 a2 a3 a4*) *s regs*
unfolding *write-mem-def traces-enabled-def*
by (*auto elim*: *bind-Traces-cases Traces-cases split*: *option.splits*)

lemma *traces-enabled-write-memt*[*traces-enabledI*]:
assumes *tag* = *B1* ⟶ *memBitsToCapability True* (*ucast w*) ∈ *derivable-caps s*
shows *traces-enabled* (*write-memt BCa BC-mword wk addr sz w tag*) *s regs*
using *assms*
unfolding *write-memt-def traces-enabled-def*
by (*cases tag*; *auto split*: *option.splits simp*: *bind-eq-Some-conv ucast-bl derivable-caps-def elim*!: *Write-memt-TracesE*)

lemma *traces-enabled-write-ram*[*traces-enabledI*]:
traces-enabled (*write-ram a0 a1 a2 a3 a4*) *s regs*
unfolding *write-ram-def MEMval-def MEMea-def*
by (*traces-enabledI intro*: *non-cap-expI*[*THEN non-cap-exp-traces-enabledI*])

lemma *traces-enabled-read-ram*[*traces-enabledI*]:
traces-enabled (*read-ram a0 a1 a2 a3*) *s regs*
unfolding *read-ram-def MEMr-def*
by (*traces-enabledI*)

lemma *traces-enabled-MEMval-tagged*[*traces-enabledI*]:
assumes *memBitsToCapability tag* (*ucast v*) ∈ *derivable-caps s*
shows *traces-enabled* (*MEMval-tagged addr sz tag v*) *s regs*
unfolding *MEMval-tagged-def*
by (*traces-enabledI intro*: *non-cap-expI*[*THEN non-cap-exp-traces-enabledI*] *assms*: *assms*)

lemma *traces-enabled-MEMval-tagged-conditional*[*traces-enabledI*]:
assumes *memBitsToCapability tag* (*ucast v*) ∈ *derivable-caps s*
shows *traces-enabled* (*MEMval-tagged-conditional addr sz tag v*) *s regs*
unfolding *MEMval-tagged-conditional-def*
by (*traces-enabledI intro*: *non-cap-expI*[*THEN non-cap-exp-traces-enabledI*] *assms*: *assms*)

lemma *traces-enabled-set-next-pcc-ex*:
assumes *arg0*: *arg0* ∈ *exception-targets ISA* (*read-from-KCC s*) **and** *ex*: *ex-traces*
shows *traces-enabled* (*set-next-pcc arg0*) *s regs*
unfolding *set-next-pcc-def bind-assoc*
by (*traces-enabledI assms*: *arg0 exception-targets-run-imp*
intro: *traces-enabled-write-reg ex no-reg-writes-traces-preserve-invariantI*

no-reg-writes-to-write-reg traces-runs-preserve-invariantI
simp: *DelayedPCC-ref-def NextPCC-ref-def*)

lemma *traces-enabled-write-reg-nth-CapRegs*[*traces-enabledI*]:
assumes $c \in$ *derivable-caps s* **and** *idx* $\in \{0..31\}$
shows *traces-enabled* (*write-reg* (*access-list-dec CapRegs idx*) *c*) *s regs*
using *assms*
unfolding *upto-31-unfold derivable-caps-def*
by (*elim insertE*; *auto intro*!: *traces-enabled-write-reg simp*: *CapRegs-def register-defs*)

lemma *traces-enabled-writeCapReg*[*traces-enabledI*]:
assumes *arg1* $\in$ *derivable-caps s*
shows *traces-enabled* (*writeCapReg arg0 arg1*) *s regs*
unfolding *writeCapReg-def bind-assoc capToString-def*
by (*traces-enabledI assms*: *assms intro*: *non-cap-expI*[*THEN non-cap-exp-traces-enabledI*] *no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*)

lemma *traces-enabled-readCapReg*[*traces-enabledI*]:
shows *traces-enabled* (*readCapReg arg0*) *s regs*
unfolding *readCapReg-def bind-assoc*
by (*traces-enabledI intro*: *non-cap-expI*[*THEN non-cap-exp-traces-enabledI*])

lemma *traces-enabled-readCapRegDDC*[*traces-enabledI*]:
shows *traces-enabled* (*readCapRegDDC arg0*) *s regs*
unfolding *readCapRegDDC-def bind-assoc*
by (*traces-enabledI*)

fun *trace-writes-cap-regs* :: *register-value trace* $\Rightarrow$ *register-name set* **where**
trace-writes-cap-regs [] = {}
| *trace-writes-cap-regs* (*e* # *t*) =
{*r*. $\exists$ *v c*. *e* = *E-write-reg r v* $\wedge$ *c* $\in$ *caps-of-regval ISA v* $\wedge$ *is-tagged-method CC c*} $\cup$
trace-writes-cap-regs t

fun *trace-allows-system-reg-access* :: *register-name set* $\Rightarrow$ *register-value trace* $\Rightarrow$ *bool* **where**
trace-allows-system-reg-access Rs [] = *False*
| *trace-allows-system-reg-access Rs* (*e* # *t*) = (*allows-system-reg-access Rs e* $\vee$ *trace-allows-system-reg-access* (*Rs* $-$ *trace-writes-cap-regs* [*e*]) *t*)

lemma *trace-allows-system-reg-access-append*:
trace-allows-system-reg-access Rs (*t1* @ *t2*) = (*trace-allows-system-reg-access Rs t1* $\vee$ *trace-allows-system-reg-access* (*Rs* $-$ *trace-writes-cap-regs t1*) *t2*)
by (*induction t1 arbitrary*: *Rs*) (*auto simp*: *Diff-eq Int-assoc*)

lemma [*simp*]: *accessible-regs s* $-$ *written-regs s* = *accessible-regs s*
by (*auto simp*: *accessible-regs-def*)

lemma *system-reg-access-run*:
system-reg-access (*run s t*) = (*system-reg-access s* ∨ *trace-allows-system-reg-access* (*accessible-regs s*) *t*)
by (*induction t arbitrary*: *s*) (*auto simp*: *accessible-regs-axiom-step Diff-Un Diff-Int-distrib Diff-Int*)

lemma *pcc-access-system-regs-allows-system-reg-access*:
assumes *Run-inv* (*pcc-access-system-regs u*) *t a regs*
shows *trace-allows-system-reg-access Rs t* ⟷ *a* ∧ *''PCC''* ∈ *Rs*
using *assms*
unfolding *pcc-access-system-regs-def Run-inv-def*
by (*auto elim*!: *Run-bindE Run-read-regE simp*: *PCC-ref-def get-regval-def regval-of-Capability-def get-cap-perms-def*)

lemma *checkCP0Access-system-reg-access*:
assumes *Run-inv* (*checkCP0Access* ()) *t* () *regs* **and** {*''PCC''*} ⊆ *accessible-regs s*
shows *trace-allows-system-reg-access* (*accessible-regs s*) *t*
using *assms pcc-access-system-regs-allows-system-reg-access*[**where** *Rs* = *accessible-regs s*]
unfolding *checkCP0Access-def checkCP0AccessHook-def Run-inv-def*
by (*auto elim*!: *Run-bindE simp*: *regstate-simp system-reg-access-run pcc-access-system-regs-allows-system-reg trace-allows-system-reg-access-append split*: *if-splits*)

lemma *Run-inv-runs-no-reg-writes-written-regs-eq*:
assumes *Run-inv m t a regs* **and** *runs-no-reg-writes-to* {*r*} *m*
shows *r* ∈ *written-regs* (*run s t*) ⟷ *r* ∈ *written-regs s*
using *assms*
by (*auto simp*: *Run-inv-def Run-runs-no-reg-writes-written-regs-eq*)

lemmas *runs-no-reg-writes-written-regs-eq* =
Run-runs-no-reg-writes-written-regs-eq Run-inv-runs-no-reg-writes-written-regs-eq

end

abbreviation *noCP0Access s* ≡ *get-StatusReg-EXL* (*CP0Status s*) = *0* ∧ *get-StatusReg-ERL* (*CP0Status s*) = *0* ∧ *get-StatusReg-KSU* (*CP0Status s*) ≠ *0* ∧ ¬(*get-StatusReg-CU* (*CP0Status s*) !! *0*)

locale *CHERI-MIPS-Fixed-Trans* =
fixes *trans-regstate* :: *regstate*
assumes *noCP0Access-trans-regstate*: *noCP0Access trans-regstate*
begin

definition *trans-regs* ≡ {*''CP0Status''*, *''TLBEntryHi''*, *''PCC''*} ∪ *TLBEntries-names*
definition *trans-inv s* ≡ (∃ *pcc*. *s*⦇*regstate.PCC* := *pcc*⦈ = *trans-regstate*)

lemma *invariant-trans-regstate*[*intro*, *simp*]:
trans-inv trans-regstate

proof −
have *trans-regstate*(|*regstate.PCC* := *regstate.PCC trans-regstate*|) = *trans-regstate*
by *auto*
then show *?thesis*
unfolding *trans-inv-def*
by *blast*
qed

fun *MemAccessType-of-acctype* :: *acctype* ⇒ *MemAccessType* **where**
MemAccessType-of-acctype Load = *LoadData*
| *MemAccessType-of-acctype Store* = *StoreData*
| *MemAccessType-of-acctype Fetch* = *Instruction*

sublocale *State-Invariant get-regval set-regval trans-inv trans-regs* **.**

definition *translate-addressM* :: *nat* ⇒ *acctype* ⇒ *nat M* **where**
translate-addressM vaddr acctype ≡
let vaddr = *word-of-int* (*int vaddr*) *in*
TLBTranslate vaddr (*MemAccessType-of-acctype acctype*) ≫= (λ*paddr*.
return (*unat paddr*))

definition *translate-address* :: *nat* ⇒ *acctype* ⇒ ′*a* ⇒ *nat option* **where**
translate-address vaddr acctype - = (*if* (∃ *t a regs*. *Run-inv* (*translate-addressM vaddr acctype*) *t a regs*) *then Some* (*the-result* (*translate-addressM vaddr acctype*)) *else None*)

sublocale *CHERI-MIPS-ISA* **where** *translate-address* = *translate-address* **.**

end

locale *CHERI-MIPS-Mem-Automaton* = *CHERI-MIPS-Fixed-Trans* +
fixes *is-fetch* :: *bool* **and** *ex-traces* :: *bool*
begin

sublocale *Mem-Inv-Automaton*
where *CC* = *CC* **and** *ISA* = *ISA* **and** *is-fetch* = *is-fetch* **and** *ex-traces* = *ex-traces*
and *translation-assm* = λ*t*. (∃ *regs*. *reads-regs-from inv-regs t regs* ∧ *trans-inv regs*)
and *get-regval* = *get-regval* **and** *set-regval* = *set-regval*
and *invariant* = *trans-inv* **and** *inv-regs* = *trans-regs*
by *unfold-locales* (*auto simp*: *ISA-def translate-address-def*)

sublocale *CHERI-MIPS-Axiom-Inv-Automaton*
where *translate-address* = *translate-address* **and** *enabled* = *enabled*
and *invariant* = *trans-inv* **and** *inv-regs* = *trans-regs* **and** *ex-traces* = *ex-traces*
by *unfold-locales*

lemma *preserves-invariant-tlbSearch*[*preserves-invariantI*]:
traces-preserve-invariant (*tlbSearch vAddr*)
unfolding *tlbSearch-def*
by (*preserves-invariantI*)

lemma *preserves-invariant-checkCP0AccessHook*[*preserves-invariantI*]:
runs-preserve-invariant (*checkCP0AccessHook u*)
unfolding *checkCP0AccessHook-def pcc-access-system-regs-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-getAccessLevel*[*preserves-invariantI*]:
traces-preserve-invariant (*getAccessLevel u*)
unfolding *getAccessLevel-def*
by (*preserves-invariantI*)

fun-cases *StatusReg-of-regval-SomeE*: *StatusReg-of-regval rv* = *Some a*

lemma *read-reg-CP0Status-inv-fields*:
assumes *Run-inv* (*read-reg CP0Status-ref*) *t a regs*
shows *get-StatusReg-EXL a* = *0* **and** *get-StatusReg-ERL a* = *0* **and** *get-StatusReg-KSU a* ≠ *0*
and ¬(*get-StatusReg-CU a* !! *0*)
using *assms noCP0Access-trans-regstate*
unfolding *Run-inv-def trans-inv-def*
by (*auto elim*!: *Run-read-regE StatusReg-of-regval-SomeE simp*: *CP0Status-ref-def trans-regs-def get-regval-def regval-of-StatusReg-def*)

lemma *bits-to-bool-iff-one*:
bits-to-bool w ⟷ *w* = *1*
by (*cases w rule*: *exhaustive-1-word*) (*auto simp*: *bits-to-bool-def*)

lemma *Run-inv-getAccessLevel-neq-Kernel*:
assumes *Run-inv* (*getAccessLevel u*) *t a regs*
shows *a* ≠ *Kernel*
using *assms*
unfolding *getAccessLevel-def Let-def or-boolM-def*
by (*auto simp*: *regstate-simp bits-to-bool-iff-one read-reg-CP0Status-inv-fields*
elim!: *Run-inv-bindE intro*!: *preserves-invariantI traces-runs-preserve-invariantI split*: *if-splits*)

lemma *Run-inv-checkCP0Access-False*[*simp*]:
Run-inv (*checkCP0Access u*) *t a regs* ⟷ *False*
proof −
define *signal-ex* :: *unit M*
where *signal-ex* ≡ *set-CauseReg-CE CP0Cause-ref 0* ≫ *SignalException CpU*
have *Run-inv signal-ex t a regs* ⟷ *False* **for** *t a regs*
unfolding *signal-ex-def Run-inv-def*
by (*auto elim*!: *Run-bindE*)

then show *?thesis*
unfolding *checkCP0Access-def and-boolM-def bind-assoc signal-ex-def*[*symmetric*]
by (*auto elim*!: *Run-inv-bindE dest*!: *Run-inv-getAccessLevel-neq-Kernel*
intro!: *preserves-invariantI traces-runs-preserve-invariantI*
simp: *read-reg-CP0Status-inv-fields*)
qed

lemma *trans-inv-regs-eq-trans-regstate*:
assumes *trans-inv regs*
shows *CP0Status regs* = *CP0Status trans-regstate* ∧ *TLBEntryHi regs* = *TLBEntryHi trans-regstate*
using *assms*
by (*auto simp*: *trans-inv-def*)

lemma *determ-runs-read-reg-CP0Status*[*determ*]: *determ-runs* (*read-reg CP0Status-ref*)
by (*intro determ-runs-read-inv-reg*) (*auto simp*: *trans-regs-def register-defs dest*: *trans-inv-regs-eq-trans-regstate*)

lemma *determ-runs-SignalExceptionBadAddr*[*determ*]: *determ-runs* (*SignalExceptionBadAddr ex badAddr*)
by (*intro determ-runsI*) (*auto simp*: *Run-inv-def*)

lemma *determ-runs-SignalExceptionTLB*[*determ*]: *determ-runs* (*SignalExceptionTLB ex badAddr*)
by (*intro determ-runsI*) (*auto simp*: *Run-inv-def*)

lemma *get-regval-TLBEntries*:
r ∈ *set TLBEntries* ⟹ *trans-inv regs* ⟹ *get-regval* (*name r*) *regs* = *Some* (*regval-of-TLBEntry* (*read-from r trans-regstate*))
by (*auto simp*: *TLBEntries-def trans-inv-def*; *simp add*: *register-defs*)

lemma *read-from-TLBEntries*:
assumes *idx* ∈ {*0..63*} **and** *trans-inv regs*
shows *read-from* (*TLBEntries* ! (*64* − *nat* (*idx* + *1*))) *trans-regstate* = *read-from* (*TLBEntries* ! (*64* − *nat* (*idx* + *1*))) *regs*
using *assms*
unfolding *upto-63-unfold*
by (*elim insertE*) (*auto simp*: *trans-inv-def TLBEntries-def register-defs*)

lemma *of-regval-TLBEntries-nth*[*simp*]:
idx ∈ {*0..63*} ⟹ *of-regval* (*TLBEntries* ! (*64* − *nat* (*idx* + *1*))) *v* = *TLBEntry-of-regval v*
unfolding *upto-63-unfold*
by (*elim insertE*) (*auto simp*: *TLBEntries-def register-defs*)

lemma *determ-runs-read-reg-access-TLBEntries*[*determ*]:
determ-traces (*read-reg* (*access-list-dec TLBEntries idx*)) **if** *idx* ∈ {*0..63*}
using *that*
by (*intro determ-traces-read-inv-reg*)

(*auto simp del*: *TLBEntries-names-unfold*
simp add: *trans-regs-def TLBEntries-names-def regval-of-TLBEntry-def*
intro!: *get-regval-TLBEntries read-from-TLBEntries*)

lemma *determ-traces-read-reg-TLBEntryHi*[*determ*]:
determ-traces (*read-reg TLBEntryHi-ref*)
by (*intro determ-traces-read-inv-reg*)
(*auto simp*: *TLBEntryHi-ref-def trans-regs-def get-regval-def dest*: *trans-inv-regs-eq-trans-regstate*)

lemma *determ-traces-tlbSearch*[*determ*]:
determ-runs (*tlbSearch vAddr*)
unfolding *tlbSearch-def Let-def*
by (*intro determ determ-traces-bindI determ-traces-foreachM determ-traces-if determ-traces-runs*
preserves-invariantI traces-runs-preserve-invariantI allI conjI impI)
auto

lemma *determ-runs-translate-addressM*: *determ-runs* (*translate-addressM vaddr is-load*)
unfolding *translate-addressM-def TLBTranslate-def TLBTranslateC-def TLBTranslate2-def*
getAccessLevel-def undefined-range-def Let-def bind-assoc
by (*intro determ-runs-bindI determ-runs-if determ determ-traces-runs*
preserves-invariantI traces-runs-preserve-invariantI allI conjI impI)
auto

lemma *TLBTranslate-LoadData-translate-address-eq*[*simp*]:
assumes *Run-inv* (*TLBTranslate vaddr LoadData*) *t paddr regs*
shows *translate-address* (*unat vaddr*) *Load* $t' =$ *Some* (*unat paddr*)
proof −
from *assms* **have** *Run-inv* (*translate-addressM* (*unat vaddr*) *Load*) *t* (*unat paddr*) *regs*
unfolding *translate-addressM-def Run-inv-def*
by (*auto simp flip*: *uint-nat intro*: *Traces-bindI*[*of - t paddr -* [], *simplified*])
then show *?thesis*
using *determ-runs-translate-addressM*
by (*auto simp*: *translate-address-def determ-the-result-eq*)
qed

lemma *TLBTranslate-StoreData-translate-address-eq*[*simp*]:
assumes *Run-inv* (*TLBTranslate vaddr StoreData*) *t paddr regs*
shows *translate-address* (*unat vaddr*) *Store* $t' =$ *Some* (*unat paddr*)
proof −
from *assms* **have** *Run-inv* (*translate-addressM* (*unat vaddr*) *Store*) *t* (*unat paddr*) *regs*
unfolding *translate-addressM-def Run-inv-def*
by (*auto simp flip*: *uint-nat intro*: *Traces-bindI*[*of - t paddr -* [], *simplified*])
then show *?thesis*
using *determ-runs-translate-addressM*
by (*auto simp*: *translate-address-def determ-the-result-eq*)

qed

lemma *TLBTranslate-Instruction-translate-address-eq*[*simp*]:
  assumes *Run-inv* (*TLBTranslate vaddr Instruction*) *t paddr regs*
  shows *translate-address* (*unat vaddr*) *Fetch* t' = *Some* (*unat paddr*)
proof −
  from *assms* **have** *Run-inv* (*translate-addressM* (*unat vaddr*) *Fetch*) *t* (*unat paddr*) *regs*
    unfolding *translate-addressM-def Run-inv-def*
    by (*auto simp flip*: *uint-nat intro*: *Traces-bindI*[*of* - *t paddr* - [], *simplified*])
  then show *?thesis*
    using *determ-runs-translate-addressM*
    by (*auto simp*: *translate-address-def determ-the-result-eq*)
qed

end

lemma *mult-mod-plus-less*:
  assumes *n dvd m* **and** $n > 0$ **and** $m > 0$ **and** $0 \leq i$ **and** $i < n$
  shows $n * q$ *mod* $m + i < (m :: int)$
  using *assms*
  by (*auto simp*: *dvd-def*)
    (*metis assms*(*2*−*5*) *mult.commute zero-less-mult-pos2 zmult2-lemma-aux4*)

lemma *dvd-nat-iff-int-dvd*:
  assumes $0 \leq i$
  shows *n dvd nat i* ⟷ *int n dvd i*
  using *assms*
  by (*auto simp*: *dvd-def nat-mult-distrib*) (*use nat-0-le* **in** ‹*fastforce*›)

lemma *sail-ones-max-word*[*simp*]: *sail-ones n* = *max-word*
  by (*intro word-eqI*) (*auto simp*: *sail-ones-def zeros-def*)

lemma *sail-mask-ucast*[*simp*]: *sail-mask n w* = *ucast w*
  by (*auto simp*: *sail-mask-def vector-truncate-def zero-extend-def*)

lemma *mod2-minus-one-mask*:
  (2 ˆ $n - 1$) = (*mask n* :: $'a$::*len word*)
  by (*auto simp*: *mask-def*)

lemma *slice-mask-nth*:
  fixes *n i l* :: *int* **and** *j* :: *nat*
  defines *w* ≡ *slice-mask n i l* :: $'n$::*len word*
  assumes *n* = *LENGTH*($'n$)
  shows *w* !! *j* ⟷ $j <$ *nat n* ∧ *nat i* $\leq j$ ∧ $j <$ *nat i* + *nat l*
  using *assms*
  by (*auto simp*: *slice-mask-def nth-shiftl Let-def mod2-minus-one-mask*)

lemma *subrange-subrange-concat-ucast-right*:

fixes *w1* :: *'a::len word* **and** *w2* :: *'b::len word*
fixes *c i1 j1 i2* :: *int*
defines $w \equiv$ *subrange-subrange-concat c w1 i1 j1 w2 i2 0* :: *'c::len word*
defines $d \equiv$ *ucast w2* :: *'d::len word*
assumes *int LENGTH('d)* $\leq$ *i2* + *1* **and** *0* $\leq$ *i2 LENGTH('b)* $\geq$ *LENGTH('d)*
LENGTH('c) $\geq$ *LENGTH('d)*
shows *ucast w* = *d*
using *assms*
by (*intro word-eqI*)
(*auto simp*: *subrange-subrange-concat-def nth-ucast word-ao-nth nth-shiftl nth-shiftr nat-add-distrib slice-mask-nth*)

context *CHERI-MIPS-Fixed-Trans*
begin

lemma [*simp*]: *tag-granule ISA* = *32* **by** (*auto simp*: *ISA-def*)

lemma *address-tag-aligned-plus-iff* [*simp*]:
fixes *addr* :: *64 word*
assumes *int* (*tag-granule ISA*) *dvd i* **and** *0* $\leq$ *i*
shows *address-tag-aligned ISA* (*unat* (*addr* + *word-of-int i*)) $\longleftrightarrow$ *address-tag-aligned ISA* (*unat addr*)
using *assms*
unfolding *address-tag-aligned-def unat-def mod-eq-0-iff-dvd uint-ge-0*[*THEN dvd-nat-iff-int-dvd*]
by (*auto simp*: *uint-word-ariths uint-word-of-int mod-add-right-eq dvd-mod-iff dvd-add-left-iff*)

lemma *TLBTranslate2-ucast-paddr-eq*:
assumes *Run* (*TLBTranslate2 vaddr acctype*) *t* (*paddr*, *flag*)
shows (*ucast paddr* :: *12 word*) = (*ucast vaddr* :: *12 word*)
using *assms*
unfolding *TLBTranslate2-def Let-def undefined-range-def*
by (*auto elim*!: *Run-bindE Run-ifE split*: *option.splits*
simp: *subrange-subrange-concat-ucast-right*)

lemma *TLBTranslateC-ucast-paddr-eq*:
assumes *Run* (*TLBTranslateC vaddr acctype*) *t* (*paddr*, *flag*)
shows (*ucast paddr* :: *12 word*) = (*ucast vaddr* :: *12 word*)
using *assms*
unfolding *TLBTranslateC-def Let-def*
by (*fastforce elim*!: *Run-bindE Run-ifE simp*: *TLBTranslate2-ucast-paddr-eq split*: *option.splits bool.splits prod.splits if-splits*)

lemma *TLBTranslate-ucast-paddr-eq*:
assumes *Run* (*TLBTranslate vaddr acctype*) *t paddr*
shows (*ucast paddr* :: *12 word*) = (*ucast vaddr* :: *12 word*)
using *assms*
unfolding *TLBTranslate-def*
by (*auto elim*!: *Run-bindE simp*: *TLBTranslateC-ucast-paddr-eq*)

lemma *address-tag-aligned-ucast5*:
  fixes *addr* :: *'a::len word*
  assumes *LENGTH('a)* $\geq$ *5*
  shows *address-tag-aligned ISA* (*unat addr*) $\longleftrightarrow$ (*ucast addr* :: *5 word*) = *0*
  using *assms*
  unfolding *unat-arith-simps*(*3*)
  by (*auto simp*: *address-tag-aligned-def unat-and-mask min-def*)

lemma *address-tag-aligned-paddr-iff-vaddr*[*simp*]:
  assumes *Run-inv* (*TLBTranslate vaddr acctype*) *t paddr regs*
  shows *address-tag-aligned ISA* (*unat paddr*) $\longleftrightarrow$ *address-tag-aligned ISA* (*unat vaddr*)
proof −
  have *paddr-vaddr*: *ucast paddr* = (*ucast vaddr* :: *12 word*)
    using *assms*
    by (*auto simp*: *Run-inv-def TLBTranslate-ucast-paddr-eq*)
  have *address-tag-aligned ISA* (*unat paddr*) $\longleftrightarrow$ (*ucast* (*ucast paddr* :: *12 word*) :: *5 word*) = *0*
    by (*auto simp*: *address-tag-aligned-ucast5*)
  also have ... $\longleftrightarrow$ *address-tag-aligned ISA* (*unat vaddr*)
    unfolding *paddr-vaddr*
    by (*auto simp*: *address-tag-aligned-ucast5*)
  finally show *?thesis* **.**
qed

lemma *TLBTranslateC-address-tag-aligned*[*simp*]:
  assumes *Run-inv* (*TLBTranslateC vaddr acctype*) *t* (*paddr*, *noStoreCap*) *regs*
  shows *address-tag-aligned ISA* (*unat paddr*) $\longleftrightarrow$ *address-tag-aligned ISA* (*unat vaddr*)
proof −
  have *paddr-vaddr*: *ucast paddr* = (*ucast vaddr* :: *12 word*)
    using *assms*
    by (*auto simp*: *Run-inv-def TLBTranslateC-ucast-paddr-eq*)
  have *address-tag-aligned ISA* (*unat paddr*) $\longleftrightarrow$ (*ucast* (*ucast paddr* :: *12 word*) :: *5 word*) = *0*
    by (*auto simp*: *address-tag-aligned-ucast5*)
  also have ... $\longleftrightarrow$ *address-tag-aligned ISA* (*unat vaddr*)
    unfolding *paddr-vaddr*
    by (*auto simp*: *address-tag-aligned-ucast5*)
  finally show *?thesis* **.**
qed

lemma *address-tag-aligned-mult-dvd*[*intro*, *simp*]:
  assumes *int* (*tag-granule ISA*) *dvd k* **and** *0* $\leq$ *k*
  shows *address-tag-aligned ISA* (*nat* (*k* $*$ *n*))
  using *assms*
  by (*auto simp*: *address-tag-aligned-def nat-mult-distrib*)

end

locale *CHERI-MIPS-Mem-Instr-Automaton* = *CHERI-MIPS-Mem-Automaton* **where**
is-fetch = *False*

locale *CHERI-MIPS-Mem-Fetch-Automaton* = *CHERI-MIPS-Mem-Automaton* **where**
is-fetch = *True*

end
theory *CHERI-MIPS-Gen-Lemmas*
imports *CHERI-MIPS-Instantiation*
begin

3.2 Footprint lemmas

context *CHERI-MIPS-Axiom-Inv-Automaton*
begin

lemma *non-cap-regsI*[*intro*, *simp*]:
non-cap-reg BranchPending-ref
non-cap-reg CID-ref
non-cap-reg CP0BadInstr-ref
non-cap-reg CP0BadInstrP-ref
non-cap-reg CP0BadVAddr-ref
non-cap-reg CP0Cause-ref
non-cap-reg CP0Compare-ref
non-cap-reg CP0ConfigK0-ref
non-cap-reg CP0Count-ref
non-cap-reg CP0HWREna-ref
non-cap-reg CP0LLAddr-ref
non-cap-reg CP0LLBit-ref
non-cap-reg CP0Status-ref
non-cap-reg CP0UserLocal-ref
non-cap-reg CurrentInstrBits-ref
non-cap-reg DelayedPC-ref
non-cap-reg GPR-ref
non-cap-reg HI-ref
non-cap-reg InBranchDelay-ref
non-cap-reg LO-ref
non-cap-reg LastInstrBits-ref
non-cap-reg NextInBranchDelay-ref
non-cap-reg NextPC-ref
non-cap-reg PC-ref
non-cap-reg TLBContext-ref
non-cap-reg TLBEntry00-ref
non-cap-reg TLBEntry01-ref
non-cap-reg TLBEntry02-ref
non-cap-reg TLBEntry03-ref
non-cap-reg TLBEntry04-ref

non-cap-reg TLBEntry05-ref
non-cap-reg TLBEntry06-ref
non-cap-reg TLBEntry07-ref
non-cap-reg TLBEntry08-ref
non-cap-reg TLBEntry09-ref
non-cap-reg TLBEntry10-ref
non-cap-reg TLBEntry11-ref
non-cap-reg TLBEntry12-ref
non-cap-reg TLBEntry13-ref
non-cap-reg TLBEntry14-ref
non-cap-reg TLBEntry15-ref
non-cap-reg TLBEntry16-ref
non-cap-reg TLBEntry17-ref
non-cap-reg TLBEntry18-ref
non-cap-reg TLBEntry19-ref
non-cap-reg TLBEntry20-ref
non-cap-reg TLBEntry21-ref
non-cap-reg TLBEntry22-ref
non-cap-reg TLBEntry23-ref
non-cap-reg TLBEntry24-ref
non-cap-reg TLBEntry25-ref
non-cap-reg TLBEntry26-ref
non-cap-reg TLBEntry27-ref
non-cap-reg TLBEntry28-ref
non-cap-reg TLBEntry29-ref
non-cap-reg TLBEntry30-ref
non-cap-reg TLBEntry31-ref
non-cap-reg TLBEntry32-ref
non-cap-reg TLBEntry33-ref
non-cap-reg TLBEntry34-ref
non-cap-reg TLBEntry35-ref
non-cap-reg TLBEntry36-ref
non-cap-reg TLBEntry37-ref
non-cap-reg TLBEntry38-ref
non-cap-reg TLBEntry39-ref
non-cap-reg TLBEntry40-ref
non-cap-reg TLBEntry41-ref
non-cap-reg TLBEntry42-ref
non-cap-reg TLBEntry43-ref
non-cap-reg TLBEntry44-ref
non-cap-reg TLBEntry45-ref
non-cap-reg TLBEntry46-ref
non-cap-reg TLBEntry47-ref
non-cap-reg TLBEntry48-ref
non-cap-reg TLBEntry49-ref
non-cap-reg TLBEntry50-ref
non-cap-reg TLBEntry51-ref
non-cap-reg TLBEntry52-ref
non-cap-reg TLBEntry53-ref

non-cap-reg TLBEntry54-ref
non-cap-reg TLBEntry55-ref
non-cap-reg TLBEntry56-ref
non-cap-reg TLBEntry57-ref
non-cap-reg TLBEntry58-ref
non-cap-reg TLBEntry59-ref
non-cap-reg TLBEntry60-ref
non-cap-reg TLBEntry61-ref
non-cap-reg TLBEntry62-ref
non-cap-reg TLBEntry63-ref
non-cap-reg TLBEntryHi-ref
non-cap-reg TLBEntryLo0-ref
non-cap-reg TLBEntryLo1-ref
non-cap-reg TLBIndex-ref
non-cap-reg TLBPageMask-ref
non-cap-reg TLBProbe-ref
non-cap-reg TLBRandom-ref
non-cap-reg TLBWired-ref
non-cap-reg TLBXContext-ref
non-cap-reg UART-RDATA-ref
non-cap-reg UART-RVALID-ref
non-cap-reg UART-WDATA-ref
non-cap-reg UART-WRITTEN-ref
non-cap-reg InstCount-ref
unfolding *BranchPending-ref-def CID-ref-def CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0BadVAddr-ref-def*
CP0Cause-ref-def CP0Compare-ref-def CP0ConfigK0-ref-def CP0Count-ref-def CP0HWREna-ref-def
CP0LLAddr-ref-def CP0LLBit-ref-def CP0Status-ref-def CP0UserLocal-ref-def CurrentInstrBits-ref-def
DelayedPC-ref-def GPR-ref-def HI-ref-def InBranchDelay-ref-def LO-ref-def
LastInstrBits-ref-def NextInBranchDelay-ref-def NextPC-ref-def PC-ref-def TLBContext-ref-def
TLBEntry00-ref-def TLBEntry01-ref-def TLBEntry02-ref-def TLBEntry03-ref-def TLBEntry04-ref-def
TLBEntry05-ref-def TLBEntry06-ref-def TLBEntry07-ref-def TLBEntry08-ref-def TLBEntry09-ref-def
TLBEntry10-ref-def TLBEntry11-ref-def TLBEntry12-ref-def TLBEntry13-ref-def TLBEntry14-ref-def
TLBEntry15-ref-def TLBEntry16-ref-def TLBEntry17-ref-def TLBEntry18-ref-def TLBEntry19-ref-def
TLBEntry20-ref-def TLBEntry21-ref-def TLBEntry22-ref-def TLBEntry23-ref-def TLBEntry24-ref-def
TLBEntry25-ref-def TLBEntry26-ref-def TLBEntry27-ref-def TLBEntry28-ref-def TLBEntry29-ref-def
TLBEntry30-ref-def TLBEntry31-ref-def TLBEntry32-ref-def TLBEntry33-ref-def TLBEntry34-ref-def
TLBEntry35-ref-def TLBEntry36-ref-def TLBEntry37-ref-def TLBEntry38-ref-def TLBEntry39-ref-def
TLBEntry40-ref-def TLBEntry41-ref-def TLBEntry42-ref-def TLBEntry43-ref-def

TLBEntry44-ref-def
TLBEntry45-ref-def TLBEntry46-ref-def TLBEntry47-ref-def TLBEntry48-ref-def
TLBEntry49-ref-def
TLBEntry50-ref-def TLBEntry51-ref-def TLBEntry52-ref-def TLBEntry53-ref-def
TLBEntry54-ref-def
TLBEntry55-ref-def TLBEntry56-ref-def TLBEntry57-ref-def TLBEntry58-ref-def
TLBEntry59-ref-def
TLBEntry60-ref-def TLBEntry61-ref-def TLBEntry62-ref-def TLBEntry63-ref-def
TLBEntryHi-ref-def
TLBEntryLo0-ref-def TLBEntryLo1-ref-def TLBIndex-ref-def TLBPageMask-ref-def
TLBProbe-ref-def
TLBRandom-ref-def TLBWired-ref-def TLBXContext-ref-def UART-RDATA-ref-def
UART-RVALID-ref-def
UART-WDATA-ref-def UART-WRITTEN-ref-def InstCount-ref-def
by (*auto simp*: *non-cap-reg-def*)

lemmas *non-cap-exp-rw-non-cap-reg*[*non-cap-expI*] =
non-cap-regsI[*THEN non-cap-exp-read-non-cap-reg*]
non-cap-regsI[*THEN non-cap-exp-write-non-cap-reg*]

lemma *non-cap-exp-undefined-option*[*non-cap-expI*]:
non-cap-exp (*undefined-option arg0*)
by (*non-cap-expI simp*: *undefined-option-def*)

lemma *non-cap-exp-undefined-exception*[*non-cap-expI*]:
non-cap-exp (*undefined-exception arg0*)
by (*non-cap-expI simp*: *undefined-exception-def*)

lemma *non-cap-exp-undefined-CauseReg*[*non-cap-expI*]:
non-cap-exp (*undefined-CauseReg arg0*)
by (*non-cap-expI simp*: *undefined-CauseReg-def*)

lemma *non-cap-exp-set-CauseReg-bits*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-CauseReg-bits arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-CauseReg-bits-def*)

lemma *non-cap-exp-set-CauseReg-BD*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-CauseReg-BD arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-CauseReg-BD-def*)

lemma *non-cap-exp-set-CauseReg-CE*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-CauseReg-CE arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-CauseReg-CE-def*)

lemma *non-cap-exp-set-CauseReg-IV*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-CauseReg-IV arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-CauseReg-IV-def*)

lemma *non-cap-exp-set-CauseReg-WP*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-CauseReg-WP arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-CauseReg-WP-def*)

lemma *non-cap-exp-set-CauseReg-IP*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-CauseReg-IP arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-CauseReg-IP-def*)

lemma *non-cap-exp-set-CauseReg-ExcCode*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-CauseReg-ExcCode arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-CauseReg-ExcCode-def*)

lemma *non-cap-exp-undefined-TLBEntryLoReg*[*non-cap-expI*]:
non-cap-exp (*undefined-TLBEntryLoReg arg0*)
by (*non-cap-expI simp*: *undefined-TLBEntryLoReg-def*)

lemma *non-cap-exp-set-TLBEntryLoReg-bits*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntryLoReg-bits arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryLoReg-bits-def*)

lemma *non-cap-exp-set-TLBEntryLoReg-CapS*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntryLoReg-CapS arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryLoReg-CapS-def*)

lemma *non-cap-exp-set-TLBEntryLoReg-CapL*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntryLoReg-CapL arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryLoReg-CapL-def*)

lemma *non-cap-exp-set-TLBEntryLoReg-PFN*[*non-cap-expI*]:
assumes *non-cap-reg arg0*

shows *non-cap-exp* (*set-TLBEntryLoReg-PFN arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryLoReg-PFN-def*)

lemma *non-cap-exp-set-TLBEntryLoReg-C*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntryLoReg-C arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryLoReg-C-def*)

lemma *non-cap-exp-set-TLBEntryLoReg-D*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntryLoReg-D arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryLoReg-D-def*)

lemma *non-cap-exp-set-TLBEntryLoReg-V*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntryLoReg-V arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryLoReg-V-def*)

lemma *non-cap-exp-set-TLBEntryLoReg-G*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntryLoReg-G arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryLoReg-G-def*)

lemma *non-cap-exp-undefined-TLBEntryHiReg*[*non-cap-expI*]:
non-cap-exp (*undefined-TLBEntryHiReg arg0*)
by (*non-cap-expI simp*: *undefined-TLBEntryHiReg-def*)

lemma *non-cap-exp-set-TLBEntryHiReg-bits*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntryHiReg-bits arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryHiReg-bits-def*)

lemma *non-cap-exp-set-TLBEntryHiReg-R*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntryHiReg-R arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryHiReg-R-def*)

lemma *non-cap-exp-set-TLBEntryHiReg-VPN2*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntryHiReg-VPN2 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryHiReg-VPN2-def*)

lemma *non-cap-exp-set-TLBEntryHiReg-ASID*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntryHiReg-ASID arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntryHiReg-ASID-def*)

lemma *non-cap-exp-undefined-ContextReg*[*non-cap-expI*]:
non-cap-exp (*undefined-ContextReg arg0*)
by (*non-cap-expI simp*: *undefined-ContextReg-def*)

lemma *non-cap-exp-set-ContextReg-bits*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-ContextReg-bits arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-ContextReg-bits-def*)

lemma *non-cap-exp-set-ContextReg-PTEBase*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-ContextReg-PTEBase arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-ContextReg-PTEBase-def*)

lemma *non-cap-exp-set-ContextReg-BadVPN2*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-ContextReg-BadVPN2 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-ContextReg-BadVPN2-def*)

lemma *non-cap-exp-undefined-XContextReg*[*non-cap-expI*]:
non-cap-exp (*undefined-XContextReg arg0*)
by (*non-cap-expI simp*: *undefined-XContextReg-def*)

lemma *non-cap-exp-set-XContextReg-bits*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-XContextReg-bits arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-XContextReg-bits-def*)

lemma *non-cap-exp-set-XContextReg-XPTEBase*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-XContextReg-XPTEBase arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-XContextReg-XPTEBase-def*)

lemma *non-cap-exp-set-XContextReg-XR*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-XContextReg-XR arg0 arg1*)
using *assms*

by (*non-cap-expI simp*: *set-XContextReg-XR-def*)

lemma *non-cap-exp-set-XContextReg-XBadVPN2*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-XContextReg-XBadVPN2 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-XContextReg-XBadVPN2-def*)

lemma *non-cap-exp-undefined-TLBEntry*[*non-cap-expI*]:
non-cap-exp (*undefined-TLBEntry arg0*)
by (*non-cap-expI simp*: *undefined-TLBEntry-def*)

lemma *non-cap-exp-set-TLBEntry-bits*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-bits arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-bits-def*)

lemma *non-cap-exp-set-TLBEntry-pagemask*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-pagemask arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-pagemask-def*)

lemma *non-cap-exp-set-TLBEntry-r*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-r arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-r-def*)

lemma *non-cap-exp-set-TLBEntry-vpn2*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-vpn2 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-vpn2-def*)

lemma *non-cap-exp-set-TLBEntry-asid*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-asid arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-asid-def*)

lemma *non-cap-exp-set-TLBEntry-g*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-g arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-g-def*)

lemma *non-cap-exp-set-TLBEntry-valid*[*non-cap-expI*]:

assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-valid arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-valid-def*)

lemma *non-cap-exp-set-TLBEntry-caps1*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-caps1 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-caps1-def*)

lemma *non-cap-exp-set-TLBEntry-capl1*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-capl1 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-capl1-def*)

lemma *non-cap-exp-set-TLBEntry-pfn1*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-pfn1 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-pfn1-def*)

lemma *non-cap-exp-set-TLBEntry-c1*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-c1 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-c1-def*)

lemma *non-cap-exp-set-TLBEntry-d1*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-d1 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-d1-def*)

lemma *non-cap-exp-set-TLBEntry-v1*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-v1 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-v1-def*)

lemma *non-cap-exp-set-TLBEntry-caps0*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-caps0 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-caps0-def*)

lemma *non-cap-exp-set-TLBEntry-capl0*[*non-cap-expI*]:
assumes *non-cap-reg arg0*

shows *non-cap-exp* (*set-TLBEntry-capl0 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-capl0-def*)

lemma *non-cap-exp-set-TLBEntry-pfn0*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-pfn0 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-pfn0-def*)

lemma *non-cap-exp-set-TLBEntry-c0*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-c0 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-c0-def*)

lemma *non-cap-exp-set-TLBEntry-d0*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-d0 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-d0-def*)

lemma *non-cap-exp-set-TLBEntry-v0*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-TLBEntry-v0 arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-TLBEntry-v0-def*)

lemma *non-cap-exp-undefined-StatusReg*[*non-cap-expI*]:
non-cap-exp (*undefined-StatusReg arg0*)
by (*non-cap-expI simp*: *undefined-StatusReg-def*)

lemma *non-cap-exp-set-StatusReg-bits*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-StatusReg-bits arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-StatusReg-bits-def*)

lemma *non-cap-exp-set-StatusReg-CU*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-StatusReg-CU arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-StatusReg-CU-def*)

lemma *non-cap-exp-set-StatusReg-BEV*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-StatusReg-BEV arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-StatusReg-BEV-def*)

lemma *non-cap-exp-set-StatusReg-IM* [*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-StatusReg-IM arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-StatusReg-IM-def*)

lemma *non-cap-exp-set-StatusReg-KX* [*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-StatusReg-KX arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-StatusReg-KX-def*)

lemma *non-cap-exp-set-StatusReg-SX* [*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-StatusReg-SX arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-StatusReg-SX-def*)

lemma *non-cap-exp-set-StatusReg-UX* [*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-StatusReg-UX arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-StatusReg-UX-def*)

lemma *non-cap-exp-set-StatusReg-KSU* [*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-StatusReg-KSU arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-StatusReg-KSU-def*)

lemma *non-cap-exp-set-StatusReg-ERL*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-StatusReg-ERL arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-StatusReg-ERL-def*)

lemma *non-cap-exp-set-StatusReg-EXL*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-StatusReg-EXL arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-StatusReg-EXL-def*)

lemma *non-cap-exp-set-StatusReg-IE* [*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-StatusReg-IE arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-StatusReg-IE-def*)

lemma *non-cap-exp-execute-branch-mips*[*non-cap-expI*]:
non-cap-exp (*execute-branch-mips arg0*)
by (*non-cap-expI simp*: *execute-branch-mips-def*)

lemma *non-cap-exp-rGPR*[*non-cap-expI*]:
non-cap-exp (*rGPR arg0*)
by (*non-cap-expI simp*: *rGPR-def*)

lemma *non-cap-exp-wGPR*[*non-cap-expI*]:
non-cap-exp (*wGPR arg0 arg1*)
by (*non-cap-expI simp*: *wGPR-def*)

lemma *non-cap-exp-MEM-sync*[*non-cap-expI*]:
non-cap-exp (*MEM-sync arg0*)
by (*non-cap-expI simp*: *MEM-sync-def*)

lemma *non-cap-exp-undefined-Exception*[*non-cap-expI*]:
non-cap-exp (*undefined-Exception arg0*)
by (*non-cap-expI simp*: *undefined-Exception-def*)

lemma *non-cap-exp-exceptionVectorOffset*[*non-cap-expI*]:
non-cap-exp (*exceptionVectorOffset arg0*)
by (*non-cap-expI simp*: *exceptionVectorOffset-def*)

lemma *non-cap-exp-exceptionVectorBase*[*non-cap-expI*]:
non-cap-exp (*exceptionVectorBase arg0*)
by (*non-cap-expI simp*: *exceptionVectorBase-def*)

lemma *non-cap-exp-updateBadInstr*[*non-cap-expI*]:
non-cap-exp (*updateBadInstr arg0*)
by (*non-cap-expI simp*: *updateBadInstr-def*)

lemma *non-cap-exp-undefined-Capability*[*non-cap-expI*]:
non-cap-exp (*undefined-Capability arg0*)
by (*non-cap-expI simp*: *undefined-Capability-def*)

lemma *non-cap-exp-undefined-MemAccessType*[*non-cap-expI*]:
non-cap-exp (*undefined-MemAccessType arg0*)
by (*non-cap-expI simp*: *undefined-MemAccessType-def*)

lemma *non-cap-exp-undefined-AccessLevel*[*non-cap-expI*]:
non-cap-exp (*undefined-AccessLevel arg0*)
by (*non-cap-expI simp*: *undefined-AccessLevel-def*)

lemma *non-cap-exp-getAccessLevel*[*non-cap-expI*]:
non-cap-exp (*getAccessLevel arg0*)
by (*non-cap-expI simp*: *getAccessLevel-def*)

lemma *non-cap-exp-undefined-CapCauseReg*[*non-cap-expI*]:

non-cap-exp (*undefined-CapCauseReg arg0*)
by (*non-cap-expI simp*: *undefined-CapCauseReg-def*)

lemma *non-cap-exp-set-CapCauseReg-ExcCode*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-CapCauseReg-ExcCode arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-CapCauseReg-ExcCode-def*)

lemma *non-cap-exp-set-CapCauseReg-RegNum*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-CapCauseReg-RegNum arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-CapCauseReg-RegNum-def*)

lemma *non-cap-exp-undefined-decode-failure*[*non-cap-expI*]:
non-cap-exp (*undefined-decode-failure arg0*)
by (*non-cap-expI simp*: *undefined-decode-failure-def*)

lemma *non-cap-exp-undefined-Comparison*[*non-cap-expI*]:
non-cap-exp (*undefined-Comparison arg0*)
by (*non-cap-expI simp*: *undefined-Comparison-def*)

lemma *non-cap-exp-undefined-WordType*[*non-cap-expI*]:
non-cap-exp (*undefined-WordType arg0*)
by (*non-cap-expI simp*: *undefined-WordType-def*)

lemma *non-cap-exp-undefined-WordTypeUnaligned*[*non-cap-expI*]:
non-cap-exp (*undefined-WordTypeUnaligned arg0*)
by (*non-cap-expI simp*: *undefined-WordTypeUnaligned-def*)

lemma *non-cap-exp-init-cp0-state*[*non-cap-expI*]:
non-cap-exp (*init-cp0-state arg0*)
by (*non-cap-expI simp*: *init-cp0-state-def*)

lemma *non-cap-exp-tlbSearch*[*non-cap-expI*]:
non-cap-exp (*tlbSearch arg0*)
by (*non-cap-expI simp*: *tlbSearch-def*)

lemma *non-cap-exp-undefined-CPtrCmpOp*[*non-cap-expI*]:
non-cap-exp (*undefined-CPtrCmpOp arg0*)
by (*non-cap-expI simp*: *undefined-CPtrCmpOp-def*)

lemma *non-cap-exp-undefined-ClearRegSet*[*non-cap-expI*]:
non-cap-exp (*undefined-ClearRegSet arg0*)
by (*non-cap-expI simp*: *undefined-ClearRegSet-def*)

lemma *non-cap-exp-capToString*[*non-cap-expI*]:
non-cap-exp (*capToString arg0 arg1*)

by (*non-cap-expI simp*: *capToString-def*)

lemma *non-cap-exp-undefined-CapEx*[*non-cap-expI*]:
non-cap-exp (*undefined-CapEx arg0*)
by (*non-cap-expI simp*: *undefined-CapEx-def*)

lemma *non-cap-exp-set-CapCauseReg-bits*[*non-cap-expI*]:
assumes *non-cap-reg arg0*
shows *non-cap-exp* (*set-CapCauseReg-bits arg0 arg1*)
using *assms*
by (*non-cap-expI simp*: *set-CapCauseReg-bits-def*)

lemma *non-cap-exp-execute-XORI*[*non-cap-expI*]:
non-cap-exp (*execute-XORI arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-XORI-def*)

lemma *non-cap-exp-execute-XOR*[*non-cap-expI*]:
non-cap-exp (*execute-XOR arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-XOR-def*)

lemma *non-cap-exp-execute-SYNC*[*non-cap-expI*]:
non-cap-exp (*execute-SYNC arg0*)
by (*non-cap-expI simp*: *execute-SYNC-def*)

lemma *non-cap-exp-execute-SUBU*[*non-cap-expI*]:
non-cap-exp (*execute-SUBU arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-SUBU-def*)

lemma *non-cap-exp-execute-SRLV*[*non-cap-expI*]:
non-cap-exp (*execute-SRLV arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-SRLV-def*)

lemma *non-cap-exp-execute-SRL*[*non-cap-expI*]:
non-cap-exp (*execute-SRL arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-SRL-def*)

lemma *non-cap-exp-execute-SRAV*[*non-cap-expI*]:
non-cap-exp (*execute-SRAV arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-SRAV-def*)

lemma *non-cap-exp-execute-SRA*[*non-cap-expI*]:
non-cap-exp (*execute-SRA arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-SRA-def*)

lemma *non-cap-exp-execute-SLTU*[*non-cap-expI*]:
non-cap-exp (*execute-SLTU arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-SLTU-def*)

lemma *non-cap-exp-execute-SLTIU*[*non-cap-expI*]:

non-cap-exp (*execute-SLTIU arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-SLTIU-def*)

lemma *non-cap-exp-execute-SLTI*[*non-cap-expI*]:
non-cap-exp (*execute-SLTI arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-SLTI-def*)

lemma *non-cap-exp-execute-SLT*[*non-cap-expI*]:
non-cap-exp (*execute-SLT arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-SLT-def*)

lemma *non-cap-exp-execute-SLLV*[*non-cap-expI*]:
non-cap-exp (*execute-SLLV arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-SLLV-def*)

lemma *non-cap-exp-execute-SLL*[*non-cap-expI*]:
non-cap-exp (*execute-SLL arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-SLL-def*)

lemma *non-cap-exp-execute-ORI*[*non-cap-expI*]:
non-cap-exp (*execute-ORI arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-ORI-def*)

lemma *non-cap-exp-execute-OR*[*non-cap-expI*]:
non-cap-exp (*execute-OR arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-OR-def*)

lemma *non-cap-exp-execute-NOR*[*non-cap-expI*]:
non-cap-exp (*execute-NOR arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-NOR-def*)

lemma *non-cap-exp-execute-MULTU*[*non-cap-expI*]:
non-cap-exp (*execute-MULTU arg0 arg1*)
by (*non-cap-expI simp*: *execute-MULTU-def*)

lemma *non-cap-exp-execute-MULT*[*non-cap-expI*]:
non-cap-exp (*execute-MULT arg0 arg1*)
by (*non-cap-expI simp*: *execute-MULT-def*)

lemma *non-cap-exp-execute-MUL*[*non-cap-expI*]:
non-cap-exp (*execute-MUL arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-MUL-def*)

lemma *non-cap-exp-execute-MTLO*[*non-cap-expI*]:
non-cap-exp (*execute-MTLO arg0*)
by (*non-cap-expI simp*: *execute-MTLO-def*)

lemma *non-cap-exp-execute-MTHI*[*non-cap-expI*]:
non-cap-exp (*execute-MTHI arg0*)

by (*non-cap-expI simp*: *execute-MTHI-def*)

lemma *non-cap-exp-execute-MSUBU*[*non-cap-expI*]:
non-cap-exp (*execute-MSUBU arg0 arg1*)
by (*non-cap-expI simp*: *execute-MSUBU-def*)

lemma *non-cap-exp-execute-MSUB*[*non-cap-expI*]:
non-cap-exp (*execute-MSUB arg0 arg1*)
by (*non-cap-expI simp*: *execute-MSUB-def*)

lemma *non-cap-exp-execute-MOVZ*[*non-cap-expI*]:
non-cap-exp (*execute-MOVZ arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-MOVZ-def*)

lemma *non-cap-exp-execute-MOVN*[*non-cap-expI*]:
non-cap-exp (*execute-MOVN arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-MOVN-def*)

lemma *non-cap-exp-execute-MFLO*[*non-cap-expI*]:
non-cap-exp (*execute-MFLO arg0*)
by (*non-cap-expI simp*: *execute-MFLO-def*)

lemma *non-cap-exp-execute-MFHI*[*non-cap-expI*]:
non-cap-exp (*execute-MFHI arg0*)
by (*non-cap-expI simp*: *execute-MFHI-def*)

lemma *non-cap-exp-execute-MADDU*[*non-cap-expI*]:
non-cap-exp (*execute-MADDU arg0 arg1*)
by (*non-cap-expI simp*: *execute-MADDU-def*)

lemma *non-cap-exp-execute-MADD*[*non-cap-expI*]:
non-cap-exp (*execute-MADD arg0 arg1*)
by (*non-cap-expI simp*: *execute-MADD-def*)

lemma *non-cap-exp-execute-LUI*[*non-cap-expI*]:
non-cap-exp (*execute-LUI arg0 arg1*)
by (*non-cap-expI simp*: *execute-LUI-def*)

lemma *non-cap-exp-execute-DSUBU*[*non-cap-expI*]:
non-cap-exp (*execute-DSUBU arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DSUBU-def*)

lemma *non-cap-exp-execute-DSRLV*[*non-cap-expI*]:
non-cap-exp (*execute-DSRLV arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DSRLV-def*)

lemma *non-cap-exp-execute-DSRL32*[*non-cap-expI*]:
non-cap-exp (*execute-DSRL32 arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DSRL32-def*)

lemma *non-cap-exp-execute-DSRL*[*non-cap-expI*]:
non-cap-exp (*execute-DSRL arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DSRL-def*)

lemma *non-cap-exp-execute-DSRAV*[*non-cap-expI*]:
non-cap-exp (*execute-DSRAV arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DSRAV-def*)

lemma *non-cap-exp-execute-DSRA32*[*non-cap-expI*]:
non-cap-exp (*execute-DSRA32 arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DSRA32-def*)

lemma *non-cap-exp-execute-DSRA*[*non-cap-expI*]:
non-cap-exp (*execute-DSRA arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DSRA-def*)

lemma *non-cap-exp-execute-DSLLV*[*non-cap-expI*]:
non-cap-exp (*execute-DSLLV arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DSLLV-def*)

lemma *non-cap-exp-execute-DSLL32*[*non-cap-expI*]:
non-cap-exp (*execute-DSLL32 arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DSLL32-def*)

lemma *non-cap-exp-execute-DSLL*[*non-cap-expI*]:
non-cap-exp (*execute-DSLL arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DSLL-def*)

lemma *non-cap-exp-execute-DMULTU*[*non-cap-expI*]:
non-cap-exp (*execute-DMULTU arg0 arg1*)
by (*non-cap-expI simp*: *execute-DMULTU-def*)

lemma *non-cap-exp-execute-DMULT*[*non-cap-expI*]:
non-cap-exp (*execute-DMULT arg0 arg1*)
by (*non-cap-expI simp*: *execute-DMULT-def*)

lemma *non-cap-exp-execute-DIVU*[*non-cap-expI*]:
non-cap-exp (*execute-DIVU arg0 arg1*)
by (*non-cap-expI simp*: *execute-DIVU-def*)

lemma *non-cap-exp-execute-DIV*[*non-cap-expI*]:
non-cap-exp (*execute-DIV arg0 arg1*)
by (*non-cap-expI simp*: *execute-DIV-def*)

lemma *non-cap-exp-execute-DDIVU*[*non-cap-expI*]:
non-cap-exp (*execute-DDIVU arg0 arg1*)
by (*non-cap-expI simp*: *execute-DDIVU-def*)

lemma *non-cap-exp-execute-DDIV*[*non-cap-expI*]:
non-cap-exp (*execute-DDIV arg0 arg1*)
by (*non-cap-expI simp*: *execute-DDIV-def*)

lemma *non-cap-exp-execute-DADDU*[*non-cap-expI*]:
non-cap-exp (*execute-DADDU arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DADDU-def*)

lemma *non-cap-exp-execute-DADDIU*[*non-cap-expI*]:
non-cap-exp (*execute-DADDIU arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-DADDIU-def*)

lemma *non-cap-exp-execute-ANDI*[*non-cap-expI*]:
non-cap-exp (*execute-ANDI arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-ANDI-def*)

lemma *non-cap-exp-execute-AND*[*non-cap-expI*]:
non-cap-exp (*execute-AND arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-AND-def*)

lemma *non-cap-exp-execute-ADDU*[*non-cap-expI*]:
non-cap-exp (*execute-ADDU arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-ADDU-def*)

lemma *non-cap-exp-execute-ADDIU*[*non-cap-expI*]:
non-cap-exp (*execute-ADDIU arg0 arg1 arg2*)
by (*non-cap-expI simp*: *execute-ADDIU-def*)

lemma *non-mem-exp-set-CauseReg-bits*[*non-mem-expI*]:
non-mem-exp (*set-CauseReg-bits arg0 arg1*)
by (*non-mem-expI simp*: *set-CauseReg-bits-def*)

lemma *non-mem-exp-set-CauseReg-BD*[*non-mem-expI*]:
non-mem-exp (*set-CauseReg-BD arg0 arg1*)
by (*non-mem-expI simp*: *set-CauseReg-BD-def*)

lemma *non-mem-exp-set-CauseReg-CE*[*non-mem-expI*]:
non-mem-exp (*set-CauseReg-CE arg0 arg1*)
by (*non-mem-expI simp*: *set-CauseReg-CE-def*)

lemma *non-mem-exp-set-CauseReg-IV*[*non-mem-expI*]:
non-mem-exp (*set-CauseReg-IV arg0 arg1*)
by (*non-mem-expI simp*: *set-CauseReg-IV-def*)

lemma *non-mem-exp-set-CauseReg-WP*[*non-mem-expI*]:
non-mem-exp (*set-CauseReg-WP arg0 arg1*)
by (*non-mem-expI simp*: *set-CauseReg-WP-def*)

lemma *non-mem-exp-set-CauseReg-IP*[*non-mem-expI*]:

non-mem-exp (*set-CauseReg-IP arg0 arg1*)
by (*non-mem-expI simp*: *set-CauseReg-IP-def*)

lemma *non-mem-exp-set-CauseReg-ExcCode*[*non-mem-expI*]:
non-mem-exp (*set-CauseReg-ExcCode arg0 arg1*)
by (*non-mem-expI simp*: *set-CauseReg-ExcCode-def*)

lemma *non-mem-exp-set-TLBEntryLoReg-bits*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryLoReg-bits arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntryLoReg-bits-def*)

lemma *non-mem-exp-set-TLBEntryLoReg-CapS*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryLoReg-CapS arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntryLoReg-CapS-def*)

lemma *non-mem-exp-set-TLBEntryLoReg-CapL*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryLoReg-CapL arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntryLoReg-CapL-def*)

lemma *non-mem-exp-set-TLBEntryLoReg-PFN*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryLoReg-PFN arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntryLoReg-PFN-def*)

lemma *non-mem-exp-set-TLBEntryLoReg-C*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryLoReg-C arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntryLoReg-C-def*)

lemma *non-mem-exp-set-TLBEntryLoReg-D*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryLoReg-D arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntryLoReg-D-def*)

lemma *non-mem-exp-set-TLBEntryLoReg-V*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryLoReg-V arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntryLoReg-V-def*)

lemma *non-mem-exp-set-TLBEntryLoReg-G*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryLoReg-G arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntryLoReg-G-def*)

lemma *non-mem-exp-set-TLBEntryHiReg-bits*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryHiReg-bits arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntryHiReg-bits-def*)

lemma *non-mem-exp-set-TLBEntryHiReg-R*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryHiReg-R arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntryHiReg-R-def*)

lemma *non-mem-exp-set-TLBEntryHiReg-VPN2*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryHiReg-VPN2 arg0 arg1*)

by (*non-mem-expI simp*: *set-TLBEntryHiReg-VPN2-def*)

lemma *non-mem-exp-set-TLBEntryHiReg-ASID*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntryHiReg-ASID arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntryHiReg-ASID-def*)

lemma *non-mem-exp-set-ContextReg-bits*[*non-mem-expI*]:
non-mem-exp (*set-ContextReg-bits arg0 arg1*)
by (*non-mem-expI simp*: *set-ContextReg-bits-def*)

lemma *non-mem-exp-set-ContextReg-PTEBase*[*non-mem-expI*]:
non-mem-exp (*set-ContextReg-PTEBase arg0 arg1*)
by (*non-mem-expI simp*: *set-ContextReg-PTEBase-def*)

lemma *non-mem-exp-set-ContextReg-BadVPN2*[*non-mem-expI*]:
non-mem-exp (*set-ContextReg-BadVPN2 arg0 arg1*)
by (*non-mem-expI simp*: *set-ContextReg-BadVPN2-def*)

lemma *non-mem-exp-set-XContextReg-bits*[*non-mem-expI*]:
non-mem-exp (*set-XContextReg-bits arg0 arg1*)
by (*non-mem-expI simp*: *set-XContextReg-bits-def*)

lemma *non-mem-exp-set-XContextReg-XPTEBase*[*non-mem-expI*]:
non-mem-exp (*set-XContextReg-XPTEBase arg0 arg1*)
by (*non-mem-expI simp*: *set-XContextReg-XPTEBase-def*)

lemma *non-mem-exp-set-XContextReg-XR*[*non-mem-expI*]:
non-mem-exp (*set-XContextReg-XR arg0 arg1*)
by (*non-mem-expI simp*: *set-XContextReg-XR-def*)

lemma *non-mem-exp-set-XContextReg-XBadVPN2*[*non-mem-expI*]:
non-mem-exp (*set-XContextReg-XBadVPN2 arg0 arg1*)
by (*non-mem-expI simp*: *set-XContextReg-XBadVPN2-def*)

lemma *non-mem-exp-set-TLBEntry-bits*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-bits arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-bits-def*)

lemma *non-mem-exp-set-TLBEntry-pagemask*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-pagemask arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-pagemask-def*)

lemma *non-mem-exp-set-TLBEntry-r*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-r arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-r-def*)

lemma *non-mem-exp-set-TLBEntry-vpn2*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-vpn2 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-vpn2-def*)

lemma *non-mem-exp-set-TLBEntry-asid*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-asid arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-asid-def*)

lemma *non-mem-exp-set-TLBEntry-g*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-g arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-g-def*)

lemma *non-mem-exp-set-TLBEntry-valid*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-valid arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-valid-def*)

lemma *non-mem-exp-set-TLBEntry-caps1*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-caps1 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-caps1-def*)

lemma *non-mem-exp-set-TLBEntry-capl1*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-capl1 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-capl1-def*)

lemma *non-mem-exp-set-TLBEntry-pfn1*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-pfn1 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-pfn1-def*)

lemma *non-mem-exp-set-TLBEntry-c1*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-c1 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-c1-def*)

lemma *non-mem-exp-set-TLBEntry-d1*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-d1 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-d1-def*)

lemma *non-mem-exp-set-TLBEntry-v1*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-v1 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-v1-def*)

lemma *non-mem-exp-set-TLBEntry-caps0*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-caps0 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-caps0-def*)

lemma *non-mem-exp-set-TLBEntry-capl0*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-capl0 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-capl0-def*)

lemma *non-mem-exp-set-TLBEntry-pfn0*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-pfn0 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-pfn0-def*)

lemma *non-mem-exp-set-TLBEntry-c0*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-c0 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-c0-def*)

lemma *non-mem-exp-set-TLBEntry-d0*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-d0 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-d0-def*)

lemma *non-mem-exp-set-TLBEntry-v0*[*non-mem-expI*]:
non-mem-exp (*set-TLBEntry-v0 arg0 arg1*)
by (*non-mem-expI simp*: *set-TLBEntry-v0-def*)

lemma *non-mem-exp-set-StatusReg-bits*[*non-mem-expI*]:
non-mem-exp (*set-StatusReg-bits arg0 arg1*)
by (*non-mem-expI simp*: *set-StatusReg-bits-def*)

lemma *non-mem-exp-set-StatusReg-CU*[*non-mem-expI*]:
non-mem-exp (*set-StatusReg-CU arg0 arg1*)
by (*non-mem-expI simp*: *set-StatusReg-CU-def*)

lemma *non-mem-exp-set-StatusReg-BEV*[*non-mem-expI*]:
non-mem-exp (*set-StatusReg-BEV arg0 arg1*)
by (*non-mem-expI simp*: *set-StatusReg-BEV-def*)

lemma *non-mem-exp-set-StatusReg-IM*[*non-mem-expI*]:
non-mem-exp (*set-StatusReg-IM arg0 arg1*)
by (*non-mem-expI simp*: *set-StatusReg-IM-def*)

lemma *non-mem-exp-set-StatusReg-KX*[*non-mem-expI*]:
non-mem-exp (*set-StatusReg-KX arg0 arg1*)
by (*non-mem-expI simp*: *set-StatusReg-KX-def*)

lemma *non-mem-exp-set-StatusReg-SX*[*non-mem-expI*]:
non-mem-exp (*set-StatusReg-SX arg0 arg1*)
by (*non-mem-expI simp*: *set-StatusReg-SX-def*)

lemma *non-mem-exp-set-StatusReg-UX*[*non-mem-expI*]:
non-mem-exp (*set-StatusReg-UX arg0 arg1*)
by (*non-mem-expI simp*: *set-StatusReg-UX-def*)

lemma *non-mem-exp-set-StatusReg-KSU*[*non-mem-expI*]:
non-mem-exp (*set-StatusReg-KSU arg0 arg1*)
by (*non-mem-expI simp*: *set-StatusReg-KSU-def*)

lemma *non-mem-exp-set-StatusReg-ERL*[*non-mem-expI*]:
non-mem-exp (*set-StatusReg-ERL arg0 arg1*)
by (*non-mem-expI simp*: *set-StatusReg-ERL-def*)

lemma *non-mem-exp-set-StatusReg-EXL*[*non-mem-expI*]:

non-mem-exp (*set-StatusReg-EXL arg0 arg1*)
by (*non-mem-expI simp*: *set-StatusReg-EXL-def*)

lemma *non-mem-exp-set-StatusReg-IE*[*non-mem-expI*]:
non-mem-exp (*set-StatusReg-IE arg0 arg1*)
by (*non-mem-expI simp*: *set-StatusReg-IE-def*)

lemma *non-mem-exp-set-CapCauseReg-ExcCode*[*non-mem-expI*]:
non-mem-exp (*set-CapCauseReg-ExcCode arg0 arg1*)
by (*non-mem-expI simp*: *set-CapCauseReg-ExcCode-def*)

lemma *non-mem-exp-set-CapCauseReg-RegNum*[*non-mem-expI*]:
non-mem-exp (*set-CapCauseReg-RegNum arg0 arg1*)
by (*non-mem-expI simp*: *set-CapCauseReg-RegNum-def*)

lemma *non-mem-exp-set-next-pcc*[*non-mem-expI*]:
non-mem-exp (*set-next-pcc arg0*)
by (*non-mem-expI simp*: *set-next-pcc-def*)

lemma *non-mem-exp-SignalException*[*non-mem-expI*]:
non-mem-exp (*SignalException arg0*)
by (*non-mem-expI simp*: *SignalException-def*)

lemma *non-mem-exp-SignalExceptionBadAddr*[*non-mem-expI*]:
non-mem-exp (*SignalExceptionBadAddr arg0 arg1*)
by (*non-mem-expI simp*: *SignalExceptionBadAddr-def*)

lemma *non-mem-exp-SignalExceptionTLB*[*non-mem-expI*]:
non-mem-exp (*SignalExceptionTLB arg0 arg1*)
by (*non-mem-expI simp*: *SignalExceptionTLB-def*)

lemma *non-mem-exp-pcc-access-system-regs*[*non-mem-expI*]:
non-mem-exp (*pcc-access-system-regs arg0*)
by (*non-mem-expI simp*: *pcc-access-system-regs-def*)

lemma *non-mem-exp-raise-c2-exception8*[*non-mem-expI*]:
non-mem-exp (*raise-c2-exception8 arg0 arg1*)
by (*non-mem-expI simp*: *raise-c2-exception8-def*)

lemma *non-mem-exp-raise-c2-exception-noreg*[*non-mem-expI*]:
non-mem-exp (*raise-c2-exception-noreg arg0*)
by (*non-mem-expI simp*: *raise-c2-exception-noreg-def*)

lemma *non-mem-exp-checkCP0AccessHook*[*non-mem-expI*]:
non-mem-exp (*checkCP0AccessHook arg0*)
by (*non-mem-expI simp*: *checkCP0AccessHook-def*)

lemma *non-mem-exp-checkCP0Access*[*non-mem-expI*]:
non-mem-exp (*checkCP0Access arg0*)

by (*non-mem-expI simp*: *checkCP0Access-def*)

lemma *non-mem-exp-incrementCP0Count*[*non-mem-expI*]:
non-mem-exp (*incrementCP0Count arg0*)
by (*non-mem-expI simp*: *incrementCP0Count-def*)

lemma *non-mem-exp-TLBTranslate2*[*non-mem-expI*]:
non-mem-exp (*TLBTranslate2 arg0 arg1*)
by (*non-mem-expI simp*: *TLBTranslate2-def*)

lemma *non-mem-exp-TLBTranslateC*[*non-mem-expI*]:
non-mem-exp (*TLBTranslateC arg0 arg1*)
by (*non-mem-expI simp*: *TLBTranslateC-def*)

lemma *non-mem-exp-TLBTranslate*[*non-mem-expI*]:
non-mem-exp (*TLBTranslate arg0 arg1*)
by (*non-mem-expI simp*: *TLBTranslate-def*)

lemma *non-mem-exp-execute-branch-pcc*[*non-mem-expI*]:
non-mem-exp (*execute-branch-pcc arg0*)
by (*non-mem-expI simp*: *execute-branch-pcc-def*)

lemma *non-mem-exp-ERETHook*[*non-mem-expI*]:
non-mem-exp (*ERETHook arg0*)
by (*non-mem-expI simp*: *ERETHook-def*)

lemma *non-mem-exp-raise-c2-exception*[*non-mem-expI*]:
non-mem-exp (*raise-c2-exception arg0 arg1*)
by (*non-mem-expI simp*: *raise-c2-exception-def*)

lemma *non-mem-exp-checkDDCPerms*[*non-mem-expI*]:
non-mem-exp (*checkDDCPerms arg0 arg1*)
by (*non-mem-expI simp*: *checkDDCPerms-def*)

lemma *non-mem-exp-addrWrapper*[*non-mem-expI*]:
non-mem-exp (*addrWrapper arg0 arg1 arg2*)
by (*non-mem-expI simp*: *addrWrapper-def*)

lemma *non-mem-exp-addrWrapperUnaligned*[*non-mem-expI*]:
non-mem-exp (*addrWrapperUnaligned arg0 arg1 arg2*)
by (*non-mem-expI simp*: *addrWrapperUnaligned-def*)

lemma *non-mem-exp-execute-branch*[*non-mem-expI*]:
non-mem-exp (*execute-branch arg0*)
by (*non-mem-expI simp*: *execute-branch-def*)

lemma *non-mem-exp-TranslatePC*[*non-mem-expI*]:
non-mem-exp (*TranslatePC arg0*)
by (*non-mem-expI simp*: *TranslatePC-def*)

lemma *non-mem-exp-checkCP2usable*[*non-mem-expI*]:
non-mem-exp (*checkCP2usable arg0*)
by (*non-mem-expI simp*: *checkCP2usable-def*)

lemma *non-mem-exp-get-CP0EPC*[*non-mem-expI*]:
non-mem-exp (*get-CP0EPC arg0*)
by (*non-mem-expI simp*: *get-CP0EPC-def*)

lemma *non-mem-exp-set-CP0EPC*[*non-mem-expI*]:
non-mem-exp (*set-CP0EPC arg0*)
by (*non-mem-expI simp*: *set-CP0EPC-def*)

lemma *non-mem-exp-get-CP0ErrorEPC*[*non-mem-expI*]:
non-mem-exp (*get-CP0ErrorEPC arg0*)
by (*non-mem-expI simp*: *get-CP0ErrorEPC-def*)

lemma *non-mem-exp-set-CP0ErrorEPC*[*non-mem-expI*]:
non-mem-exp (*set-CP0ErrorEPC arg0*)
by (*non-mem-expI simp*: *set-CP0ErrorEPC-def*)

lemma *non-mem-exp-dump-cp2-state*[*non-mem-expI*]:
non-mem-exp (*dump-cp2-state arg0*)
by (*non-mem-expI simp*: *dump-cp2-state-def*)

lemma *non-mem-exp-TLBWriteEntry*[*non-mem-expI*]:
non-mem-exp (*TLBWriteEntry arg0*)
by (*non-mem-expI simp*: *TLBWriteEntry-def*)

lemma *non-mem-exp-execute-WAIT*[*non-mem-expI*]:
non-mem-exp (*execute-WAIT arg0*)
by (*non-mem-expI simp*: *execute-WAIT-def*)

lemma *non-mem-exp-execute-TRAPREG*[*non-mem-expI*]:
non-mem-exp (*execute-TRAPREG arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-TRAPREG-def*)

lemma *non-mem-exp-execute-TRAPIMM*[*non-mem-expI*]:
non-mem-exp (*execute-TRAPIMM arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-TRAPIMM-def*)

lemma *non-mem-exp-execute-TLBWR*[*non-mem-expI*]:
non-mem-exp (*execute-TLBWR arg0*)
by (*non-mem-expI simp*: *execute-TLBWR-def*)

lemma *non-mem-exp-execute-TLBWI*[*non-mem-expI*]:
non-mem-exp (*execute-TLBWI arg0*)
by (*non-mem-expI simp*: *execute-TLBWI-def*)

lemma *non-mem-exp-execute-TLBR*[*non-mem-expI*]:
non-mem-exp (*execute-TLBR arg0*)
by (*non-mem-expI simp*: *execute-TLBR-def*)

lemma *non-mem-exp-execute-TLBP*[*non-mem-expI*]:
non-mem-exp (*execute-TLBP arg0*)
by (*non-mem-expI simp*: *execute-TLBP-def*)

lemma *non-mem-exp-execute-SYSCALL*[*non-mem-expI*]:
non-mem-exp (*execute-SYSCALL arg0*)
by (*non-mem-expI simp*: *execute-SYSCALL-def*)

lemma *non-mem-exp-execute-SUB*[*non-mem-expI*]:
non-mem-exp (*execute-SUB arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-SUB-def*)

lemma *non-mem-exp-execute-RI*[*non-mem-expI*]:
non-mem-exp (*execute-RI arg0*)
by (*non-mem-expI simp*: *execute-RI-def*)

lemma *non-mem-exp-execute-RDHWR*[*non-mem-expI*]:
non-mem-exp (*execute-RDHWR arg0 arg1*)
by (*non-mem-expI simp*: *execute-RDHWR-def*)

lemma *non-mem-exp-execute-MTC0*[*non-mem-expI*]:
non-mem-exp (*execute-MTC0 arg0 arg1 arg2 arg3*)
by (*non-mem-expI simp*: *execute-MTC0-def*)

lemma *non-mem-exp-execute-MFC0*[*non-mem-expI*]:
non-mem-exp (*execute-MFC0 arg0 arg1 arg2 arg3*)
by (*non-mem-expI simp*: *execute-MFC0-def*)

lemma *non-mem-exp-execute-JR*[*non-mem-expI*]:
non-mem-exp (*execute-JR arg0*)
by (*non-mem-expI simp*: *execute-JR-def*)

lemma *non-mem-exp-execute-JALR*[*non-mem-expI*]:
non-mem-exp (*execute-JALR arg0 arg1*)
by (*non-mem-expI simp*: *execute-JALR-def*)

lemma *non-mem-exp-execute-JAL*[*non-mem-expI*]:
non-mem-exp (*execute-JAL arg0*)
by (*non-mem-expI simp*: *execute-JAL-def*)

lemma *non-mem-exp-execute-J*[*non-mem-expI*]:
non-mem-exp (*execute-J arg0*)
by (*non-mem-expI simp*: *execute-J-def*)

lemma *non-mem-exp-execute-ERET*[*non-mem-expI*]:

non-mem-exp (*execute-ERET arg0*)
by (*non-mem-expI simp*: *execute-ERET-def*)

lemma *non-mem-exp-execute-DSUB*[*non-mem-expI*]:
non-mem-exp (*execute-DSUB arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-DSUB-def*)

lemma *non-mem-exp-execute-DADDI*[*non-mem-expI*]:
non-mem-exp (*execute-DADDI arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-DADDI-def*)

lemma *non-mem-exp-execute-DADD*[*non-mem-expI*]:
non-mem-exp (*execute-DADD arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-DADD-def*)

lemma *non-mem-exp-execute-ClearRegs*[*non-mem-expI*]:
non-mem-exp (*execute-ClearRegs arg0 arg1*)
by (*non-mem-expI simp*: *execute-ClearRegs-def*)

lemma *non-mem-exp-execute-CWriteHwr*[*non-mem-expI*]:
non-mem-exp (*execute-CWriteHwr arg0 arg1*)
by (*non-mem-expI simp*: *execute-CWriteHwr-def*)

lemma *non-mem-exp-execute-CUnseal*[*non-mem-expI*]:
non-mem-exp (*execute-CUnseal arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CUnseal-def*)

lemma *non-mem-exp-execute-CToPtr*[*non-mem-expI*]:
non-mem-exp (*execute-CToPtr arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CToPtr-def*)

lemma *non-mem-exp-execute-CTestSubset*[*non-mem-expI*]:
non-mem-exp (*execute-CTestSubset arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CTestSubset-def*)

lemma *non-mem-exp-execute-CSub*[*non-mem-expI*]:
non-mem-exp (*execute-CSub arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CSub-def*)

lemma *non-mem-exp-execute-CSetOffset*[*non-mem-expI*]:
non-mem-exp (*execute-CSetOffset arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CSetOffset-def*)

lemma *non-mem-exp-execute-CSetFlags*[*non-mem-expI*]:
non-mem-exp (*execute-CSetFlags arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CSetFlags-def*)

lemma *non-mem-exp-execute-CSetCause*[*non-mem-expI*]:
non-mem-exp (*execute-CSetCause arg0*)

by (*non-mem-expI simp*: *execute-CSetCause-def*)

lemma *non-mem-exp-execute-CSetCID*[*non-mem-expI*]:
non-mem-exp (*execute-CSetCID arg0*)
by (*non-mem-expI simp*: *execute-CSetCID-def*)

lemma *non-mem-exp-execute-CSetBoundsImmediate*[*non-mem-expI*]:
non-mem-exp (*execute-CSetBoundsImmediate arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CSetBoundsImmediate-def*)

lemma *non-mem-exp-execute-CSetBoundsExact*[*non-mem-expI*]:
non-mem-exp (*execute-CSetBoundsExact arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CSetBoundsExact-def*)

lemma *non-mem-exp-execute-CSetBounds*[*non-mem-expI*]:
non-mem-exp (*execute-CSetBounds arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CSetBounds-def*)

lemma *non-mem-exp-execute-CSetAddr*[*non-mem-expI*]:
non-mem-exp (*execute-CSetAddr arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CSetAddr-def*)

lemma *non-mem-exp-execute-CSeal*[*non-mem-expI*]:
non-mem-exp (*execute-CSeal arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CSeal-def*)

lemma *non-mem-exp-execute-CReturn*[*non-mem-expI*]:
non-mem-exp (*execute-CReturn arg0*)
by (*non-mem-expI simp*: *execute-CReturn-def*)

lemma *non-mem-exp-execute-CReadHwr*[*non-mem-expI*]:
non-mem-exp (*execute-CReadHwr arg0 arg1*)
by (*non-mem-expI simp*: *execute-CReadHwr-def*)

lemma *non-mem-exp-execute-CRAP*[*non-mem-expI*]:
non-mem-exp (*execute-CRAP arg0 arg1*)
by (*non-mem-expI simp*: *execute-CRAP-def*)

lemma *non-mem-exp-execute-CRAM*[*non-mem-expI*]:
non-mem-exp (*execute-CRAM arg0 arg1*)
by (*non-mem-expI simp*: *execute-CRAM-def*)

lemma *non-mem-exp-execute-CPtrCmp*[*non-mem-expI*]:
non-mem-exp (*execute-CPtrCmp arg0 arg1 arg2 arg3*)
by (*non-mem-expI simp*: *execute-CPtrCmp-def*)

lemma *non-mem-exp-execute-CMove*[*non-mem-expI*]:
non-mem-exp (*execute-CMove arg0 arg1*)
by (*non-mem-expI simp*: *execute-CMove-def*)

lemma *non-mem-exp-execute-CMOVX*[*non-mem-expI*]:
non-mem-exp (*execute-CMOVX arg0 arg1 arg2 arg3*)
by (*non-mem-expI simp*: *execute-CMOVX-def*)

lemma *non-mem-exp-execute-CJALR*[*non-mem-expI*]:
non-mem-exp (*execute-CJALR arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CJALR-def*)

lemma *non-mem-exp-execute-CIncOffsetImmediate*[*non-mem-expI*]:
non-mem-exp (*execute-CIncOffsetImmediate arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CIncOffsetImmediate-def*)

lemma *non-mem-exp-execute-CIncOffset*[*non-mem-expI*]:
non-mem-exp (*execute-CIncOffset arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CIncOffset-def*)

lemma *non-mem-exp-execute-CGetType*[*non-mem-expI*]:
non-mem-exp (*execute-CGetType arg0 arg1*)
by (*non-mem-expI simp*: *execute-CGetType-def*)

lemma *non-mem-exp-execute-CGetTag*[*non-mem-expI*]:
non-mem-exp (*execute-CGetTag arg0 arg1*)
by (*non-mem-expI simp*: *execute-CGetTag-def*)

lemma *non-mem-exp-execute-CGetSealed*[*non-mem-expI*]:
non-mem-exp (*execute-CGetSealed arg0 arg1*)
by (*non-mem-expI simp*: *execute-CGetSealed-def*)

lemma *non-mem-exp-execute-CGetPerm*[*non-mem-expI*]:
non-mem-exp (*execute-CGetPerm arg0 arg1*)
by (*non-mem-expI simp*: *execute-CGetPerm-def*)

lemma *non-mem-exp-execute-CGetPCCSetOffset*[*non-mem-expI*]:
non-mem-exp (*execute-CGetPCCSetOffset arg0 arg1*)
by (*non-mem-expI simp*: *execute-CGetPCCSetOffset-def*)

lemma *non-mem-exp-execute-CGetPCC*[*non-mem-expI*]:
non-mem-exp (*execute-CGetPCC arg0*)
by (*non-mem-expI simp*: *execute-CGetPCC-def*)

lemma *non-mem-exp-execute-CGetOffset*[*non-mem-expI*]:
non-mem-exp (*execute-CGetOffset arg0 arg1*)
by (*non-mem-expI simp*: *execute-CGetOffset-def*)

lemma *non-mem-exp-execute-CGetLen*[*non-mem-expI*]:
non-mem-exp (*execute-CGetLen arg0 arg1*)
by (*non-mem-expI simp*: *execute-CGetLen-def*)

lemma *non-mem-exp-execute-CGetFlags*[*non-mem-expI*]:
non-mem-exp (*execute-CGetFlags arg0 arg1*)
by (*non-mem-expI simp*: *execute-CGetFlags-def*)

lemma *non-mem-exp-execute-CGetCause*[*non-mem-expI*]:
non-mem-exp (*execute-CGetCause arg0*)
by (*non-mem-expI simp*: *execute-CGetCause-def*)

lemma *non-mem-exp-execute-CGetCID*[*non-mem-expI*]:
non-mem-exp (*execute-CGetCID arg0*)
by (*non-mem-expI simp*: *execute-CGetCID-def*)

lemma *non-mem-exp-execute-CGetBase*[*non-mem-expI*]:
non-mem-exp (*execute-CGetBase arg0 arg1*)
by (*non-mem-expI simp*: *execute-CGetBase-def*)

lemma *non-mem-exp-execute-CGetAndAddr*[*non-mem-expI*]:
non-mem-exp (*execute-CGetAndAddr arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CGetAndAddr-def*)

lemma *non-mem-exp-execute-CGetAddr*[*non-mem-expI*]:
non-mem-exp (*execute-CGetAddr arg0 arg1*)
by (*non-mem-expI simp*: *execute-CGetAddr-def*)

lemma *non-mem-exp-execute-CFromPtr*[*non-mem-expI*]:
non-mem-exp (*execute-CFromPtr arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CFromPtr-def*)

lemma *non-mem-exp-execute-CCopyType*[*non-mem-expI*]:
non-mem-exp (*execute-CCopyType arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CCopyType-def*)

lemma *non-mem-exp-execute-CClearTag*[*non-mem-expI*]:
non-mem-exp (*execute-CClearTag arg0 arg1*)
by (*non-mem-expI simp*: *execute-CClearTag-def*)

lemma *non-mem-exp-execute-CCheckType*[*non-mem-expI*]:
non-mem-exp (*execute-CCheckType arg0 arg1*)
by (*non-mem-expI simp*: *execute-CCheckType-def*)

lemma *non-mem-exp-execute-CCheckTag*[*non-mem-expI*]:
non-mem-exp (*execute-CCheckTag arg0*)
by (*non-mem-expI simp*: *execute-CCheckTag-def*)

lemma *non-mem-exp-execute-CCheckPerm*[*non-mem-expI*]:
non-mem-exp (*execute-CCheckPerm arg0 arg1*)
by (*non-mem-expI simp*: *execute-CCheckPerm-def*)

lemma *non-mem-exp-execute-CCall*[*non-mem-expI*]:

non-mem-exp (*execute-CCall arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CCall-def*)

lemma *non-mem-exp-execute-CCSeal*[*non-mem-expI*]:
non-mem-exp (*execute-CCSeal arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CCSeal-def*)

lemma *non-mem-exp-execute-CBuildCap*[*non-mem-expI*]:
non-mem-exp (*execute-CBuildCap arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CBuildCap-def*)

lemma *non-mem-exp-execute-CBZ*[*non-mem-expI*]:
non-mem-exp (*execute-CBZ arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CBZ-def*)

lemma *non-mem-exp-execute-CBX*[*non-mem-expI*]:
non-mem-exp (*execute-CBX arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CBX-def*)

lemma *non-mem-exp-execute-CAndPerm*[*non-mem-expI*]:
non-mem-exp (*execute-CAndPerm arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CAndPerm-def*)

lemma *non-mem-exp-execute-CAndAddr*[*non-mem-expI*]:
non-mem-exp (*execute-CAndAddr arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CAndAddr-def*)

lemma *non-mem-exp-execute-CACHE*[*non-mem-expI*]:
non-mem-exp (*execute-CACHE arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-CACHE-def*)

lemma *non-mem-exp-execute-BREAK*[*non-mem-expI*]:
non-mem-exp (*execute-BREAK arg0*)
by (*non-mem-expI simp*: *execute-BREAK-def*)

lemma *non-mem-exp-execute-BEQ*[*non-mem-expI*]:
non-mem-exp (*execute-BEQ arg0 arg1 arg2 arg3 arg4*)
by (*non-mem-expI simp*: *execute-BEQ-def*)

lemma *non-mem-exp-execute-BCMPZ*[*non-mem-expI*]:
non-mem-exp (*execute-BCMPZ arg0 arg1 arg2 arg3 arg4*)
by (*non-mem-expI simp*: *execute-BCMPZ-def*)

lemma *non-mem-exp-execute-ADDI*[*non-mem-expI*]:
non-mem-exp (*execute-ADDI arg0 arg1 arg2*)
by (*non-mem-expI simp*: *execute-ADDI-def*)

lemma *non-mem-exp-execute-ADD*[*non-mem-expI*]:
non-mem-exp (*execute-ADD arg0 arg1 arg2*)

by (*non-mem-expI simp*: *execute-ADD-def*)

lemma *non-mem-exp-cp2-next-pc*[*non-mem-expI*]:
non-mem-exp (*cp2-next-pc u*)
by (*non-mem-expI simp*: *cp2-next-pc-def*)

lemma *no-reg-writes-to-undefined-option*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-option arg0*)
unfolding *undefined-option-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MIPS-write*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MIPS-write arg0 arg1 arg2*)
unfolding *MIPS-write-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MIPS-read*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MIPS-read arg0 arg1*)
unfolding *MIPS-read-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-exception*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-exception arg0*)
unfolding *undefined-exception-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-CauseReg*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-CauseReg arg0*)
unfolding *undefined-CauseReg-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CauseReg-bits*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-CauseReg-bits arg0 arg1*)
using *assms*
unfolding *set-CauseReg-bits-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CauseReg-BD*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-CauseReg-BD arg0 arg1*)
using *assms*
unfolding *set-CauseReg-BD-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CauseReg-CE*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-CauseReg-CE arg0 arg1*)
using *assms*

unfolding *set-CauseReg-CE-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CauseReg-IV*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-CauseReg-IV arg0 arg1*)
using *assms*
unfolding *set-CauseReg-IV-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CauseReg-WP*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-CauseReg-WP arg0 arg1*)
using *assms*
unfolding *set-CauseReg-WP-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CauseReg-IP*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-CauseReg-IP arg0 arg1*)
using *assms*
unfolding *set-CauseReg-IP-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CauseReg-ExcCode*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-CauseReg-ExcCode arg0 arg1*)
using *assms*
unfolding *set-CauseReg-ExcCode-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-TLBEntryLoReg*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-TLBEntryLoReg arg0*)
unfolding *undefined-TLBEntryLoReg-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryLoReg-bits*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryLoReg-bits arg0 arg1*)
using *assms*
unfolding *set-TLBEntryLoReg-bits-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryLoReg-CapS*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryLoReg-CapS arg0 arg1*)
using *assms*
unfolding *set-TLBEntryLoReg-CapS-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryLoReg-CapL*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryLoReg-CapL arg0 arg1*)
using *assms*
unfolding *set-TLBEntryLoReg-CapL-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryLoReg-PFN*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryLoReg-PFN arg0 arg1*)
using *assms*
unfolding *set-TLBEntryLoReg-PFN-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryLoReg-C*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryLoReg-C arg0 arg1*)
using *assms*
unfolding *set-TLBEntryLoReg-C-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryLoReg-D*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryLoReg-D arg0 arg1*)
using *assms*
unfolding *set-TLBEntryLoReg-D-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryLoReg-V*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryLoReg-V arg0 arg1*)
using *assms*
unfolding *set-TLBEntryLoReg-V-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryLoReg-G*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryLoReg-G arg0 arg1*)
using *assms*
unfolding *set-TLBEntryLoReg-G-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-TLBEntryHiReg*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-TLBEntryHiReg arg0*)
unfolding *undefined-TLBEntryHiReg-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryHiReg-bits*[*no-reg-writes-toI*, *simp*]:

assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryHiReg-bits arg0 arg1*)
using *assms*
unfolding *set-TLBEntryHiReg-bits-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryHiReg-R*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryHiReg-R arg0 arg1*)
using *assms*
unfolding *set-TLBEntryHiReg-R-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryHiReg-VPN2*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryHiReg-VPN2 arg0 arg1*)
using *assms*
unfolding *set-TLBEntryHiReg-VPN2-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntryHiReg-ASID*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntryHiReg-ASID arg0 arg1*)
using *assms*
unfolding *set-TLBEntryHiReg-ASID-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-ContextReg*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-ContextReg arg0*)
unfolding *undefined-ContextReg-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-ContextReg-bits*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-ContextReg-bits arg0 arg1*)
using *assms*
unfolding *set-ContextReg-bits-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-ContextReg-PTEBase*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-ContextReg-PTEBase arg0 arg1*)
using *assms*
unfolding *set-ContextReg-PTEBase-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-ContextReg-BadVPN2*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-ContextReg-BadVPN2 arg0 arg1*)

using *assms*
unfolding *set-ContextReg-BadVPN2-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-XContextReg*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-XContextReg arg0*)
unfolding *undefined-XContextReg-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-XContextReg-bits*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-XContextReg-bits arg0 arg1*)
using *assms*
unfolding *set-XContextReg-bits-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-XContextReg-XPTEBase*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-XContextReg-XPTEBase arg0 arg1*)
using *assms*
unfolding *set-XContextReg-XPTEBase-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-XContextReg-XR*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-XContextReg-XR arg0 arg1*)
using *assms*
unfolding *set-XContextReg-XR-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-XContextReg-XBadVPN2*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-XContextReg-XBadVPN2 arg0 arg1*)
using *assms*
unfolding *set-XContextReg-XBadVPN2-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-TLBEntry*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-TLBEntry arg0*)
unfolding *undefined-TLBEntry-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-bits*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-bits arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-bits-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-pagemask*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-pagemask arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-pagemask-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-r*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-r arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-r-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-vpn2*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-vpn2 arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-vpn2-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-asid*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-asid arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-asid-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-g*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-g arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-g-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-valid*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-valid arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-valid-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-caps1*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-caps1 arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-caps1-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-capl1* [*no-reg-writes-toI*, *simp*]:
  assumes *name arg0* $\notin$ *Rs*
  shows *no-reg-writes-to Rs* (*set-TLBEntry-capl1 arg0 arg1*)
  using *assms*
  unfolding *set-TLBEntry-capl1-def bind-assoc*
  by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-pfn1* [*no-reg-writes-toI*, *simp*]:
  assumes *name arg0* $\notin$ *Rs*
  shows *no-reg-writes-to Rs* (*set-TLBEntry-pfn1 arg0 arg1*)
  using *assms*
  unfolding *set-TLBEntry-pfn1-def bind-assoc*
  by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-c1* [*no-reg-writes-toI*, *simp*]:
  assumes *name arg0* $\notin$ *Rs*
  shows *no-reg-writes-to Rs* (*set-TLBEntry-c1 arg0 arg1*)
  using *assms*
  unfolding *set-TLBEntry-c1-def bind-assoc*
  by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-d1* [*no-reg-writes-toI*, *simp*]:
  assumes *name arg0* $\notin$ *Rs*
  shows *no-reg-writes-to Rs* (*set-TLBEntry-d1 arg0 arg1*)
  using *assms*
  unfolding *set-TLBEntry-d1-def bind-assoc*
  by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-v1* [*no-reg-writes-toI*, *simp*]:
  assumes *name arg0* $\notin$ *Rs*
  shows *no-reg-writes-to Rs* (*set-TLBEntry-v1 arg0 arg1*)
  using *assms*
  unfolding *set-TLBEntry-v1-def bind-assoc*
  by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-caps0* [*no-reg-writes-toI*, *simp*]:
  assumes *name arg0* $\notin$ *Rs*
  shows *no-reg-writes-to Rs* (*set-TLBEntry-caps0 arg0 arg1*)
  using *assms*
  unfolding *set-TLBEntry-caps0-def bind-assoc*
  by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-capl0* [*no-reg-writes-toI*, *simp*]:
  assumes *name arg0* $\notin$ *Rs*
  shows *no-reg-writes-to Rs* (*set-TLBEntry-capl0 arg0 arg1*)
  using *assms*
  unfolding *set-TLBEntry-capl0-def bind-assoc*
  by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-pfn0*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-pfn0 arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-pfn0-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-c0*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-c0 arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-c0-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-d0*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-d0 arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-d0-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-TLBEntry-v0*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-TLBEntry-v0 arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-v0-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-StatusReg*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-StatusReg arg0*)
unfolding *undefined-StatusReg-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-StatusReg-bits*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-StatusReg-bits arg0 arg1*)
using *assms*
unfolding *set-StatusReg-bits-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-StatusReg-CU*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-StatusReg-CU arg0 arg1*)
using *assms*
unfolding *set-StatusReg-CU-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-StatusReg-BEV*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*

shows *no-reg-writes-to Rs* (*set-StatusReg-BEV arg0 arg1*)
using *assms*
unfolding *set-StatusReg-BEV-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-StatusReg-IM*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-StatusReg-IM arg0 arg1*)
using *assms*
unfolding *set-StatusReg-IM-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-StatusReg-KX*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-StatusReg-KX arg0 arg1*)
using *assms*
unfolding *set-StatusReg-KX-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-StatusReg-SX*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-StatusReg-SX arg0 arg1*)
using *assms*
unfolding *set-StatusReg-SX-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-StatusReg-UX*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-StatusReg-UX arg0 arg1*)
using *assms*
unfolding *set-StatusReg-UX-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-StatusReg-KSU*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-StatusReg-KSU arg0 arg1*)
using *assms*
unfolding *set-StatusReg-KSU-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-StatusReg-ERL*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-StatusReg-ERL arg0 arg1*)
using *assms*
unfolding *set-StatusReg-ERL-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-StatusReg-EXL*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*

shows *no-reg-writes-to Rs* (*set-StatusReg-EXL arg0 arg1*)
using *assms*
unfolding *set-StatusReg-EXL-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-StatusReg-IE*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-StatusReg-IE arg0 arg1*)
using *assms*
unfolding *set-StatusReg-IE-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-execute-branch-mips*[*no-reg-writes-toI*, *simp*]:
assumes {*''BranchPending''*, *''DelayedPC''*, *''NextInBranchDelay''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*execute-branch-mips arg0*)
using *assms*
unfolding *execute-branch-mips-def bind-assoc*
by (*no-reg-writes-toI simp*: *BranchPending-ref-def DelayedPC-ref-def NextInBranchDelay-ref-def*)

lemma *no-reg-writes-to-rGPR*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*rGPR arg0*)
unfolding *rGPR-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-wGPR*[*no-reg-writes-toI*, *simp*]:
assumes {*''GPR''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*wGPR arg0 arg1*)
using *assms*
unfolding *wGPR-def bind-assoc*
by (*no-reg-writes-toI simp*: *GPR-ref-def*)

lemma *no-reg-writes-to-MEMr*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMr arg0 arg1*)
unfolding *MEMr-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MEMr-reserve*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMr-reserve arg0 arg1*)
unfolding *MEMr-reserve-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MEM-sync*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEM-sync arg0*)
unfolding *MEM-sync-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MEMea*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMea arg0 arg1*)
unfolding *MEMea-def bind-assoc*

by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MEMea-conditional*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMea-conditional arg0 arg1*)
unfolding *MEMea-conditional-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MEMval*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMval arg0 arg1 arg2*)
unfolding *MEMval-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MEMval-conditional*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMval-conditional arg0 arg1 arg2*)
unfolding *MEMval-conditional-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-Exception*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-Exception arg0*)
unfolding *undefined-Exception-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-exceptionVectorOffset*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*exceptionVectorOffset arg0*)
unfolding *exceptionVectorOffset-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-exceptionVectorBase*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*exceptionVectorBase arg0*)
unfolding *exceptionVectorBase-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-updateBadInstr*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*updateBadInstr arg0*)
using *assms*
unfolding *updateBadInstr-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def*)

lemma *no-reg-writes-to-undefined-Capability*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-Capability arg0*)
unfolding *undefined-Capability-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-next-pcc*[*no-reg-writes-toI*, *simp*]:
assumes {*''DelayedPCC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*set-next-pcc arg0*)
using *assms*
unfolding *set-next-pcc-def bind-assoc*

by (*no-reg-writes-toI simp*: *DelayedPCC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-SignalException*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Status''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*SignalException arg0*)
using *assms*
unfolding *SignalException-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def CP0Status-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-SignalExceptionBadAddr*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0BadVAddr''*, *''CP0Cause''*, *''CP0Status''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*SignalExceptionBadAddr arg0 arg1*)
using *assms*
unfolding *SignalExceptionBadAddr-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0BadVAddr-ref-def CP0Cause-ref-def CP0Status-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-SignalExceptionTLB*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0BadVAddr''*, *''CP0Cause''*, *''CP0Status''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*, *''TLBContext''*, *''TLBEntryHi''*, *''TLBXContext''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*SignalExceptionTLB arg0 arg1*)
using *assms*
unfolding *SignalExceptionTLB-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0BadVAddr-ref-def CP0Cause-ref-def CP0Status-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def TLBContext-ref-def TLBEntryHi-ref-def TLBXContext-ref-def*)

lemma *no-reg-writes-to-undefined-MemAccessType*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-MemAccessType arg0*)
unfolding *undefined-MemAccessType-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-AccessLevel*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-AccessLevel arg0*)
unfolding *undefined-AccessLevel-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-getAccessLevel*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*getAccessLevel arg0*)
unfolding *getAccessLevel-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-pcc-access-system-regs*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*pcc-access-system-regs arg0*)

unfolding *pcc-access-system-regs-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-CapCauseReg*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-CapCauseReg arg0*)
unfolding *undefined-CapCauseReg-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CapCauseReg-ExcCode*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-CapCauseReg-ExcCode arg0 arg1*)
using *assms*
unfolding *set-CapCauseReg-ExcCode-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CapCauseReg-RegNum*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-CapCauseReg-RegNum arg0 arg1*)
using *assms*
unfolding *set-CapCauseReg-RegNum-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-raise-c2-exception8*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Status''*, *''CapCause''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*raise-c2-exception8 arg0 arg1*)
using *assms*
unfolding *raise-c2-exception8-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def CP0Status-ref-def CapCause-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-raise-c2-exception-noreg*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Status''*, *''CapCause''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*raise-c2-exception-noreg arg0*)
using *assms*
unfolding *raise-c2-exception-noreg-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def CP0Status-ref-def CapCause-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-checkCP0AccessHook*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Status''*, *''CapCause''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*checkCP0AccessHook arg0*)
using *assms*
unfolding *checkCP0AccessHook-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def*

CP0Status-ref-def CapCause-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def)

lemma *no-reg-writes-to-checkCP0Access*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Status''*, *''CapCause''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*checkCP0Access arg0*)
using *assms*
unfolding *checkCP0Access-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def CP0Status-ref-def CapCause-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-incrementCP0Count*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Count''*, *''CP0Status''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*, *''TLBRandom''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*incrementCP0Count arg0*)
using *assms*
unfolding *incrementCP0Count-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def CP0Count-ref-def CP0Status-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def TLBRandom-ref-def*)

lemma *no-reg-writes-to-undefined-decode-failure*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-decode-failure arg0*)
unfolding *undefined-decode-failure-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-Comparison*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-Comparison arg0*)
unfolding *undefined-Comparison-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-WordType*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-WordType arg0*)
unfolding *undefined-WordType-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-WordTypeUnaligned*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-WordTypeUnaligned arg0*)
unfolding *undefined-WordTypeUnaligned-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MEMr-wrapper*[*no-reg-writes-toI*, *simp*]:
assumes {*''UART-RVALID''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*MEMr-wrapper arg0 arg1*)
using *assms*
unfolding *MEMr-wrapper-def bind-assoc*
by (*no-reg-writes-toI simp*: *UART-RVALID-ref-def*)

lemma *no-reg-writes-to-MEMr-reserve-wrapper*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMr-reserve-wrapper arg0 arg1*)
unfolding *MEMr-reserve-wrapper-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-init-cp0-state*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0Status''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*init-cp0-state arg0*)
using *assms*
unfolding *init-cp0-state-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0Status-ref-def*)

lemma *no-reg-writes-to-tlbSearch*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*tlbSearch arg0*)
unfolding *tlbSearch-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-TLBTranslate2*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0BadVAddr''*, *''CP0Cause''*, *''CP0Status''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*, *''TLBContext''*, *''TLBEntryHi''*, *''TLBXContext''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*TLBTranslate2 arg0 arg1*)
using *assms*
unfolding *TLBTranslate2-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0BadVAddr-ref-def CP0Cause-ref-def CP0Status-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def TLBContext-ref-def TLBEntryHi-ref-def TLBXContext-ref-def*)

lemma *no-reg-writes-to-TLBTranslateC*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0BadVAddr''*, *''CP0Cause''*, *''CP0Status''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*, *''TLBContext''*, *''TLBEntryHi''*, *''TLBXContext''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*TLBTranslateC arg0 arg1*)
using *assms*
unfolding *TLBTranslateC-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0BadVAddr-ref-def CP0Cause-ref-def CP0Status-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def TLBContext-ref-def TLBEntryHi-ref-def TLBXContext-ref-def*)

lemma *no-reg-writes-to-TLBTranslate*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0BadVAddr''*, *''CP0Cause''*, *''CP0Status''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*, *''TLBContext''*, *''TLBEntryHi''*, *''TLBXContext''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*TLBTranslate arg0 arg1*)
using *assms*
unfolding *TLBTranslate-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0BadVAddr-ref-def CP0Cause-ref-def CP0Status-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def*

NextPCC-ref-def TLBContext-ref-def TLBEntryHi-ref-def TLBXContext-ref-def)

lemma *no-reg-writes-to-undefined-CPtrCmpOp*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-CPtrCmpOp arg0*)
unfolding *undefined-CPtrCmpOp-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-ClearRegSet*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-ClearRegSet arg0*)
unfolding *undefined-ClearRegSet-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-capToString*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*capToString arg0 arg1*)
unfolding *capToString-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-undefined-CapEx*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*undefined-CapEx arg0*)
unfolding *undefined-CapEx-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CapCauseReg-bits*[*no-reg-writes-toI*, *simp*]:
assumes *name arg0* $\notin$ *Rs*
shows *no-reg-writes-to Rs* (*set-CapCauseReg-bits arg0 arg1*)
using *assms*
unfolding *set-CapCauseReg-bits-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-execute-branch-pcc*[*no-reg-writes-toI*, *simp*]:
assumes {*''BranchPending''*, *''DelayedPC''*, *''DelayedPCC''*, *''NextInBranchDelay''*} $\cap$ *Rs* = {}
shows *no-reg-writes-to Rs* (*execute-branch-pcc arg0*)
using *assms*
unfolding *execute-branch-pcc-def bind-assoc*
by (*no-reg-writes-toI simp*: *BranchPending-ref-def DelayedPC-ref-def DelayedPCC-ref-def NextInBranchDelay-ref-def*)

lemma *no-reg-writes-to-ERETHook*[*no-reg-writes-toI*, *simp*]:
assumes {*''DelayedPCC''*, *''NextPCC''*} $\cap$ *Rs* = {}
shows *no-reg-writes-to Rs* (*ERETHook arg0*)
using *assms*
unfolding *ERETHook-def bind-assoc*
by (*no-reg-writes-toI simp*: *DelayedPCC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-raise-c2-exception*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Status''*, *''CapCause''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*} $\cap$ *Rs* = {}
shows *no-reg-writes-to Rs* (*raise-c2-exception arg0 arg1*)

using *assms*
unfolding *raise-c2-exception-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def CP0Status-ref-def CapCause-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-MEMr-tagged*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMr-tagged arg0 arg1 arg2*)
unfolding *MEMr-tagged-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MEMr-tagged-reserve*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMr-tagged-reserve arg0 arg1 arg2*)
unfolding *MEMr-tagged-reserve-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MEMw-tagged*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMw-tagged arg0 arg1 arg2 arg3*)
unfolding *MEMw-tagged-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MEMw-tagged-conditional*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMw-tagged-conditional arg0 arg1 arg2 arg3*)
unfolding *MEMw-tagged-conditional-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-MEMw-wrapper*[*no-reg-writes-toI*, *simp*]:
assumes {*''UART-WDATA''*, *''UART-WRITTEN''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*MEMw-wrapper arg0 arg1 arg2*)
using *assms*
unfolding *MEMw-wrapper-def bind-assoc*
by (*no-reg-writes-toI simp*: *UART-WDATA-ref-def UART-WRITTEN-ref-def*)

lemma *no-reg-writes-to-MEMw-conditional-wrapper*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*MEMw-conditional-wrapper arg0 arg1 arg2*)
unfolding *MEMw-conditional-wrapper-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-checkDDCPerms*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Status''*, *''CapCause''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*checkDDCPerms arg0 arg1*)
using *assms*
unfolding *checkDDCPerms-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def CP0Status-ref-def CapCause-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-addrWrapper*[*no-reg-writes-toI*, *simp*]:

assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Status''*, *''CapCause''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*addrWrapper arg0 arg1 arg2*)
using *assms*
unfolding *addrWrapper-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def CP0Status-ref-def CapCause-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-addrWrapperUnaligned*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Status''*, *''CapCause''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*addrWrapperUnaligned arg0 arg1 arg2*)
using *assms*
unfolding *addrWrapperUnaligned-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def CP0Status-ref-def CapCause-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-execute-branch*[*no-reg-writes-toI*, *simp*]:
assumes {*''BranchPending''*, *''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Status''*, *''CapCause''*, *''DelayedPC''*, *''DelayedPCC''*, *''EPCC''*, *''NextInBranchDelay''*, *''NextPC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*execute-branch arg0*)
using *assms*
unfolding *execute-branch-def bind-assoc*
by (*no-reg-writes-toI simp*: *BranchPending-ref-def CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def CP0Status-ref-def CapCause-ref-def DelayedPC-ref-def DelayedPCC-ref-def EPCC-ref-def NextInBranchDelay-ref-def NextPC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-TranslatePC*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0BadVAddr''*, *''CP0Cause''*, *''CP0Count''*, *''CP0Status''*, *''CapCause''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*, *''TLBContext''*, *''TLBEntryHi''*, *''TLBRandom''*, *''TLBXContext''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*TranslatePC arg0*)
using *assms*
unfolding *TranslatePC-def bind-assoc*
by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0BadVAddr-ref-def CP0Cause-ref-def CP0Count-ref-def CP0Status-ref-def CapCause-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def TLBContext-ref-def TLBEntryHi-ref-def TLBRandom-ref-def TLBXContext-ref-def*)

lemma *no-reg-writes-to-checkCP2usable*[*no-reg-writes-toI*, *simp*]:
assumes {*''CP0BadInstr''*, *''CP0BadInstrP''*, *''CP0Cause''*, *''CP0Status''*, *''DelayedPCC''*, *''EPCC''*, *''NextPC''*, *''NextPCC''*} ∩ *Rs* = {}
shows *no-reg-writes-to Rs* (*checkCP2usable arg0*)
using *assms*
unfolding *checkCP2usable-def bind-assoc*

by (*no-reg-writes-toI simp*: *CP0BadInstr-ref-def CP0BadInstrP-ref-def CP0Cause-ref-def CP0Status-ref-def DelayedPCC-ref-def EPCC-ref-def NextPC-ref-def NextPCC-ref-def*)

lemma *no-reg-writes-to-get-CP0EPC*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*get-CP0EPC arg0*)
unfolding *get-CP0EPC-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CP0EPC*[*no-reg-writes-toI*, *simp*]:
assumes $\{''EPCC''\} \cap Rs = \{\}$
shows *no-reg-writes-to Rs* (*set-CP0EPC arg0*)
using *assms*
unfolding *set-CP0EPC-def bind-assoc*
by (*no-reg-writes-toI simp*: *EPCC-ref-def*)

lemma *no-reg-writes-to-get-CP0ErrorEPC*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*get-CP0ErrorEPC arg0*)
unfolding *get-CP0ErrorEPC-def bind-assoc*
by (*no-reg-writes-toI*)

lemma *no-reg-writes-to-set-CP0ErrorEPC*[*no-reg-writes-toI*, *simp*]:
assumes $\{''ErrorEPCC''\} \cap Rs = \{\}$
shows *no-reg-writes-to Rs* (*set-CP0ErrorEPC arg0*)
using *assms*
unfolding *set-CP0ErrorEPC-def bind-assoc*
by (*no-reg-writes-toI simp*: *ErrorEPCC-ref-def*)

lemma *no-reg-writes-to-dump-cp2-state*[*no-reg-writes-toI*, *simp*]:
no-reg-writes-to Rs (*dump-cp2-state arg0*)
unfolding *dump-cp2-state-def bind-assoc*
by (*no-reg-writes-toI*)

end

end
theory *CHERI-MIPS-Reg-Axioms*
imports *CHERI-MIPS-Gen-Lemmas*
begin

3.3 Register and capability derivability properties of instructions

context *CHERI-MIPS-Reg-Automaton*
begin

lemma *preserves-invariant-write-non-inv-regs*[*preserves-invariantI*]:
$\bigwedge v$. *traces-preserve-invariant* (*write-reg BranchPending-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg C01-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg C02-ref v*)

$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C03-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C04-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C05-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C06-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C07-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C08-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C09-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C10-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C11-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C12-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C13-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C14-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C15-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C16-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C17-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C18-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C19-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C20-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C21-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C22-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C23-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C24-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C25-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C26-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C27-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C28-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C29-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C30-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg C31-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CID-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0BadInstr-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0BadInstrP-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0BadVAddr-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0Cause-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0Compare-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0ConfigK0-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0Count-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0HWREna-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0LLAddr-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0LLBit-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0Status-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CP0UserLocal-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CPLR-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CULR-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CapCause-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg CurrentInstrBits-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg DDC-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg DelayedPC-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg DelayedPCC-ref v*)

$\bigwedge v.$ *traces-preserve-invariant* (*write-reg EPCC-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg ErrorEPCC-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg GPR-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg HI-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg InBranchDelay-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg KCC-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg KDC-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg KR1C-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg KR2C-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg LO-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg LastInstrBits-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg NextInBranchDelay-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg NextPC-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg NextPCC-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg PC-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBContext-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry00-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry01-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry02-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry03-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry04-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry05-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry06-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry07-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry08-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry09-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry10-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry11-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry12-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry13-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry14-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry15-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry16-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry17-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry18-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry19-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry20-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry21-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry22-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry23-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry24-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry25-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry26-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry27-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry28-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry29-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry30-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry31-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry32-ref v*)

$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry33-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry34-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry35-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry36-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry37-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry38-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry39-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry40-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry41-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry42-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry43-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry44-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry45-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry46-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry47-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry48-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry49-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry50-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry51-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry52-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry53-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry54-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry55-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry56-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry57-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry58-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry59-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry60-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry61-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry62-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntry63-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntryHi-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntryLo0-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntryLo1-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBIndex-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBPageMask-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBProbe-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBRandom-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBWired-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBXContext-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg UART-RDATA-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg UART-RVALID-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg UART-WDATA-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg UART-WRITTEN-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg InstCount-ref v*)
unfolding *BranchPending-ref-def C01-ref-def C02-ref-def C03-ref-def C04-ref-def C05-ref-def C06-ref-def C07-ref-def C08-ref-def C09-ref-def C10-ref-def C11-ref-def C12-ref-def C13-ref-def C14-ref-def C15-ref-def C16-ref-def C17-ref-def C18-ref-def C19-ref-def*

C20-ref-def C21-ref-def C22-ref-def C23-ref-def C24-ref-def
C25-ref-def C26-ref-def C27-ref-def C28-ref-def C29-ref-def
C30-ref-def C31-ref-def CID-ref-def CP0BadInstr-ref-def CP0BadInstrP-ref-def
CP0BadVAddr-ref-def CP0Cause-ref-def CP0Compare-ref-def CP0ConfigK0-ref-def CP0Count-ref-def
CP0HWREna-ref-def CP0LLAddr-ref-def CP0LLBit-ref-def CP0Status-ref-def CP0UserLocal-ref-def
CPLR-ref-def CULR-ref-def CapCause-ref-def CurrentInstrBits-ref-def DDC-ref-def
DelayedPC-ref-def DelayedPCC-ref-def EPCC-ref-def ErrorEPCC-ref-def GPR-ref-def
HI-ref-def InBranchDelay-ref-def KCC-ref-def KDC-ref-def KR1C-ref-def
KR2C-ref-def LO-ref-def LastInstrBits-ref-def NextInBranchDelay-ref-def NextPC-ref-def
NextPCC-ref-def PC-ref-def TLBContext-ref-def TLBEntry00-ref-def TLBEntry01-ref-def
TLBEntry02-ref-def TLBEntry03-ref-def TLBEntry04-ref-def TLBEntry05-ref-def TLBEntry06-ref-def
TLBEntry07-ref-def TLBEntry08-ref-def TLBEntry09-ref-def TLBEntry10-ref-def TLBEntry11-ref-def
TLBEntry12-ref-def TLBEntry13-ref-def TLBEntry14-ref-def TLBEntry15-ref-def TLBEntry16-ref-def
TLBEntry17-ref-def TLBEntry18-ref-def TLBEntry19-ref-def TLBEntry20-ref-def TLBEntry21-ref-def
TLBEntry22-ref-def TLBEntry23-ref-def TLBEntry24-ref-def TLBEntry25-ref-def TLBEntry26-ref-def
TLBEntry27-ref-def TLBEntry28-ref-def TLBEntry29-ref-def TLBEntry30-ref-def TLBEntry31-ref-def
TLBEntry32-ref-def TLBEntry33-ref-def TLBEntry34-ref-def TLBEntry35-ref-def TLBEntry36-ref-def
TLBEntry37-ref-def TLBEntry38-ref-def TLBEntry39-ref-def TLBEntry40-ref-def TLBEntry41-ref-def
TLBEntry42-ref-def TLBEntry43-ref-def TLBEntry44-ref-def TLBEntry45-ref-def TLBEntry46-ref-def
TLBEntry47-ref-def TLBEntry48-ref-def TLBEntry49-ref-def TLBEntry50-ref-def TLBEntry51-ref-def
TLBEntry52-ref-def TLBEntry53-ref-def TLBEntry54-ref-def TLBEntry55-ref-def TLBEntry56-ref-def
TLBEntry57-ref-def TLBEntry58-ref-def TLBEntry59-ref-def TLBEntry60-ref-def TLBEntry61-ref-def
TLBEntry62-ref-def TLBEntry63-ref-def TLBEntryHi-ref-def TLBEntryLo0-ref-def TLBEntryLo1-ref-def
TLBIndex-ref-def TLBPageMask-ref-def TLBProbe-ref-def TLBRandom-ref-def TLBWired-ref-def
TLBXContext-ref-def UART-RDATA-ref-def UART-RVALID-ref-def UART-WDATA-ref-def UART-WRITTEN-ref-def
InstCount-ref-def
by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*; *simp*)+

lemma *preserves-invariant-no-writes-to-inv-regs*[*preserves-invariantI*]:
$\bigwedge$*arg0 arg1 arg2*. *traces-preserve-invariant* (*MIPS-write arg0 arg1 arg2*)

$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (MIPS\text{-}read\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}CauseReg\text{-}BD\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}CauseReg\text{-}CE\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}CauseReg\text{-}IV\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}CauseReg\text{-}IP\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}CauseReg\text{-}ExcCode\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}bits\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}CapS\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}CapL\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}PFN\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}C\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}D\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}V\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}G\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryHiReg\text{-}R\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryHiReg\text{-}VPN2\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryHiReg\text{-}ASID\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}ContextReg\text{-}PTEBase\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}ContextReg\text{-}BadVPN2\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}XContextReg\text{-}XPTEBase\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}XContextReg\text{-}XR\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}XContextReg\text{-}XBadVPN2\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}pagemask\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}r\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}vpn2\ arg0\ arg1)$

$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}asid\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}g\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}valid\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}caps1\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}capl1\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}pfn1\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}c1\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}d1\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}v1\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}caps0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}capl0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}pfn0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}c0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}d0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}v0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}CU\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}BEV\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}IM\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}KX\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}SX\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}UX\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}KSU\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}ERL\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}EXL\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}IE$

$arg0\ arg1)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (execute\text{-}branch\text{-}mips\ arg0)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (rGPR\ arg0)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (wGPR\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (MEMr\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (MEMr\text{-}reserve\ arg0\ arg1)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (MEM\text{-}sync\ arg0)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (MEMea\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (MEMea\text{-}conditional\ arg0\ arg1)$
$\bigwedge arg0\ arg1\ arg2.\ traces\text{-}preserve\text{-}invariant\ (MEMval\ arg0\ arg1\ arg2)$
$\bigwedge arg0\ arg1\ arg2.\ traces\text{-}preserve\text{-}invariant\ (MEMval\text{-}conditional\ arg0\ arg1\ arg2)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (exceptionVectorOffset\ arg0)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (exceptionVectorBase\ arg0)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (updateBadInstr\ arg0)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (set\text{-}next\text{-}pcc\ arg0)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (SignalException\ arg0)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (SignalExceptionBadAddr\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (SignalExceptionTLB\ arg0\ arg1)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (getAccessLevel\ arg0)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (pcc\text{-}access\text{-}system\text{-}regs\ arg0)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}CapCauseReg\text{-}ExcCode\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin inv\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}CapCauseReg\text{-}RegNum\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (raise\text{-}c2\text{-}exception8\ arg0\ arg1)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (raise\text{-}c2\text{-}exception\text{-}noreg\ arg0)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (checkCP0AccessHook\ arg0)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (checkCP0Access\ arg0)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (incrementCP0Count\ arg0)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (MEMr\text{-}wrapper\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (MEMr\text{-}reserve\text{-}wrapper\ arg0\ arg1)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (tlbSearch\ arg0)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (TLBTranslate2\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (TLBTranslateC\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (TLBTranslate\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (capToString\ arg0\ arg1)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (execute\text{-}branch\text{-}pcc\ arg0)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (ERETHook\ arg0)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (raise\text{-}c2\text{-}exception\ arg0\ arg1)$
$\bigwedge arg0\ arg1\ arg2.\ traces\text{-}preserve\text{-}invariant\ (MEMr\text{-}tagged\ arg0\ arg1\ arg2)$
$\bigwedge arg0\ arg1\ arg2.\ traces\text{-}preserve\text{-}invariant\ (MEMr\text{-}tagged\text{-}reserve\ arg0\ arg1\ arg2)$
$\bigwedge arg0\ arg1\ arg2\ arg3.\ traces\text{-}preserve\text{-}invariant\ (MEMw\text{-}tagged\ arg0\ arg1\ arg2\ arg3)$
$\bigwedge arg0\ arg1\ arg2\ arg3.\ traces\text{-}preserve\text{-}invariant\ (MEMw\text{-}tagged\text{-}conditional\ arg0\ arg1\ arg2\ arg3)$
$\bigwedge arg0\ arg1\ arg2.\ traces\text{-}preserve\text{-}invariant\ (MEMw\text{-}wrapper\ arg0\ arg1\ arg2)$
$\bigwedge arg0\ arg1\ arg2.\ traces\text{-}preserve\text{-}invariant\ (MEMw\text{-}conditional\text{-}wrapper\ arg0\ arg1\ arg2)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (checkDDCPerms\ arg0\ arg1)$

$\bigwedge$*arg0 arg1 arg2. traces-preserve-invariant (addrWrapper arg0 arg1 arg2)*
$\bigwedge$*arg0 arg1 arg2. traces-preserve-invariant (addrWrapperUnaligned arg0 arg1 arg2)*
$\bigwedge$*arg0. traces-preserve-invariant (execute-branch arg0)*
$\bigwedge$*arg0. traces-preserve-invariant (checkCP2usable arg0)*
$\bigwedge$*arg0. traces-preserve-invariant (get-CP0EPC arg0)*
$\bigwedge$*arg0. traces-preserve-invariant (set-CP0EPC arg0)*
$\bigwedge$*arg0. traces-preserve-invariant (get-CP0ErrorEPC arg0)*
$\bigwedge$*arg0. traces-preserve-invariant (set-CP0ErrorEPC arg0)*
by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-toI*; *simp*)+

lemma *preserves-invariant-write-reg*[*preserves-invariantI*]:
assumes *name r* $\notin$ *inv-regs*
shows *traces-preserve-invariant (write-reg r v)*
using *assms*
by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-toI*)

lemma *preserves-invariant-TLBWriteEntry*[*preserves-invariantI*]:
traces-preserve-invariant (TLBWriteEntry idx)
unfolding *TLBWriteEntry-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-option*[*preserves-invariantI*]:
runs-preserve-invariant (undefined-option arg0)
unfolding *undefined-option-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-exception*[*preserves-invariantI*]:
runs-preserve-invariant (undefined-exception arg0)
unfolding *undefined-exception-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-CauseReg*[*preserves-invariantI*]:
runs-preserve-invariant (undefined-CauseReg arg0)
unfolding *undefined-CauseReg-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-set-CauseReg-bits*[*preserves-invariantI*]:
assumes *name arg0* $\notin$ *inv-regs*
shows *runs-preserve-invariant (set-CauseReg-bits arg0 arg1)*
using *assms*
unfolding *set-CauseReg-bits-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-set-CauseReg-WP*[*preserves-invariantI*]:
assumes *name arg0* $\notin$ *inv-regs*
shows *runs-preserve-invariant (set-CauseReg-WP arg0 arg1)*
using *assms*
unfolding *set-CauseReg-WP-def bind-assoc*

by *preserves-invariantI*

lemma *preserves-invariant-undefined-TLBEntryLoReg*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-TLBEntryLoReg arg0*)
unfolding *undefined-TLBEntryLoReg-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-TLBEntryHiReg*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-TLBEntryHiReg arg0*)
unfolding *undefined-TLBEntryHiReg-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-set-TLBEntryHiReg-bits*[*preserves-invariantI*]:
assumes *name arg0* $\notin$ *inv-regs*
shows *runs-preserve-invariant* (*set-TLBEntryHiReg-bits arg0 arg1*)
using *assms*
unfolding *set-TLBEntryHiReg-bits-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-ContextReg*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-ContextReg arg0*)
unfolding *undefined-ContextReg-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-set-ContextReg-bits*[*preserves-invariantI*]:
assumes *name arg0* $\notin$ *inv-regs*
shows *runs-preserve-invariant* (*set-ContextReg-bits arg0 arg1*)
using *assms*
unfolding *set-ContextReg-bits-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-XContextReg*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-XContextReg arg0*)
unfolding *undefined-XContextReg-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-set-XContextReg-bits*[*preserves-invariantI*]:
assumes *name arg0* $\notin$ *inv-regs*
shows *runs-preserve-invariant* (*set-XContextReg-bits arg0 arg1*)
using *assms*
unfolding *set-XContextReg-bits-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-TLBEntry*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-TLBEntry arg0*)
unfolding *undefined-TLBEntry-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-set-TLBEntry-bits*[*preserves-invariantI*]:

assumes *name arg0* $\notin$ *inv-regs*
shows *runs-preserve-invariant* (*set-TLBEntry-bits arg0 arg1*)
using *assms*
unfolding *set-TLBEntry-bits-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-StatusReg*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-StatusReg arg0*)
unfolding *undefined-StatusReg-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-set-StatusReg-bits*[*preserves-invariantI*]:
assumes *name arg0* $\notin$ *inv-regs*
shows *runs-preserve-invariant* (*set-StatusReg-bits arg0 arg1*)
using *assms*
unfolding *set-StatusReg-bits-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-Exception*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-Exception arg0*)
unfolding *undefined-Exception-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-Capability*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-Capability arg0*)
unfolding *undefined-Capability-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-MemAccessType*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-MemAccessType arg0*)
unfolding *undefined-MemAccessType-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-AccessLevel*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-AccessLevel arg0*)
unfolding *undefined-AccessLevel-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-CapCauseReg*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-CapCauseReg arg0*)
unfolding *undefined-CapCauseReg-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-decode-failure*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-decode-failure arg0*)
unfolding *undefined-decode-failure-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-Comparison*[*preserves-invariantI*]:

runs-preserve-invariant (*undefined-Comparison arg0*)
unfolding *undefined-Comparison-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-WordType*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-WordType arg0*)
unfolding *undefined-WordType-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-WordTypeUnaligned*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-WordTypeUnaligned arg0*)
unfolding *undefined-WordTypeUnaligned-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-init-cp0-state*[*preserves-invariantI*]:
runs-preserve-invariant (*init-cp0-state arg0*)
unfolding *init-cp0-state-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-CPtrCmpOp*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-CPtrCmpOp arg0*)
unfolding *undefined-CPtrCmpOp-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-ClearRegSet*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-ClearRegSet arg0*)
unfolding *undefined-ClearRegSet-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-undefined-CapEx*[*preserves-invariantI*]:
runs-preserve-invariant (*undefined-CapEx arg0*)
unfolding *undefined-CapEx-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-set-CapCauseReg-bits*[*preserves-invariantI*]:
assumes *name arg0* $\notin$ *inv-regs*
shows *runs-preserve-invariant* (*set-CapCauseReg-bits arg0 arg1*)
using *assms*
unfolding *set-CapCauseReg-bits-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-TranslatePC*[*preserves-invariantI*]:
runs-preserve-invariant (*TranslatePC arg0*)
unfolding *TranslatePC-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-dump-cp2-state*[*preserves-invariantI*]:
runs-preserve-invariant (*dump-cp2-state arg0*)
unfolding *dump-cp2-state-def bind-assoc*

by *preserves-invariantI*

lemma *preserves-invariant-execute-XORI*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-XORI arg0 arg1 arg2*)
unfolding *execute-XORI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-XOR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-XOR arg0 arg1 arg2*)
unfolding *execute-XOR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-WAIT*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-WAIT arg0*)
unfolding *execute-WAIT-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-TRAPREG*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-TRAPREG arg0 arg1 arg2*)
unfolding *execute-TRAPREG-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-TRAPIMM*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-TRAPIMM arg0 arg1 arg2*)
unfolding *execute-TRAPIMM-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-TLBWR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-TLBWR arg0*)
unfolding *execute-TLBWR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-TLBWI*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-TLBWI arg0*)
unfolding *execute-TLBWI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-TLBR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-TLBR arg0*)
unfolding *execute-TLBR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-TLBP*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-TLBP arg0*)
unfolding *execute-TLBP-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-Store*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-Store arg0 arg1 arg2 arg3 arg4*)

unfolding *execute-Store-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SYSCALL*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SYSCALL arg0*)
unfolding *execute-SYSCALL-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SYNC*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SYNC arg0*)
unfolding *execute-SYNC-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SWR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SWR arg0 arg1 arg2*)
unfolding *execute-SWR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SWL*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SWL arg0 arg1 arg2*)
unfolding *execute-SWL-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SUBU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SUBU arg0 arg1 arg2*)
unfolding *execute-SUBU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SUB*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SUB arg0 arg1 arg2*)
unfolding *execute-SUB-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SRLV*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SRLV arg0 arg1 arg2*)
unfolding *execute-SRLV-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SRL*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SRL arg0 arg1 arg2*)
unfolding *execute-SRL-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SRAV*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SRAV arg0 arg1 arg2*)
unfolding *execute-SRAV-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SRA*[*preserves-invariantI*]:

runs-preserve-invariant (*execute-SRA arg0 arg1 arg2*)
unfolding *execute-SRA-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SLTU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SLTU arg0 arg1 arg2*)
unfolding *execute-SLTU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SLTIU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SLTIU arg0 arg1 arg2*)
unfolding *execute-SLTIU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SLTI*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SLTI arg0 arg1 arg2*)
unfolding *execute-SLTI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SLT*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SLT arg0 arg1 arg2*)
unfolding *execute-SLT-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SLLV*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SLLV arg0 arg1 arg2*)
unfolding *execute-SLLV-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SLL*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SLL arg0 arg1 arg2*)
unfolding *execute-SLL-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SDR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SDR arg0 arg1 arg2*)
unfolding *execute-SDR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-SDL*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-SDL arg0 arg1 arg2*)
unfolding *execute-SDL-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-RI*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-RI arg0*)
unfolding *execute-RI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-RDHWR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-RDHWR arg0 arg1*)
unfolding *execute-RDHWR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-ORI*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-ORI arg0 arg1 arg2*)
unfolding *execute-ORI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-OR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-OR arg0 arg1 arg2*)
unfolding *execute-OR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-NOR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-NOR arg0 arg1 arg2*)
unfolding *execute-NOR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MULTU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MULTU arg0 arg1*)
unfolding *execute-MULTU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MULT*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MULT arg0 arg1*)
unfolding *execute-MULT-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MUL*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MUL arg0 arg1 arg2*)
unfolding *execute-MUL-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MTLO*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MTLO arg0*)
unfolding *execute-MTLO-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MTHI*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MTHI arg0*)
unfolding *execute-MTHI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MTC0*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MTC0 arg0 arg1 arg2 arg3*)
unfolding *execute-MTC0-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MSUBU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MSUBU arg0 arg1*)
unfolding *execute-MSUBU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MSUB*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MSUB arg0 arg1*)
unfolding *execute-MSUB-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MOVZ*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MOVZ arg0 arg1 arg2*)
unfolding *execute-MOVZ-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MOVN*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MOVN arg0 arg1 arg2*)
unfolding *execute-MOVN-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MFLO*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MFLO arg0*)
unfolding *execute-MFLO-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MFHI*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MFHI arg0*)
unfolding *execute-MFHI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MFC0*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MFC0 arg0 arg1 arg2 arg3*)
unfolding *execute-MFC0-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MADDU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MADDU arg0 arg1*)
unfolding *execute-MADDU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-MADD*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-MADD arg0 arg1*)
unfolding *execute-MADD-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-Load*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-Load arg0 arg1 arg2 arg3 arg4 arg5*)
unfolding *execute-Load-def bind-assoc*

by *preserves-invariantI*

lemma *preserves-invariant-execute-LWR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-LWR arg0 arg1 arg2*)
unfolding *execute-LWR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-LWL*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-LWL arg0 arg1 arg2*)
unfolding *execute-LWL-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-LUI*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-LUI arg0 arg1*)
unfolding *execute-LUI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-LDR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-LDR arg0 arg1 arg2*)
unfolding *execute-LDR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-LDL*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-LDL arg0 arg1 arg2*)
unfolding *execute-LDL-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-JR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-JR arg0*)
unfolding *execute-JR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-JALR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-JALR arg0 arg1*)
unfolding *execute-JALR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-JAL*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-JAL arg0*)
unfolding *execute-JAL-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-J*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-J arg0*)
unfolding *execute-J-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-ERET*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-ERET arg0*)

unfolding *execute-ERET-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DSUBU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DSUBU arg0 arg1 arg2*)
unfolding *execute-DSUBU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DSUB*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DSUB arg0 arg1 arg2*)
unfolding *execute-DSUB-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DSRLV*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DSRLV arg0 arg1 arg2*)
unfolding *execute-DSRLV-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DSRL32*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DSRL32 arg0 arg1 arg2*)
unfolding *execute-DSRL32-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DSRL*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DSRL arg0 arg1 arg2*)
unfolding *execute-DSRL-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DSRAV*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DSRAV arg0 arg1 arg2*)
unfolding *execute-DSRAV-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DSRA32*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DSRA32 arg0 arg1 arg2*)
unfolding *execute-DSRA32-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DSRA*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DSRA arg0 arg1 arg2*)
unfolding *execute-DSRA-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DSLLV*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DSLLV arg0 arg1 arg2*)
unfolding *execute-DSLLV-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DSLL32*[*preserves-invariantI*]:

runs-preserve-invariant (*execute-DSLL32 arg0 arg1 arg2*)
unfolding *execute-DSLL32-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DSLL*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DSLL arg0 arg1 arg2*)
unfolding *execute-DSLL-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DMULTU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DMULTU arg0 arg1*)
unfolding *execute-DMULTU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DMULT*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DMULT arg0 arg1*)
unfolding *execute-DMULT-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DIVU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DIVU arg0 arg1*)
unfolding *execute-DIVU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DIV*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DIV arg0 arg1*)
unfolding *execute-DIV-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DDIVU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DDIVU arg0 arg1*)
unfolding *execute-DDIVU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DDIV*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DDIV arg0 arg1*)
unfolding *execute-DDIV-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DADDU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DADDU arg0 arg1 arg2*)
unfolding *execute-DADDU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DADDIU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DADDIU arg0 arg1 arg2*)
unfolding *execute-DADDIU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DADDI*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DADDI arg0 arg1 arg2*)
unfolding *execute-DADDI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-DADD*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-DADD arg0 arg1 arg2*)
unfolding *execute-DADD-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-ClearRegs*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-ClearRegs arg0 arg1*)
unfolding *execute-ClearRegs-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CWriteHwr*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CWriteHwr arg0 arg1*)
unfolding *execute-CWriteHwr-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CUnseal*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CUnseal arg0 arg1 arg2*)
unfolding *execute-CUnseal-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CToPtr*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CToPtr arg0 arg1 arg2*)
unfolding *execute-CToPtr-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CTestSubset*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CTestSubset arg0 arg1 arg2*)
unfolding *execute-CTestSubset-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CSub*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSub arg0 arg1 arg2*)
unfolding *execute-CSub-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CStoreConditional*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CStoreConditional arg0 arg1 arg2 arg3*)
unfolding *execute-CStoreConditional-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CStore*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CStore arg0 arg1 arg2 arg3 arg4*)
unfolding *execute-CStore-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CSetOffset*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSetOffset arg0 arg1 arg2*)
unfolding *execute-CSetOffset-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CSetFlags*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSetFlags arg0 arg1 arg2*)
unfolding *execute-CSetFlags-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CSetCause*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSetCause arg0*)
unfolding *execute-CSetCause-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CSetCID*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSetCID arg0*)
unfolding *execute-CSetCID-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CSetBoundsImmediate*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSetBoundsImmediate arg0 arg1 arg2*)
unfolding *execute-CSetBoundsImmediate-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CSetBoundsExact*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSetBoundsExact arg0 arg1 arg2*)
unfolding *execute-CSetBoundsExact-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CSetBounds*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSetBounds arg0 arg1 arg2*)
unfolding *execute-CSetBounds-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CSetAddr*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSetAddr arg0 arg1 arg2*)
unfolding *execute-CSetAddr-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CSeal*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSeal arg0 arg1 arg2*)
unfolding *execute-CSeal-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CSCC*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSCC arg0 arg1 arg2*)
unfolding *execute-CSCC-def bind-assoc*

by *preserves-invariantI*

lemma *preserves-invariant-execute-CSC*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CSC arg0 arg1 arg2 arg3*)
unfolding *execute-CSC-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CReturn*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CReturn arg0*)
unfolding *execute-CReturn-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CReadHwr*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CReadHwr arg0 arg1*)
unfolding *execute-CReadHwr-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CRAP*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CRAP arg0 arg1*)
unfolding *execute-CRAP-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CRAM*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CRAM arg0 arg1*)
unfolding *execute-CRAM-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CPtrCmp*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CPtrCmp arg0 arg1 arg2 arg3*)
unfolding *execute-CPtrCmp-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CMove*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CMove arg0 arg1*)
unfolding *execute-CMove-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CMOVX*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CMOVX arg0 arg1 arg2 arg3*)
unfolding *execute-CMOVX-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CLoadTags*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CLoadTags arg0 arg1*)
unfolding *execute-CLoadTags-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CLoadLinked*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CLoadLinked arg0 arg1 arg2 arg3*)

unfolding *execute-CLoadLinked-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CLoad*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CLoad arg0 arg1 arg2 arg3 arg4 arg5*)
unfolding *execute-CLoad-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CLLC*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CLLC arg0 arg1*)
unfolding *execute-CLLC-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CLCBI*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CLCBI arg0 arg1 arg2*)
unfolding *execute-CLCBI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CLC*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CLC arg0 arg1 arg2 arg3*)
unfolding *execute-CLC-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CJALR*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CJALR arg0 arg1 arg2*)
unfolding *execute-CJALR-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CIncOffsetImmediate*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CIncOffsetImmediate arg0 arg1 arg2*)
unfolding *execute-CIncOffsetImmediate-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CIncOffset*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CIncOffset arg0 arg1 arg2*)
unfolding *execute-CIncOffset-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetType*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetType arg0 arg1*)
unfolding *execute-CGetType-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetTag*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetTag arg0 arg1*)
unfolding *execute-CGetTag-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetSealed*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetSealed arg0 arg1*)
unfolding *execute-CGetSealed-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetPerm*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetPerm arg0 arg1*)
unfolding *execute-CGetPerm-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetPCCSetOffset*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetPCCSetOffset arg0 arg1*)
unfolding *execute-CGetPCCSetOffset-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetPCC*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetPCC arg0*)
unfolding *execute-CGetPCC-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetOffset*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetOffset arg0 arg1*)
unfolding *execute-CGetOffset-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetLen*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetLen arg0 arg1*)
unfolding *execute-CGetLen-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetFlags*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetFlags arg0 arg1*)
unfolding *execute-CGetFlags-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetCause*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetCause arg0*)
unfolding *execute-CGetCause-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetCID*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetCID arg0*)
unfolding *execute-CGetCID-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetBase*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetBase arg0 arg1*)
unfolding *execute-CGetBase-def bind-assoc*

by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetAndAddr*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetAndAddr arg0 arg1 arg2*)
unfolding *execute-CGetAndAddr-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CGetAddr*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CGetAddr arg0 arg1*)
unfolding *execute-CGetAddr-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CFromPtr*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CFromPtr arg0 arg1 arg2*)
unfolding *execute-CFromPtr-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CCopyType*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CCopyType arg0 arg1 arg2*)
unfolding *execute-CCopyType-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CClearTag*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CClearTag arg0 arg1*)
unfolding *execute-CClearTag-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CCheckType*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CCheckType arg0 arg1*)
unfolding *execute-CCheckType-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CCheckTag*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CCheckTag arg0*)
unfolding *execute-CCheckTag-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CCheckPerm*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CCheckPerm arg0 arg1*)
unfolding *execute-CCheckPerm-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CCall*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CCall arg0 arg1 arg2*)
unfolding *execute-CCall-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CCSeal*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CCSeal arg0 arg1 arg2*)

unfolding *execute-CCSeal-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CBuildCap*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CBuildCap arg0 arg1 arg2*)
unfolding *execute-CBuildCap-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CBZ*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CBZ arg0 arg1 arg2*)
unfolding *execute-CBZ-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CBX*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CBX arg0 arg1 arg2*)
unfolding *execute-CBX-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CAndPerm*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CAndPerm arg0 arg1 arg2*)
unfolding *execute-CAndPerm-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CAndAddr*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CAndAddr arg0 arg1 arg2*)
unfolding *execute-CAndAddr-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-CACHE*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-CACHE arg0 arg1 arg2*)
unfolding *execute-CACHE-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-BREAK*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-BREAK arg0*)
unfolding *execute-BREAK-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-BEQ*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-BEQ arg0 arg1 arg2 arg3 arg4*)
unfolding *execute-BEQ-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-BCMPZ*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-BCMPZ arg0 arg1 arg2 arg3 arg4*)
unfolding *execute-BCMPZ-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-ANDI*[*preserves-invariantI*]:

runs-preserve-invariant (*execute-ANDI arg0 arg1 arg2*)
unfolding *execute-ANDI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-AND*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-AND arg0 arg1 arg2*)
unfolding *execute-AND-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-ADDU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-ADDU arg0 arg1 arg2*)
unfolding *execute-ADDU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-ADDIU*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-ADDIU arg0 arg1 arg2*)
unfolding *execute-ADDIU-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-ADDI*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-ADDI arg0 arg1 arg2*)
unfolding *execute-ADDI-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute-ADD*[*preserves-invariantI*]:
runs-preserve-invariant (*execute-ADD arg0 arg1 arg2*)
unfolding *execute-ADD-def bind-assoc*
by *preserves-invariantI*

lemma *preserves-invariant-execute*[*preserves-invariantI*]:
runs-preserve-invariant (*execute instr*)
by (*cases instr rule*: *execute.cases*; *simp*; *preserves-invariantI*)

lemma *traces-enabled-write-cap-regs*[*traces-enabledI*]:
assumes $c \in$ *derivable-caps s*
shows *traces-enabled* (*write-reg C01-ref c*) *s regs*
and *traces-enabled* (*write-reg C02-ref c*) *s regs*
and *traces-enabled* (*write-reg C03-ref c*) *s regs*
and *traces-enabled* (*write-reg C04-ref c*) *s regs*
and *traces-enabled* (*write-reg C05-ref c*) *s regs*
and *traces-enabled* (*write-reg C06-ref c*) *s regs*
and *traces-enabled* (*write-reg C07-ref c*) *s regs*
and *traces-enabled* (*write-reg C08-ref c*) *s regs*
and *traces-enabled* (*write-reg C09-ref c*) *s regs*
and *traces-enabled* (*write-reg C10-ref c*) *s regs*
and *traces-enabled* (*write-reg C11-ref c*) *s regs*
and *traces-enabled* (*write-reg C12-ref c*) *s regs*
and *traces-enabled* (*write-reg C13-ref c*) *s regs*
and *traces-enabled* (*write-reg C14-ref c*) *s regs*

and *traces-enabled* (*write-reg C15-ref c*) *s regs*
and *traces-enabled* (*write-reg C16-ref c*) *s regs*
and *traces-enabled* (*write-reg C17-ref c*) *s regs*
and *traces-enabled* (*write-reg C18-ref c*) *s regs*
and *traces-enabled* (*write-reg C19-ref c*) *s regs*
and *traces-enabled* (*write-reg C20-ref c*) *s regs*
and *traces-enabled* (*write-reg C21-ref c*) *s regs*
and *traces-enabled* (*write-reg C22-ref c*) *s regs*
and *traces-enabled* (*write-reg C23-ref c*) *s regs*
and *traces-enabled* (*write-reg C24-ref c*) *s regs*
and *traces-enabled* (*write-reg C25-ref c*) *s regs*
and *traces-enabled* (*write-reg C26-ref c*) *s regs*
and *traces-enabled* (*write-reg C27-ref c*) *s regs*
and *traces-enabled* (*write-reg C28-ref c*) *s regs*
and *traces-enabled* (*write-reg C29-ref c*) *s regs*
and *traces-enabled* (*write-reg C30-ref c*) *s regs*
and *traces-enabled* (*write-reg C31-ref c*) *s regs*
and *traces-enabled* (*write-reg CPLR-ref c*) *s regs*
and *traces-enabled* (*write-reg CULR-ref c*) *s regs*
and *traces-enabled* (*write-reg DDC-ref c*) *s regs*
and *traces-enabled* (*write-reg DelayedPCC-ref c*) *s regs*
and *traces-enabled* (*write-reg EPCC-ref c*) *s regs*
and *traces-enabled* (*write-reg ErrorEPCC-ref c*) *s regs*
and *traces-enabled* (*write-reg KCC-ref c*) *s regs*
and *traces-enabled* (*write-reg KDC-ref c*) *s regs*
and *traces-enabled* (*write-reg KR1C-ref c*) *s regs*
and *traces-enabled* (*write-reg KR2C-ref c*) *s regs*
and *traces-enabled* (*write-reg NextPCC-ref c*) *s regs*
and *traces-enabled* (*write-reg PCC-ref c*) *s regs*
using *assms*
by (*intro traces-enabled-write-reg*; *auto simp*: *register-defs derivable-caps-def*)+

lemma *traces-enabled-write-reg-CapCause*[*traces-enabledI*]:
traces-enabled (*write-reg CapCause-ref c*) *s regs*
by (*intro traces-enabled-write-reg*; *auto simp*: *register-defs derivable-caps-def*)+

lemma *traces-enabled-read-cap-regs*[*traces-enabledI*]:
traces-enabled (*read-reg C01-ref*) *s regs*
traces-enabled (*read-reg C02-ref*) *s regs*
traces-enabled (*read-reg C03-ref*) *s regs*
traces-enabled (*read-reg C04-ref*) *s regs*
traces-enabled (*read-reg C05-ref*) *s regs*
traces-enabled (*read-reg C06-ref*) *s regs*
traces-enabled (*read-reg C07-ref*) *s regs*
traces-enabled (*read-reg C08-ref*) *s regs*
traces-enabled (*read-reg C09-ref*) *s regs*
traces-enabled (*read-reg C10-ref*) *s regs*
traces-enabled (*read-reg C11-ref*) *s regs*
traces-enabled (*read-reg C12-ref*) *s regs*

traces-enabled (*read-reg C13-ref*) *s regs*
traces-enabled (*read-reg C14-ref*) *s regs*
traces-enabled (*read-reg C15-ref*) *s regs*
traces-enabled (*read-reg C16-ref*) *s regs*
traces-enabled (*read-reg C17-ref*) *s regs*
traces-enabled (*read-reg C18-ref*) *s regs*
traces-enabled (*read-reg C19-ref*) *s regs*
traces-enabled (*read-reg C20-ref*) *s regs*
traces-enabled (*read-reg C21-ref*) *s regs*
traces-enabled (*read-reg C22-ref*) *s regs*
traces-enabled (*read-reg C23-ref*) *s regs*
traces-enabled (*read-reg C24-ref*) *s regs*
traces-enabled (*read-reg C25-ref*) *s regs*
traces-enabled (*read-reg C26-ref*) *s regs*
traces-enabled (*read-reg C27-ref*) *s regs*
traces-enabled (*read-reg C28-ref*) *s regs*
traces-enabled (*read-reg C29-ref*) *s regs*
traces-enabled (*read-reg C30-ref*) *s regs*
traces-enabled (*read-reg C31-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg CPLR-ref*) *s regs*
traces-enabled (*read-reg CULR-ref*) *s regs*
traces-enabled (*read-reg DDC-ref*) *s regs*
traces-enabled (*read-reg DelayedPCC-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg EPCC-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg ErrorEPCC-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg KCC-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg KDC-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg KR1C-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg KR2C-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg CapCause-ref*) *s regs*
traces-enabled (*read-reg NextPCC-ref*) *s regs*
traces-enabled (*read-reg PCC-ref*) *s regs*
by (*intro traces-enabled-read-reg*; *auto simp*: *register-defs*)+

lemma *read-cap-regs-derivable*[*derivable-capsE*]:
⋀*t c regs s*. *Run-inv* (*read-reg C01-ref*) *t c regs* ⟹ {*''C01''*} ⊆ *accessible-regs s* ⟹ *c* ∈ *derivable-caps* (*run s t*)
⋀*t c regs s*. *Run-inv* (*read-reg C02-ref*) *t c regs* ⟹ {*''C02''*} ⊆ *accessible-regs s* ⟹ *c* ∈ *derivable-caps* (*run s t*)
⋀*t c regs s*. *Run-inv* (*read-reg C03-ref*) *t c regs* ⟹ {*''C03''*} ⊆ *accessible-regs s* ⟹ *c* ∈ *derivable-caps* (*run s t*)
⋀*t c regs s*. *Run-inv* (*read-reg C04-ref*) *t c regs* ⟹ {*''C04''*} ⊆ *accessible-regs s* ⟹ *c* ∈ *derivable-caps* (*run s t*)
⋀*t c regs s*. *Run-inv* (*read-reg C05-ref*) *t c regs* ⟹ {*''C05''*} ⊆ *accessible-regs s* ⟹ *c* ∈ *derivable-caps* (*run s t*)
⋀*t c regs s*. *Run-inv* (*read-reg C06-ref*) *t c regs* ⟹ {*''C06''*} ⊆ *accessible-regs*

$s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C07-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C07''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C08-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C08''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C09-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C09''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C10-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C10''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C11-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C11''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C12-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C12''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C13-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C13''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C14-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C14''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C15-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C15''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C16-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C16''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C17-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C17''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C18-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C18''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C19-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C19''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C20-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C20''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C21-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C21''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C22-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C22''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C23-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C23''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C24-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C24''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C25-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C25''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C26-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C26''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C27-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C27''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C28-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C28''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C29-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C29''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$
$\bigwedge t\ c\ \textit{regs}\ s.\ \textit{Run-inv}\ (\textit{read-reg}\ \textit{C30-ref})\ t\ c\ \textit{regs} \Longrightarrow \{''C30''\} \subseteq \textit{accessible-regs}\ s \Longrightarrow c \in \textit{derivable-caps}\ (\textit{run}\ s\ t)$

$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C31\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C31''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ CPLR\text{-}ref)\ t\ c\ regs \Longrightarrow \{''CPLR''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ CULR\text{-}ref)\ t\ c\ regs \Longrightarrow \{''CULR''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ DDC\text{-}ref)\ t\ c\ regs \Longrightarrow \{''DDC''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ DelayedPCC\text{-}ref)\ t\ c\ regs \Longrightarrow \{''DelayedPCC''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ EPCC\text{-}ref)\ t\ c\ regs \Longrightarrow \{''EPCC''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ ErrorEPCC\text{-}ref)\ t\ c\ regs \Longrightarrow \{''ErrorEPCC''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ KCC\text{-}ref)\ t\ c\ regs \Longrightarrow \{''KCC''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ KDC\text{-}ref)\ t\ c\ regs \Longrightarrow \{''KDC''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ KR1C\text{-}ref)\ t\ c\ regs \Longrightarrow \{''KR1C''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ KR2C\text{-}ref)\ t\ c\ regs \Longrightarrow \{''KR2C''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ NextPCC\text{-}ref)\ t\ c\ regs \Longrightarrow \{''NextPCC''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ PCC\text{-}ref)\ t\ c\ regs \Longrightarrow \{''PCC''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
unfolding *C01-ref-def C02-ref-def C03-ref-def C04-ref-def C05-ref-def C06-ref-def C07-ref-def C08-ref-def C09-ref-def C10-ref-def C11-ref-def C12-ref-def C13-ref-def C14-ref-def C15-ref-def C16-ref-def C17-ref-def C18-ref-def C19-ref-def C20-ref-def C21-ref-def C22-ref-def C23-ref-def C24-ref-def C25-ref-def C26-ref-def C27-ref-def C28-ref-def C29-ref-def C30-ref-def C31-ref-def CPLR-ref-def CULR-ref-def DDC-ref-def DelayedPCC-ref-def EPCC-ref-def ErrorEPCC-ref-def KCC-ref-def KDC-ref-def KR1C-ref-def KR2C-ref-def NextPCC-ref-def PCC-ref-def Run-inv-def derivable-caps-def*
by (*auto elim*!: *Run-read-regE intro*!: *derivable.Copy*)

end

context *CHERI-MIPS-Reg-Automaton*
begin

lemmas *non-cap-exp-traces-enabled*[*traces-enabledI*] = *non-cap-expI*[*THEN non-cap-exp-traces-enabledI*]

lemma *traces-enabled-MIPS-write*[*traces-enabledI*]:
shows *traces-enabled* (*MIPS-write arg0 arg1 arg2*) *s regs*
unfolding *MIPS-write-def bind-assoc*

by (*traces-enabledI*)

lemma *traces-enabled-MIPS-read*[*traces-enabledI*]:
shows *traces-enabled* (*MIPS-read arg0 arg1*) *s regs*
unfolding *MIPS-read-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMr*[*traces-enabledI*]:
shows *traces-enabled* (*MEMr arg0 arg1*) *s regs*
unfolding *MEMr-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMr-reserve*[*traces-enabledI*]:
shows *traces-enabled* (*MEMr-reserve arg0 arg1*) *s regs*
unfolding *MEMr-reserve-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMea*[*traces-enabledI*]:
shows *traces-enabled* (*MEMea arg0 arg1*) *s regs*
unfolding *MEMea-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMea-conditional*[*traces-enabledI*]:
shows *traces-enabled* (*MEMea-conditional arg0 arg1*) *s regs*
unfolding *MEMea-conditional-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMval*[*traces-enabledI*]:
shows *traces-enabled* (*MEMval arg0 arg1 arg2*) *s regs*
unfolding *MEMval-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMval-conditional*[*traces-enabledI*]:
shows *traces-enabled* (*MEMval-conditional arg0 arg1 arg2*) *s regs*
unfolding *MEMval-conditional-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-set-next-pcc*[*traces-enabledI*]:
assumes *arg0* ∈ *derivable-caps s*
shows *traces-enabled* (*set-next-pcc arg0*) *s regs*
unfolding *set-next-pcc-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *Run-inv-read-reg-PCC-not-sealed*:
assumes *Run-inv* (*read-reg PCC-ref*) *t c regs*
shows *Capability-sealed c* = *False*
using *assms*
unfolding *Run-inv-def*
by (*auto elim*!: *Run-read-regE simp*: *PCC-ref-def get-regval-def regval-of-Capability-def*)

lemma *traces-enabled-SignalException*[*traces-enabledI*]:
  assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
  shows *traces-enabled* (*SignalException arg0*) *s regs*
proof *cases*
  assume *ex*: *ex-traces*
  note [*derivable-capsE*] = *read-reg-KCC-exception-targets*
  show *?thesis*
    unfolding *SignalException-def bind-assoc*
    by (*traces-enabledI assms*: *assms intro*: *traces-enabled-set-next-pcc-ex ex simp*: *Run-inv-read-reg-PCC-not-sealed*)
next
  assume ¬*ex-traces*
  then show *?thesis*
    unfolding *traces-enabled-def finished-def isException-def*
    by *auto*
qed

lemma *traces-enabled-SignalExceptionBadAddr*[*traces-enabledI*]:
  assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
  shows *traces-enabled* (*SignalExceptionBadAddr arg0 arg1*) *s regs*
  unfolding *SignalExceptionBadAddr-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-SignalExceptionTLB*[*traces-enabledI*]:
  assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
  shows *traces-enabled* (*SignalExceptionTLB arg0 arg1*) *s regs*
  unfolding *SignalExceptionTLB-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-pcc-access-system-regs*[*traces-enabledI*]:
  shows *traces-enabled* (*pcc-access-system-regs arg0*) *s regs*
  unfolding *pcc-access-system-regs-def bind-assoc*
  by (*traces-enabledI*)

lemma *Run-raise-c2-exception8-False*[*simp*]: *Run* (*raise-c2-exception8 arg0 arg1*) *t a* $\longleftrightarrow$ *False*
  unfolding *raise-c2-exception8-def*
  by (*auto elim*!: *Run-bindE*)

lemma *traces-enabled-raise-c2-exception8*[*traces-enabledI*]:
  assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
  shows *traces-enabled* (*raise-c2-exception8 arg0 arg1*) *s regs*
proof *cases*
  assume *ex*: *ex-traces*
  have *set-ExcCode*: *traces-enabled* (*set-CapCauseReg-ExcCode CapCause-ref exc*) *s regs* **for** *exc s regs*
    unfolding *set-CapCauseReg-ExcCode-def*
    by (*traces-enabledI intro*: *ex*)

have *set-RegNum*: *traces-enabled* (*set-CapCauseReg-RegNum CapCause-ref r*) *s regs* **for** *r s regs*
unfolding *set-CapCauseReg-RegNum-def*
by (*traces-enabledI intro*: *ex*)
show *?thesis*
unfolding *raise-c2-exception8-def bind-assoc*
by (*traces-enabledI intro*: *set-ExcCode set-RegNum assms*: *assms simp*: *CapCause-ref-def*)
next
assume $\neg$*ex-traces*
then show *?thesis*
unfolding *traces-enabled-def finished-def isException-def*
by *auto*
qed

lemma *traces-enabled-raise-c2-exception-noreg*[*traces-enabledI*]:
assumes {''*PCC*''} ⊆ *accessible-regs s*
shows *traces-enabled* (*raise-c2-exception-noreg arg0*) *s regs*
unfolding *raise-c2-exception-noreg-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-checkCP0AccessHook*[*traces-enabledI*]:
assumes {''*PCC*''} ⊆ *accessible-regs s*
shows *traces-enabled* (*checkCP0AccessHook arg0*) *s regs*
unfolding *checkCP0AccessHook-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-checkCP0Access*[*traces-enabledI*]:
assumes {''*PCC*''} ⊆ *accessible-regs s*
shows *traces-enabled* (*checkCP0Access arg0*) *s regs*
unfolding *checkCP0Access-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-incrementCP0Count*[*traces-enabledI*]:
assumes {''*PCC*''} ⊆ *accessible-regs s*
shows *traces-enabled* (*incrementCP0Count arg0*) *s regs*
unfolding *incrementCP0Count-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMr-wrapper*[*traces-enabledI*]:
shows *traces-enabled* (*MEMr-wrapper arg0 arg1*) *s regs*
unfolding *MEMr-wrapper-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMr-reserve-wrapper*[*traces-enabledI*]:
shows *traces-enabled* (*MEMr-reserve-wrapper arg0 arg1*) *s regs*
unfolding *MEMr-reserve-wrapper-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-TLBTranslate2*[*traces-enabledI*]:

assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*TLBTranslate2 arg0 arg1*) *s regs*
unfolding *TLBTranslate2-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-TLBTranslateC*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*TLBTranslateC arg0 arg1*) *s regs*
unfolding *TLBTranslateC-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-TLBTranslate*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*TLBTranslate arg0 arg1*) *s regs*
unfolding *TLBTranslate-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-branch-pcc*[*traces-enabledI*]:
assumes *arg0* $\in$ *derivable-caps s*
shows *traces-enabled* (*execute-branch-pcc arg0*) *s regs*
unfolding *execute-branch-pcc-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-ERETHook*[*traces-enabledI*]:
assumes $\{''EPCC'', ''ErrorEPCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*ERETHook arg0*) *s regs*
unfolding *ERETHook-def bind-assoc*
by (*traces-enabledI assms*: *assms simp*: *accessible-regs-def*)

lemma *traces-enabled-raise-c2-exception*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*raise-c2-exception arg0 arg1*) *s regs*
unfolding *raise-c2-exception-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMr-tagged*[*traces-enabledI*]:
shows *traces-enabled* (*MEMr-tagged arg0 arg1 arg2*) *s regs*
unfolding *MEMr-tagged-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMr-tagged-reserve*[*traces-enabledI*]:
shows *traces-enabled* (*MEMr-tagged-reserve arg0 arg1 arg2*) *s regs*
unfolding *MEMr-tagged-reserve-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMw-tagged*[*traces-enabledI*]:
assumes *memBitsToCapability tag* (*ucast v*) $\in$ *derivable-caps s*
shows *traces-enabled* (*MEMw-tagged addr sz tag v*) *s regs*
unfolding *MEMw-tagged-def bind-assoc*

by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMw-tagged-conditional*[*traces-enabledI*]:
assumes *memBitsToCapability tag* (*ucast v*) $\in$ *derivable-caps s*
shows *traces-enabled* (*MEMw-tagged-conditional addr sz tag v*) *s regs*
unfolding *MEMw-tagged-conditional-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMw-wrapper*[*traces-enabledI*]:
shows *traces-enabled* (*MEMw-wrapper arg0 arg1 arg2*) *s regs*
unfolding *MEMw-wrapper-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMw-conditional-wrapper*[*traces-enabledI*]:
shows *traces-enabled* (*MEMw-conditional-wrapper arg0 arg1 arg2*) *s regs*
unfolding *MEMw-conditional-wrapper-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-checkDDCPerms*[*traces-enabledI*]:
assumes {''*PCC*''} $\subseteq$ *accessible-regs s* **and** *arg0* $\in$ *derivable-caps s*
shows *traces-enabled* (*checkDDCPerms arg0 arg1*) *s regs*
unfolding *checkDDCPerms-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-addrWrapper*[*traces-enabledI*]:
assumes {''*DDC*'', ''*PCC*''} $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*addrWrapper arg0 arg1 arg2*) *s regs*
unfolding *addrWrapper-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-addrWrapperUnaligned*[*traces-enabledI*]:
assumes {''*DDC*'', ''*PCC*''} $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*addrWrapperUnaligned arg0 arg1 arg2*) *s regs*
unfolding *addrWrapperUnaligned-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-branch*[*traces-enabledI*]:
assumes {''*PCC*''} $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-branch arg0*) *s regs*
unfolding *execute-branch-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-TranslatePC*[*traces-enabledI*]:
assumes {''*PCC*''} $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*TranslatePC arg0*) *s regs*
unfolding *TranslatePC-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-checkCP2usable*[*traces-enabledI*]:

assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*checkCP2usable arg0*) *s regs*
unfolding *checkCP2usable-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-get-CP0EPC*[*traces-enabledI*]:
assumes $\{''EPCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*get-CP0EPC arg0*) *s regs*
unfolding *get-CP0EPC-def bind-assoc*
by (*traces-enabledI assms*: *assms simp*: *accessible-regs-def*)

lemma *traces-enabled-set-CP0EPC*[*traces-enabledI*]:
assumes $\{''EPCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*set-CP0EPC arg0*) *s regs*
unfolding *set-CP0EPC-def bind-assoc*
by (*traces-enabledI assms*: *assms simp*: *accessible-regs-def*)

lemma *traces-enabled-get-CP0ErrorEPC*[*traces-enabledI*]:
assumes $\{''ErrorEPCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*get-CP0ErrorEPC arg0*) *s regs*
unfolding *get-CP0ErrorEPC-def bind-assoc*
by (*traces-enabledI assms*: *assms simp*: *accessible-regs-def*)

lemma *traces-enabled-set-CP0ErrorEPC*[*traces-enabledI*]:
assumes $\{''ErrorEPCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*set-CP0ErrorEPC arg0*) *s regs*
unfolding *set-CP0ErrorEPC-def bind-assoc*
by (*traces-enabledI assms*: *assms simp*: *accessible-regs-def system-reg-access-run*)

lemma *traces-enabled-TLBWriteEntry*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*TLBWriteEntry arg0*) *s regs*
unfolding *TLBWriteEntry-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-WAIT*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-WAIT arg0*) *s regs*
unfolding *execute-WAIT-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-TRAPREG*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-TRAPREG arg0 arg1 arg2*) *s regs*
unfolding *execute-TRAPREG-def bind-assoc*

by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-TRAPIMM*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-TRAPIMM arg0 arg1 arg2*) *s regs*
unfolding *execute-TRAPIMM-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-TLBWR*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-TLBWR arg0*) *s regs*
unfolding *execute-TLBWR-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-TLBWI*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-TLBWI arg0*) *s regs*
unfolding *execute-TLBWI-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-TLBR*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-TLBR arg0*) *s regs*
unfolding *execute-TLBR-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-TLBP*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-TLBP arg0*) *s regs*
unfolding *execute-TLBP-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-Store*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-Store arg0 arg1 arg2 arg3 arg4*) *s regs*
unfolding *execute-Store-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-SYSCALL*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-SYSCALL arg0*) *s regs*
unfolding *execute-SYSCALL-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-SWR*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-SWR arg0 arg1 arg2*) *s regs*
unfolding *execute-SWR-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-SWL*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-SWL arg0 arg1 arg2*) *s regs*
unfolding *execute-SWL-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-SUB*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-SUB arg0 arg1 arg2*) *s regs*
unfolding *execute-SUB-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-SDR*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-SDR arg0 arg1 arg2*) *s regs*
unfolding *execute-SDR-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-SDL*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-SDL arg0 arg1 arg2*) *s regs*
unfolding *execute-SDL-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-RI*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-RI arg0*) *s regs*
unfolding *execute-RI-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-RDHWR*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-RDHWR arg0 arg1*) *s regs*
unfolding *execute-RDHWR-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *Run-inv-pcc-access-system-regs-accessible-regs*:
assumes *Run-inv* (*pcc-access-system-regs u*) *t a regs* **and** *a*
and *Rs* ∩ *written-regs s* = {} **and** {*''PCC''*} ⊆ *accessible-regs s*
shows *Rs* ⊆ *accessible-regs* (*run s t*)
using *assms*
by (*auto simp*: *accessible-regs-def system-reg-access-run pcc-access-system-regs-allows-system-reg-access runs-no-reg-writes-written-regs-eq no-reg-writes-runs-no-reg-writes*)

lemmas *Run-inv-pcc-access-system-regs-privileged-regs-accessible*[*accessible-regsE*]
=
Run-inv-pcc-access-system-regs-accessible-regs[**where** *Rs* = {*''KCC''*}]
Run-inv-pcc-access-system-regs-accessible-regs[**where** *Rs* = {*''KDC''*}]

Run-inv-pcc-access-system-regs-accessible-regs[**where** $Rs = \{''EPCC''\}$]
Run-inv-pcc-access-system-regs-accessible-regs[**where** $Rs = \{''ErrorEPCC''\}$]
Run-inv-pcc-access-system-regs-accessible-regs[**where** $Rs = \{''KR1C''\}$]
Run-inv-pcc-access-system-regs-accessible-regs[**where** $Rs = \{''KR2C''\}$]
Run-inv-pcc-access-system-regs-accessible-regs[**where** $Rs = \{''CapCause''\}$]
Run-inv-pcc-access-system-regs-accessible-regs[**where** $Rs = \{''CPLR''\}$]

lemma *Run-inv-SignalException-False*[*simp*]: *Run-inv* (*SignalException exc*) *t a regs* $\longleftrightarrow$ *False*
unfolding *Run-inv-def*
by *auto*

lemma *Run-inv-checkCP0Access-accessible-regs*:
assumes *Run-inv* (*checkCP0Access u*) *t a regs*
and $Rs \cap$ *written-regs s* $= \{\}$ **and** $\{''PCC''\} \subseteq$ *accessible-regs s*
shows $Rs \subseteq$ *accessible-regs* (*run s t*)
using *assms*
unfolding *checkCP0Access-def checkCP0AccessHook-def bind-assoc*
by (*auto elim*!: *Run-inv-bindE intro*!: *preserves-invariantI traces-runs-preserve-invariantI split*: *if-splits simp*: *CP0Cause-ref-def*)
(*auto simp*: *accessible-regs-def runs-no-reg-writes-written-regs-eq no-reg-writes-runs-no-reg-writes system-reg-access-run pcc-access-system-regs-allows-system-reg-access*)

lemmas *Run-inv-checkCP0Access-privileged-regs-accessible*[*accessible-regsE*] =
Run-inv-checkCP0Access-accessible-regs[**where** $Rs = \{''KCC''\}$]
Run-inv-checkCP0Access-accessible-regs[**where** $Rs = \{''KDC''\}$]
Run-inv-checkCP0Access-accessible-regs[**where** $Rs = \{''EPCC''\}$]
Run-inv-checkCP0Access-accessible-regs[**where** $Rs = \{''ErrorEPCC''\}$]
Run-inv-checkCP0Access-accessible-regs[**where** $Rs = \{''KR1C''\}$]
Run-inv-checkCP0Access-accessible-regs[**where** $Rs = \{''KR2C''\}$]
Run-inv-checkCP0Access-accessible-regs[**where** $Rs = \{''CapCause''\}$]
Run-inv-checkCP0Access-accessible-regs[**where** $Rs = \{''CPLR''\}$]

lemma *Run-inv-no-reg-writes-written-regs*[*accessible-regsE*]:
assumes *Run-inv m t a regs*
and *runs-no-reg-writes-to Rs m* **and** $Rs \cap$ *written-regs s* $= \{\}$
shows $Rs \cap$ *written-regs* (*run s t*) $= \{\}$
using *assms*
by (*auto simp*: *runs-no-reg-writes-written-regs-eq runs-no-reg-writes-to-def*)

lemma *Run-inv-assert-exp-iff*[*iff*]:
Run-inv (*assert-exp c msg*) *t a regs* $\longleftrightarrow c \wedge t = [] \wedge$ *invariant regs*
unfolding *Run-inv-def*
by *auto*

lemma *throw-bind-eq*[*simp*]: (*throw e* $\gg=$ *m*) = *throw e*
by (*auto simp*: *throw-def*)

lemma *SignalException-bind-eq*[*simp*]: (*SignalException ex* $\gg=$ *m*) = *SignalEx-*

ception ex
  unfolding *SignalException-def Let-def bind-assoc throw-bind-eq* **..**

lemma *runs-no-reg-writes-to-checkCP2usable*[*runs-no-reg-writes-toI*, *simp*]:
  shows *runs-no-reg-writes-to Rs* (*checkCP2usable u*)
  unfolding *checkCP2usable-def runs-no-reg-writes-to-def*
  by (*auto elim*!: *Run-bindE Run-read-regE split*: *if-splits*)

lemma *traces-enabled-execute-MTC0*[*traces-enabledI*]:
 assumes {*''PCC''*} ⊆ *accessible-regs s* **and** {*''EPCC''*, *''ErrorEPCC''*} ∩ *written-regs s* = {}
  shows *traces-enabled* (*execute-MTC0 arg0 arg1 arg2 arg3*) *s regs*
  unfolding *execute-MTC0-def bind-assoc*
 by (*intro traces-enabled-if-ignore-cond traces-enabledI preserves-invariantI traces-runs-preserve-invariantI*; *accessible-regsI assms*: *assms*)

lemma *traces-enabled-execute-MFC0*[*traces-enabledI*]:
 assumes {*''PCC''*} ⊆ *accessible-regs s* **and** {*''EPCC''*, *''ErrorEPCC''*} ∩ *written-regs s* = {}
  shows *traces-enabled* (*execute-MFC0 arg0 arg1 arg2 arg3*) *s regs*
  unfolding *execute-MFC0-def bind-assoc*
 by (*intro traces-enabled-if-ignore-cond traces-enabledI preserves-invariantI traces-runs-preserve-invariantI conjI allI impI*; *accessible-regsI assms*: *assms*)

lemma *traces-enabled-execute-Load*[*traces-enabledI*]:
  assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
  shows *traces-enabled* (*execute-Load arg0 arg1 arg2 arg3 arg4 arg5*) *s regs*
  unfolding *execute-Load-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-LWR*[*traces-enabledI*]:
  assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
  shows *traces-enabled* (*execute-LWR arg0 arg1 arg2*) *s regs*
  unfolding *execute-LWR-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-LWL*[*traces-enabledI*]:
  assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
  shows *traces-enabled* (*execute-LWL arg0 arg1 arg2*) *s regs*
  unfolding *execute-LWL-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-LDR*[*traces-enabledI*]:
  assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
  shows *traces-enabled* (*execute-LDR arg0 arg1 arg2*) *s regs*
  unfolding *execute-LDR-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-LDL*[*traces-enabledI*]:

assumes $\{''DDC'', ''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-LDL arg0 arg1 arg2*) *s regs*
unfolding *execute-LDL-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-JR*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-JR arg0*) *s regs*
unfolding *execute-JR-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-JALR*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-JALR arg0 arg1*) *s regs*
unfolding *execute-JALR-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-JAL*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-JAL arg0*) *s regs*
unfolding *execute-JAL-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-J*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-J arg0*) *s regs*
unfolding *execute-J-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *runs-no-reg-writes-to-checkCP0Access*[*runs-no-reg-writes-toI*, *simp*]:
runs-no-reg-writes-to Rs (*checkCP0Access u*)
using *no-reg-writes-to-pcc-access-system-regs no-reg-writes-to-getAccessLevel no-reg-writes-to-read-reg*[**where**
r = *CP0Status-ref*]
unfolding *checkCP0Access-def checkCP0AccessHook-def runs-no-reg-writes-to-def no-reg-writes-to-def and-boolM-def*
by (*fastforce elim*!: *Run-bindE split*: *if-splits*)

lemma *traces-enabled-execute-ERET*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** $\{''EPCC'', ''ErrorEPCC''\} \cap$ *written-regs s* $= \{\}$
shows *traces-enabled* (*execute-ERET arg0*) *s regs*
unfolding *execute-ERET-def bind-assoc*
by (*traces-enabledI assms*: *assms checkCP0Access-system-reg-access simp*: *accessible-regs-def runs-no-reg-writes-written-regs-eq no-reg-writes-runs-no-reg-writes system-reg-access-run*)

lemma *traces-enabled-execute-DSUB*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-DSUB arg0 arg1 arg2*) *s regs*
unfolding *execute-DSUB-def bind-assoc*

by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-DADDI*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-DADDI arg0 arg1 arg2*) *s regs*
unfolding *execute-DADDI-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-DADD*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-DADD arg0 arg1 arg2*) *s regs*
unfolding *execute-DADD-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

declare *traces-enabled-foreachM-inv*[**where** $P = \lambda$- - -. *True*, *traces-enabledI*]

lemma *traces-enabled-execute-ClearRegs*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-ClearRegs arg0 arg1*) *s regs*
unfolding *execute-ClearRegs-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CWriteHwr*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CWriteHwr arg0 arg1*) *s regs*
unfolding *execute-CWriteHwr-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *unsealCap-derivable-caps*[*derivable-capsI*]:
assumes $c \in$ *derivable-caps s* **and** $c' \in$ *derivable-caps s*
and *Capability-tag c* **and** *Capability-tag c'*
and *Capability-sealed c* **and** $\neg$*Capability-sealed c'*
and *Capability-permit-unseal c'*
and *getCapCursor c'* = *uint* (*Capability-otype c*)
shows (*unsealCap c*)(|*Capability-global* := *Capability-global c* $\wedge$ *Capability-global c'*|) $\in$ *derivable-caps s*
(**is** *?unseal c c'* $\in$ *derivable-caps s*)
proof −
have *unseal CC c* (*get-global-method CC c'*) $\in$ *derivable* (*accessed-caps s*)
using *assms*
by (*intro derivable.Unseal*) (*auto simp*: *derivable-caps-def unat-def*[*symmetric*] *get-cap-perms-def*)
then have *?unseal c c'* $\in$ *derivable* (*accessed-caps s*)
by (*elim derivable.Restrict*)
(*auto simp*: *leq-cap-def unseal-def unsealCap-def getCapBase-def getCapTop-def get-cap-perms-def*)
then show *?thesis*
by (*auto simp*: *derivable-caps-def*)
qed

lemma *traces-enabled-execute-CUnseal*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CUnseal arg0 arg1 arg2*) *s regs*
unfolding *execute-CUnseal-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CToPtr*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CToPtr arg0 arg1 arg2*) *s regs*
unfolding *execute-CToPtr-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CTestSubset*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CTestSubset arg0 arg1 arg2*) *s regs*
unfolding *execute-CTestSubset-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CSub*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CSub arg0 arg1 arg2*) *s regs*
unfolding *execute-CSub-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CStoreConditional*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CStoreConditional arg0 arg1 arg2 arg3*) *s regs*
unfolding *execute-CStoreConditional-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CStore*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CStore arg0 arg1 arg2 arg3 arg4*) *s regs*
unfolding *execute-CStore-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CSetOffset*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CSetOffset arg0 arg1 arg2*) *s regs*
unfolding *execute-CSetOffset-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *setCapFlags-derivable-caps*[*derivable-capsI*]:
assumes $c \in$ *derivable-caps s*
shows *setCapFlags c f* $\in$ *derivable-caps s*
using *assms*
by (*auto simp*: *setCapFlags-def*)

lemma *traces-enabled-execute-CSetFlags*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CSetFlags arg0 arg1 arg2*) *s regs*
unfolding *execute-CSetFlags-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-set-CapCauseReg-ExcCode*:
assumes *system-reg-access s* $\vee$ *ex-traces*
shows *traces-enabled* (*set-CapCauseReg-ExcCode CapCause-ref exc*) *s regs*
unfolding *set-CapCauseReg-ExcCode-def*
by (*traces-enabledI intro*: *traces-enabled-read-reg traces-enabled-write-reg simp*: *CapCause-ref-def assms*: *assms*)

lemma *traces-enabled-set-CapCauseReg-RegNum*:
assumes *system-reg-access s* $\vee$ *ex-traces*
shows *traces-enabled* (*set-CapCauseReg-RegNum CapCause-ref exc*) *s regs*
unfolding *set-CapCauseReg-RegNum-def*
by (*traces-enabledI intro*: *traces-enabled-read-reg traces-enabled-write-reg simp*: *CapCause-ref-def assms*: *assms*)

lemma *system-reg-access-run-ex-tracesI*[*accessible-regsI*]:
assumes $\neg$*trace-allows-system-reg-access* (*accessible-regs s*) *t* $\Longrightarrow$ *system-reg-access s* $\vee$ *ex-traces*
shows *system-reg-access* (*run s t*) $\vee$ *ex-traces*
using *assms*
by (*auto simp*: *system-reg-access-run*)

lemma *pcc-access-system-regs-allows-system-reg-access-ex-tracesI*[*accessible-regsE*]:
assumes *Run-inv* (*pcc-access-system-regs u*) *t a regs* **and** *a* **and** $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *system-reg-access* (*run s t*) $\vee$ *ex-traces*
using *assms*
by (*auto simp*: *system-reg-access-run pcc-access-system-regs-allows-system-reg-access*)

lemma *traces-enabled-execute-CSetCause*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CSetCause arg0*) *s regs*
unfolding *execute-CSetCause-def bind-assoc*
by (*traces-enabledI assms*: *assms intro*: *traces-enabled-set-CapCauseReg-ExcCode traces-enabled-set-CapCauseReg-RegNum simp*: *CapCause-ref-def*)

lemma *traces-enabled-execute-CSetCID*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CSetCID arg0*) *s regs*
unfolding *execute-CSetCID-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *unat-add-nat-uint-add*: *unat a* + *unat b* = *nat* (*uint a* + *uint b*)
by (*auto simp*: *unat-def nat-add-distrib*)

lemma [*simp*]: $0 \leq i \Longrightarrow nat\ i \leq unat\ j \longleftrightarrow i \leq uint\ j$
  by (*auto simp*: *unat-def nat-le-eq-zle*)

lemma *setCapBounds-derivable-caps*[*derivable-capsE*]:
  assumes $setCapBounds\ c\ b\ t = (e,\ c')$
    and $c \in derivable\text{-}caps\ s$ **and** $\neg Capability\text{-}sealed\ c$
    and $getCapBase\ c \leq uint\ b$ **and** $uint\ b \leq uint\ t$ **and** $uint\ t \leq getCapTop\ c$
  shows $c' \in derivable\text{-}caps\ s$
proof −
  have $getCapTop\ c' \leq uint\ t$
    using *assms Divides.mod-less-eq-dividend*[**where** $a = uint\ (t - ucast\ b)$ **and** $b = 2\ \hat{}\ 64$]
    unfolding *setCapBounds-def getCapTop-def getCapBase-def*
    by (*auto simp*: *uint-and-mask uint-sub-if-size*)
  then have *leq-cap CC c′ c*
    using *assms*
    by (*auto simp*: *leq-cap-def setCapBounds-def getCapBase-def getCapTop-def nat-le-eq-zle get-cap-perms-def*)
  from *derivable.Restrict*[*OF - this*]
  show *?thesis*
    using *assms*
    by (*auto simp*: *derivable-caps-def setCapBounds-def*)
qed

lemma *to-bits-uint-ucast*[*simp*]:
  $n = int\ (LENGTH('a)) \Longrightarrow to\text{-}bits\ n\ (uint\ w) = (ucast\ w{::}'a{::}len\ word)$
  by (*auto simp*: *to-bits-def of-bl-bin-word-of-int ucast-def*)

lemma *to-bits-add*[*simp*]:
  $n = int\ (LENGTH('a)) \Longrightarrow to\text{-}bits\ n\ (a + b) = (to\text{-}bits\ n\ a + to\text{-}bits\ n\ b :: 'a{::}len\ word)$
  by (*auto simp*: *to-bits-def of-bl-bin-word-of-int wi-hom-syms*)

lemma *to-bits-64-getCapCursor*[*simp*]: $to\text{-}bits\ 64\ (getCapCursor\ c) = Capability\text{-}address\ c$
  by (*auto simp*: *getCapCursor-def*)

lemma *traces-enabled-execute-CSetBoundsImmediate*[*traces-enabledI*]:
  assumes $\{''PCC''\} \subseteq accessible\text{-}regs\ s$ **and** $CapRegs\text{-}names \subseteq accessible\text{-}regs\ s$
  shows *traces-enabled* (*execute-CSetBoundsImmediate arg0 arg1 arg2*) *s regs*
  unfolding *execute-CSetBoundsImmediate-def bind-assoc*
  by (*traces-enabledI assms*: *assms simp*: *getCapCursor-def getCapTop-def*)

lemma *traces-enabled-execute-CSetBoundsExact*[*traces-enabledI*]:
  assumes $\{''PCC''\} \subseteq accessible\text{-}regs\ s$ **and** $CapRegs\text{-}names \subseteq accessible\text{-}regs\ s$
  shows *traces-enabled* (*execute-CSetBoundsExact arg0 arg1 arg2*) *s regs*
  unfolding *execute-CSetBoundsExact-def bind-assoc*
  by (*traces-enabledI assms*: *assms simp*: *getCapCursor-def getCapTop-def*)

lemma *traces-enabled-execute-CSetBounds*[*traces-enabledI*]:
  assumes {''*PCC*''} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
  shows *traces-enabled* (*execute-CSetBounds arg0 arg1 arg2*) *s regs*
  unfolding *execute-CSetBounds-def bind-assoc*
  by (*traces-enabledI assms*: *assms simp*: *getCapCursor-def getCapTop-def*)

lemma *setCapAddr-derivable-caps*[*derivable-capsE*]:
  assumes *setCapAddr c a'* = (*success*, *c'*)
    and *c* ∈ *derivable-caps s*
    and *Capability-tag c* ⟶ ¬*Capability-sealed c*
  shows *c'* ∈ *derivable-caps s*
proof −
  have *leq-cap CC c' c* **and** *Capability-tag c'* ⟷ *Capability-tag c*
    using *assms*
   by (*auto simp*: *setCapAddr-def leq-cap-def getCapBase-def getCapTop-def get-cap-perms-def*)
  then show *?thesis*
    using *assms*
    by (*auto simp*: *derivable-caps-def elim*: *derivable.Restrict*)
qed

lemma *traces-enabled-execute-CSetAddr*[*traces-enabledI*]:
  assumes {''*PCC*''} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
  shows *traces-enabled* (*execute-CSetAddr arg0 arg1 arg2*) *s regs*
  unfolding *execute-CSetAddr-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *sealCap-derivable-caps*[*derivable-capsE*]:
  assumes *sealCap c* (*to-bits 24* (*getCapCursor c'*)) = (*success*, *c''*)
    and *c* ∈ *derivable-caps s* **and** *c'* ∈ *derivable-caps s*
    and *Capability-tag c* **and** *Capability-tag c'*
    and ¬*Capability-sealed c* **and** ¬*Capability-sealed c'*
    and *Capability-permit-seal c'*
  shows *c''* ∈ *derivable-caps s*
proof −
  have *seal CC c* (*get-cursor-method CC c'*) ∈ *derivable* (*accessed-caps s*)
    using *assms*
    by (*intro derivable.Seal*) (*auto simp*: *derivable-caps-def get-cap-perms-def*)
  moreover have *seal CC c* (*get-cursor-method CC c'*) = *c''*
    using *assms*
    by (*cases c*)
     (*auto simp*: *sealCap-def seal-def to-bits-def getCapCursor-def of-bl-bin-word-of-int*
word-of-int-nat)
  ultimately show *?thesis*
    by (*simp add*: *derivable-caps-def*)
qed

lemma *traces-enabled-execute-CSeal*[*traces-enabledI*]:
  assumes {''*PCC*''} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*

shows *traces-enabled* (*execute-CSeal arg0 arg1 arg2*) *s regs*
unfolding *execute-CSeal-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CSCC*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CSCC arg0 arg1 arg2*) *s regs*
unfolding *execute-CSCC-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CSC*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CSC arg0 arg1 arg2 arg3*) *s regs*
unfolding *execute-CSC-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CReturn*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CReturn arg0*) *s regs*
unfolding *execute-CReturn-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CReadHwr*[*traces-enabledI*]:
assumes {*''CULR''*, *''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
and *privileged-regs ISA* ∩ *written-regs s* = {}
shows *traces-enabled* (*execute-CReadHwr arg0 arg1*) *s regs*
proof −
have *uint arg1* ∈ {*0..31*}
by *auto*
then show *?thesis*
unfolding *upto-31-unfold execute-CReadHwr-def bind-assoc*
by (*elim insertE*; *simp cong*: *if-cong*; *use nothing* **in** ‹*traces-enabledI assms*: *assms*›)
qed

lemma *traces-enabled-execute-CRAP*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CRAP arg0 arg1*) *s regs*
unfolding *execute-CRAP-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CRAM*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CRAM arg0 arg1*) *s regs*
unfolding *execute-CRAM-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CPtrCmp*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*

shows *traces-enabled* (*execute-CPtrCmp arg0 arg1 arg2 arg3*) *s regs*
unfolding *execute-CPtrCmp-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CMove*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CMove arg0 arg1*) *s regs*
unfolding *execute-CMove-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CMOVX*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CMOVX arg0 arg1 arg2 arg3*) *s regs*
unfolding *execute-CMOVX-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CLoadTags*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CLoadTags arg0 arg1*) *s regs*
unfolding *execute-CLoadTags-def bind-assoc*
by (*traces-enabledI intro*: *traces-enabled-foreachM-inv*[**where** *s* = *s* **and** *P* = *λvars s' regs'*. {*''PCC''*} ⊆ *accessible-regs s'* **for** *s*] *assms*: *assms*)

lemma *traces-enabled-execute-CLoadLinked*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CLoadLinked arg0 arg1 arg2 arg3*) *s regs*
unfolding *execute-CLoadLinked-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CLoad*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CLoad arg0 arg1 arg2 arg3 arg4 arg5*) *s regs*
unfolding *execute-CLoad-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *Run-inv-read-memt-derivable-caps*[*derivable-capsE*]:
assumes *Run-inv* (*read-memt BCa BC-mword rk addr sz*) *t a regs*
and *tag* ⟶ *a* = (*mem*, *B1*)
shows *memBitsToCapability tag mem* ∈ *derivable-caps* (*run s t*)
using *assms*
unfolding *Run-inv-def read-memt-def read-memt-bytes-def maybe-fail-def bind-assoc*
by (*cases tag*)
(*auto simp*: *derivable-caps-def BC-mword-defs memBitsToCapability-def capBitsToCapability-def elim*!: *Run-bindE intro*!: *derivable.Copy elim*: *Traces-cases split*: *option.splits*)

lemma *Run-inv-maybe-fail-iff*[*simp*]:
Run-inv (*maybe-fail msg x*) *t a regs* ⟷ (*x* = *Some a* ∧ *t* = [] ∧ *invariant regs*)
by (*auto simp*: *Run-inv-def maybe-fail-def split*: *option.splits*)

lemma *Run-inv-MEMr-tagged-reserve-derivable-caps*[*derivable-capsE*]:
  assumes *Run-inv* (*MEMr-tagged-reserve addr sz allow-tag*) *t a regs*
    and *tag* $\longrightarrow$ *a* = (*True*, *mem*)
  shows *memBitsToCapability tag mem* $\in$ *derivable-caps* (*run s t*)
  using *assms*
  unfolding *MEMr-tagged-reserve-def*
  by (*auto elim*!: *Run-inv-bindE Run-inv-read-memt-derivable-caps intro*: *preserves-invariantI traces-runs-preserve-invariantI*
          split: *option.splits bitU.splits if-splits simp*: *bool-of-bitU-def*)

lemma *Run-inv-MEMr-tagged-derivable-caps*[*derivable-capsE*]:
  assumes *Run-inv* (*MEMr-tagged addr sz allow-tag*) *t a regs*
    and *tag* $\longrightarrow$ *a* = (*True*, *mem*)
  shows *memBitsToCapability tag mem* $\in$ *derivable-caps* (*run s t*)
  using *assms*
  unfolding *MEMr-tagged-def*
  by (*auto elim*!: *Run-inv-bindE Run-inv-read-memt-derivable-caps intro*: *preserves-invariantI traces-runs-preserve-invariantI*
          split: *option.splits bitU.splits if-splits simp*: *bool-of-bitU-def*)

lemma *traces-enabled-execute-CLLC*[*traces-enabledI*]:
  assumes {''*PCC*''} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
  shows *traces-enabled* (*execute-CLLC arg0 arg1*) *s regs*
  unfolding *execute-CLLC-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CLCBI*[*traces-enabledI*]:
  assumes {''*PCC*''} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
  shows *traces-enabled* (*execute-CLCBI arg0 arg1 arg2*) *s regs*
  unfolding *execute-CLCBI-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CLC*[*traces-enabledI*]:
  assumes {''*PCC*''} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
  shows *traces-enabled* (*execute-CLC arg0 arg1 arg2 arg3*) *s regs*
  unfolding *execute-CLC-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CJALR*[*traces-enabledI*]:
  assumes {''*PCC*''} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
  shows *traces-enabled* (*execute-CJALR arg0 arg1 arg2*) *s regs*
  unfolding *execute-CJALR-def bind-assoc*
  by (*traces-enabledI assms*: *assms simp*: *Run-inv-read-reg-PCC-not-sealed*)

lemma *incCapOffset-derivable-caps*[*derivable-capsE*]:
  assumes c': *incCapOffset c i* = (*success*, c')

and *c*: $c \in$ *derivable-caps s* **and** *noseal*: *Capability-tag c* $\wedge$ $i \neq 0 \longrightarrow \neg$*Capability-sealed c*
shows $c' \in$ *derivable-caps s*
proof –
have *leq-cap CC c′ c* **if** *Capability-tag c*
using *that c′*[*symmetric*] *noseal*
by (*auto simp*: *leq-cap-def incCapOffset-def getCapTop-def getCapBase-def get-cap-perms-def*)
moreover have *Capability-tag c′* $\longleftrightarrow$ *Capability-tag c*
using *c′*
by (*auto simp*: *incCapOffset-def*)
ultimately show *?thesis*
using *c*
unfolding *derivable-caps-def*
by (*auto elim*: *derivable.Restrict*)
qed

lemma *traces-enabled-execute-CIncOffsetImmediate*[*traces-enabledI*]:
assumes {*′′PCC′′*} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CIncOffsetImmediate arg0 arg1 arg2*) *s regs*
unfolding *execute-CIncOffsetImmediate-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CIncOffset*[*traces-enabledI*]:
assumes {*′′PCC′′*} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CIncOffset arg0 arg1 arg2*) *s regs*
unfolding *execute-CIncOffset-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetType*[*traces-enabledI*]:
assumes {*′′PCC′′*} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetType arg0 arg1*) *s regs*
unfolding *execute-CGetType-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetTag*[*traces-enabledI*]:
assumes {*′′PCC′′*} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetTag arg0 arg1*) *s regs*
unfolding *execute-CGetTag-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetSealed*[*traces-enabledI*]:
assumes {*′′PCC′′*} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetSealed arg0 arg1*) *s regs*
unfolding *execute-CGetSealed-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetPerm*[*traces-enabledI*]:
assumes {*′′PCC′′*} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*

shows *traces-enabled* (*execute-CGetPerm arg0 arg1*) *s regs*
unfolding *execute-CGetPerm-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetPCCSetOffset*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetPCCSetOffset arg0 arg1*) *s regs*
unfolding *execute-CGetPCCSetOffset-def bind-assoc*
by (*traces-enabledI assms*: *assms simp*: *Run-inv-read-reg-PCC-not-sealed*)

lemma *traces-enabled-execute-CGetPCC*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetPCC arg0*) *s regs*
unfolding *execute-CGetPCC-def bind-assoc*
by (*traces-enabledI assms*: *assms simp*: *Run-inv-read-reg-PCC-not-sealed*)

lemma *traces-enabled-execute-CGetOffset*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetOffset arg0 arg1*) *s regs*
unfolding *execute-CGetOffset-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetLen*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetLen arg0 arg1*) *s regs*
unfolding *execute-CGetLen-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetFlags*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetFlags arg0 arg1*) *s regs*
unfolding *execute-CGetFlags-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetCause*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetCause arg0*) *s regs*
unfolding *execute-CGetCause-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetCID*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetCID arg0*) *s regs*
unfolding *execute-CGetCID-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetBase*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetBase arg0 arg1*) *s regs*

unfolding *execute-CGetBase-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetAndAddr*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetAndAddr arg0 arg1 arg2*) *s regs*
unfolding *execute-CGetAndAddr-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CGetAddr*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CGetAddr arg0 arg1*) *s regs*
unfolding *execute-CGetAddr-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CFromPtr*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CFromPtr arg0 arg1 arg2*) *s regs*
unfolding *execute-CFromPtr-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *update-Capability-address-derivable-caps*[*derivable-capsI*]:
assumes $c \in$ *derivable-caps s* **and** $\neg$*Capability-sealed c*
shows $c(\!|$*Capability-address* $:= a|\!) \in$ *derivable-caps s*
proof −
have *leq-cap CC* ($c(\!|$*Capability-address* $:= a|\!)$) *c*
using *assms*
by (*auto simp*: *leq-cap-def getCapBase-def getCapTop-def get-cap-perms-def*)
then show *?thesis*
using *assms*
by (*auto simp*: *derivable-caps-def elim*: *derivable.Restrict*)
qed

lemma *null-cap-not-sealed*[*simp*, *intro*]: $\neg$*Capability-sealed null-cap*
by (*auto simp*: *null-cap-def*)

declare *null-cap-derivable*[*derivable-capsI*]

lemma *traces-enabled-execute-CCopyType*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CCopyType arg0 arg1 arg2*) *s regs*
unfolding *execute-CCopyType-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CClearTag*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CClearTag arg0 arg1*) *s regs*
unfolding *execute-CClearTag-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CCheckType*[*traces-enabledI*]:
  assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
  shows *traces-enabled* (*execute-CCheckType arg0 arg1*) *s regs*
  unfolding *execute-CCheckType-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CCheckTag*[*traces-enabledI*]:
  assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
  shows *traces-enabled* (*execute-CCheckTag arg0*) *s regs*
  unfolding *execute-CCheckTag-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CCheckPerm*[*traces-enabledI*]:
  assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
  shows *traces-enabled* (*execute-CCheckPerm arg0 arg1*) *s regs*
  unfolding *execute-CCheckPerm-def bind-assoc*
  by (*traces-enabledI assms*: *assms*)

lemma *set-next-pcc-invoked-caps*:
  assumes *cc* $\in$ *derivable-caps s*
    and $\exists$ *cd*. *cd* $\in$ *derivable-caps s* $\wedge$ *invokable CC cc cd* **and** *invocation-traces*
  shows *traces-enabled* (*set-next-pcc* (*unsealCap cc*)) *s regs*
proof −
  have *leq-cap CC* (*unsealCap cc*) (*unseal CC cc True*)
   by (*auto simp*: *leq-cap-def unsealCap-def unseal-def getCapBase-def getCapTop-def get-cap-perms-def*)
  moreover obtain *cd*
   where *cc*: *cc* $\in$ *derivable* (*accessed-caps s*) **and** *cd*: *cd* $\in$ *derivable* (*accessed-caps s*)
      and *Capability-tag cc* **and** *Capability-tag cd* **and** *invokable CC cc cd*
    using *assms*
    by (*auto simp*: *derivable-caps-def invokable-def get-cap-perms-def*)
   moreover have *cc* $\in$ *derivable* (*accessed-caps* (*run s t*)) **and** *cd* $\in$ *derivable* (*accessed-caps* (*run s t*)) **for** *t*
    using *cc cd derivable-mono*[*OF accessed-caps-run-mono*]
    by *auto*
  ultimately show *?thesis*
    unfolding *set-next-pcc-def*
   by (*intro traces-enabled-write-reg traces-enabledI preserves-invariantI traces-runs-preserve-invariantI*)
      (*auto simp add*: *NextPCC-ref-def DelayedPCC-ref-def derivable-caps-def intro*: ‹*invocation-traces*›)
qed

lemma *write-reg-C26-invoked-caps*:
  assumes *cd* $\in$ *derivable-caps s*
    and $\exists$ *cc*. *cc* $\in$ *derivable-caps s* $\wedge$ *invokable CC cc cd* **and** *invocation-traces*
  shows *traces-enabled* (*write-reg C26-ref* (*unsealCap cd*)) *s regs*
proof −

have *leq-cap CC* (*unsealCap cd*) (*unseal CC cd True*)
by (*auto simp*: *leq-cap-def unsealCap-def unseal-def getCapBase-def getCapTop-def get-cap-perms-def*)
moreover obtain *cc*
where *cc*: *cc* ∈ *derivable* (*accessed-caps s*) **and** *cd*: *cd* ∈ *derivable* (*accessed-caps s*)
and *Capability-tag cc* **and** *Capability-tag cd* **and** *invokable CC cc cd*
using *assms*
by (*auto simp*: *derivable-caps-def invokable-def get-cap-perms-def*)
ultimately show *?thesis*
by (*intro traces-enabled-write-reg*)
(*auto simp*: *C26-ref-def derivable-caps-def intro*: ‹*invocation-traces*›)
qed

lemma *getCapCursor-nonneg*[*simp*]: *0* ≤ *getCapCursor c*
by (*auto simp*: *getCapCursor-def*)

lemma *getCapTop-nonneg*[*simp*]: *0* ≤ *getCapTop c*
by (*auto simp*: *getCapTop-def*)

lemma *invokable-data-cap-derivable*:
assumes ¬ *Capability-permit-execute cd* **and** *Capability-permit-execute cc*
and *Capability-tag cd* **and** *Capability-tag cc*
and *Capability-sealed cd* **and** *Capability-sealed cc*
and *Capability-permit-ccall cd* **and** *Capability-permit-ccall cc*
and *Capability-otype cc* = *Capability-otype cd*
and *getCapBase cc* ≤ *getCapCursor cc*
and *getCapCursor cc* < *getCapTop cc*
and *cd* ∈ *derivable-caps s*
shows ∃ *cd*. *cd* ∈ *derivable-caps s* ∧ *invokable CC cc cd*
using *assms getCapCursor-nonneg*[*of cc*] *getCapTop-nonneg*[*of cc*]
unfolding *le-less*
by (*auto simp*: *invokable-def nat-le-eq-zle get-cap-perms-def*)

lemma *invokable-code-cap-derivable*:
assumes ¬¬ *Capability-permit-execute cc* **and** ¬*Capability-permit-execute cd*
and *Capability-tag cd* **and** *Capability-tag cc*
and *Capability-sealed cd* **and** *Capability-sealed cc*
and *Capability-permit-ccall cd* **and** *Capability-permit-ccall cc*
and *Capability-otype cc* = *Capability-otype cd*
and *getCapBase cc* ≤ *getCapCursor cc*
and *getCapCursor cc* < *getCapTop cc*
and *cc* ∈ *derivable-caps s*
shows ∃ *cc*. *cc* ∈ *derivable-caps s* ∧ *invokable CC cc cd*
using *assms getCapCursor-nonneg*[*of cc*] *getCapTop-nonneg*[*of cc*]
unfolding *le-less*
by (*auto simp*: *invokable-def nat-le-eq-zle get-cap-perms-def*)

lemma *traces-enabled-execute-CCall*[*traces-enabledI*]:

assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
and *invocation-traces*
shows *traces-enabled* (*execute-CCall arg0 arg1 arg2*) *s regs*
unfolding *execute-CCall-def bind-assoc*
by (*traces-enabledI assms*: *assms intro*: *set-next-pcc-invoked-caps write-reg-C26-invoked-caps elim*: *invokable-data-cap-derivable invokable-code-cap-derivable*)

lemma *traces-enabled-execute-CCSeal*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CCSeal arg0 arg1 arg2*) *s regs*
unfolding *execute-CCSeal-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

definition *perms-of-bits* :: *31 word* $\Rightarrow$ *perms* **where**
perms-of-bits p =
(|*permit-ccall* = *p* !! *8*,
permit-execute = *p* !! *1*,
permit-load = *p* !! *2*,
permit-load-capability = *p* !! *4*,
permit-seal = *p* !! *7*,
permit-store = *p* !! *3*,
permit-store-capability = *p* !! *5*,
permit-store-local-capability = *p* !! *6*,
permit-system-access = *p* !! *10*,
permit-unseal = *p* !! *9*|)

definition *and-perms* :: *perms* $\Rightarrow$ *perms* $\Rightarrow$ *perms* **where**
and-perms p1 p2 =
(|*permit-ccall* = *permit-ccall p1* $\wedge$ *permit-ccall p2*,
permit-execute = *permit-execute p1* $\wedge$ *permit-execute p2*,
permit-load = *permit-load p1* $\wedge$ *permit-load p2*,
permit-load-capability = *permit-load-capability p1* $\wedge$ *permit-load-capability p2*,
permit-seal = *permit-seal p1* $\wedge$ *permit-seal p2*,
permit-store = *permit-store p1* $\wedge$ *permit-store p2*,
permit-store-capability = *permit-store-capability p1* $\wedge$ *permit-store-capability p2*,
permit-store-local-capability = *permit-store-local-capability p1* $\wedge$ *permit-store-local-capability p2*,
permit-system-access = *permit-system-access p1* $\wedge$ *permit-system-access p2*,
permit-unseal = *permit-unseal p1* $\wedge$ *permit-unseal p2*|)

lemma *setCapPerms-derivable-caps*[*derivable-capsI*]:
assumes $c \in$ *derivable-caps s* **and** *Capability-tag c* $\longrightarrow$ *leq-perms* (*perms-of-bits p*) (*get-cap-perms c*) $\wedge$ $\neg$*Capability-sealed c* $\wedge$ (*p* !! *0* $\longrightarrow$ *Capability-global c*)
shows *setCapPerms c p* $\in$ *derivable-caps s*
proof −
have *leq-cap CC* (*setCapPerms c p*) *c* **and** *Capability-tag* (*setCapPerms c p*) =

Capability-tag c
using *assms*
by (*auto simp*: *setCapPerms-def leq-cap-def getCapBase-def getCapTop-def perms-of-bits-def get-cap-perms-def*)
then show *?thesis*
using *assms*
by (*auto simp*: *derivable-caps-def elim*!: *derivable.Restrict*)
qed

lemma *bool-to-bits-nth*[*simp*]: *bool-to-bits b* !! *n* $\longleftrightarrow$ *b* $\wedge$ *n* = *0*
by (*auto simp*: *bool-to-bits-def*)

lemma *perms-of-bits-getCapPerms-get-cap-perms*[*simp*]:
perms-of-bits (*getCapPerms c*) = *get-cap-perms c*
by (*auto simp*: *perms-of-bits-def getCapPerms-def getCapHardPerms-def get-cap-perms-def test-bit-cat nth-ucast*)

lemma *getCapPerms-0th-iff-global*[*simp*]:
getCapPerms c !! *0* = *Capability-global c*
by (*auto simp*: *getCapPerms-def getCapHardPerms-def test-bit-cat nth-ucast*)

lemma *perms-of-bits-AND-and-perms*[*simp*]:
perms-of-bits (*x AND y*) = *and-perms* (*perms-of-bits x*) (*perms-of-bits y*)
by (*auto simp*: *perms-of-bits-def and-perms-def word-ao-nth*)

lemma *leq-perms-and-perms*[*simp*, *intro*]:
leq-perms (*and-perms p1 p2*) *p1*
by (*auto simp*: *leq-perms-def and-perms-def*)

lemma *traces-enabled-execute-CBuildCap*[*traces-enabledI*]:
assumes {*''PCC''*} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CBuildCap arg0 arg1 arg2*) *s regs*
proof −
have [*simp*]:
Capability-global c' = *Capability-global c*
Capability-sealed c' $\longleftrightarrow$ *Capability-sealed c*
get-cap-perms c' = *get-cap-perms c*
if *setCapOffset c offset* = (*success*, *c'*) **for** *c c' offset success*
using *that*
by (*auto simp*: *setCapOffset-def get-cap-perms-def*)
have [*simp*]:
Capability-global c' = *Capability-global c*
Capability-sealed c' $\longleftrightarrow$ *Capability-sealed c*
get-cap-perms c' = *get-cap-perms c*
if *setCapBounds c t b* = (*success*, *c'*) **for** *c c' t b success*
using *that*
by (*auto simp*: *setCapBounds-def get-cap-perms-def*)

have [*simp*]: *Capability-global c* $\longrightarrow$ *Capability-global c′*
if *getCapPerms c AND getCapPerms c′ = getCapPerms c* **for** *c c′*
unfolding *getCapPerms-0th-iff-global*[*symmetric*]
by (*subst that*[*symmetric*]) (*auto simp add*: *word-ao-nth simp del*: *getCapPerms-0th-iff-global*)
have [*elim*]: *leq-perms* (*get-cap-perms c*) (*get-cap-perms c′*)
if *getCapPerms c AND getCapPerms c′ = getCapPerms c* **for** *c c′*
unfolding *perms-of-bits-getCapPerms-get-cap-perms*[*symmetric*]
by (*subst that*[*symmetric*]) (*auto simp add*: *leq-perms-def perms-of-bits-def word-ao-nth*)
show *?thesis*
unfolding *execute-CBuildCap-def bind-assoc*
by (*traces-enabledI assms*: *assms simp*: *getCapBase-def getCapTop-def*)
qed

lemma *traces-enabled-execute-CBZ*[*traces-enabledI*]:
assumes {*″PCC″*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CBZ arg0 arg1 arg2*) *s regs*
unfolding *execute-CBZ-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CBX*[*traces-enabledI*]:
assumes {*″PCC″*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CBX arg0 arg1 arg2*) *s regs*
unfolding *execute-CBX-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CAndPerm*[*traces-enabledI*]:
assumes {*″PCC″*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CAndPerm arg0 arg1 arg2*) *s regs*
unfolding *execute-CAndPerm-def bind-assoc*
by (*traces-enabledI assms*: *assms simp*: *word-ao-nth*)

lemma *traces-enabled-execute-CAndAddr*[*traces-enabledI*]:
assumes {*″PCC″*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CAndAddr arg0 arg1 arg2*) *s regs*
unfolding *execute-CAndAddr-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CACHE*[*traces-enabledI*]:
assumes {*″PCC″*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CACHE arg0 arg1 arg2*) *s regs*
unfolding *execute-CACHE-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-BREAK*[*traces-enabledI*]:
assumes {*″PCC″*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-BREAK arg0*) *s regs*
unfolding *execute-BREAK-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-BEQ*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-BEQ arg0 arg1 arg2 arg3 arg4*) *s regs*
unfolding *execute-BEQ-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-BCMPZ*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-BCMPZ arg0 arg1 arg2 arg3 arg4*) *s regs*
unfolding *execute-BCMPZ-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-ADDI*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-ADDI arg0 arg1 arg2*) *s regs*
unfolding *execute-ADDI-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-ADD*[*traces-enabledI*]:
assumes $\{''PCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-ADD arg0 arg1 arg2*) *s regs*
unfolding *execute-ADD-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-instr-sem*[*traces-enabledI*]:
assumes $\{''CULR'', ''DDC'', ''PCC''\} \subseteq$ *accessible-regs s*
and *CapRegs-names* $\subseteq$ *accessible-regs s*
and *privileged-regs ISA* $\cap$ *written-regs s* $= \{\}$
and *invokes-caps ISA instr* [] $\longrightarrow$ *invocation-traces*
shows *traces-enabled* (*instr-sem ISA instr*) *s regs*
by (*cases instr rule*: *execute.cases*; *simp*; *use nothing* **in** ‹*traces-enabledI assms*: *assms*›)

lemma *hasTrace-instr-reg-axioms*:
assumes *hasTrace t* (*instr-sem ISA instr*)
and *reads-regs-from inv-regs t regs* **and** *invariant regs*
and *hasException t* (*instr-sem ISA instr*) $\vee$ *hasFailure t* (*instr-sem ISA instr*) $\longrightarrow$ *ex-traces*
and *invokes-caps ISA instr t* $\longrightarrow$ *invocation-traces*
shows *store-cap-reg-axiom CC ISA ex-traces invocation-traces t*
and *store-cap-mem-axiom CC ISA t*
and *read-reg-axiom CC ISA ex-traces t*
using *assms*
by (*intro traces-enabled-reg-axioms*[**where** $m =$ *instr-sem ISA instr* **and** *regs* $=$ *regs*] *traces-enabled-instr-sem*; *auto*)+

lemma *preserves-invariant-write-reg-PCC*[*preserves-invariantI*]:
assumes *Capability-tag c* **and** $\neg$*Capability-sealed c*

shows *traces-preserve-invariant* (*write-reg PCC-ref c*)
using *assms*
unfolding *traces-preserve-invariant-def trace-preserves-invariant-def*
by (*auto simp*: *write-reg-def register-defs elim*: *Write-reg-TracesE*)

end

end
theory *CHERI-MIPS-Mem-Axioms*
imports *CHERI-MIPS-Gen-Lemmas*
begin

3.4 Memory access properties of instructions

context *CHERI-MIPS-Mem-Automaton*
begin

lemma *preserves-invariant-write-non-inv-regs*[*preserves-invariantI*]:
$\bigwedge v$. *traces-preserve-invariant* (*write-reg BranchPending-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg C26-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CID-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CP0BadInstr-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CP0BadInstrP-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CP0BadVAddr-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CP0Cause-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CP0Compare-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CP0ConfigK0-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CP0Count-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CP0HWREna-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CP0LLAddr-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CP0LLBit-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CP0UserLocal-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CPLR-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CULR-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CapCause-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg CurrentInstrBits-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg DDC-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg DelayedPC-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg DelayedPCC-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg EPCC-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg ErrorEPCC-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg GPR-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg HI-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg InBranchDelay-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg KCC-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg KDC-ref v*)
$\bigwedge v$. *traces-preserve-invariant* (*write-reg KR1C-ref v*)

$\bigwedge v.$ *traces-preserve-invariant* (*write-reg KR2C-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg LO-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg LastInstrBits-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg NextInBranchDelay-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg NextPC-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg NextPCC-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg PC-ref v*)

$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntryLo0-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBEntryLo1-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBIndex-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBPageMask-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBProbe-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBRandom-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg TLBWired-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg UART-RDATA-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg UART-RVALID-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg UART-WDATA-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg UART-WRITTEN-ref v*)
$\bigwedge v.$ *traces-preserve-invariant* (*write-reg InstCount-ref v*)
unfolding *BranchPending-ref-def C26-ref-def CID-ref-def CP0BadInstr-ref-def CP0BadInstrP-ref-def*
CP0BadVAddr-ref-def CP0Cause-ref-def CP0Compare-ref-def CP0ConfigK0-ref-def CP0Count-ref-def
CP0HWREna-ref-def CP0LLAddr-ref-def CP0LLBit-ref-def CP0UserLocal-ref-def CPLR-ref-def
CULR-ref-def CapCause-ref-def CurrentInstrBits-ref-def DDC-ref-def DelayedPC-ref-def
DelayedPCC-ref-def EPCC-ref-def ErrorEPCC-ref-def GPR-ref-def HI-ref-def
InBranchDelay-ref-def KCC-ref-def KDC-ref-def KR1C-ref-def KR2C-ref-def
LO-ref-def LastInstrBits-ref-def NextInBranchDelay-ref-def NextPC-ref-def NextPCC-ref-def
PC-ref-def PCC-ref-def TLBEntryLo0-ref-def TLBEntryLo1-ref-def TLBIndex-ref-def
TLBPageMask-ref-def TLBProbe-ref-def TLBRandom-ref-def TLBWired-ref-def UART-RDATA-ref-def
UART-RVALID-ref-def UART-WDATA-ref-def UART-WRITTEN-ref-def InstCount-ref-def
by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*; *simp add*: *trans-regs-def*)+

declare *MemAccessType.split*[**where** $P = \lambda m.$ *runs-preserve-invariant m*, *THEN iffD2*, *preserves-invariantI*]

lemma *preserves-invariant-no-writes-to-inv-regs*[*preserves-invariantI*]:
$\bigwedge$*arg0 arg1 arg2*. *traces-preserve-invariant* (*MIPS-write arg0 arg1 arg2*)
$\bigwedge$*arg0 arg1*. *traces-preserve-invariant* (*MIPS-read arg0 arg1*)
$\bigwedge$*arg0 arg1*. *name arg0* $\notin$ *trans-regs* $\Longrightarrow$ *traces-preserve-invariant* (*set-CauseReg-BD arg0 arg1*)
$\bigwedge$*arg0 arg1*. *name arg0* $\notin$ *trans-regs* $\Longrightarrow$ *traces-preserve-invariant* (*set-CauseReg-CE arg0 arg1*)
$\bigwedge$*arg0 arg1*. *name arg0* $\notin$ *trans-regs* $\Longrightarrow$ *traces-preserve-invariant* (*set-CauseReg-IV arg0 arg1*)

$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}CauseReg\text{-}IP\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}CauseReg\text{-}ExcCode\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}bits\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}CapS\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}CapL\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}PFN\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}C\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}D\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}V\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryLoReg\text{-}G\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryHiReg\text{-}R\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryHiReg\text{-}VPN2\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntryHiReg\text{-}ASID\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}ContextReg\text{-}PTEBase\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}ContextReg\text{-}BadVPN2\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}XContextReg\text{-}XPTEBase\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}XContextReg\text{-}XR\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}XContextReg\text{-}XBadVPN2\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}pagemask\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}r\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}vpn2\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}asid\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}g\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}valid\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}caps1$

$arg0\ arg1$)
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}capl1\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}pfn1\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}c1\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}d1\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}v1\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}caps0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}capl0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}pfn0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}c0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}d0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}TLBEntry\text{-}v0\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}CU\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}BEV\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}IM\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}KX\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}SX\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}UX\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}KSU\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}ERL\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}EXL\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ name\ arg0 \notin trans\text{-}regs \Longrightarrow traces\text{-}preserve\text{-}invariant\ (set\text{-}StatusReg\text{-}IE\ arg0\ arg1)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (execute\text{-}branch\text{-}mips\ arg0)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (rGPR\ arg0)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (wGPR\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (MEMr\ arg0\ arg1)$
$\bigwedge arg0\ arg1.\ traces\text{-}preserve\text{-}invariant\ (MEMr\text{-}reserve\ arg0\ arg1)$
$\bigwedge arg0.\ traces\text{-}preserve\text{-}invariant\ (MEM\text{-}sync\ arg0)$

$\bigwedge$*arg0 arg1*. *traces-preserve-invariant* (*MEMea arg0 arg1*)
$\bigwedge$*arg0 arg1*. *traces-preserve-invariant* (*MEMea-conditional arg0 arg1*)
$\bigwedge$*arg0 arg1 arg2*. *traces-preserve-invariant* (*MEMval arg0 arg1 arg2*)
$\bigwedge$*arg0 arg1 arg2*. *traces-preserve-invariant* (*MEMval-conditional arg0 arg1 arg2*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*exceptionVectorOffset arg0*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*exceptionVectorBase arg0*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*updateBadInstr arg0*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*set-next-pcc arg0*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*getAccessLevel arg0*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*pcc-access-system-regs arg0*)
$\bigwedge$*arg0 arg1*. *name arg0* $\notin$ *trans-regs* $\Longrightarrow$ *traces-preserve-invariant* (*set-CapCauseReg-ExcCode arg0 arg1*)
$\bigwedge$*arg0 arg1*. *name arg0* $\notin$ *trans-regs* $\Longrightarrow$ *traces-preserve-invariant* (*set-CapCauseReg-RegNum arg0 arg1*)
$\bigwedge$*arg0 arg1*. *traces-preserve-invariant* (*MEMr-wrapper arg0 arg1*)
$\bigwedge$*arg0 arg1*. *traces-preserve-invariant* (*MEMr-reserve-wrapper arg0 arg1*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*tlbSearch arg0*)
$\bigwedge$*arg0 arg1*. *traces-preserve-invariant* (*capToString arg0 arg1*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*execute-branch-pcc arg0*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*ERETHook arg0*)
$\bigwedge$*arg0 arg1 arg2*. *traces-preserve-invariant* (*MEMr-tagged arg0 arg1 arg2*)
$\bigwedge$*arg0 arg1 arg2*. *traces-preserve-invariant* (*MEMr-tagged-reserve arg0 arg1 arg2*)
$\bigwedge$*arg0 arg1 arg2 arg3*. *traces-preserve-invariant* (*MEMw-tagged arg0 arg1 arg2 arg3*)
$\bigwedge$*arg0 arg1 arg2 arg3*. *traces-preserve-invariant* (*MEMw-tagged-conditional arg0 arg1 arg2 arg3*)
$\bigwedge$*arg0 arg1 arg2*. *traces-preserve-invariant* (*MEMw-wrapper arg0 arg1 arg2*)
$\bigwedge$*arg0 arg1 arg2*. *traces-preserve-invariant* (*MEMw-conditional-wrapper arg0 arg1 arg2*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*get-CP0EPC arg0*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*set-CP0EPC arg0*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*get-CP0ErrorEPC arg0*)
$\bigwedge$*arg0*. *traces-preserve-invariant* (*set-CP0ErrorEPC arg0*)
by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-toI*; *simp add*: *trans-regs-def*)+

lemma *preserves-invariant-undefined-option*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-option arg0*)
unfolding *undefined-option-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-exception*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-exception arg0*)
unfolding *undefined-exception-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-CauseReg*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-CauseReg arg0*)
unfolding *undefined-CauseReg-def bind-assoc*

by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-TLBEntryLoReg*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-TLBEntryLoReg arg0*)
unfolding *undefined-TLBEntryLoReg-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-TLBEntryHiReg*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-TLBEntryHiReg arg0*)
unfolding *undefined-TLBEntryHiReg-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-ContextReg*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-ContextReg arg0*)
unfolding *undefined-ContextReg-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-XContextReg*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-XContextReg arg0*)
unfolding *undefined-XContextReg-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-TLBEntry*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-TLBEntry arg0*)
unfolding *undefined-TLBEntry-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-StatusReg*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-StatusReg arg0*)
unfolding *undefined-StatusReg-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-Exception*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-Exception arg0*)
unfolding *undefined-Exception-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-Capability*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-Capability arg0*)
unfolding *undefined-Capability-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-MemAccessType*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-MemAccessType arg0*)
unfolding *undefined-MemAccessType-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-AccessLevel*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-AccessLevel arg0*)
unfolding *undefined-AccessLevel-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-CapCauseReg*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-CapCauseReg arg0*)
unfolding *undefined-CapCauseReg-def bind-assoc*
by (*preserves-invariantI*)

lemma *trans-regs-non-members*[*simp*]:
name CP0Cause-ref $\notin$ *trans-regs*
name CapCause-ref $\notin$ *trans-regs*
by (*auto simp*: *trans-regs-def CP0Cause-ref-def CapCause-ref-def*)

lemma *preserves-invariant-incrementCP0Count*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*incrementCP0Count arg0*)
unfolding *incrementCP0Count-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-decode-failure*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-decode-failure arg0*)
unfolding *undefined-decode-failure-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-Comparison*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-Comparison arg0*)
unfolding *undefined-Comparison-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-WordType*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-WordType arg0*)
unfolding *undefined-WordType-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-WordTypeUnaligned*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-WordTypeUnaligned arg0*)
unfolding *undefined-WordTypeUnaligned-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-TLBTranslate2*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*TLBTranslate2 arg0 arg1*)
unfolding *TLBTranslate2-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-TLBTranslateC*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*TLBTranslateC arg0 arg1*)
unfolding *TLBTranslateC-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-TLBTranslate*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*TLBTranslate arg0 arg1*)
unfolding *TLBTranslate-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-CPtrCmpOp*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-CPtrCmpOp arg0*)
unfolding *undefined-CPtrCmpOp-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-ClearRegSet*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-ClearRegSet arg0*)
unfolding *undefined-ClearRegSet-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-undefined-CapEx*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*undefined-CapEx arg0*)
unfolding *undefined-CapEx-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-set-CapCauseReg-bits*[*preserves-invariantI*]:
assumes *name arg0* $\notin$ *trans-regs*
shows *runs-preserve-invariant* (*set-CapCauseReg-bits arg0 arg1*)
using *assms*
unfolding *set-CapCauseReg-bits-def bind-assoc*
by (*preserves-invariantI*; *intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-write-reg*)

lemma *preserves-invariant-raise-c2-exception*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*raise-c2-exception arg0 arg1*)
unfolding *raise-c2-exception-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-checkDDCPerms*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*checkDDCPerms arg0 arg1*)
unfolding *checkDDCPerms-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-addrWrapper*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*addrWrapper arg0 arg1 arg2*)
unfolding *addrWrapper-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-addrWrapperUnaligned*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*addrWrapperUnaligned arg0 arg1 arg2*)
unfolding *addrWrapperUnaligned-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-branch*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-branch arg0*)
unfolding *execute-branch-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-TranslatePC*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*TranslatePC arg0*)
unfolding *TranslatePC-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-checkCP2usable*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*checkCP2usable arg0*)
unfolding *checkCP2usable-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-dump-cp2-state*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*dump-cp2-state arg0*)
unfolding *dump-cp2-state-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-XORI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-XORI arg0 arg1 arg2*)
unfolding *execute-XORI-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-XOR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-XOR arg0 arg1 arg2*)
unfolding *execute-XOR-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-WAIT*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-WAIT arg0*)

unfolding *execute-WAIT-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-TRAPREG*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-TRAPREG arg0 arg1 arg2*)
unfolding *execute-TRAPREG-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-TRAPIMM*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-TRAPIMM arg0 arg1 arg2*)
unfolding *execute-TRAPIMM-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-bind-checkCP0Access*:
runs-preserve-invariant (*checkCP0Access u* $\gg$ *m*)
using *Run-inv-checkCP0Access-False*
unfolding *Run-inv-def runs-preserve-invariant-def trace-preserves-invariant-def*
by (*auto simp*: *regstate-simp elim*!: *Run-bindE*)

lemma *preserves-invariant-execute-TLBWR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-TLBWR arg0*)
unfolding *execute-TLBWR-def bind-assoc*
by (*intro preserves-invariant-bind-checkCP0Access*)

lemma *preserves-invariant-execute-TLBWI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-TLBWI arg0*)
unfolding *execute-TLBWI-def bind-assoc*
by (*intro preserves-invariant-bind-checkCP0Access*)

lemma *preserves-invariant-execute-TLBR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-TLBR arg0*)
unfolding *execute-TLBR-def bind-assoc*
by (*intro preserves-invariant-bind-checkCP0Access*)

lemma *preserves-invariant-execute-TLBP*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-TLBP arg0*)
unfolding *execute-TLBP-def bind-assoc*
by (*intro preserves-invariant-bind-checkCP0Access*)

lemma *preserves-invariant-execute-Store*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-Store arg0 arg1 arg2 arg3 arg4*)
unfolding *execute-Store-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SYSCALL*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SYSCALL arg0*)
unfolding *execute-SYSCALL-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SYNC*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SYNC arg0*)
unfolding *execute-SYNC-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SWR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SWR arg0 arg1 arg2*)
unfolding *execute-SWR-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SWL*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SWL arg0 arg1 arg2*)
unfolding *execute-SWL-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SUBU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SUBU arg0 arg1 arg2*)
unfolding *execute-SUBU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SUB*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SUB arg0 arg1 arg2*)
unfolding *execute-SUB-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SRLV*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SRLV arg0 arg1 arg2*)
unfolding *execute-SRLV-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SRL*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SRL arg0 arg1 arg2*)
unfolding *execute-SRL-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SRAV*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SRAV arg0 arg1 arg2*)
unfolding *execute-SRAV-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SRA*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SRA arg0 arg1 arg2*)
unfolding *execute-SRA-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SLTU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SLTU arg0 arg1 arg2*)
unfolding *execute-SLTU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SLTIU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SLTIU arg0 arg1 arg2*)
unfolding *execute-SLTIU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SLTI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SLTI arg0 arg1 arg2*)
unfolding *execute-SLTI-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SLT*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SLT arg0 arg1 arg2*)
unfolding *execute-SLT-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SLLV*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SLLV arg0 arg1 arg2*)
unfolding *execute-SLLV-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SLL*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SLL arg0 arg1 arg2*)
unfolding *execute-SLL-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SDR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SDR arg0 arg1 arg2*)
unfolding *execute-SDR-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-SDL*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-SDL arg0 arg1 arg2*)
unfolding *execute-SDL-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-RI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-RI arg0*)
unfolding *execute-RI-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-RDHWR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-RDHWR arg0 arg1*)
unfolding *execute-RDHWR-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-ORI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-ORI arg0 arg1 arg2*)
unfolding *execute-ORI-def bind-assoc*

by (*preserves-invariantI*)

lemma *preserves-invariant-execute-OR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-OR arg0 arg1 arg2*)
unfolding *execute-OR-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-NOR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-NOR arg0 arg1 arg2*)
unfolding *execute-NOR-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MULTU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MULTU arg0 arg1*)
unfolding *execute-MULTU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MULT*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MULT arg0 arg1*)
unfolding *execute-MULT-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MUL*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MUL arg0 arg1 arg2*)
unfolding *execute-MUL-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MTLO*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MTLO arg0*)
unfolding *execute-MTLO-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MTHI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MTHI arg0*)
unfolding *execute-MTHI-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MTC0*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MTC0 arg0 arg1 arg2 arg3*)
unfolding *execute-MTC0-def bind-assoc*
by (*intro preserves-invariant-bind-checkCP0Access*)

lemma *preserves-invariant-execute-MSUBU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MSUBU arg0 arg1*)
unfolding *execute-MSUBU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MSUB*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MSUB arg0 arg1*)

unfolding *execute-MSUB-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MOVZ*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MOVZ arg0 arg1 arg2*)
unfolding *execute-MOVZ-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MOVN*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MOVN arg0 arg1 arg2*)
unfolding *execute-MOVN-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MFLO*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MFLO arg0*)
unfolding *execute-MFLO-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MFHI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MFHI arg0*)
unfolding *execute-MFHI-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MFC0*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MFC0 arg0 arg1 arg2 arg3*)
unfolding *execute-MFC0-def bind-assoc*
by (*intro preserves-invariant-bind-checkCP0Access*)

lemma *preserves-invariant-execute-MADDU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MADDU arg0 arg1*)
unfolding *execute-MADDU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-MADD*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-MADD arg0 arg1*)
unfolding *execute-MADD-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-Load*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-Load arg0 arg1 arg2 arg3 arg4 arg5*)
unfolding *execute-Load-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-LWR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-LWR arg0 arg1 arg2*)
unfolding *execute-LWR-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-LWL*[*preserves-invariantI*]:

shows *runs-preserve-invariant* (*execute-LWL arg0 arg1 arg2*)
unfolding *execute-LWL-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-LUI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-LUI arg0 arg1*)
unfolding *execute-LUI-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-LDR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-LDR arg0 arg1 arg2*)
unfolding *execute-LDR-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-LDL*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-LDL arg0 arg1 arg2*)
unfolding *execute-LDL-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-JR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-JR arg0*)
unfolding *execute-JR-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-JALR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-JALR arg0 arg1*)
unfolding *execute-JALR-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-JAL*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-JAL arg0*)
unfolding *execute-JAL-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-J*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-J arg0*)
unfolding *execute-J-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-ERET*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-ERET arg0*)
unfolding *execute-ERET-def bind-assoc*
by (*intro preserves-invariant-bind-checkCP0Access*)

lemma *preserves-invariant-execute-DSUBU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DSUBU arg0 arg1 arg2*)
unfolding *execute-DSUBU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DSUB*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DSUB arg0 arg1 arg2*)
unfolding *execute-DSUB-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DSRLV*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DSRLV arg0 arg1 arg2*)
unfolding *execute-DSRLV-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DSRL32*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DSRL32 arg0 arg1 arg2*)
unfolding *execute-DSRL32-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DSRL*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DSRL arg0 arg1 arg2*)
unfolding *execute-DSRL-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DSRAV*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DSRAV arg0 arg1 arg2*)
unfolding *execute-DSRAV-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DSRA32*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DSRA32 arg0 arg1 arg2*)
unfolding *execute-DSRA32-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DSRA*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DSRA arg0 arg1 arg2*)
unfolding *execute-DSRA-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DSLLV*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DSLLV arg0 arg1 arg2*)
unfolding *execute-DSLLV-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DSLL32*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DSLL32 arg0 arg1 arg2*)
unfolding *execute-DSLL32-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DSLL*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DSLL arg0 arg1 arg2*)
unfolding *execute-DSLL-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DMULTU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DMULTU arg0 arg1*)
unfolding *execute-DMULTU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DMULT*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DMULT arg0 arg1*)
unfolding *execute-DMULT-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DIVU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DIVU arg0 arg1*)
unfolding *execute-DIVU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DIV*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DIV arg0 arg1*)
unfolding *execute-DIV-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DDIVU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DDIVU arg0 arg1*)
unfolding *execute-DDIVU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DDIV*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DDIV arg0 arg1*)
unfolding *execute-DDIV-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DADDU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DADDU arg0 arg1 arg2*)
unfolding *execute-DADDU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DADDIU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DADDIU arg0 arg1 arg2*)
unfolding *execute-DADDIU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DADDI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DADDI arg0 arg1 arg2*)
unfolding *execute-DADDI-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-DADD*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-DADD arg0 arg1 arg2*)
unfolding *execute-DADD-def bind-assoc*

by (*preserves-invariantI*)

lemma *preserves-invariant-writeCapReg*[*preserves-invariantI*]:
shows *traces-preserve-invariant* (*writeCapReg n v*)
by (*intro no-reg-writes-traces-preserve-invariantI no-reg-writes-to-writeCapReg*)
(*simp add*: *CapRegs-names-def trans-regs-def*)

lemma *preserves-invariant-execute-ClearRegs*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-ClearRegs arg0 arg1*)
unfolding *execute-ClearRegs-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CWriteHwr*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CWriteHwr arg0 arg1*)
unfolding *execute-CWriteHwr-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CUnseal*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CUnseal arg0 arg1 arg2*)
unfolding *execute-CUnseal-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CToPtr*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CToPtr arg0 arg1 arg2*)
unfolding *execute-CToPtr-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CTestSubset*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CTestSubset arg0 arg1 arg2*)
unfolding *execute-CTestSubset-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSub*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CSub arg0 arg1 arg2*)
unfolding *execute-CSub-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CStoreConditional*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CStoreConditional arg0 arg1 arg2 arg3*)
unfolding *execute-CStoreConditional-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CStore*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CStore arg0 arg1 arg2 arg3 arg4*)
unfolding *execute-CStore-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSetOffset*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CSetOffset arg0 arg1 arg2*)

unfolding *execute-CSetOffset-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSetFlags*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CSetFlags arg0 arg1 arg2*)
unfolding *execute-CSetFlags-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSetCause*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CSetCause arg0*)
unfolding *execute-CSetCause-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSetCID*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CSetCID arg0*)
unfolding *execute-CSetCID-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSetBoundsImmediate*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CSetBoundsImmediate arg0 arg1 arg2*)
unfolding *execute-CSetBoundsImmediate-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSetBoundsExact*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CSetBoundsExact arg0 arg1 arg2*)
unfolding *execute-CSetBoundsExact-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSetBounds*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CSetBounds arg0 arg1 arg2*)
unfolding *execute-CSetBounds-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSetAddr*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CSetAddr arg0 arg1 arg2*)
unfolding *execute-CSetAddr-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSeal*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CSeal arg0 arg1 arg2*)
unfolding *execute-CSeal-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSCC*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CSCC arg0 arg1 arg2*)
unfolding *execute-CSCC-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CSC*[*preserves-invariantI*]:

shows *runs-preserve-invariant* (*execute-CSC arg0 arg1 arg2 arg3*)
unfolding *execute-CSC-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CReturn*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CReturn arg0*)
unfolding *execute-CReturn-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CReadHwr*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CReadHwr arg0 arg1*)
unfolding *execute-CReadHwr-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CRAP*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CRAP arg0 arg1*)
unfolding *execute-CRAP-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CRAM*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CRAM arg0 arg1*)
unfolding *execute-CRAM-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CPtrCmp*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CPtrCmp arg0 arg1 arg2 arg3*)
unfolding *execute-CPtrCmp-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CMove*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CMove arg0 arg1*)
unfolding *execute-CMove-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CMOVX*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CMOVX arg0 arg1 arg2 arg3*)
unfolding *execute-CMOVX-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CLoadTags*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CLoadTags arg0 arg1*)
unfolding *execute-CLoadTags-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CLoadLinked*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CLoadLinked arg0 arg1 arg2 arg3*)
unfolding *execute-CLoadLinked-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CLoad*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CLoad arg0 arg1 arg2 arg3 arg4 arg5*)
unfolding *execute-CLoad-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CLLC*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CLLC arg0 arg1*)
unfolding *execute-CLLC-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CLCBI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CLCBI arg0 arg1 arg2*)
unfolding *execute-CLCBI-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CLC*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CLC arg0 arg1 arg2 arg3*)
unfolding *execute-CLC-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CJALR*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CJALR arg0 arg1 arg2*)
unfolding *execute-CJALR-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CIncOffsetImmediate*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CIncOffsetImmediate arg0 arg1 arg2*)
unfolding *execute-CIncOffsetImmediate-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CIncOffset*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CIncOffset arg0 arg1 arg2*)
unfolding *execute-CIncOffset-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetType*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetType arg0 arg1*)
unfolding *execute-CGetType-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetTag*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetTag arg0 arg1*)
unfolding *execute-CGetTag-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetSealed*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetSealed arg0 arg1*)

unfolding *execute-CGetSealed-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetPerm*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetPerm arg0 arg1*)
unfolding *execute-CGetPerm-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetPCCSetOffset*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetPCCSetOffset arg0 arg1*)
unfolding *execute-CGetPCCSetOffset-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetPCC*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetPCC arg0*)
unfolding *execute-CGetPCC-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetOffset*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetOffset arg0 arg1*)
unfolding *execute-CGetOffset-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetLen*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetLen arg0 arg1*)
unfolding *execute-CGetLen-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetFlags*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetFlags arg0 arg1*)
unfolding *execute-CGetFlags-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetCause*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetCause arg0*)
unfolding *execute-CGetCause-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetCID*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetCID arg0*)
unfolding *execute-CGetCID-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetBase*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetBase arg0 arg1*)
unfolding *execute-CGetBase-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetAndAddr*[*preserves-invariantI*]:

shows *runs-preserve-invariant* (*execute-CGetAndAddr arg0 arg1 arg2*)
unfolding *execute-CGetAndAddr-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CGetAddr*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CGetAddr arg0 arg1*)
unfolding *execute-CGetAddr-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CFromPtr*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CFromPtr arg0 arg1 arg2*)
unfolding *execute-CFromPtr-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CCopyType*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CCopyType arg0 arg1 arg2*)
unfolding *execute-CCopyType-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CClearTag*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CClearTag arg0 arg1*)
unfolding *execute-CClearTag-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CCheckType*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CCheckType arg0 arg1*)
unfolding *execute-CCheckType-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CCheckTag*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CCheckTag arg0*)
unfolding *execute-CCheckTag-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CCheckPerm*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CCheckPerm arg0 arg1*)
unfolding *execute-CCheckPerm-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CCall*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CCall arg0 arg1 arg2*)
unfolding *execute-CCall-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CCSeal*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CCSeal arg0 arg1 arg2*)
unfolding *execute-CCSeal-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CBuildCap*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CBuildCap arg0 arg1 arg2*)
unfolding *execute-CBuildCap-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CBZ*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CBZ arg0 arg1 arg2*)
unfolding *execute-CBZ-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CBX*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CBX arg0 arg1 arg2*)
unfolding *execute-CBX-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CAndPerm*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CAndPerm arg0 arg1 arg2*)
unfolding *execute-CAndPerm-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-CAndAddr*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CAndAddr arg0 arg1 arg2*)
unfolding *execute-CAndAddr-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-checkCP0Access*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*checkCP0Access u*)
using *Run-inv-checkCP0Access-False*
unfolding *runs-preserve-invariant-def trace-preserves-invariant-def Run-inv-def*
by *auto*

lemma *preserves-invariant-execute-CACHE*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-CACHE arg0 arg1 arg2*)
unfolding *execute-CACHE-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-BREAK*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-BREAK arg0*)
unfolding *execute-BREAK-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-BEQ*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-BEQ arg0 arg1 arg2 arg3 arg4*)
unfolding *execute-BEQ-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-BCMPZ*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-BCMPZ arg0 arg1 arg2 arg3 arg4*)
unfolding *execute-BCMPZ-def bind-assoc*

by (*preserves-invariantI*)

lemma *preserves-invariant-execute-ANDI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-ANDI arg0 arg1 arg2*)
unfolding *execute-ANDI-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-AND*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-AND arg0 arg1 arg2*)
unfolding *execute-AND-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-ADDU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-ADDU arg0 arg1 arg2*)
unfolding *execute-ADDU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-ADDIU*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-ADDIU arg0 arg1 arg2*)
unfolding *execute-ADDIU-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-ADDI*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-ADDI arg0 arg1 arg2*)
unfolding *execute-ADDI-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute-ADD*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute-ADD arg0 arg1 arg2*)
unfolding *execute-ADD-def bind-assoc*
by (*preserves-invariantI*)

lemma *preserves-invariant-execute*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*execute instr*)
by (*cases instr rule*: *execute.cases*; *simp*; *preserves-invariantI*)

lemma *preserves-invariant-write-reg-PCC*[*preserves-invariantI*]:
traces-preserve-invariant (*write-reg PCC-ref v*)
by (*auto simp*: *write-reg-def traces-preserve-invariant-def elim*!: *Write-reg-TracesE*)
(*auto simp*: *trace-preserves-invariant-def trans-inv-def register-defs split*: *option.splits*)

lemma *preserves-invariant-cp2-next-pc*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*cp2-next-pc u*)
unfolding *cp2-next-pc-def*
by (*preserves-invariantI*)

lemma *preserves-invariant-fetch*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*fetch u*)

unfolding *fetch-def*
by (*preserves-invariantI*)

end

context *CHERI-MIPS-Mem-Instr-Automaton*
begin

lemmas *non-cap-exp-traces-enabled*[*traces-enabledI*] = *non-cap-expI*[*THEN non-cap-exp-traces-enabledI*]

lemmas *non-mem-exp-traces-enabled*[*traces-enabledI*] = *non-mem-expI*[*THEN non-mem-exp-traces-enabledI*]

lemma *notnotE*[*derivable-capsE*]:
assumes $\neg\neg P$
obtains P
using *assms*
by *blast*

lemma *getCapCursor-mod-pow2-64*[*simp*]:
getCapCursor c mod 18446744073709551616 = getCapCursor c
using *uint-idem*[*of Capability-address c*]
by (*auto simp*: *getCapCursor-def*)

lemma *mem-val-is-local-cap-Capability-global*[*simp*]:
mem-val-is-local-cap CC ISA (*mem-bytes-of-word* (*capToMemBits c*)) *tag* $\longleftrightarrow$
$\neg$*Capability-global c* $\wedge$ *tag* $\neq$ *BU*
by (*cases tag*) (*auto simp*: *mem-val-is-local-cap-def bind-eq-Some-conv*)

declare *cap-size-def*[*simp*]

lemma *access-enabled-Store*[*derivable-capsE*]:
assumes *Capability-permit-store c*
and *tag* $\neq$ *B0* $\longrightarrow$ *Capability-permit-store-cap c*
and *mem-val-is-local-cap CC ISA v tag* $\wedge$ *tag* = *B1* $\longrightarrow$ *Capability-permit-store-local-cap c*
and *Capability-tag c* **and** $\neg$*Capability-sealed c*
and *paddr-in-mem-region c Store paddr sz*
and *c* $\in$ *derivable-caps s*
and *tag* = *B0* $\vee$ *tag* = *B1* **and** *length v* = *sz*
and *tag* $\neq$ *B0* $\longrightarrow$ *address-tag-aligned ISA paddr* $\wedge$ *sz* = *tag-granule ISA*
shows *access-enabled s Store paddr sz v tag*
using *assms*
unfolding *access-enabled-def authorises-access-def has-access-permission-def*
by (*auto simp*: *get-cap-perms-def derivable-caps-def*)

lemma *access-enabled-Load*[*derivable-capsE*]:
assumes *Capability-permit-load c*
and *tag* $\neq$ *B0* $\longrightarrow$ *Capability-permit-load-cap c*

and *Capability-tag c* **and** ¬*Capability-sealed c*
and *paddr-in-mem-region c Load paddr sz*
and $c \in$ *derivable-caps s*
and *tag* $\neq$ *B0* $\longrightarrow$ *address-tag-aligned ISA paddr* $\wedge$ *sz* = *tag-granule ISA*
shows *access-enabled s Load paddr sz v tag*
using *assms*
unfolding *access-enabled-def authorises-access-def has-access-permission-def*
by (*auto simp*: *get-cap-perms-def derivable-caps-def*)

lemma [*simp*]: *isa.translate-address ISA* = *translate-address*
by (*auto simp*: *ISA-def*)

fun *acctype-of-bool* **where**
acctype-of-bool True = *LoadData*
| *acctype-of-bool False* = *StoreData*

lemma *Run-raise-c2-exception-False*[*simp*]:
Run (*raise-c2-exception ex r*) *t a* $\longleftrightarrow$ *False*
Run-inv (*raise-c2-exception ex r*) *t a regs* $\longleftrightarrow$ *False*
unfolding *Run-inv-def*
by (*auto simp*: *raise-c2-exception-def raise-c2-exception8-def elim*!: *Run-bindE*)

lemma *Run-if-then-raise-c2-exception-else*[*simp*]:
Run (*if c then raise-c2-exception ex r else m*) *t a* $\longleftrightarrow$ ¬*c* $\wedge$ *Run m t a*
Run-inv (*if c then raise-c2-exception ex r else m*) *t a regs* $\longleftrightarrow$ ¬*c* $\wedge$ *Run-inv m t a regs*
by *auto*

lemma *no-translation-tables*[*simp*]: *translation-tables ISA t* = {}
by (*auto simp*: *ISA-def*)

lemma *Run-read-reg-DDC-derivable-caps*:
assumes *Run* (*read-reg DDC-ref*) *t c* **and** {*''DDC''*} $\subseteq$ *accessible-regs s*
shows $c \in$ *derivable-caps* (*run s t*)
using *assms*
by (*auto elim*!: *Run-read-regE simp*: *DDC-ref-def derivable-caps-def intro*!: *derivable.Copy*)

abbreviation *empty-trace* :: *register-value trace* **where** *empty-trace* ≡ []

lemma *Run-inv-addrWrapper-access-enabled*[*derivable-capsE*]:
assumes *Run-inv* (*addrWrapper addr acctype width*) *t vaddr regs*
and *translate-address* (*unat vaddr*) *acctype′ empty-trace* = *Some paddr*
and {*''DDC''*} $\subseteq$ *accessible-regs s*
and *acctype* = *MemAccessType-of-acctype acctype′*
and *acctype′* = *Store* $\longrightarrow$ *length v* = *nat sz*
and *sz* = *wordWidthBytes width*
shows *access-enabled* (*run s t*) *acctype′ paddr* (*nat sz*) *v B0*
using *assms*

unfolding *Run-inv-def addrWrapper-def checkDDCPerms-def Let-def*
unfolding *access-enabled-def authorises-access-def has-access-permission-def paddr-in-mem-region-def*
apply (*cases acctype'*)
apply (*auto elim*!: *Run-bindE simp*: *get-cap-perms-def getCapBounds-def address-range-def derivable-caps-def dest*!: *Run-read-reg-DDC-derivable-caps*)
subgoal for *c*
apply (*rule bexI*[**where** $x = c$])
apply (*clarify*)
apply (*rule exI*[**where** $x =$ *unat vaddr*])
by *auto*
subgoal for *c*
apply (*rule bexI*[**where** $x = c$])
apply (*clarify*)
apply (*rule exI*[**where** $x =$ *unat vaddr*])
by *auto*
done

lemma *Run-read-reg-DDC-access-enabled*:
assumes *Run* (*read-reg DDC-ref*) *t c*
and {''*DDC*''} $\subseteq$ *accessible-regs s*
and *Capability-tag c* **and** $\neg$*Capability-sealed c*
and *paddr-in-mem-region c acctype paddr sz*
and *acctype* = *Store* $\longrightarrow$ *length v* = *nat sz*
and *acctype* = *Load* $\wedge$ *Capability-permit-load c* $\vee$ *acctype* = *Store* $\wedge$ *Capability-permit-store c*
shows *access-enabled* (*run s t*) *acctype paddr sz v B0*
using *assms*
unfolding *access-enabled-def authorises-access-def has-access-permission-def*
by (*auto simp*: *get-cap-perms-def derivable-caps-def dest*!: *Run-read-reg-DDC-derivable-caps*)

lemma *translate-address-paddr-in-mem-region*:
assumes *translate-address* (*nat vaddr*) *is-load empty-trace* = *Some paddr*
and *getCapBase c* $\leq$ *vaddr* **and** *vaddr* + *sz* $\leq$ *getCapTop c*
and $0 \leq$ *vaddr*
shows *paddr-in-mem-region c is-load paddr* (*nat sz*)
using *assms*
unfolding *paddr-in-mem-region-def*
by (*intro exI*[**where** $x =$ *nat vaddr*])
(*auto simp*: *paddr-in-mem-region-def address-range-def simp flip*: *nat-add-distrib*)

lemma *pos-mod-le*[*simp*]:
$0 < b \Longrightarrow a \bmod b \leq (b :: int)$
by (*auto simp*: *le-less*)

lemma *mod-diff-mod-eq*:
fixes *a b c* :: *int*
assumes *c dvd b* **and** $0 < b$ **and** $0 < c$
shows $(a \bmod b - a \bmod c) \bmod b = a \bmod b - a \bmod c$
using *assms*

apply (*auto simp*: *dvd-def*)
by (*smt Divides.pos-mod-bound assms*(*1*) *int-mod-eq′ mod-mod-cancel unique-euclidean-semiring-numeral-cla*

lemma *mod-le-dvd-divisor*:
fixes *a b c* :: *int*
assumes *c dvd b* **and** *0* < *b* **and** *0* < *c*
shows *a mod c* ≤ *a mod b*
using *assms*
apply (*auto simp*: *dvd-def*)
by (*metis assms*(*1*) *assms*(*2*) *mod-mod-cancel pos-mod-conj zmod-le-nonneg-dividend*)

lemma *Run-inv-addrWrapperUnaligned-access-enabled*[*derivable-capsE*]:
assumes *Run-inv* (*addrWrapperUnaligned addr acctype width*) *t* (*vaddr*, *sz*) *regs*
and *translate-address* (*unat vaddr*) *acctype′ empty-trace* = *Some paddr*
and {*′′DDC′′*} ⊆ *accessible-regs s*
and *acctype* = *MemAccessType-of-acctype acctype′*
and *acctype′* = *Store* ⟶ *length v* = *nat sz*
shows *access-enabled* (*run s t*) *acctype′ paddr* (*nat sz*) *v B0*
using *assms*
unfolding *Run-inv-def addrWrapperUnaligned-def unalignedBytesTouched-def checkDDCPerms-def Let-def*
by (*cases width*; *cases acctype′*;
auto elim!: *Run-bindE Run-read-reg-DDC-access-enabled translate-address-paddr-in-mem-region simp*: *getCapBounds-def mod-mod-cancel mod-diff-mod-eq mod-le-dvd-divisor*)

lemma *access-enabled-run-mono*:
assumes *access-enabled s is-load paddr sz v tag*
shows *access-enabled* (*run s t*) *is-load paddr sz v tag*
using *assms derivable-mono*[*OF accessed-caps-run-mono*[**where** *s* = *s* **and** *t* = *t*]]
unfolding *access-enabled-def*
by *blast*

declare *Run-inv-addrWrapperUnaligned-access-enabled*[*THEN access-enabled-run-mono*, *derivable-capsE*]

lemma *TLBTranslateC-translate-address-eq*[*simp*]:
assumes *Run-inv* (*TLBTranslateC vaddr acctype*) *t* (*paddr*, *noStoreCap*) *regs*
and *acctype* = *MemAccessType-of-acctype acctype′*
shows *translate-address* (*unat vaddr*) *acctype′ t′* = *Some* (*unat paddr*)
proof −
from *assms* **have** *Run-inv* (*translate-addressM* (*unat vaddr*) *acctype′*) *t* (*unat paddr*) *regs*
unfolding *translate-addressM-def TLBTranslate-def bind-assoc Run-inv-def*
by (*auto simp flip*: *uint-nat intro*: *Traces-bindI*[*of - t - -* [], *simplified*])
then show *?thesis*

using *determ-runs-translate-addressM*
by (*auto simp*: *translate-address-def determ-the-result-eq*)
qed

lemma *TLBTranslate-translate-address-eq*[*simp*]:
assumes *Run-inv* (*TLBTranslate vaddr acctype*) *t paddr regs*
and *acctype* = *MemAccessType-of-acctype acctype'*
shows *translate-address* (*unat vaddr*) *acctype' t'* = *Some* (*unat paddr*)
proof −
from *assms* **have** *Run-inv* (*translate-addressM* (*unat vaddr*) *acctype'*) *t* (*unat paddr*) *regs*
unfolding *translate-addressM-def bind-assoc Run-inv-def*
by (*auto simp flip*: *uint-nat intro*: *Traces-bindI*[*of - t - -* [], *simplified*])
then show *?thesis*
using *determ-runs-translate-addressM*
by (*auto simp*: *translate-address-def determ-the-result-eq*)
qed

lemma *traces-enabled-bind-prod-split*[*traces-enabled-combinatorI*]:
assumes $\bigwedge t\ a\ b$. *Run-inv m t* (*a*, *b*) *regs* $\Longrightarrow$ *traces-enabled* (*f a b*) (*run s t*) (*the* (*updates-regs trans-regs t regs*))
and *runs-preserve-invariant m* **and** *traces-enabled m s regs*
shows *traces-enabled* (*m* $\gg=$ (λ*vars*. *let* (*a*, *b*) = *vars in f a b*)) *s regs*
using *assms*
by (*auto intro*: *traces-enabled-bind*)

lemma *TLBTranslate-paddr-in-mem-region*[*derivable-capsE*]:
assumes *Run-inv* (*TLBTranslate vaddr acctype*) *t paddr regs*
and *getCapBase c* $\leq$ *uint vaddr* **and** *uint vaddr* + *sz* $\leq$ *getCapTop c* **and** *0* $\leq$ *sz*
and *acctype* = *MemAccessType-of-acctype acctype'*
shows *paddr-in-mem-region c acctype'* (*unat paddr*) (*nat sz*)
using *assms TLBTranslate-translate-address-eq*[*OF assms*(*1*), **where** *t'* = []]
unfolding *paddr-in-mem-region-def*
by (*intro exI*[**where** *x* = *unat vaddr*])
(*auto simp add*: *address-range-def unat-def simp flip*: *nat-add-distrib*)

lemma *TLBTranslateC-paddr-in-mem-region*[*derivable-capsE*]:
assumes *Run-inv* (*TLBTranslateC vaddr acctype*) *t* (*paddr*, *noStoreCap*) *regs*
and *getCapBase c* $\leq$ *uint vaddr* **and** *uint vaddr* + *sz* $\leq$ *getCapTop c* **and** *0* $\leq$ *sz*
and *acctype* = *MemAccessType-of-acctype acctype'*
shows *paddr-in-mem-region c acctype'* (*unat paddr*) (*nat sz*)
using *assms TLBTranslateC-translate-address-eq*[*OF assms*(*1*), **where** *t'* = []]
unfolding *paddr-in-mem-region-def*
by (*intro exI*[**where** *x* = *unat vaddr*])
(*auto simp add*: *address-range-def unat-def simp flip*: *nat-add-distrib*)

lemma *non-cap-exp-MEMea*[*non-cap-expI*]:
non-cap-exp (*MEMea addr sz*)
unfolding *MEMea-def write-mem-ea-def maybe-fail-def*
by (*auto simp*: *non-cap-exp-def elim*: *Traces-cases*)

lemma *non-cap-exp-MEMea-conditional*[*non-cap-expI*]:
non-cap-exp (*MEMea-conditional addr sz*)
unfolding *MEMea-conditional-def write-mem-ea-def maybe-fail-def*
by (*auto simp*: *non-cap-exp-def elim*: *Traces-cases*)

lemma *traces-enabled-write-mem-ea*[*traces-enabledI*]:
shows *traces-enabled* (*write-mem-ea BC-mword wk addr-sz addr sz*) *s regs*
by (*auto simp*: *write-mem-ea-def maybe-fail-def traces-enabled-def split*: *option.splits elim*: *Traces-cases*)

lemma *traces-enabled-write-mem*[*traces-enabledI*]:
assumes *access-enabled s Store* (*unat addr*) (*nat sz*) (*mem-bytes-of-word v*) *B0*
shows *traces-enabled* (*write-mem BC-mword BC-mword wk addr-sz addr sz v*) *s regs*
using *assms*
by (*auto simp*: *write-mem-def traces-enabled-def split*: *option.splits elim*: *Traces-cases*)

lemma *traces-enabled-write-memt*[*traces-enabledI*]:
assumes *access-enabled s Store* (*unat addr*) (*nat sz*) (*mem-bytes-of-word v*) *tag*
shows *traces-enabled* (*write-memt BC-mword BC-mword wk addr sz v tag*) *s regs*
using *assms*
by (*auto simp*: *write-memt-def traces-enabled-def split*: *option.splits elim*: *Traces-cases*)

lemma *traces-enabled-read-mem-bytes*[*traces-enabledI*]:
assumes $\bigwedge$*bytes*. *access-enabled s Load* (*unat addr*) (*nat sz*) *bytes B0*
shows *traces-enabled* (*read-mem-bytes BC-mword BC-mword rk addr sz*) *s regs*
using *assms*
by (*auto simp*: *read-mem-bytes-def maybe-fail-def traces-enabled-def split*: *option.splits elim*: *Traces-cases*)

lemma *traces-enabled-read-mem*[*traces-enabledI*]:
assumes $\bigwedge$*bytes*. *access-enabled s Load* (*unat addr*) (*nat sz*) *bytes B0*
shows *traces-enabled* (*read-mem BC-mword BC-mword rk addr-sz addr sz*) *s regs*
unfolding *read-mem-def*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-read-memt-bytes*[*traces-enabledI*]:
assumes $\bigwedge$*bytes tag*. *access-enabled s Load* (*unat addr*) (*nat sz*) *bytes tag*
shows *traces-enabled* (*read-memt-bytes BC-mword BC-mword rk addr sz*) *s regs*

using *assms*
by (*auto simp*: *read-memt-bytes-def maybe-fail-def traces-enabled-def split*: *option.splits elim*: *Traces-cases*)

lemma *traces-enabled-read-memt*[*traces-enabledI*]:
assumes $\bigwedge$*bytes tag*. *access-enabled s Load* (*unat addr*) (*nat sz*) *bytes tag*
shows *traces-enabled* (*read-memt BC-mword BC-mword rk addr sz*) *s regs*
unfolding *read-memt-def*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMea*[*traces-enabledI*]:
shows *traces-enabled* (*MEMea arg0 arg1*) *s regs*
unfolding *MEMea-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMea-conditional*[*traces-enabledI*]:
shows *traces-enabled* (*MEMea-conditional arg0 arg1*) *s regs*
unfolding *MEMea-conditional-def bind-assoc*
by (*traces-enabledI*)

lemma *traces-enabled-MEMval*[*traces-enabledI*]:
assumes *access-enabled s Store* (*unat addr*) (*nat sz*) (*mem-bytes-of-word v*) *B0*
shows *traces-enabled* (*MEMval addr sz v*) *s regs*
unfolding *MEMval-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMr*[*traces-enabledI*]:
assumes $\bigwedge$*bytes*. *access-enabled s Load* (*unat addr*) (*nat sz*) *bytes B0*
shows *traces-enabled* (*MEMr addr sz*) *s regs*
unfolding *MEMr-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MIPS-write*[*traces-enabledI*]:
assumes *access-enabled s Store* (*unat addr*) (*nat sz*) (*mem-bytes-of-word v*) *B0*
shows *traces-enabled* (*MIPS-write addr sz v*) *s regs*
unfolding *MIPS-write-def write-ram-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MIPS-read*[*traces-enabledI*]:
assumes $\bigwedge$*bytes*. *access-enabled s Load* (*unat addr*) (*nat sz*) *bytes B0*
shows *traces-enabled* (*MIPS-read addr sz*) *s regs*
unfolding *MIPS-read-def read-ram-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMr-reserve*[*traces-enabledI*]:
assumes $\bigwedge$*bytes*. *access-enabled s Load* (*unat addr*) (*nat sz*) *bytes B0*
shows *traces-enabled* (*MEMr-reserve addr sz*) *s regs*
unfolding *MEMr-reserve-def bind-assoc*

by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMval-conditional*[*traces-enabledI*]:
assumes *access-enabled s Store* (*unat addr*) (*nat sz*) (*mem-bytes-of-word v*) *B0*
shows *traces-enabled* (*MEMval-conditional addr sz v*) *s regs*
unfolding *MEMval-conditional-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMr-wrapper*[*traces-enabledI*]:
assumes $\bigwedge$*bytes*. *access-enabled s Load* (*unat addr*) (*nat sz*) *bytes B0*
shows *traces-enabled* (*MEMr-wrapper addr sz*) *s regs*
unfolding *MEMr-wrapper-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMr-reserve-wrapper*[*traces-enabledI*]:
assumes $\bigwedge$*bytes*. *access-enabled s Load* (*unat addr*) (*nat sz*) *bytes B0*
shows *traces-enabled* (*MEMr-reserve-wrapper addr sz*) *s regs*
unfolding *MEMr-reserve-wrapper-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMr-tagged*[*traces-enabledI*]:
assumes $\bigwedge$*bytes tag*. *tag* $\neq$ *B0* $\longrightarrow$ *allow-tag* $\Longrightarrow$ *access-enabled s Load* (*unat addr*) (*nat sz*) *bytes tag*
shows *traces-enabled* (*MEMr-tagged addr sz allow-tag*) *s regs*
unfolding *MEMr-tagged-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMr-tagged-reserve*[*traces-enabledI*]:
assumes $\bigwedge$*bytes tag*. *tag* $\neq$ *B0* $\longrightarrow$ *allow-tag* $\Longrightarrow$ *access-enabled s Load* (*unat addr*) (*nat sz*) *bytes tag*
shows *traces-enabled* (*MEMr-tagged-reserve addr sz allow-tag*) *s regs*
unfolding *MEMr-tagged-reserve-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMw-tagged*[*traces-enabledI*]:
assumes *access-enabled s Store* (*unat addr*) (*nat sz*) (*mem-bytes-of-word v*) (*bitU-of-bool tag*)
shows *traces-enabled* (*MEMw-tagged addr sz tag v*) *s regs*
unfolding *MEMw-tagged-def MEMval-tagged-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMw-tagged-conditional*[*traces-enabledI*]:
assumes *access-enabled s Store* (*unat addr*) (*nat sz*) (*mem-bytes-of-word v*) (*bitU-of-bool tag*)
shows *traces-enabled* (*MEMw-tagged-conditional addr sz tag v*) *s regs*
unfolding *MEMw-tagged-conditional-def MEMval-tagged-conditional-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMw-wrapper*[*traces-enabledI*]:

assumes *access-enabled s Store* (*unat addr*) (*nat sz*) (*mem-bytes-of-word v*) *B0*
shows *traces-enabled* (*MEMw-wrapper addr sz v*) *s regs*
unfolding *MEMw-wrapper-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMw-conditional-wrapper*[*traces-enabledI*]:
assumes *access-enabled s Store* (*unat addr*) (*nat sz*) (*mem-bytes-of-word v*) *B0*
shows *traces-enabled* (*MEMw-conditional-wrapper addr sz v*) *s regs*
unfolding *MEMw-conditional-wrapper-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

declare *Run-inv-addrWrapper-access-enabled*[*THEN access-enabled-run-mono*, *derivable-capsE*]

lemma *traces-enabled-execute-Store*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-Store arg0 arg1 arg2 arg3 arg4*) *s regs*
unfolding *execute-Store-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-SWR*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-SWR arg0 arg1 arg2*) *s regs*
unfolding *execute-SWR-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-SWL*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-SWL arg0 arg1 arg2*) *s regs*
unfolding *execute-SWL-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-SDR*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-SDR arg0 arg1 arg2*) *s regs*
unfolding *execute-SDR-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-SDL*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-SDL arg0 arg1 arg2*) *s regs*
unfolding *execute-SDL-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-Load*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-Load arg0 arg1 arg2 arg3 arg4 arg5*) *s regs*

unfolding *execute-Load-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-LWR*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-LWR arg0 arg1 arg2*) *s regs*
unfolding *execute-LWR-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-LWL*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-LWL arg0 arg1 arg2*) *s regs*
unfolding *execute-LWL-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-LDR*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-LDR arg0 arg1 arg2*) *s regs*
unfolding *execute-LDR-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-LDL*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-LDL arg0 arg1 arg2*) *s regs*
unfolding *execute-LDL-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CStoreConditional*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CStoreConditional arg0 arg1 arg2 arg3*) *s regs*
unfolding *execute-CStoreConditional-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CStore*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CStore arg0 arg1 arg2 arg3 arg4*) *s regs*
unfolding *execute-CStore-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CSCC*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CSCC arg0 arg1 arg2*) *s regs*
unfolding *execute-CSCC-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CSC*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CSC arg0 arg1 arg2 arg3*) *s regs*
unfolding *execute-CSC-def bind-assoc*

by (*traces-enabledI assms*: *assms*)

declare *traces-enabled-foreachM-inv*[**where** $P = \lambda$*vars s regs*. *True*, *simplified*, *traces-enabledI*]
thm *traces-enabled-foreachM-inv*[**where** $s = s$ **and** $P = \lambda$*vars s' regs'*. *derivable-caps* $s \subseteq$ *derivable-caps s'* **for** *s*]

lemma *uint-cacheline-plus-cap-size*:
assumes *getCapCursor* $c = 128 * q$ **and** $0 \leq x$ **and** $x \leq 3$
shows *uint* (*to-bits 64 128* $*$ *to-bits 64 q* + (*word-of-int* ($x * 32$) :: *64 word*)) $= 128 * q + x * 32$
proof −
have $128 * q < 2^{64}$ **and** $*$: $0 \leq 128 * q$
using *uint-bounded*[*of Capability-address c*]
unfolding *assms*(*1*)[*symmetric*] *getCapCursor-def*
by (*auto*)
moreover have $0 \leq q$
using $*$
by *auto*
ultimately show *?thesis*
using *assms*
by (*auto simp*: *uint-word-ariths getCapCursor-def uint-word-of-int*)
qed

lemma *traces-enabled-execute-CLoadTags*[*traces-enabledI*]:
assumes {*''PCC''*} $\subseteq$ *accessible-regs s* **and** *CapRegs-names* $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*execute-CLoadTags arg0 arg1*) *s regs*
unfolding *execute-CLoadTags-def bind-assoc*
apply (*traces-enabledI-with* ‹−› *intro*: *traces-enabled-foreachM-inv*[**where** $s = s$ **and** $P = \lambda$*vars s' regs'*. *derivable-caps s* $\subseteq$ *derivable-caps s'* **for** *s*])
apply (*derivable-caps-step*)
apply (*derivable-caps-step*)
apply (*derivable-caps-step*)
apply (*derivable-caps-step*)
apply (*derivable-caps-step*)
apply (*auto*)[]
apply (*auto*)[]
apply (*auto*)[]

apply (*derivable-caps-step*)
apply (*auto simp*: *caps-per-cacheline-def uint-cacheline-plus-cap-size*)[]
apply (*auto simp*: *caps-per-cacheline-def uint-cacheline-plus-cap-size*)[]
apply (*auto simp*: *caps-per-cacheline-def*)[]
apply (*auto simp*: *caps-per-cacheline-def*)[]

apply (*derivable-caps-step*)

apply (*elim set-mp*)
apply (*derivable-capsI assms*: *assms*)[]
apply (*auto simp*: *caps-per-cacheline-def*)[]

apply (*elim subset-trans*)
apply (*intro derivable-caps-run-mono*)
apply (*auto simp*: *caps-per-cacheline-def*)[]
done

lemma *traces-enabled-execute-CLoadLinked*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CLoadLinked arg0 arg1 arg2 arg3*) *s regs*
unfolding *execute-CLoadLinked-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma [*simp*]: *integerOfString ''18446744073709551616''* = *18446744073709551616*
by *eval*

lemma *traces-enabled-execute-CLoad*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CLoad arg0 arg1 arg2 arg3 arg4 arg5*) *s regs*
unfolding *execute-CLoad-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CLLC*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CLLC arg0 arg1*) *s regs*
unfolding *execute-CLLC-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CLCBI*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CLCBI arg0 arg1 arg2*) *s regs*
unfolding *execute-CLCBI-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-execute-CLC*[*traces-enabledI*]:
assumes {*''PCC''*} ⊆ *accessible-regs s* **and** *CapRegs-names* ⊆ *accessible-regs s*
shows *traces-enabled* (*execute-CLC arg0 arg1 arg2 arg3*) *s regs*
unfolding *execute-CLC-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-instr-sem*[*traces-enabledI*]:
assumes {*''DDC''*, *''PCC''*} ⊆ *accessible-regs s*
and *CapRegs-names* ⊆ *accessible-regs s*

shows *traces-enabled* (*instr-sem ISA instr*) *s regs*
by (*cases instr rule*: *execute.cases*; *simp*; *use nothing* **in** ‹*traces-enabledI assms*: *assms*›)

lemma *hasTrace-instr-mem-axioms*:
assumes *hasTrace t* (*instr-sem ISA instr*)
and *reads-regs-from trans-regs t regs* **and** *trans-inv regs*
and *instr-raises-ex ISA instr t* $\longrightarrow$ *ex-traces*
shows *store-mem-axiom CC ISA t*
and *store-tag-axiom CC ISA t*
and *load-mem-axiom CC ISA False t*
using *assms*
by (*intro traces-enabled-mem-axioms*[**where** *m* = *instr-sem ISA instr* **and** *regs* = *regs*] *traces-enabled-instr-sem*;
auto)+

end

3.5 Instruction fetch properties

context *CHERI-MIPS-Mem-Fetch-Automaton*
begin

lemmas *non-cap-exp-traces-enabled*[*traces-enabledI*] = *non-cap-expI*[*THEN non-cap-exp-traces-enabledI*]

lemmas *non-mem-exp-traces-enabled*[*traces-enabledI*] = *non-mem-expI*[*THEN non-mem-exp-traces-enabledI*]

thm *Run-bind-trace-enabled traces-enabled-bind*

lemma *Run-inv-bind-trace-enabled*:
assumes *Run-inv* ($m \gg= f$) *t a regs* **and** *runs-preserve-invariant m*
and $\bigwedge$*tm tf am*. *t* = *tm* @ *tf* $\Longrightarrow$ *Run-inv m tm am regs* $\Longrightarrow$ *trace-enabled s tm*
and $\bigwedge$*tm tf am*. *t* = *tm* @ *tf* $\Longrightarrow$ *Run-inv m tm am regs* $\Longrightarrow$ *Run-inv* (*f am*) *tf a* (*the* (*updates-regs trans-regs tm regs*)) $\Longrightarrow$ *trace-enabled* (*run s tm*) *tf*
shows *trace-enabled s t*
using *assms*
by (*elim Run-inv-bindE*) (*auto simp*: *trace-enabled-append-iff*)

lemma *traces-enabled-read-mem-bytes*[*traces-enabledI*]:
assumes $\bigwedge$*bytes*. *access-enabled s Fetch* (*unat addr*) (*nat sz*) *bytes B0*
shows *traces-enabled* (*read-mem-bytes BC-mword BC-mword rk addr sz*) *s regs*
using *assms*
by (*auto simp*: *read-mem-bytes-def maybe-fail-def traces-enabled-def split*: *option.splits elim*: *Traces-cases*)

lemma *traces-enabled-MEMr-wrapper*[*traces-enabledI*]:

assumes $\bigwedge$*bytes*. *access-enabled s Fetch* (*unat addr*) (*nat sz*) *bytes B0*
shows *traces-enabled* (*MEMr-wrapper addr sz*) *s regs*
unfolding *MEMr-wrapper-def MEMr-def read-mem-def*
by (*traces-enabledI assms*: *assms*)

lemma [*simp*]: *translation-tables ISA t* = {}
by (*auto simp*: *ISA-def*)

lemma [*simp*]: *isa.translate-address ISA vaddr Fetch t* = *translate-address vaddr Fetch t*
by (*auto simp*: *ISA-def*)

lemma *access-enabled-FetchI*:
assumes *c* ∈ *derivable-caps s* **and** *Capability-tag c* **and** ¬*Capability-sealed c*
and *translate-address vaddr Fetch* ([] :: *register-value trace*) = *Some paddr*
and *vaddr* ≥ *nat* (*getCapBase c*) **and** *vaddr* + *sz* ≤ *nat* (*getCapTop c*)
and *Capability-permit-execute c* **and** *sz* > *0*
shows *access-enabled s Fetch paddr sz bytes B0*
using *assms*
by (*auto simp*: *access-enabled-defs derivable-caps-def address-range-def get-cap-perms-def*)

lemma *Run-inv-no-reg-writes-to-updates-regs-inv*[*simp*]:
assumes *Run-inv m t a regs* **and** *no-reg-writes-to Rs m*
shows *updates-regs Rs t regs'* = *Some regs'*
using *assms*
unfolding *Run-inv-def*
by *auto*

lemma *Run-inv-read-regE*:
assumes *Run-inv* (*read-reg r*) *t v regs*
obtains *rv* **where** *t* = [*E-read-reg* (*name r*) *rv*] **and** *of-regval r rv* = *Some v*
using *assms*
unfolding *Run-inv-def*
by (*auto elim*!: *Run-read-regE*)

lemma [*simp*]: *Run-inv* (*SignalExceptionBadAddr ex badAddr*) *t a regs* ⟷ *False*
by (*auto simp*: *Run-inv-def*)

lemma [*simp*]: *''PCC''* ∈ *trans-regs*
by (*auto simp*: *trans-regs-def*)

lemma *runs-no-reg-writes-to-incrementCP0Count*[*runs-no-reg-writes-toI*, *simp*]:
assumes {*''TLBRandom''*, *''CP0Count''*, *''CP0Cause''*} ∩ *Rs* = {}
shows *runs-no-reg-writes-to Rs* (*incrementCP0Count u*)
using *assms*
unfolding *incrementCP0Count-def Let-def bind-assoc*
by (*no-reg-writes-toI simp*: *register-defs*)

lemma [*simp*]: *runs-no-reg-writes-to trans-regs* (*incrementCP0Count u*)
  by (*auto simp*: *trans-regs-def*)
find-theorems *updates-regs no-reg-writes-to*

lemma *Run-inv-runs-no-reg-writes-to-updates-regs-inv*[*simp*]:
  assumes *Run-inv m t a regs*
    and *runs-no-reg-writes-to trans-regs m*
  shows *updates-regs trans-regs t regs* = *Some regs*
  using *assms*
proof −
  have $\forall$ *r* $\in$ *trans-regs*. $\forall$ *v*. *E-write-reg r v* $\notin$ *set t*
    using *assms*
    by (*auto simp*: *runs-no-reg-writes-to-def Run-inv-def*)
  then show *updates-regs trans-regs t regs* = *Some regs*
    by (*induction trans-regs t regs rule*: *updates-regs.induct*) *auto*
qed

lemma *Run-inv-read-reg-PCC*[*simp*]:
  assumes *Run-inv* (*read-reg PCC-ref*) *t c regs*
  shows *regstate.PCC regs* = *c*
  using *assms*
  by (*auto simp*: *Run-inv-def register-defs regval-of-Capability-def elim*!: *Run-read-regE*)

lemma *foo*:
  assumes ¬ *getCapTop c* < *getCapBase c* + *uint vaddr* + *4* **and** *getCapTop c* ≤ *pow2 64*
  shows *unat* (*to-bits 64* (*getCapBase c* + *uint vaddr*) :: *64 word*) = *nat* (*getCapBase c* + *uint vaddr*) ∧ *nat* (*getCapBase c*) ≤ *nat* (*getCapBase c* + *uint vaddr*) ∧ *nat* (*getCapBase c* + *uint vaddr*) + *4* = *nat* (*getCapBase c* + *uint vaddr* + *4*)
  using *assms*
  by (*auto simp*: *nat-add-distrib getCapBase-def*)

lemma *Run-inv-TranslatePC-access-enabled-Fetch*:
  assumes *Run-inv* (*TranslatePC vaddr*) *t paddr regs*
    and *regstate.PCC regs* ∈ *derivable-caps s*
  shows *access-enabled* (*run s t*) *Fetch* (*unat paddr*) (*nat 4*) *bytes B0*
proof −
  { fix *c*
    assume ¬ *getCapTop c* < *getCapBase c* + *uint vaddr* + *4* **and** *getCapTop c* ≤ *pow2 64*
    then have *unat* (*to-bits 64* (*getCapBase c* + *uint vaddr*) :: *64 word*) = *nat* (*getCapBase c* + *uint vaddr*) ∧ *nat* (*getCapBase c*) ≤ *nat* (*getCapBase c* + *uint vaddr*) ∧ *nat* (*getCapBase c* + *uint vaddr*) + *4* = *nat* (*getCapBase c* + *uint vaddr* + *4*)
      by (*auto simp*: *nat-add-distrib getCapBase-def*)
  }
  from *this*[*of regstate.PCC regs*]
  show *?thesis*

using *assms*
unfolding *TranslatePC-def bind-assoc Let-def*
by (*intro access-enabled-FetchI*[**where** c = *regstate.PCC regs* **and** *vaddr* = *unat* (*to-bits 64* (*getCapBase* (*regstate.PCC regs*) + *uint vaddr*) :: *64 word*)])
(*auto elim*!: *Run-inv-bindE Run-inv-ifE intro*!: *preserves-invariantI intro*: *traces-runs-preserve-invariantI derivable-caps-run-imp simp add*: *getCapBounds-def simp del*: *unat-to-bits dest*!: *TLBTranslate-Instruction-translate-address-eq*[**where** t' = [] :: *register-value trace*])
qed

lemma [*simp*]:
name UART-WRITTEN-ref ∉ *trans-regs*
name InstCount-ref ∉ *trans-regs*
name NextPCC-ref ∉ *trans-regs*
by (*auto simp*: *trans-regs-def register-defs*)

lemma *Run-write-regE*:
assumes *Run* (*write-reg r v*) *t a*
obtains t = [*E-write-reg* (*name r*) (*regval-of r v*)]
using *assms*
by (*auto simp*: *write-reg-def elim*!: *Write-reg-TracesE*)

lemma *Run-inv-write-reg-PCC-updates-regs*[*simp*]:
assumes *Run-inv* (*write-reg PCC-ref c*) *t a regs*
shows *updates-regs trans-regs t regs'* = *Some* (*regs'*(|*regstate.PCC* := *c*|))
using *assms*
unfolding *Run-inv-def*
by (*auto simp*: *register-defs elim*: *Run-write-regE*)

lemma *Run-inv-read-reg-NextPCC-derivable-caps*[*derivable-capsE*]:
assumes *Run-inv* (*read-reg NextPCC-ref*) *t c regs*
and {*''NextPCC''*} ⊆ *accessible-regs s*
shows c ∈ *derivable-caps* (*run s t*)
using *assms*
by (*auto simp*: *step-defs register-defs derivable-caps-def intro*: *derivable.Copy elim*!: *Run-inv-read-regE*)

lemma *Run-inv-cp2-next-pc-PCC-derivable*:
assumes *Run-inv* (*cp2-next-pc* ()) *t a regs*
and {*''NextPCC''*} ⊆ *accessible-regs s*
shows *regstate.PCC* (*the* (*updates-regs trans-regs t regs*)) ∈ *derivable-caps* (*run s t*)
using *assms*(*1*)
unfolding *cp2-next-pc-def*
by (*auto elim*!: *Run-inv-bindE Run-inv-ifE intro*: *preserves-invariantI traces-runs-preserve-invariantI simp*: *regstate-simp*)
(*derivable-capsI assms*: *assms*(*2*))+

lemma *traces-enabled-fetch*[*traces-enabledI*]:

assumes $\{''NextPCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*fetch u*) *s regs*
unfolding *fetch-def bind-assoc*
by (*traces-enabledI elim*: *Run-inv-TranslatePC-access-enabled-Fetch Run-inv-cp2-next-pc-PCC-derivable assms*: *assms*)

lemma *traces-enabled-instr-fetch*[*traces-enabledI*]:
assumes $\{''NextPCC''\} \subseteq$ *accessible-regs s*
shows *traces-enabled* (*instr-fetch ISA*) *s regs*
unfolding *ISA-simps*
by (*traces-enabledI assms*: *assms*)

lemma *hasTrace-fetch-mem-axioms*:
assumes *hasTrace t* (*instr-fetch ISA*)
and *reads-regs-from trans-regs t regs* **and** *trans-inv regs*
and *fetch-raises-ex ISA t* $\longrightarrow$ *ex-traces*
shows *store-mem-axiom CC ISA t*
and *store-tag-axiom CC ISA t*
and *load-mem-axiom CC ISA True t*
using *assms*
by (*intro traces-enabled-mem-axioms*[**where** *m* = *instr-fetch ISA* **and** *regs* = *regs*] *traces-enabled-instr-fetch*; *auto*)+

end

locale *CHERI-MIPS-Reg-Fetch-Automaton* = *CHERI-MIPS-Fixed-Trans* +
fixes *ex-traces* :: *bool*
begin

sublocale *Reg-Automaton?*: *Write-Cap-Inv-Automaton CC ISA ex-traces False get-regval set-regval trans-inv trans-regs* **..**

sublocale *CHERI-MIPS-Axiom-Inv-Automaton* **where** *enabled* = *enabled* **and** *invariant* = *trans-inv* **and** *inv-regs* = *trans-regs* **and** *translate-address* = *translate-address* **..**

sublocale *Mem-Automaton*: *CHERI-MIPS-Mem-Fetch-Automaton trans-regstate ex-traces* **..**

lemmas *non-cap-exp-traces-enabled*[*traces-enabledI*] = *non-cap-expI*[*THEN non-cap-exp-traces-enabledI*]

definition *PCC-accessible s regs* $\equiv$ *''PCC''* $\in$ *accessible-regs s* $\vee$ *regstate.PCC regs* $\in$ *derivable-caps s*

lemma
assumes *Run-inv* (*read-reg PCC-ref*) *t c regs* **and** *PCC-accessible s regs*
shows *c* $\in$ *derivable-caps* (*run s t*)

using *assms derivable-mono*[**where** $C = C$ **and** $C' = C \cup C'$ **for** C C']
unfolding *Run-inv-def PCC-accessible-def derivable-caps-def*
by (*fastforce simp*: *register-defs regval-of-Capability-def elim*!: *Run-read-regE intro*: *derivable.Copy*)

lemma *traces-enabled-write-cap-regs*[*traces-enabledI*]:
assumes $c \in$ *derivable-caps s*
shows *traces-enabled* (*write-reg C01-ref c*) *s regs*
and *traces-enabled* (*write-reg C02-ref c*) *s regs*
and *traces-enabled* (*write-reg C03-ref c*) *s regs*
and *traces-enabled* (*write-reg C04-ref c*) *s regs*
and *traces-enabled* (*write-reg C05-ref c*) *s regs*
and *traces-enabled* (*write-reg C06-ref c*) *s regs*
and *traces-enabled* (*write-reg C07-ref c*) *s regs*
and *traces-enabled* (*write-reg C08-ref c*) *s regs*
and *traces-enabled* (*write-reg C09-ref c*) *s regs*
and *traces-enabled* (*write-reg C10-ref c*) *s regs*
and *traces-enabled* (*write-reg C11-ref c*) *s regs*
and *traces-enabled* (*write-reg C12-ref c*) *s regs*
and *traces-enabled* (*write-reg C13-ref c*) *s regs*
and *traces-enabled* (*write-reg C14-ref c*) *s regs*
and *traces-enabled* (*write-reg C15-ref c*) *s regs*
and *traces-enabled* (*write-reg C16-ref c*) *s regs*
and *traces-enabled* (*write-reg C17-ref c*) *s regs*
and *traces-enabled* (*write-reg C18-ref c*) *s regs*
and *traces-enabled* (*write-reg C19-ref c*) *s regs*
and *traces-enabled* (*write-reg C20-ref c*) *s regs*
and *traces-enabled* (*write-reg C21-ref c*) *s regs*
and *traces-enabled* (*write-reg C22-ref c*) *s regs*
and *traces-enabled* (*write-reg C23-ref c*) *s regs*
and *traces-enabled* (*write-reg C24-ref c*) *s regs*
and *traces-enabled* (*write-reg C25-ref c*) *s regs*
and *traces-enabled* (*write-reg C26-ref c*) *s regs*
and *traces-enabled* (*write-reg C27-ref c*) *s regs*
and *traces-enabled* (*write-reg C28-ref c*) *s regs*
and *traces-enabled* (*write-reg C29-ref c*) *s regs*
and *traces-enabled* (*write-reg C30-ref c*) *s regs*
and *traces-enabled* (*write-reg C31-ref c*) *s regs*
and *traces-enabled* (*write-reg CPLR-ref c*) *s regs*
and *traces-enabled* (*write-reg CULR-ref c*) *s regs*
and *traces-enabled* (*write-reg DDC-ref c*) *s regs*
and *traces-enabled* (*write-reg DelayedPCC-ref c*) *s regs*
and *traces-enabled* (*write-reg EPCC-ref c*) *s regs*
and *traces-enabled* (*write-reg ErrorEPCC-ref c*) *s regs*
and *traces-enabled* (*write-reg KCC-ref c*) *s regs*
and *traces-enabled* (*write-reg KDC-ref c*) *s regs*
and *traces-enabled* (*write-reg KR1C-ref c*) *s regs*
and *traces-enabled* (*write-reg KR2C-ref c*) *s regs*
and *traces-enabled* (*write-reg NextPCC-ref c*) *s regs*

and *traces-enabled* (*write-reg PCC-ref c*) *s regs*
using *assms*
by (*intro traces-enabled-write-reg*; *auto simp*: *register-defs derivable-caps-def*)+

lemma *traces-enabled-write-reg-CapCause*[*traces-enabledI*]:
traces-enabled (*write-reg CapCause-ref c*) *s regs*
by (*intro traces-enabled-write-reg*; *auto simp*: *register-defs derivable-caps-def*)+

lemma *traces-enabled-read-cap-regs*[*traces-enabledI*]:
traces-enabled (*read-reg C01-ref*) *s regs*
traces-enabled (*read-reg C02-ref*) *s regs*
traces-enabled (*read-reg C03-ref*) *s regs*
traces-enabled (*read-reg C04-ref*) *s regs*
traces-enabled (*read-reg C05-ref*) *s regs*
traces-enabled (*read-reg C06-ref*) *s regs*
traces-enabled (*read-reg C07-ref*) *s regs*
traces-enabled (*read-reg C08-ref*) *s regs*
traces-enabled (*read-reg C09-ref*) *s regs*
traces-enabled (*read-reg C10-ref*) *s regs*
traces-enabled (*read-reg C11-ref*) *s regs*
traces-enabled (*read-reg C12-ref*) *s regs*
traces-enabled (*read-reg C13-ref*) *s regs*
traces-enabled (*read-reg C14-ref*) *s regs*
traces-enabled (*read-reg C15-ref*) *s regs*
traces-enabled (*read-reg C16-ref*) *s regs*
traces-enabled (*read-reg C17-ref*) *s regs*
traces-enabled (*read-reg C18-ref*) *s regs*
traces-enabled (*read-reg C19-ref*) *s regs*
traces-enabled (*read-reg C20-ref*) *s regs*
traces-enabled (*read-reg C21-ref*) *s regs*
traces-enabled (*read-reg C22-ref*) *s regs*
traces-enabled (*read-reg C23-ref*) *s regs*
traces-enabled (*read-reg C24-ref*) *s regs*
traces-enabled (*read-reg C25-ref*) *s regs*
traces-enabled (*read-reg C26-ref*) *s regs*
traces-enabled (*read-reg C27-ref*) *s regs*
traces-enabled (*read-reg C28-ref*) *s regs*
traces-enabled (*read-reg C29-ref*) *s regs*
traces-enabled (*read-reg C30-ref*) *s regs*
traces-enabled (*read-reg C31-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg CPLR-ref*) *s regs*
traces-enabled (*read-reg CULR-ref*) *s regs*
traces-enabled (*read-reg DDC-ref*) *s regs*
traces-enabled (*read-reg DelayedPCC-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg EPCC-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg ErrorEPCC-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg KCC-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg KDC-ref*) *s regs*

system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg KR1C-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg KR2C-ref*) *s regs*
system-reg-access s ∨ *ex-traces* ⟹ *traces-enabled* (*read-reg CapCause-ref*) *s regs*
traces-enabled (*read-reg NextPCC-ref*) *s regs*
traces-enabled (*read-reg PCC-ref*) *s regs*
by (*intro traces-enabled-read-reg*; *auto simp*: *register-defs*)+

lemma *read-cap-regs-derivable*[*derivable-capsE*]:
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C01\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C01''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C02\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C02''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C03\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C03''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C04\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C04''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C05\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C05''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C06\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C06''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C07\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C07''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C08\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C08''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C09\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C09''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C10\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C10''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C11\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C11''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C12\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C12''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C13\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C13''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C14\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C14''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C15\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C15''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C16\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C16''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C17\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C17''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C18\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C18''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C19\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C19''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C20\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C20''\} \subseteq accessible\text{-}regs\ s \Longrightarrow c \in derivable\text{-}caps\ (run\ s\ t)$
$\bigwedge t\ c\ regs\ s.\ Run\text{-}inv\ (read\text{-}reg\ C21\text{-}ref)\ t\ c\ regs \Longrightarrow \{''C21''\} \subseteq accessible\text{-}regs$

$s \Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg C22-ref*) *t c regs* $\Longrightarrow \{''C22''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg C23-ref*) *t c regs* $\Longrightarrow \{''C23''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg C24-ref*) *t c regs* $\Longrightarrow \{''C24''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg C25-ref*) *t c regs* $\Longrightarrow \{''C25''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg C26-ref*) *t c regs* $\Longrightarrow \{''C26''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg C27-ref*) *t c regs* $\Longrightarrow \{''C27''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg C28-ref*) *t c regs* $\Longrightarrow \{''C28''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg C29-ref*) *t c regs* $\Longrightarrow \{''C29''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg C30-ref*) *t c regs* $\Longrightarrow \{''C30''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg C31-ref*) *t c regs* $\Longrightarrow \{''C31''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg CPLR-ref*) *t c regs* $\Longrightarrow \{''CPLR''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg CULR-ref*) *t c regs* $\Longrightarrow \{''CULR''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg DDC-ref*) *t c regs* $\Longrightarrow \{''DDC''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg DelayedPCC-ref*) *t c regs* $\Longrightarrow \{''DelayedPCC''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg EPCC-ref*) *t c regs* $\Longrightarrow \{''EPCC''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg ErrorEPCC-ref*) *t c regs* $\Longrightarrow \{''ErrorEPCC''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg KCC-ref*) *t c regs* $\Longrightarrow \{''KCC''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg KDC-ref*) *t c regs* $\Longrightarrow \{''KDC''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg KR1C-ref*) *t c regs* $\Longrightarrow \{''KR1C''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg KR2C-ref*) *t c regs* $\Longrightarrow \{''KR2C''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg NextPCC-ref*) *t c regs* $\Longrightarrow \{''NextPCC''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
$\bigwedge t$ *c regs s. Run-inv* (*read-reg PCC-ref*) *t c regs* $\Longrightarrow \{''PCC''\} \subseteq$ *accessible-regs s* $\Longrightarrow c \in$ *derivable-caps* (*run s t*)
unfolding *C01-ref-def C02-ref-def C03-ref-def C04-ref-def C05-ref-def C06-ref-def C07-ref-def C08-ref-def C09-ref-def C10-ref-def C11-ref-def C12-ref-def C13-ref-def C14-ref-def C15-ref-def C16-ref-def C17-ref-def C18-ref-def C19-ref-def C20-ref-def*

C21-ref-def C22-ref-def C23-ref-def C24-ref-def C25-ref-def
C26-ref-def C27-ref-def C28-ref-def C29-ref-def C30-ref-def
C31-ref-def CPLR-ref-def CULR-ref-def DDC-ref-def DelayedPCC-ref-def
EPCC-ref-def ErrorEPCC-ref-def KCC-ref-def KDC-ref-def KR1C-ref-def
KR2C-ref-def NextPCC-ref-def PCC-ref-def Run-inv-def derivable-caps-def
by (*auto elim*!: *Run-read-regE intro*!: *derivable.Copy*)

lemma *traces-enabled-cp2-next-pc*[*traces-enabledI*]:
assumes {*''DelayedPCC''*, *''NextPCC''*} $\subseteq$ *accessible-regs s*
shows *traces-enabled* (*cp2-next-pc u*) *s regs*
unfolding *cp2-next-pc-def bind-assoc*
by (*traces-enabledI assms*: *assms simp*: *register-defs*)

lemma *traces-enabled-set-next-pcc-ex*:
assumes *arg0*: *arg0* $\in$ *exception-targets ISA* (*read-from-KCC s*) **and** *ex*: *ex-traces*
shows *traces-enabled* (*set-next-pcc arg0*) *s regs*
unfolding *set-next-pcc-def bind-assoc*
by (*traces-enabledI intro*: *traces-enabled-write-reg assms exception-targets-run-imp simp*: *register-defs*)

lemma *read-reg-PCC-from-iff*:
assumes *reads-regs-from trans-regs t regs*
defines *pcc* $\equiv$ *regstate.PCC regs*
and *e* $\equiv$ *E-read-reg ''PCC''* (*Regval-Capability* (*regstate.PCC regs*))
shows *Run* (*read-reg PCC-ref*) *t c* $\longleftrightarrow$ (*c* = *pcc* $\wedge$ *t* = [*e*])
using *assms*
by (*auto simp*: *read-reg-def register-defs regval-of-Capability-def elim*!: *Read-reg-TracesE*)

lemma *read-reg-PCC-from-bind-iff*:
assumes *reads-regs-from trans-regs t regs*
defines *pcc* $\equiv$ *regstate.PCC regs*
and *e* $\equiv$ *E-read-reg ''PCC''* (*Regval-Capability* (*regstate.PCC regs*))
shows *Run* (*read-reg PCC-ref* $\gg=$ *f*) *t a* $\longleftrightarrow$ ($\exists$ *tf*. *t* = *e* # *tf* $\wedge$ *Run* (*f pcc*) *tf a*)
using *assms*
by (*auto elim*!: *Run-bindE simp*: *read-reg-PCC-from-iff regstate-simp*
intro!: *Traces-bindI*[*of read-reg PCC-ref* [*e*], *unfolded e-def*, *simplified*])

lemmas *read-reg-PCC-from-iffs* = *read-reg-PCC-from-iff read-reg-PCC-from-bind-iff*

lemma *Run-read-accessible-PCC-derivable*:
assumes *Run* (*read-reg PCC-ref*) *t c* **and** *reads-regs-from trans-regs t regs* **and** *PCC-accessible s regs*
shows *c* $\in$ *derivable-caps* (*run s t*)
using *assms derivable-mono*[*OF Un-upper1*, *THEN in-mono*]
by (*auto simp*: *register-defs regval-of-Capability-def derivable-caps-def PCC-accessible-def*
elim!: *Run-read-regE intro*: *derivable.Copy*)

lemma *Run-write-derivable-PCC-accessible*:

assumes *Run* (*write-reg PCC-ref c*) *t a* **and** *reads-regs-from Rs t regs* **and** $''PCC'' \in Rs$
and $c \in$ *derivable-caps s*
shows *PCC-accessible* (*run s t*) (*the* (*updates-regs Rs t regs*))
using *assms*
by (*auto simp*: *PCC-accessible-def register-defs derivable-caps-def elim*!: *Mem-Automaton.Run-write-regE*)

lemma *Run-PCC-accessible-run*:
assumes *Run m t a* **and** *runs-no-reg-writes-to* $\{''PCC''\}$ *m* **and** *PCC-accessible s regs*
shows *PCC-accessible* (*run s t*) *regs*
using *assms derivable-caps-run-mono*[*of s t*]
by (*auto simp*: *PCC-accessible-def accessible-regs-def Run-runs-no-reg-writes-written-regs-eq*)

lemmas *Run-inv-PCC-accessible-run* = *Run-inv-RunI*[*THEN Run-PCC-accessible-run*]

lemma *Run-runs-no-reg-writes-to-updates-regs-inv*[*simp*]:
assumes *Run m t a* **and** *reads-regs-from Rs t regs* **and** *runs-no-reg-writes-to Rs m*
shows *updates-regs Rs t regs* = *Some regs*
proof −
have $\forall r \in Rs.\ \forall v.$ *E-write-reg r v* $\notin$ *set t*
using *assms*
by (*auto simp*: *runs-no-reg-writes-to-def Run-inv-def*)
then show *updates-regs Rs t regs* = *Some regs*
by (*induction Rs t regs rule*: *updates-regs.induct*) *auto*
qed

lemma *Run-runs-no-reg-writes-to-get-regval-eq*[*simp*]:
assumes *Run m t a* **and** *reads-regs-from Rs t regs* **and** *runs-no-reg-writes-to* $\{r\}$ *m*
shows *get-regval r* (*the* (*updates-regs Rs t regs*)) = *get-regval r regs*
proof −
have $\forall v.$ *E-write-reg r v* $\notin$ *set t*
using *assms*
by (*auto simp*: *runs-no-reg-writes-to-def Run-inv-def*)
then show *?thesis*
using *assms*(*2*)
by (*induction Rs t regs rule*: *updates-regs.induct*)
(*auto split*: *Option.bind-splits if-splits simp*: *get-ignore-set-regval*)
qed

lemma *Run-PCC-accessible-update*:
assumes *Run m t a* **and** *reads-regs-from Rs t regs* **and** *runs-no-reg-writes-to* $\{''PCC''\}$ *m*
and *PCC-accessible s regs*
shows *PCC-accessible s* (*the* (*updates-regs Rs t regs*))
proof −

have *get-regval ''PCC''* (*the* (*updates-regs Rs t regs*)) = *get-regval ''PCC'' regs*
using *assms*
by *auto*
then show *?thesis*
using ‹*PCC-accessible s regs*›
by (*auto simp*: *PCC-accessible-def register-defs regval-of-Capability-def*)
qed

lemma *Run-inv-PCC-accessible-update*:
assumes *Run-inv m t a regs* **and** *runs-no-reg-writes-to* {*''PCC''*} *m*
and *PCC-accessible s regs*
shows *PCC-accessible s* (*the* (*updates-regs trans-regs t regs*))
using *assms*
by (*intro Run-PCC-accessible-update*) (*auto simp*: *Run-inv-def*)

lemma *Run-PCC-accessible-run-update*:
assumes *Run m t a* **and** *reads-regs-from Rs t regs* **and** *runs-no-reg-writes-to* {*''PCC''*} *m*
and *PCC-accessible s regs*
shows *PCC-accessible* (*run s t*) (*the* (*updates-regs Rs t regs*))
using *assms*
by (*blast intro*: *Run-PCC-accessible-run Run-PCC-accessible-update*)

lemma *Run-inv-PCC-accessible-run-update*:
assumes *Run-inv m t a regs* **and** *runs-no-reg-writes-to* {*''PCC''*} *m*
and *PCC-accessible s regs*
shows *PCC-accessible* (*run s t*) (*the* (*updates-regs trans-regs t regs*))
using *assms*
by (*blast intro*: *Run-inv-PCC-accessible-update Run-inv-PCC-accessible-run*)

lemmas *Run-PCC-accessibleE*[*derivable-capsE*] =
Run-PCC-accessible-run-update Run-PCC-accessible-update Run-PCC-accessible-run
Run-inv-PCC-accessible-run-update Run-inv-PCC-accessible-update Run-inv-PCC-accessible-run

lemma (**in** *Register-State*) *reads-regs-bind-updates-regs-the*[*simp*]:
assumes *reads-regs-from R t s*
shows *Option.bind* (*updates-regs R t s*) *f* = *f* (*the* (*updates-regs R t s*))
using *assms*
by (*elim reads-regs-from-updates-regs-Some*) *auto*

find-theorems *NextPCC-ref derivable-caps*

lemma *Run-inv-cp2-next-pc-PCC-accessible*:
assumes *Run-inv* (*cp2-next-pc u*) *t a regs* **and** {*''NextPCC''*} ⊆ *accessible-regs s*
shows *PCC-accessible* (*run s t*) (*the* (*updates-regs trans-regs t regs*))
proof −
have ∗: *PCC-accessible s* (*regs*⦇*regstate.PCC* := *c*⦈) **if** *c* ∈ *derivable-caps s* **for** *s c* **and** *regs* :: *regstate*

using *that*
by (*auto simp*: *PCC-accessible-def*)
show *?thesis*
using *assms*
unfolding *cp2-next-pc-def bind-assoc*
by (*auto elim*!: *Run-inv-bindE Run-inv-ifE intro*: *preserves-invariantI traces-runs-preserve-invariantI intro*!: ∗ *simp*: *regstate-simp*)
(*derivable-capsI*)+
qed

lemma *SignalException-trace-enabled*:
assumes (*SignalException arg0*, *t*, *m′*) ∈ *Traces* **and** *reads-regs-from trans-regs t regs*
and *PCC-accessible s regs* **and** *ex*: *ex-traces*
shows *trace-enabled s t*
proof −
note [*trace-elim*] = *non-cap-expI*[*THEN non-cap-exp-trace-enabledI*]
have [*trace-elim*]: (*read-reg PCC-ref*, *t*, *m′*) ∈ *Traces* ⟹ *trace-enabled s t* **for** *s t* **and** *m′* :: *Capability M*
by (*elim read-reg-trace-enabled*; *auto simp*: *register-defs*)
have [*trace-elim*]: (*read-reg KCC-ref*, *t*, *m′*) ∈ *Traces* ⟹ *trace-enabled s t* **for** *s t* **and** *m′* :: *Capability M*
by (*elim read-reg-trace-enabled*; *auto simp*: *register-defs intro*: *ex*)
have [*trace-elim*]: (*write-reg EPCC-ref c*, *t*, *m′*) ∈ *Traces* ⟹ *c* ∈ *derivable-caps s* ⟹ *trace-enabled s t* **for** *s t c* **and** *m′* :: *unit M*
by (*elim write-reg-trace-enabled*) (*auto simp*: *derivable-caps-def register-defs*)
have [*trace-elim*]: (*set-next-pcc c*, *t*, *m′*) ∈ *Traces* ⟹ *c* ∈ *exception-targets ISA* (*read-from-KCC s*) ⟹ *trace-enabled s t* **for** *s t c* **and** *m′* :: *unit M*
unfolding *set-next-pcc-def*
by (*elim trace-elim write-reg-trace-enabled*)
(*auto simp*: *register-defs intro*: *ex exception-targets-run-imp*)
have [*trace-elim*]: (*set-CauseReg-BD CP0Cause-ref x*, *t*, *m′*) ∈ *Traces* ⟹ *trace-enabled s t* **for** *s t x m′*
unfolding *set-CauseReg-BD-def*
by (*elim trace-elim*)
have [*trace-elim*]: (*set-CauseReg-ExcCode CP0Cause-ref x*, *t*, *m′*) ∈ *Traces* ⟹ *trace-enabled s t* **for** *s t x m′*
unfolding *set-CauseReg-ExcCode-def*
by (*elim trace-elim*)
have [*trace-elim*]: (*set-StatusReg-EXL CP0Status-ref x*, *t*, *m′*) ∈ *Traces* ⟹ *trace-enabled s t* **for** *s t x m′*
unfolding *set-StatusReg-EXL-def*
by (*elim trace-elim*)
have *read-KCC-ex-target*: *c* ∈ *exception-targets ISA* (*read-from-KCC* (*Mem-Automaton.run s t*))
if *Run* (*read-reg KCC-ref*) *t c* **for** *s t c*
using *that*
by (*auto elim*!: *Run-read-regE simp*: *register-defs*)
note [*derivable-capsE*] = *Run-read-accessible-PCC-derivable*[**where** *regs* = *regs*]

show *?thesis*
using *assms*(*1*−*3*)
unfolding *SignalException-def bind-assoc*
by (*elim trace-elim read-KCC-ex-target*)
(*derivable-capsI simp*: *regstate-simp read-reg-PCC-from-iffs*)
qed

lemma *traces-enabled-SignalException*[*traces-enabledI*]:
assumes *PCC-accessible s regs*
shows *traces-enabled* (*SignalException arg0* :: *'a M*) *s regs*
proof *cases*
assume *ex*: *ex-traces*
then show *?thesis*
using *assms SignalException-trace-enabled*
unfolding *traces-enabled-def*
by *blast*
next
assume ¬ *ex-traces*
then show *?thesis*
unfolding *traces-enabled-def finished-def isException-def*
by *auto*
qed

lemma [*simp*]:
name CP0Count-ref = *''CP0Count''*
name TLBRandom-ref = *''TLBRandom''*
name CP0BadVAddr-ref = *''CP0BadVAddr''*
name CapCause-ref = *''CapCause''*
name BranchPending-ref = *''BranchPending''*
name NextInBranchDelay-ref = *''NextInBranchDelay''*
name InBranchDelay-ref = *''InBranchDelay''*
name PC-ref = *''PC''*
name NextPC-ref = *''NextPC''*
name InstCount-ref = *''InstCount''*
name CurrentInstrBits-ref = *''CurrentInstrBits''*
by (*auto simp*: *register-defs*)

lemma [*simp*]:
''CP0Count'' ∉ *trans-regs*
''TLBRandom'' ∉ *trans-regs*
''CP0BadVAddr'' ∉ *trans-regs*
''CapCause'' ∉ *trans-regs*
''BranchPending'' ∉ *trans-regs*
''NextInBranchDelay'' ∉ *trans-regs*
''InBranchDelay'' ∉ *trans-regs*
''PC'' ∉ *trans-regs*
''NextPC'' ∉ *trans-regs*
''InstCount'' ∉ *trans-regs*
''CurrentInstrBits'' ∉ *trans-regs*

by (*auto simp*: *trans-regs-def*)

lemma *traces-enabled-SignalExceptionBadAddr*[*traces-enabledI*]:
assumes *PCC-accessible s regs*
shows *traces-enabled* (*SignalExceptionBadAddr arg0 arg1*) *s regs*
unfolding *SignalExceptionBadAddr-def*
by (*traces-enabledI assms*: *assms*)

lemma *SignalExceptionTLB-trace-enabled*:
assumes (*SignalExceptionTLB arg0 arg1* :: $'a$ *M*, *t*, m') ∈ *Traces* **and** *reads-regs-from trans-regs t regs*
and *PCC-accessible s regs* **and** *ex*: *ex-traces*
shows *trace-enabled s t*
proof −
have [*trace-elim*]: (*write-reg CP0BadVAddr-ref v*, *t*, m') ∈ *Traces* ⟹ *trace-enabled s t* **for** *s t v* **and** m' :: *unit M*
by (*auto elim*!: *write-reg-trace-enabled simp*: *register-defs*)
have [*trace-elim*]: (*set-ContextReg-BadVPN2 TLBContext-ref v*, *t*, m') ∈ *Traces* ⟹ *trace-enabled s t* **for** *s t v* **and** m' :: *unit M*
by (*auto elim*!: *trace-elim read-reg-trace-enabled write-reg-trace-enabled simp*: *set-ContextReg-BadVPN2-def register-defs*)
have [*trace-elim*]: (*set-XContextReg-XBadVPN2 TLBXContext-ref v*, *t*, m') ∈ *Traces* ⟹ *trace-enabled s t* **for** *s t v* **and** m' :: *unit M*
by (*auto elim*!: *trace-elim read-reg-trace-enabled write-reg-trace-enabled simp*: *set-XContextReg-XBadVPN2-def register-defs*)
have [*trace-elim*]: (*set-XContextReg-XR TLBXContext-ref v*, *t*, m') ∈ *Traces* ⟹ *trace-enabled s t* **for** *s t v* **and** m' :: *unit M*
by (*auto elim*!: *trace-elim read-reg-trace-enabled write-reg-trace-enabled simp*: *set-XContextReg-XR-def register-defs*)
have [*trace-elim*]: (*set-TLBEntryHiReg-R TLBEntryHi-ref v*, *t*, m') ∈ *Traces* ⟹ *trace-enabled s t* **for** *s t v* **and** m' :: *unit M*
by (*auto elim*!: *trace-elim read-reg-trace-enabled write-reg-trace-enabled simp*: *set-TLBEntryHiReg-R-def register-defs*)
have [*trace-elim*]: (*set-TLBEntryHiReg-VPN2 TLBEntryHi-ref v*, *t*, m') ∈ *Traces* ⟹ *trace-enabled s t* **for** *s t v* **and** m' :: *unit M*
by (*auto elim*!: *trace-elim read-reg-trace-enabled write-reg-trace-enabled simp*: *set-TLBEntryHiReg-VPN2-def register-defs*)
note [*derivable-capsI*] = *ex*
show *?thesis*
using *assms*(*1*−*3*)
unfolding *SignalExceptionTLB-def bind-assoc*
by (*elim trace-elim SignalException-trace-enabled*)
(*derivable-capsI simp*: *regstate-simp*)+
qed

lemma *traces-enabled-SignalExceptionTLB*[*traces-enabledI*]:
assumes *PCC-accessible s regs*
shows *traces-enabled* (*SignalExceptionTLB arg0 arg1*) *s regs*
proof *cases*

assume *ex*: *ex-traces*
show *?thesis*
unfolding *traces-enabled-def*
using *assms*
by (*auto elim*!: *SignalExceptionTLB-trace-enabled intro*: *ex*)
next
assume ¬*ex-traces*
then show *?thesis*
unfolding *traces-enabled-def finished-def isException-def*
by *auto*
qed

lemma *traces-enabled-incrementCP0Count*[*traces-enabledI*]:
assumes *PCC-accessible s regs*
shows *traces-enabled* (*incrementCP0Count u*) *s regs*
unfolding *incrementCP0Count-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-raise-c2-exception8*[*traces-enabledI*]:
assumes *PCC-accessible s regs*
shows *traces-enabled* (*raise-c2-exception8 arg0 arg1*) *s regs*
proof *cases*
assume *ex*: *ex-traces*
have *1*: *traces-enabled* (*set-CapCauseReg-ExcCode CapCause-ref x*) *s regs* **for** *x s regs*
unfolding *set-CapCauseReg-ExcCode-def*
by (*traces-enabledI assms*: *ex*)
have *2*: *traces-enabled* (*set-CapCauseReg-RegNum CapCause-ref x*) *s regs* **for** *x s regs*
unfolding *set-CapCauseReg-RegNum-def*
by (*traces-enabledI assms*: *ex*)
show *?thesis*
unfolding *raise-c2-exception8-def bind-assoc*
by (*traces-enabledI assms*: *assms intro*: *1 2*)
next
assume ¬ *ex-traces*
then show *?thesis*
unfolding *traces-enabled-def finished-def isException-def*
by *auto*
qed

lemma *traces-enabled-raise-c2-exception-noreg*[*traces-enabledI*]:
assumes *PCC-accessible s regs*
shows *traces-enabled* (*raise-c2-exception-noreg arg0*) *s regs*
unfolding *raise-c2-exception-noreg-def*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-TLBTranslate2*[*traces-enabledI*]:
assumes *PCC-accessible s regs*

shows *traces-enabled* (*TLBTranslate2 arg0 arg1*) *s regs*
unfolding *TLBTranslate2-def*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-TLBTranslateC*[*traces-enabledI*]:
assumes *PCC-accessible s regs*
shows *traces-enabled* (*TLBTranslateC arg0 arg1*) *s regs*
unfolding *TLBTranslateC-def*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-TLBTranslate*[*traces-enabledI*]:
assumes *PCC-accessible s regs*
shows *traces-enabled* (*TLBTranslate arg0 arg1*) *s regs*
unfolding *TLBTranslate-def*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-TranslatePC*[*traces-enabledI*]:
assumes *PCC-accessible s regs*
shows *traces-enabled* (*TranslatePC vaddr*) *s regs*
unfolding *TranslatePC-def bind-assoc*
by (*traces-enabledI assms*: *assms*)

lemma *traces-enabled-MEMr*[*traces-enabledI*]:
shows *traces-enabled* (*MEMr arg0 arg1*) *s regs*
unfolding *MEMr-def read-mem-def read-mem-bytes-def maybe-fail-def bind-assoc*
by (*auto simp*: *traces-enabled-def elim*!: *bind-Traces-cases split*: *option.splits elim*: *Traces-cases*)

lemma *traces-enabled-MEMr-wrapper*[*traces-enabledI*]:
shows *traces-enabled* (*MEMr-wrapper arg0 arg1*) *s regs*
unfolding *MEMr-wrapper-def bind-assoc*
by (*traces-enabledI-with* ‹−›)

lemma *traces-enabled-fetch*[*traces-enabledI*]:
assumes {''*DelayedPCC*'', ''*NextPCC*'', ''*PCC*''} ⊆ *accessible-regs s*
shows *traces-enabled* (*fetch u*) *s regs*
unfolding *fetch-def bind-assoc*
by (*traces-enabledI elim*: *Run-inv-cp2-next-pc-PCC-accessible assms*: *assms simp*: *register-defs*)

lemma *traces-enabled-instr-fetch*[*traces-enabledI*]:
assumes {''*DelayedPCC*'', ''*NextPCC*'', ''*PCC*''} ⊆ *accessible-regs s*
shows *traces-enabled* (*instr-fetch ISA*) *s regs*
unfolding *ISA-simps*
by (*traces-enabledI assms*: *assms*)

lemma *hasTrace-fetch-reg-axioms*:
assumes *hasTrace t* (*instr-fetch ISA*)
and *reads-regs-from trans-regs t regs* **and** *trans-inv regs*

and *fetch-raises-ex ISA t* $\longrightarrow$ *ex-traces*
shows *store-cap-reg-axiom CC ISA ex-traces False t*
and *store-cap-mem-axiom CC ISA t*
and *read-reg-axiom CC ISA ex-traces t*
using *assms*
by (*intro traces-enabled-reg-axioms*[**where** *m* = *instr-fetch ISA* **and** *regs* = *regs*] *traces-enabled-instr-fetch*; *auto*)+

end

end
theory *CHERI-MIPS-Properties*
imports *CHERI-MIPS-Reg-Axioms CHERI-MIPS-Mem-Axioms Properties*
begin

3.6 Instantiation of monotonicity result

context *CHERI-MIPS-Reg-Automaton*
begin

lemma *runs-no-reg-writes-to-incrementCP0Count*[*runs-no-reg-writes-toI*]:
assumes {''*TLBRandom*'', ''*CP0Count*'', ''*CP0Cause*''} ∩ *Rs* = {}
shows *runs-no-reg-writes-to Rs* (*incrementCP0Count u*)
using *assms*
unfolding *incrementCP0Count-def bind-assoc Let-def*
by (*no-reg-writes-toI simp*: *TLBRandom-ref-def CP0Count-ref-def CP0Cause-ref-def*)

lemma *TranslatePC-establishes-inv*:
assumes *Run* (*TranslatePC vaddr*) *t a* **and** *reads-regs-from* {''*PCC*''} *t s*
shows *invariant s*
using *assms*
unfolding *TranslatePC-def bind-assoc Let-def*
by (*auto elim*!: *Run-bindE Run-read-regE split*: *if-splits*
simp: *regstate-simp register-defs regval-of-Capability-def*)

lemma *not-PCC-regs*[*simp*]:
name PC-ref ≠ ''*PCC*''
name InBranchDelay-ref ≠ ''*PCC*''
name NextPC-ref ≠ ''*PCC*''
name NextInBranchDelay-ref ≠ ''*PCC*''
name BranchPending-ref ≠ ''*PCC*''
name CurrentInstrBits-ref ≠ ''*PCC*''
name LastInstrBits-ref ≠ ''*PCC*''
name UART-WRITTEN-ref ≠ ''*PCC*''
name InstCount-ref ≠ ''*PCC*''
by (*auto simp*: *register-defs*)

lemma *fetch-establishes-inv*:
assumes *Run* (*fetch u*) *t a* **and** *reads-regs-from* {''*PCC*''} *t s*

shows *invariant* (*the* (*updates-regs* {''*PCC*''} *t s*))
using *assms*
unfolding *fetch-def bind-assoc Let-def*
by (*auto elim*!: *Run-bindE simp*: *regstate-simp dest*: *TranslatePC-establishes-inv*)

lemma *instr-fetch-establishes-inv*:
assumes *Run* (*instr-fetch ISA*) *t a* **and** *reads-regs-from* {''*PCC*''} *t s*
shows *invariant* (*the* (*updates-regs* {''*PCC*''} *t s*))
using *assms*
by (*auto simp*: *ISA-def elim*!: *Run-bindE split*: *option.splits dest*: *fetch-establishes-inv*)

end

lemma (**in** *CHERI-MIPS-Mem-Automaton*) *preserves-invariant-instr-fetch*[*preserves-invariantI*]:
shows *runs-preserve-invariant* (*instr-fetch ISA*)
by (*auto simp*: *ISA-def intro*!: *preserves-invariantI*; *simp add*: *runs-preserve-invariant-def*)

context *CHERI-MIPS-Fixed-Trans*
begin

definition *state-assms t reg-s mem-s* ≡ *reads-regs-from trans-regs t mem-s* ∧ *reads-regs-from* {''*PCC*''} *t reg-s* ∧ *trans-inv mem-s*
definition *fetch-assms t* ≡ (∃ *reg-s mem-s*. *state-assms t reg-s mem-s*)
definition *instr-assms t* ≡ (∃ *reg-s mem-s*. *state-assms t reg-s mem-s* ∧ *CHERI-MIPS-Reg-Automaton.invariant reg-s*)

sublocale *CHERI-ISA* **where** *CC* = *CC* **and** *ISA* = *ISA* **and** *fetch-assms* = *fetch-assms* **and** *instr-assms* = *instr-assms*
proof
fix *t instr*
interpret *Reg-Axioms*: *CHERI-MIPS-Reg-Automaton*
where *ex-traces* = *instr-raises-ex ISA instr t*
and *invocation-traces* = *invokes-caps ISA instr t*
and *translate-address* = *translate-address* **.**
interpret *Mem-Axioms*: *CHERI-MIPS-Mem-Instr-Automaton trans-regstate instr-raises-ex ISA instr t*
by *unfold-locales*
assume *t*: *hasTrace t* (*instr-sem ISA instr*) **and** *instr-assms t*
then obtain *reg-s mem-s*
where *reg-assms*: *reads-regs-from* {''*PCC*''} *t reg-s Reg-Axioms.invariant reg-s*
and *mem-assms*: *reads-regs-from trans-regs t mem-s trans-inv mem-s*
by (*auto simp*: *instr-assms-def state-assms-def*)
show *cheri-axioms CC ISA False* (*instr-raises-ex ISA instr t*)
(*invokes-caps ISA instr t*) *t*
unfolding *cheri-axioms-def*
using *Reg-Axioms.hasTrace-instr-reg-axioms*[*OF t reg-assms*]
using *Mem-Axioms.hasTrace-instr-mem-axioms*[*OF t mem-assms*]
by *auto*
next

fix *t*
interpret *Fetch-Axioms*: *CHERI-MIPS-Reg-Fetch-Automaton trans-regstate fetch-raises-ex ISA t* **..**
assume *t*: *hasTrace t* (*instr-fetch ISA*) **and** *fetch-assms t*
then obtain *regs* **where** $*$: *reads-regs-from trans-regs t regs trans-inv regs*
by (*auto simp*: *fetch-assms-def state-assms-def*)
then show *cheri-axioms CC ISA True* (*fetch-raises-ex ISA t*) *False t*
unfolding *cheri-axioms-def*
using *Fetch-Axioms.hasTrace-fetch-reg-axioms*[*OF t* $*$]
using *Fetch-Axioms.Mem-Automaton.hasTrace-fetch-mem-axioms*[*OF t* $*$]
by *auto*
next
fix *t* t' *instr*
interpret *Mem-Axioms*: *CHERI-MIPS-Mem-Instr-Automaton trans-regstate* **by** *unfold-locales*
assume $*$: *instr-assms* (t @ t') **and** $**$: *Run* (*instr-sem ISA instr*) *t* ()
have *trans-inv* (*the* (*updates-regs trans-regs t mem-s*))
if *trans-inv mem-s* **and** *reads-regs-from trans-regs t mem-s* **for** *mem-s*
using *Mem-Axioms.preserves-invariant-execute*[*of instr*] *that* $**$
by (*elim runs-preserve-invariantE*[**where** $t = t$ **and** $s =$ *mem-s* **and** $a = ()$])
(*auto simp*: *instr-assms-def state-assms-def regstate-simp*)
with $*$ **show** *instr-assms t* $\wedge$ *fetch-assms* t'
by (*auto simp*: *instr-assms-def fetch-assms-def state-assms-def regstate-simp*)
next
fix *t* t' *instr*
interpret *Reg-Axioms*: *CHERI-MIPS-Reg-Automaton*
where *ex-traces* = *fetch-raises-ex ISA t*
and *invocation-traces* = *False*
and *translate-address* = *translate-address* **.**
interpret *Mem-Axioms*: *CHERI-MIPS-Mem-Automaton trans-regstate* **by** *unfold-locales*
assume $*$: *fetch-assms* (t @ t') **and** $**$: *Run* (*instr-fetch ISA*) *t instr*
have $***$: *trans-inv* (*the* (*updates-regs trans-regs t regs*))
if *reads-regs-from trans-regs t regs* **and** *trans-inv regs* **for** *regs*
using $**$ *that*
by (*elim runs-preserve-invariantE*[*OF Mem-Axioms.preserves-invariant-instr-fetch*]) *auto*
show *fetch-assms t* $\wedge$ *instr-assms* t'
using $*$ $**$
unfolding *fetch-assms-def instr-assms-def state-assms-def*
by (*fastforce simp*: *regstate-simp elim*!: *Run-bindE split*: *option.splits*
dest: *Reg-Axioms.fetch-establishes-inv* $***$)
qed

lemma *translate-address-tag-aligned*:
fixes *s* :: *regstate sequential-state*
assumes *translate-address vaddr acctype s* = *Some paddr*
shows *address-tag-aligned ISA paddr* = *address-tag-aligned ISA vaddr*
(**is** *?aligned paddr* = *?aligned vaddr*)
proof −

interpret *CHERI-MIPS-Mem-Automaton* **..**
have [*simp*]: *?aligned* (*unat* (*word-of-int* (*int vaddr*) :: *64 word*)) ⟷ *?aligned vaddr*
unfolding *address-tag-aligned-def*
by (*auto simp*: *unat-def uint-word-of-int nat-mod-distrib nat-power-eq mod-mod-cancel*)
from *assms* **obtain** *t regs* **where** *Run-inv* (*translate-addressM vaddr acctype*) *t paddr regs*
by (*auto simp*: *translate-address-def determ-the-result-eq*[*OF determ-runs-translate-addressM*] *split*: *if-splits*)
then show *?thesis*
by (*auto simp*: *translate-addressM-def elim*!: *Run-inv-bindE intro*: *preserves-invariantI*)
qed

sublocale *CHERI-ISA-State* **where** *CC* = *CC* **and** *ISA* = *ISA*
and *read-regval* = *get-regval* **and** *write-regval* = *set-regval*
and *fetch-assms* = *fetch-assms* **and** *instr-assms* = *instr-assms*
and *s-translation-tables* = λ-. {} **and** *s-translate-address* = *translate-address*
using *get-absorb-set-regval get-ignore-set-regval translate-address-tag-aligned*
by *unfold-locales* (*auto simp*: *ISA-def translate-address-def*)

thm *reachable-caps-instrs-trace-intradomain-monotonicity*

lemma *regstate-put-mem-bytes-eq*[*simp*]:
regstate (*put-mem-bytes addr sz v tag s*) = *regstate s*
by (*auto simp*: *put-mem-bytes-def Let-def*)

lemma *set-regval-Some-Some*:
assumes *set-regval r v s* = *Some s1*
obtains *s1'* **where** *set-regval r v s'* = *Some s1'*
using *assms*
by (*elim set-regval-cases of-regval-SomeE*) (*auto simp*: *register-defs*)

lemma *get-regval-eq-reads-regs-imp*:
assumes ∀ *r* ∈ *Rs*. *get-regval r s* = *get-regval r s'*
and *reads-regs-from Rs t s'*
shows *reads-regs-from Rs t s*
proof (*use assms* **in** ‹*induction t arbitrary*: *s s'*›)
case (*Cons e t*)
then show *?case*
proof (*cases e*)
fix *r v*
assume *e*: *e* = *E-write-reg r v*
with *Cons* **show** *?thesis*
proof *cases*
assume *r*: *r* ∈ *Rs*
with *Cons.prems e* **obtain** *s1'* **where** *s1'*: *set-regval r v s'* = *Some s1'* **and** ∗: *reads-regs-from Rs t s1'*
by (*auto split*: *if-splits option.splits*)
moreover obtain *s1* **where** *s1*: *set-regval r v s* = *Some s1*

by (*rule set-regval-Some-Some*[*OF s1'*])
have $**$: $\forall r' \in Rs.$ *get-regval* r' *s1* = *get-regval* r' *s1'*
proof
fix r'
assume $r' \in Rs$
then show *get-regval* r' *s1* = *get-regval* r' *s1'*
using *s1 s1' Cons.prems*
by (*cases* $r' = r$) (*auto simp*: *get-absorb-set-regval get-ignore-set-regval*)
qed
show *?thesis*
using *e r s1 Cons.IH*[*OF* $**$ $*$]
by *auto*
qed *auto*
qed (*auto split*: *if-splits option.splits*)
qed *auto*

lemma *set-other-reg-reads-regs-iff*:
assumes *set-regval r v s* = *Some* s' **and** $r \notin Rs$
shows *reads-regs-from Rs t* s' = *reads-regs-from Rs t s*
proof −
have $\forall r' \in Rs.$ *get-regval* r' *s* = *get-regval* r' s'
using *assms get-ignore-set-regval*
by *fastforce*
then show *?thesis*
using *get-regval-eq-reads-regs-imp*[*of Rs s* s' *t*]
using *get-regval-eq-reads-regs-imp*[*of Rs* s' *s t*]
by *auto*
qed

lemma *reads-regs-from-mono*:
assumes *reads-regs-from Rs t s*
and $Rs' \subseteq Rs$
shows *reads-regs-from* Rs' *t s*
using *assms*
by (*induction Rs t s rule*: *reads-regs-from.induct*)
(*auto split*: *if-splits option.splits dest*: *set-other-reg-reads-regs-iff*[**where** *Rs* = *Rs'*])

lemma *s-invariant-trivial*:
assumes *t*: *s-allows-trace t s* **and** *f*: $\bigwedge s'.\ f\ s' = f\ s$
shows *s-invariant f t s*
proof −
have *f*: *f s1* = *f s2* **for** *s1 s2*
using *f*[*of s1*] *f*[*of s2*, *symmetric*]
by *auto*
obtain s' **where** *s-run-trace t s* = *Some* s'
using *t*

by *blast*
then show *s-invariant f t s*
by (*induction* (*get-regval*, *set-regval*) *t s rule*: *runTraceS.induct*)
(*auto split*: *Option.bind-splits intro*: *f*)
qed

theorem *cheri-mips-cap-monotonicity*:
assumes *t*: *hasTrace t* (*fetch-execute-loop ISA n*)
and *s*: *s-run-trace t s* = *Some s′*
and *regs*: *reads-regs-from trans-regs t trans-regstate* — Fixes contents of address translation control registers and implies that we are in user mode via the assumption *noCP0Access trans-regstate*.
and *no-ex*: ¬ *instrs-raise-ex ISA n t*
and *no-ccall*: ¬ *instrs-invoke-caps ISA n t*
shows *reachable-caps s′* ⊆ *reachable-caps s*
proof (*intro reachable-caps-instrs-trace-intradomain-monotonicity*[*OF t - s no-ex no-ccall*])
have *state-assms t trans-regstate trans-regstate*
using *regs*
by (*auto simp*: *state-assms-def trans-regs-def elim*: *reads-regs-from-mono*)
then show *fetch-assms t*
by (*auto simp*: *fetch-assms-def*)
show *s-invariant* (λ*s′ addr load*. *local.translate-address addr load s′*) *t s*
and *s-invariant* (λ-. {}) *t s*
and *s-invariant-holds no-caps-in-translation-tables t s*
using *s*
by (*auto simp*: *translate-address-def no-caps-in-translation-tables-def intro*: *s-invariant-trivial*)
qed

end

end